Organización de Computadoras II (Sistemas Embebidos ATOM)

Juan Meza
Janeth Aguilar
Salomon Ibarra
Ruben Estrada

Introduccion a las herramientas de programacion GNU para embebidos

Una característica de las GCC, que además es muy común encontrar en el software libre, es que el proyecto no ha seguido un proceso formal de ingenierías con fases de análisis, diseño, implementación y pruebas. GCC esta siendo creado por cientos de programadores de distintas nacionalidades trabajando sobre esta entremezclada con el código fuente a base de comentarios. Aun asi han conseguido crear una herramienta de calidad, estable, rápida y con un bajo consumo de memoria.

Frontend y backend

A groso modo podemos dividir el proceso de compilación en dos partes: Frontend y backend. El frontend (donde se realiza el análisis léxico y gramatical) lee un archivo de código fuente y crea una estructura en forma de árbol en memoria de acuerdo a la gramatica del lenguaje. El backend lee este árbol y lo convierte en una secuencia de instrucciones ensamblador para la maquina destino. Dentro del backend podemos incluir tanto el análisis semántico, como la generación de código y la optimización.

Aunque los lenguajes que GCC puede compilar son muy distintos, a medida que avanzan las frases del frontend se va a generando una estructura comin a todos los lenguajes, de forma que un mismo backend puede generar código para distintos lenguajes. Otra característica de GCC es que usando distintos backend podemos generar código para distintas plataformas sin necesidad de disponer de un frontend para cada plataforma. Esto ha permitido que actualmente las GCC sean capaces de generar código para casi todos los tipos de hardware existentes.

Fases del compilador

Antes de empezar a describir como usar el comando gcc, conviene recordar al lectot las tres fases que tiene la generación de un ejecutable, por parte en compilador:

1. Preprocesado: Expande las directivas que empiezan por # como #define, #include, o #indef
2. Compilacion: Traduce el código fuente a código objeto, un código cercano al código maquina donde las llamadas externas están sin resolver y las direcciones de memoria son relativas.
3. Enlazado: Combina los diferentes archivos de código objeto resolviendo las llamadas que ah quedado pendientes y asignando direcciones definitivas al ejecutable final.

En el caso de gcc, aunque llamaremos compilación al proceso global, la etapa de compilación consta de dos partes:

1. Compilacion: Donde se traduce el código fuente a código ensamblador
2. Ensamblado: Donde se traduce el código ensamblador en código objeto.

El proceso de compilación lo realiza en parte el frontend (generando un árbol gramatical), y en parte el backend (generando el código ensamblador para la maquina que corresponda). Tanto el proceso de ensamblado como el enlazado son realizados por el backend.

Comandos disponibles

Como ya hemos comentado, actualmente las gcc constan de un gran numero de comandos que iremos viendo, y que resumimos en la tabla,

Comando	Descipcion
Gcc	Compilador de las gcc
Ccl	El compilador actual de C en ensamblador
As	Ensamblador que traduce de lenguaje ensamblador a código objeto
Ld	El ensalzador estatico de GCC
gcc_select	Cambia la versión del compilador que estamos usando.
C++filt	Elimina los nombres con name magling de la entrada estándar e imprime el resultado en la salida estándar.
File	Muestra binarios incluidos en un binario universal
Lipo	Muestra y manipula información de binarios universales.
Ndisasm	Desensamblador de Netwide Assembler Project distribuido con Xcode
Nm	Muestra la tabla de símbolos de un archico de código objeto o de una librería.
Otool	Muestra información sobre un archivo mach-O. Permite incluso desensamblar estos archivos.
Otx	Desensamblador bastante potente
Libtool	Permite generar librerías tanto de enlace estatico como de enlace dinamico a partir de archivos de código objeto
Cmpdylib	Compara la compatibilidad entre dos librerías de enlace dinamico
Strip	Elimina símbolos inncesarios de un archivo de código objeto.
g++	Una versión de gcc que fija el lenguaje por defecto a c++, e incluye las librerías estándar de c++
c++	Equicalente al anterior
collect2	Genera las inicializaciones de los objetos globales
Gcj	Compilador Java de las GCC
Gij	Interprete Java de las GCC
jcf-dump	Nos permite obtenet información sobre contenido de un archivo .class
jv-scan	Recibe un archivo de código fuente java y produce distinta información sobre este
Grepjar	Busca una expresión regular en un archivo .jar
Fastjar	Una implementación del comando jar de sun que ejecuta considerablemente mas rápido
class-dump	Permite obtener las clases Objective-C contenidas

	en un archivo Mach-o
gcjh	Permite generar los prototipos CNI (o JNI si usamos la opción –jni) de los métodos a implementar.
Gdb	Es el depurador de GNU. Permite ejecutar paso a paso un programa e identificar errores.
Leaks	Permite identificar fugas de memoria, es decir memoria reservada dinámicamente que nunca se libera.
malloc_history	Permite inspeccionar un log con toda la memoria dinámica que ha sido reservada y liberada durante la ejecución de un programa
Gprof	Permite perfilar un programa, es decir, detectar partes de programa que si se optimizan podríamos conseguir una considerable mejora.
Gcov	Permite realizar test de cobertura a un programa, es decir, saber cuantas veces se ejecuta cada línea del programa.

Actualmente no todos los comandos que se describen estan disponibles por defecto, con lo que si no se encuentra algun de estos comandos en su terminal, dejamos como ejercicio para buscarlo e instalarlo.

Opciones de la linea de comandos

Las opciones de la linea de comandos siempre empiezan por uno o dos guiones. En general, cuando la opcion tiene una letra se ponen dos guiones, cuando solo tienen una letra se pone un guion. Por ejemplo, para compilar el archivo hola.c y genenrar el archivo de codigo objeto hola.o usando ANSI C estandar se usa el comando.

$ gcc hola.c –ansi –c –o hola.o

Vemos que se indica primero los archivos de entrada (hola.c en este caso), y luego las opciones, algunas de las cuales tienen parametros y otras no. En general gcc es bastante flexible y suele acpetar que le pasemos las opciones en cualquier orden.

Las opciones de la linea de comandos se pueden clasificar en tres categorias:

1. Especificas del lenguaje. Son Opciones que solo se pueden usar junto con determinados lenguajes de programacion. Por ejemplo, -C89 solo se usa para indicar que queremos usar el estandar ISO de 1989.
2. Especifica de la plataforma. Son opciones que solo se usan en determinada plataforma. Por ejemplo –f-ret-in-387 se usa solo en INTEL para indicar que el retoro en punto flotante de las funciones se haga en un registro de punto flotante.
3. Generales. Son opciones que se pueden usar en todos los lenguajes y que todas las plataformas, como por ejemplo –o usada para indicar que queremos optimizar el codigo objeto resultante.

Como veremos mas adelante, hay opciones que no van dirigidas al compilador si no al encalzador

estatico. Cuando gcc recibe una opcion que no entiende simplemente la pasa al enlazador estatico esperando que este la sepa interpretar.

Embezar a manejar gcc

Si nosotros pasamos a gcc un programa como

```
/*hola.c*/
#include <stdio.h>
int main(){
        printf("Hola Mundo \n");
        return (0);
}
```

Y ejecutando: $gcc hola.c

Este lo compilara y enlazara generando como salida el archivo ejecutable a.out , que podras ejecutarlo de la siguiente manera:

```
$./a.out
Hola Mundo
```

Si preferimos que el archivo de salida tenga otro nombre podemos indicarlo con la opcion –o. Por ejemplo, el siguiente comando genera el archivo ejecutable hola.

```
$ gcc hola.c –o hola
```

El preprocesador

El concepto de preprocesador fue introducido inicialmente por C, pero despues los otros lenguajes como C++ o Objective-C lo han heredado.

El programa encargado de realizar el preprocesado es un comando llamado cpp.

Opciones relacionadas con el preprocesador

Existe una serie de opciones relacionadas con el preprocesador que se resumen en la tabla siguiente:

Opcion	Descripcion
-D	Define un identificador del preprocesador (que puede ser un macro). Para ello usamos –D nombre=[valor], donde si no asigamos valor, por defecto nombre vale 1.
-U	Usar la opción –U nombre cancela el identificador del preprocesador previamente definidos. El identificador podría haber sido

	definido en un archivo, o con la opción -D
-I	-I directorio incluye directorio en la lista de directorios donde busca archivos de cabecera. El directorio aquí pasado se usa antes que los directorios de inclusión estándar, con lo que esta opción nos permite sobre-escribir archivos de cabecera estándar.
-W	Existen algunos warnings relacionados con el preprocesador como por ejemplo –Wunused-macros que indica que queremos generar un warning si un identificador del preprocesador definido en un .c no es usado en todo el archivo. Esta opción no afecta a los identificadores definidos, y no usados, en los archivos de cabecera, o en las opciones del preprocesador.

Driver's

UNA INTRODUCCIÓN A LOS DEVICE DRIVERS

A medida que la popularidad del sistema Linux continúa creciendo, el interés en escribir controladores de dispositivos Linux aumenta constantemente. La mayoría de Linux es independiente del hardware que se ejecuta en, y la mayoría de los usuarios pueden ser (feliz) desconocen los aspectos de hardware. Pero, por cada pieza de hardware compatible con Linux, alguien en algún lugar se ha escrito un driver para que funcione con el sistema. Sin controladores de dispositivos, no existe ningún sistema en funcionamiento.

Los controladores de dispositivos adquieren un papel especial en el kernel de Linux. Son diferentes "cajas negras" que hacen que una determinada pieza de hardware responder a una interfaz de programación bien definida interno, sino que ocultar completamente los detalles de cómo funciona el dispositivo. Actividades del usuario se realiza por medio de un conjunto de llamadas estandarizadas que son independientes de la controlador específico; mapear esas llamadas a operaciones específicas del dispositivo que actúan en hardware real es entonces el papel del controlador de dispositivo. Esta interfaz de programación es tal que los conductores pueden ser construidos por separado del resto del núcleo, y "conectados" en tiempo de ejecución cuando sea necesario. Esta modularidad hace que los controladores de Linux fáciles de escribir, hasta el punto que en la actualidad hay cientos de ellos disponibles.

Hay una serie de razones para estar interesado en la escritura de controladores de dispositivos de Linux. La velocidad a la que se disponga de nuevo hardware (y obsoleto!) Solo garantiza que los escritores del conductor estará ocupado por el futuro previsible. Los individuos pueden necesitar saber acerca de los controladores con el fin de obtener acceso a un dispositivo particular que es de interés para ellos. Los proveedores de hardware, por lo que un controlador de Linux disponible para sus productos, puede añadir la base de usuarios de Linux grande y creciente de sus mercados potenciales. Y la naturaleza de código abierto del

sistema Linux significa que si los deseos del escritor conductor, la fuente para un conductor puede ser rápidamente difundido a millones de usuarios.

Este libro le enseñará cómo escribir sus propios controladores y cómo vaguear en las partes conexas del kernel. Hemos adoptado un enfoque independiente del dispositivo, las técnicas de programación e interfaces son presentados, siempre que sea posible, sin estar atado a ningún dispositivo específico. Cada driver es diferente, como escritor driver, usted tendrá que entender bien su dispositivo específico. Pero la mayoría de los principios y técnicas básicos son los mismos para todos los drivers. Este libro no puede enseñarle acerca de su dispositivo, pero le dará una mano en el fondo que necesita para que su dispositivo funcione.

A medida que aprenda a escribir controladores, usted descubrirá mucho sobre el kernel de Linux en general, lo que puede ayudar a entender cómo su equipo funciona y por qué las cosas no son siempre lo más rápido que esperar o no hacer absolutamente lo que quiere . Vamos a introducir nuevas ideas poco a poco, empezando con pilotos muy simples y basándose en ellos, cada nuevo concepto estará acompañado por el código de ejemplo que no necesita hardware especial para ser probado.

Los siguientes programas y ejemplos, muestran como se monta y desmonta un driver, siga los pasos y podrá entender claramente como se hace esto

1.- se crea un directorio:

[inforce@localhost ~]$ **mkdir ojala_funcione**

2.- me traslado al directorio

[inforce@localhost ~]$ **cd ojala_funcione**
[inforce@localhost ojala_funcione]$

Un módulo se crea a partir de un fuente en "C". A continuación tenemos, un módulo mínimo:

[inforce@localhost ojala_funcione]$ **gedit ejemplo.c &**

Archivo ejemplo.c:

```
/*_________ejemplo.c_____________________*/
#ifndef __KERNEL__
# define __KERNEL__
#endif
#ifndef MODULE
# define MODULE
#endif

#include <linux/module.h>
#include <linux/kernel.h>
```

```
#include <linux/init.h>
MODULE_LICENSE("Dual BSD/GPL");

int n = 1;
module_param(n, int, S_IRUGO);

static int __init entrando(void) {
printk(KERN_INFO "Entrando.n=%d\n",n);
return 0;
}

static void __exit saliendo(void) {
printk(KERN_INFO "Saliendo.\n");
}

module_init(entrando);
module_exit(saliendo);
/*_____*/
```

Imprimiendo mensajes desde el núcleo de Linux

El núcleo no dispone de salida estándar, por lo que no podemos utilizar la función printf(). A cambio, el núcleo ofrece una versión de ésta, llamada printk(), que funciona casi igual, a excepción de que el resultado lo imprime sobre un buffer circular de mensajes (kernel ring buffer).

En el kernel ring buffer es donde se escriben todos los mensajes del núcleo. De hecho, son los mensajes que vemos cuando arranca Linux. En cualquier momento podemos ver su contenido reciente con la orden dmesg o su contenido inmediato consultando el archivo /proc/kmsg.

El manejo de los mensajes por el kernel es muy flexible y algo complejo y /usr/src/linux/include/linux/kernel.h). Concretamente, el primer parámetro de printk debe ser una de las siguientes constantes, dónde <n> es un número entre 0 y 7, ambos inclusive, que indican el nivel de prioridad del mensaje.

KERN_EMERG <0> System is unusable
KERN_ALERT <1> Action must be taken immediately
KERN_CRIT <2> Critical conditions
KERN_ERR <3> Error conditions
KERN_WARNING <4> Warning conditions
KERN_NOTICE <5> Normal but significant condition
KERN_INFO <6> Informational
KERN_DEBUG <7> Debug-level messages

Normalmente, el núcleo está configurado para mostrar por la consola activa los mensajes de prioridad superior a 6. (Los terminales gráficos no son consolas, a no ser que se hayan lanzado explícitamente como tales).

Compilación de los modulos

El archivo Makefile necesario para compilar el módulo de ejemplo sería:

KVERSION = $(shell uname -r)

obj-m = ejemplo.o

all:

make -C /lib/modules/$(KVERSION)/build M=$(PWD) modules

clean:

make -C /lib/modules/$(KVERSION)/build M=$(PWD) clean

El nombre del archivo makefile debe ser Makefile (Nota: es importante notar que en la parte dentro del Makefile, en la parte de "**make -C /lib"** lleva el espacio, de un tap, si no tiene ese espacion no compila)

[inforce@localhost ojala_funcione]$ **gedit Makefile &**

y incluimos

KVERSION = $(shell uname -r)

obj-m = ejemplo.o

all:

make -C /lib/modules/$(KVERSION)/build M=$(PWD) modules

clean:

make -C /lib/modules/$(KVERSION)/build M=$(PWD) clean

La primera línea averigua automáticamente qué versión del núcleo estás ejecutando (uname -r) y supone que los módulos que vas a compilar son para ese núcleo. Si se compilan los módulos para un núcleo distinto del actual, habría que cambiar esa línea poniendo la versión

concreta del núcleo (v.g. KVERSION = 2.6.20.3-ubuntu1).

La opción "-C" indica la ejecución del makefile desde el directorio indicado por el primer parámetro, en este caso /lib/modules/$(KVERSION)/build y compila y crea los módulos cuyos archivos se encuentran en el directorio indicado por el segundo parámetro, $(PWD).

La opción "M=" indica al compilador que lo que se va a compilar y generar son módulos.

Para compilar los módulos especificados en la variable obj-m se utilizará la orden make all.

Si no entiendes algo, pregúntalo, después de pensarlo. ¡¡No te quedes con ninguna duda al respecto!!

Si se está utilizando un núcleo diferente al que viene en la distribución, para que estén disponibles los archivos de cabecera necesarios para compilar, los fuentes del núcleo deben estar en su sitio (/usr/src/linux-num_de_version) y se debe haber ejecutado al menos make menuconfig (grabando la configuración) y make depend, o haber recompilado el núcleo completo (make menuconfig y make depend bzImage modules modules_install).

LO COMPILAMOS COMO:

```
[inforce@localhost ojala_funcione]$ make
make -C /lib/modules/2.6.29-10/build M=/home/inforce/ojala_funcione modules
make[1]: Entering directory `/usr/src/kernels/2.6.29-10'
make[2]: *** No rule to make target
`/home/inforce/ojala_funcione/ejemplo.c', needed by
`/home/inforce/ojala_funcione/ejemplo.o'. Stop.
make[1]: *** [_module_/home/inforce/ojala_funcione] Error 2
make[1]: Leaving directory `/usr/src/kernels/2.6.29-10'
make: *** [all] Error 2
[2]+ Done gedit Makefile
```

para ver lo creado por el make escribimos el comando ls –l , como se muestra a continuacion

```
[inforce@localhost ojala_funcione]$ ls -l
total 180
-rw-rw-r-- 1 inforce inforce 450 2012-10-19 19:37 ejemplo.c
-rw-rw-r-- 1 inforce inforce 77378 2012-10-19 19:38 ejemplo.ko
-rw-rw-r-- 1 inforce inforce 497 2012-10-19 19:38 ejemplo.mod.c
-rw-rw-r-- 1 inforce inforce 38132 2012-10-19 19:38 ejemplo.mod.o
-rw-rw-r-- 1 inforce inforce 40276 2012-10-19 19:38 ejemplo.o
-rw-rw-r-- 1 inforce inforce 174 2012-10-19 19:35 Makefile
-rw-rw-r-- 1 inforce inforce 2171 2012-10-19 19:38 Module.markers
-rw-rw-r-- 1 inforce inforce 47 2012-10-19 19:38 modules.order
-rw-rw-r-- 1 inforce inforce 0 2012-10-19 19:38 Module.symvers
[inforce@localhost ojala_funcione]$
```

Utilización de los módulos

La carga de un módulo se lleva a cabo mediante la orden insmod, que realizará todas las acciones comentadas antes para insertar el **código** en el núcleo. Haz, desde una consola:

[inforce@localhost ojala_funcione]$ **su**
Password:
[root@localhost ojala_funcione]# **insmod ejemplo.ko**
[root@localhost ojala_funcione]#

Acabamos de instalar ejemplo y ejecutar su macro module_init. Si se le pasa a insmod un nombre de archivo sin ruta, se busca en los directorios estándar.

La orden lsmod permite listar los módulos que en un momento dado tenemos instalados:

[root@localhost ojala_funcione]# **lsmod**

Y, finalmente, con rmmod podemos extraer del núcleo el módulo (el nombre puede no incluir la extensión .ko):

[root@localhost ojala_funcione]# **rmmod ejemplo**

Para pasar parámetros a un módulo, no hay más que asignar valores a las variables globales declaradas como parámetros con la macro module_param(nombre, tipo, modo). Como hemos visto en el programa de ejemplo, module_param recibe como primer argumento el nombre de la variable y como segundo argumento, el tipo. Como tercer parámetro tenemos el modo en que puede ser accedida la variable.

El tipo debe ser uno de los siguientes: int, long, short, uint, ulong, ushort, charp (puntero a caracter) y bool (un booleano, cuya variale asociada debe ser de tipo entero).

El modo S_IRUGO indica que este parámetro será leído por todo el mundo pero nadie puede modificarlo (Read User, Group y Others).

La definición de module_param puede consultarse en el archivo /usr/src/linux/include/linux/moduleparam.h.

La sintaxis es muy sencilla, ya que basta con escribir la asignación como parámetro de

insmod. Por ejemplo (pruébalo mejor desde una consola):

[root@localhost ojala_funcione]# **insmod ejemplo.ko n=4**

Con modinfo -p ejemplo.ko podemos averiguar qué parámetros puede recibir el módulo.

¿Para qué el código del kernel?

Esto es realmente importante, así que echaremos un vistazo un poco más profundo. Cuando compilamos un módulo para Linux es necesario tener el código fuente de algunas partes del kernel, puesto que muchas instrucciones de preprocesador usadas no se encuentran en los headers estándar de desarrollo. En vez de ello, se encuentran en los headers de kernel.

Creamos un directorio

[root@localhost inforce]# **mkdir woorale**

nos transladamos al directorio creado

[root@localhost inforce]# **cd woorale**

“Hola, kernel!”

Bien, es hora de crear nuestro primer módulo.

[root@localhost woorale]# **gedit hello.c &**

Sin más rodeos, el código sería el siguiente:

```
/* El nombre del archivo es 'hello.c'. */
#include <linux/module.h>

MODULE_LICENSE("GPL");

static int myParam =33;
module_param(myParam,int, S_IRUGO|S_IWUSR);
MODULE_PARM_DESC(myParam, "My parameter (default 33)");
static int hello_init(void)
{
printk("hello world, myParam = %d\n",myParam);
return 0;
}
static void hello_exit(void)
{
printk("goodbye world, myParam = %d\n",myParam);
```

```
}
module_init(hello_init);
module_exit(hello_exit);
```

Verificamos que este en el mismo directorio

[root@localhost woorale]# **ls -l**
total 4
-rw-r--r-- 1 root root 1094 2012-10-19 19:53 hola.c

Algunas observaciones acerca del código de arriba:

- Técnicamente, no es necesario imprimir cosas cada vez que se carga o remueve un módulo (con printk). Pero puesto que es nuestro primer módulo, y aún no hace nada especial, es más divertido si lo dejamos así.
- Es necesario hacer que la función de inicio retorne 0, si queremos indicar que la carga fue satisfactoria.
- No, no es necesario poner una coma después de indicar el nivel de logs (KERN_INFO). Es un error común hacerlo.

Eso es todo… ¡vamos a compilarlo! El archivo Makefile, Este es el archivo Makefile que necesitaremos:

[root@localhost woorale]# **gedit Makefile &**

```
obj-m=hello.o
KVERSION=$(shell uname -r)
all:
	make -C /lib/modules/$(KVERSION)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(KVERSION)/build M=$(PWD) clean
```

Como has de saber, los archivos Makefile indican las reglas necesarias para compilar código. En este caso, explicándolo a groso modo, lo que hace el Makefile es detectar que aún nos encontramos en el directorio de desarrollo de nuestro módulo, y por lo tanto se dirige al directorio del kernel, compila el módulo desde ahí y se devuelve. Para probarlo basta con ejecutar el comando make:

root@localhost practica2]# **make**

Cargar o remover el módulo

Llego la hora de cargar nuestro módulo. Para ello, como comenté anteriormente, es

necesario poseer privilegios administrativos:

[root@localhost practica2]# **insmod hello.ko**

para verificar si esta cargado el modulo

[root@localhost practica2]# **lsmod**

Para remover el modulo

[root@localhost practica2]# **rmmod hello**

¿Y donde está lo que imprimimos con printk? Bien, no es común imprimir en consola cosas mientras un módulo es cargado o removido; en este caso, la salida va a dar al archivo de logs principal de Linux (/var/log/messages); puedes ver la salida con el comando dmesg o directamente en dicho archivo:

[root@localhost practica2]# **dmesg | tail**

Conclusión

Estas son apenas las bases que deberíamos tener para comenzar con la construcción de un módulo para el kernel de Linux. Es de valiosa ayuda jugar un poco con este ejemplo, de tal manera que podamos estar seguros que todo irá bien cuando hagamos algo un poco más complejo

Damos tres ejemplos de módulos a fin de probarlos y mostrar la estructura básica.

"Hola Mundo" en espacio de kernel

El siguiente ejemplo imprime un `"Hello, world"` al cargar el módulo, y un `"Goodbye, cruel world"` cuando se lo quita del sistema.
Solamente se define la inicialización y destrucción del módulo.

También se puede ver el uso de la funcion `printk` para imprimir mensajes informativos.

```
#include <linux/init.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");

static int hello_init(void)
{
  printk(KERN_ALERT "Hello, world\n");
  return 0;
```

```
}

static void hello_exit(void)
{
  printk(KERN_ALERT "Goodbye, cruel world\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

El archivo Makefile es sencillo y bastante genérico por lo que servirá, adaptando los nombres, para el desarrollo del proyecto.

```
obj-m := hello.o

all:
  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Podemos compilarlo para luego insertarlo y removerlo del sistema y ver los efectos que produce.

```
$ make
...
$ sudo insmod ./hello.ko
$ sudo rmmod hello
$ dmesg | tail -2
[ 3889.892180] Hello, world
[ 3892.863703] Goodbye, cruel world
```

Una modificación sencilla permite aceptar **parámetros de entrada** para nuestro módulo.

```
#include <linux/init.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");

static char *whom = "world";
static unsigned int howmany = 1;
module_param(howmany, int, S_IRUGO);
module_param(whom, charp, S_IRUGO);


static int hello_init(void)
```

```
{
  unsigned int i = 0;
  for (i = 0; i < howmany; ++i) {
          printk(KERN_ALERT "Hello, %s\n", whom);
  }
  return 0;
}

static void hello_exit(void)
{
  printk(KERN_ALERT "Goodbye, cruel %s\n", whom);
}

module_init(hello_init);
module_exit(hello_exit);
```

Para pasarle los parámetros hacemos:

```
$ insmod ./hello_param.ko howmany=8 whom=JuanMeza
$ rmmod hello_param
```

Y con `dmesg` miramos sus efectos:

```
[ 3824.384669] Hello, JuanMeza
[ 3824.384678] Hello, JuanMeza
[ 3824.384681] Hello, JuanMeza
[ 3824.384683] Hello, JuanMeza
[ 3824.384686] Hello, JuanMeza
[ 3824.384688] Hello, JuanMeza
[ 3824.384691] Hello, JuanMeza
[ 3824.384693] Hello, JuanMeza
[ 3831.493057] Goodbye, cruel JuanMeza
```

El driver completo "memoria": parte inicial del driver

Ahora realizaremos un driver completo, memoria.c, utilizando la memoria del ordenador que nos permitirá escribir y leer un carácter en memoria. Este dispositivo, aunque no muy útil, es muy ilustrativo dado que es un driver completo y fácil de implementar ya que no se necesita un dispositivo real.

Para realizar un driver, en la parte inicial de él, tendremos que definir las constantes MODULE y __KERNEL__. Además tendremos que incluir, con #include, una serie de archivos habituales en los drivers:

```
  <<memoria inicio>>=

/* Definiciones e includes necesarios para los drivers */
#define MODULE
#define __KERNEL__
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h> /* printk() */
#include <linux/malloc.h> /* kmalloc() */
#include <linux/fs.h> /* everything... */
#include <linux/errno.h> /* error codes */
#include <linux/types.h> /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h> /* O_ACCMODE */
#include <asm/system.h> /* cli(), *_flags */
#include <asm/uaccess.h> /* copy_from/to_user */

/* Declaracion de funciones de memoria.c */
int memoria_open(struct inode *inode, struct file *filp);
int memoria_release(struct inode *inode, struct file *filp);
ssize_t memoria_read(struct file *filp, char *buf,
  size_t count, loff_t *f_pos);
ssize_t memoria_write(struct file *filp, char *buf,
  size_t count, loff_t *f_pos);
void cleanup_module(void);

/* Estructura que declara las funciones tipicas */
/* de acceso a ficheros */
struct file_operations memoria_fops = {
  read: memoria_read,
  write: memoria_write,
  open: memoria open,
  release: memoria_release
};

/* Variables globales del driver */
/* Numero mayor */
int memoria_major = 60;
/* Buffer donde guardar los datos */
char *memoria_buffer;
```

Detrás de los archivos #include, aparecen las declaraciones de las funciones que definiremos en el programa más adelante. Posteriormente aparece la definición de la estructura file_operations que define las funciones típicas que se utilizan al manipular archivos y que veremos después. Finalmente están las variables globales del driver, una de ellas es el número mayor del dispositivo y la otra un puntero a la región de memoria, memoria_buffer, que utilizaremos como almacén de datos del driver.

El driver "memoria": conexión de dispositivos con sus ficheros

En UNIX y Linux se accede a los dispositivos desde el espacio de usuario de idéntica forma a como se hace con un archivo. Dichos archivos suelen colgar del directorio /dev.

Para ligar archivos con dispositivos se utilizan dos números: número mayor y número menor. El número mayor es el que utiliza el kernel para relacionar el archivo con su driver. El número menor es para uso interno del dispositivo y por simplicidad no lo veremos aquí.

Para conseguir este propósito primero se tiene que crear el archivo que sirva como dispositivo con el comando, como usuario root,

mknod /dev/memoria c 60 0

donde la c significa que se trata de un dispositivo tipo char, el 60 es el número mayor y el 0 el número menor. Para ligar un driver con su archivo /dev correspondiente se utiliza la función register_chrdev que tiene como argumento el número mayor del dispositivo. Esta función se llama con tres argumentos: número mayor, cadena de caracteres indicando el nombre del módulo y una estructura file operations que asocia esta llamada con las funciones aplicables a ficheros definidas dentro de ella. Se invoca, al instalar el módulo, de esta forma:

```
  <<memoria init module>>=
int init_module(void) {
  int result;

  /* Registrando dispositivo */
  result = register_chrdev(memoria_major, "memoria",
      &memoria_fops);
  if (result < 0) {
    printk(
      "<1>memoria: no puedo obtener numero mayor %d\n",
      memoria_major);
    return result;
  }
```

```
  /* Reservando memoria para el buffer */
  memoria_buffer = kmalloc(1, GFP_KERNEL);
  if (!memoria_buffer) {
    result = -ENOMEM;
    goto fallo;
  }
  memset(memoria_buffer, 0, 1);

  printk("<1>Insertando modulo\n");
  return 0;

  fallo:
    cleanup_module();
    return result;
}
```

Además reservamos espacio en memoria para el buffer de nuestro dispositivo, memoria_buffer, a través de la función kmalloc, la cual es muy similar a la común malloc. Finalmente actuamos en consecuencia ante posibles errores al registrar el número mayor o al reservar memoria.

El driver "memoria": eliminando el módulo

Para eliminar el módulo, dentro de la función cleanup_module, insertamos la función unregister_chrdev para liberar el número mayor dentro del kernel.

```
  <<memoria cleanup module>>=
void cleanup_module(void) {

  /* Liberamos numero mayor */
  unregister_chrdev(memoria_major, "memoria");

  /* Liberamos memoria del buffer */
  if (memoria_buffer) {
    kfree(memoria_buffer);
  }

  printk("<1>Quitando modulo\n");
}
```

En esta subrutina también liberamos la memoria del buffer del dispositivo para dejar el kernel limpio al quitar el módulo

El driver "memoria": abriendo el dispositivo como archivo

La función en el espacio de kernel correspondiente a la apertura de un archivo en el espacio de usuario (fopen) es el miembro open: de la estructura file operations en la llamada a register_chrdev. En este caso se trata de memoria_open. Tiene como argumentos una estructura inode que pasa información del kernel al driver tal como el número mayor y el número menor y una estructura file con información relativa a las distintas operaciones que se pueden realizar con el fichero. Ninguna de estas dos funciones las veremos en profundidad aquí.

El kernel lleva un contador de cuantas veces está siendo utilizado un driver. El valor para cada driver se puede ver en la última columna numérica del comando lsmod. Cuando se abre un dispositivo para leer o escribir en él, la cuenta de uso se debe incrementar, tal y como aparece en la función memoria_open.

Además de esta operación, en la apertura de un archivo, se suelen iniciar las variables pertinentes al driver y el propio dispositivo en si. En este ejemplo, y debido a su extrema sencillez, no realizaremos operaciones de este tipo en dicha función.

Podemos ver la función memoria_open a continuación:

```
  <<memoria open>>=
int memoria_open(struct inode *inode, struct file *filp) {
  /* Aumentamos la cuenta de uso */
  MOD_INC_USE_COUNT;

  /* Exito */
  return 0;

}
```

En la Tabla se puede ver esta nueva función.

Eventos	Funciones de usuarios	Funciones del kernel
Carga de módulo	insmod	init_module
Abrir dispositivo	fopen	file operations: open
Leer dispositivo		
Escribir dispositivo		
Cerrar dispositivo		
Quitar módulo	rmmod	cleanup_module

Tabla Eventos de los drivers y sus funciones asociadas de intercambio

entre el espacio de kernel y el espacio de usuario.

El driver "memoria": cerrando el dispositivo como fichero

La función correspondiente a cerrar el archivo en el espacio de usuario (fclose), es el miembro release: de la estructura file operations en la llamada a register_chrdev. En este caso se trata de la función memoria_release. Tiene como argumentos la estructura inode y la estructura file anteriores.

Al liberar un archivo del espacio de usuario, se debe decrementar la cuenta de uso para restablecerla a su valor original. El módulo no se podrá descargar del kernel si dicha cuenta es distinta de cero.

Además de esta operación, cuando se cierra un fichero, se suele liberar memoria y variables relacionadas con la apertura del dispositivo. En este caso, a causa de su simplicidad, no se hacen este tipo de operaciones.

La función memoria_release aparece a continuación:

```
  <<memoria release>>=
int memoria_release(struct inode *inode, struct file *filp) {

  /* Decrementamos la cuenta de uso */
  MOD_DEC_USE_COUNT;

  /* Exito */
  return 0;
}
```

En la Tabla se puede ver esta nueva función.

Eventos	Funciones de usuarios	Funciones del kernel
Carga de módulo	insmod	init_module
Abrir dispositivo	fopen	file operations: open
Leer dispositivo		
Escribir dispositivo		
Cerrar dispositivo	fclose	file operations: release
Quitar módulo	rmmod	cleanup_module

Tabla. Eventos de los drivers y sus funciones asociadas de

intercambio entre el espacio de kernel y el espacio de usuario.

El driver "memoria": leyendo del dispositivo

Para leer el dispositivo mediante la función de usuario fread o similar se utiliza el miembro read: de la estructura file operations en la llamada a register_chrdev. En este caso es la función memoria_read. Tiene como argumentos una estructura tipo file, un buffer, buf, del cual leerá la función del espacio de usuario (fread), un contador del número de bytes a transferir, count, que tiene el mismo valor que el contador habitual de la función de lectura del espacio de usuario (fread) y finalmente la posición del fichero donde empezar a leer, f_pos.

En este caso simple, la función memoria_read procede a transferir el byte del buffer del driver, memoria_buffer, al espacio de usuario, mediante la función copy_to_user:

```
  <<memoria read>>=
ssize_t memoria_read(struct file *filp, char *buf,
  size_t count, loff_t *f_pos) {

  /* Transferimos datos al espacio de usuario */
  copy_to_user(buf,memoria_buffer,1);

  /* Cambiamos posición de lectura segun convenga */
  if (*f_pos == 0) {
    *f_pos+=1;
    return 1;
  } else {
    return 0;
  }
}
```

Además cambiamos la posición de lectura en el archivo, f_pos. Si dicha posición está en el inicio del fichero, la aumentamos en una unidad y damos como valor de retorno el número de bytes leídos correctamente, 1. Si no estamos en el inicio del fichero devolvemos un fin de fichero, 0, ya que el archivo o dispositivo únicamente almacena un byte.

En la Tabla se puede ver esta nueva función.

Eventos	Funciones de usuarios	Funciones del kernel
Carga de módulo	insmod	init_module
Abrir dispositivo	fopen	file operations: open
Leer dispositivo	fread	file operations: read
Escribir dispositivo		
Cerrar dispositivo	fclose	file operations: release
Quitar módulo	rmmod	cleanup_module

Tabla. Eventos de los drivers y sus funciones asociadas de intercambio entre el espacio de kernel y el espacio de usuario.

El driver "memoria": escribiendo al dispositivo

Para escribir en el dispositivo mediante la función de usuario fwrite o similar se utiliza el miembro write: que aparece en la estructura file operations en la llamada a register_chrdev. En este ejemplo es la función memoria_write. Tiene como argumentos una estructura tipo file, un buffer, buf, al cual escribirá la función del espacio de usuario (fwrite), un contador del número de bytes a transferir, count, que tiene el mismo valor que el contador habitual de la función de escritura del espacio de usuario (fwrite) y finalmente la posición del archivo donde empezar a escribir, f_pos.

```
  <<memoria write>>=
ssize_t memoria_write( struct file *filp, char *buf,
  size_t count, loff_t *f_pos) {
  char *tmp;
  tmp=buf+count-1;
  copy_from_user(memoria_buffer,tmp,1);
  return 1;
}
```

Además resulta útil la función copy_from_user que transfiere datos del espacio de usuario al espacio de kernel.

En la Tabla se puede ver esta nueva función.

Eventos	Funciones de usuarios	Funciones del kernel
Carga de módulo	insmod	init_module
Abrir dispositivo	fopen	file operations: open
Leer dispositivo	fread	file operations: read
Escribir dispositivo	fwrite	file operations: write
Cerrar dispositivo	fclose	file operations: release
Quitar módulo	rmmod	cleanup_module

Tabla. Eventos de los drivers y sus funciones asociadas de intercambio entre el espacio de kernel y el espacio de usuario

El driver "memoria" al completo

Juntando todas las operaciones que hemos visto obtenemos el driver completo

```
  <<memoria.c>>=
<<memoria inicio>>
<<memoria init module>>
<<memoria cleanup module>>
<<memoria open>>
<<memoria release>>
<<memoria read>>
<<memoria write>>
```

Para utilizar este módulo, primero lo compilamos de la misma forma que el hola.c. Posteriormente cargamos el módulo con

insmod memoria.o

Conviene también desproteger el dispositivo

chmod 666 /dev/memoria

A partir de entonces, y si todo ha ido bien, tendremos un dispositivo /dev/memoria al cual le podemos escribir una cadena de caracteres y guardará el último de ellos. Podemos realizar esta operación así

$ echo -n abcdef > /dev/memoria

Para ver el contenido del dispositivo podemos usar un simple cat:

$ cat /dev/memoria

Este carácter no cambiará en memoria hasta que no se sobreescriba o se desinstale el módulo.

El archivo de “memoria.c”, se escribe como:

```
/* Definiciones e includes necesarios para los drivers */

#define MODULE
#define __KERNEL__
//#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h> /* printk() */
//#include <linux/malloc.h> /* kmalloc() */
#include <linux/fs.h> /* everything... */
#include <linux/errno.h> /* error codes */
#include <linux/types.h> /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h> /* O_ACCMODE */
#include <asm/system.h> /* cli(), *_flags */
#include <asm/uaccess.h> /* copy_from/to_user */

/* Declaracion de funciones de memoria.c */

int memoria_open(struct inode *inode, struct file *filp);
int memoria_release(struct inode *inode, struct file *filp);
```

```
ssize_t memoria_read(struct file *filp, char *buf,
  size_t count, loff_t *f_pos);
ssize_t memoria_write(struct file *filp, char *buf,
  size_t count, loff_t *f_pos);
void cleanup_module(void);
/* Estructura que declara las funciones tipicas */

/* de acceso a ficheros */

struct file_operations memoria_fops = {
  read: memoria_read,
  write: memoria_write,
  open: memoria_open,
  release: memoria_release
};

/* Variables globales del driver */
/* Numero mayor */
int memoria_major = 60;
/* Buffer donde guardar los datos */
char *memoria_buffer;
int init_module(void) {
  int result;
  /* Registrando dispositivo */
  result = register_chrdev(memoria_major, "memoria",&memoria_fops);
  if (result < 0) {
    printk("<1>memoria: no puedo obtener numero mayor %d\n",memoria_major);
    return result;
  }

/* Reservando memoria para el buffer */
  memoria_buffer = kmalloc(1, GFP_KERNEL);
  if (!memoria_buffer) {
    result = -ENOMEM;
    goto fallo;
  }
  memset(memoria_buffer, 0, 1);
  printk("<1>Insertando modulo\n");
  return 0;
  fallo:
    cleanup_module();
    return result;
}
void cleanup_module(void) {
  /* Liberamos numero mayor */
  unregister_chrdev(memoria_major, "memoria");
  /* Liberamos memoria del buffer */
  if (memoria_buffer) {
    kfree(memoria_buffer);
  }
  printk("<1>Quitando modulo\n");
}

int memoria_open(struct inode *inode, struct file *filp) {
  /* Aumentamos la cuenta de uso */
//  MOD_INC_USE_COUNT;
  /* Exito */
  return 0;
}

int memoria_release(struct inode *inode, struct file *filp) {
  /* Decrementamos la cuenta de uso */
//  MOD_DEC_USE_COUNT;
  /* Exito */
  return 0;
```

```
}
ssize_t memoria_read(struct file *filp, char *buf,
  size_t count, loff_t *f_pos) {
  /* Transferimos datos al espacio de usuario */
  copy_to_user(buf,memoria_buffer,1);
/* Cambiamos posiciÃ³n de lectura segun convenga */
  if (*f_pos == 0) {
    *f_pos+=1;
    return 1;
  } else {
    return 0;
  }
}
ssize_t memoria_write( struct file *filp, char *buf,
  size_t count, loff_t *f_pos) {
  char *tmp;
  tmp=buf+count-1;
  copy_from_user(memoria_buffer,tmp,1);
  return 1;
}
```

El archivo "Makefile" para compilar este programa lo escribimos commo sigue:

```
KVERSION = $(shell uname -r)

obj-m = memoria.o

all:

	make -C /lib/modules/$(KVERSION)/build M=$(PWD) modules

clean:

	make -C /lib/modules/$(KVERSION)/build M=$(PWD) clean
```

El driver real "puertopar": descripción del puerto paralelo

Procederemos ahora a modificar el anterior driver "memoria" para realizar uno que haga una tarea real sobre un dispositivo real. Utilizaremos el ubicuo y sencillo puerto paralelo del ordenador y el módulo se llamará "puertopar".

El puerto paralelo es en realidad un dispositivo que permite la entrada y salida de información digital. Externamente tiene un conector hembra D-25 con veinticinco patillas. Internamente, desde el punto de vista de la CPU, ocupa tres bytes de memoria. La dirección base, es decir, la del primer byte del dispositivo, es habitualmente la 0x378 en un PC. En este ejemplo sencillo usaremos únicamente el primer byte. el cual consta enteramente de salidas digitales.

La conexión de dicho byte con el patillaje del conector exterior aparece en la Fig.

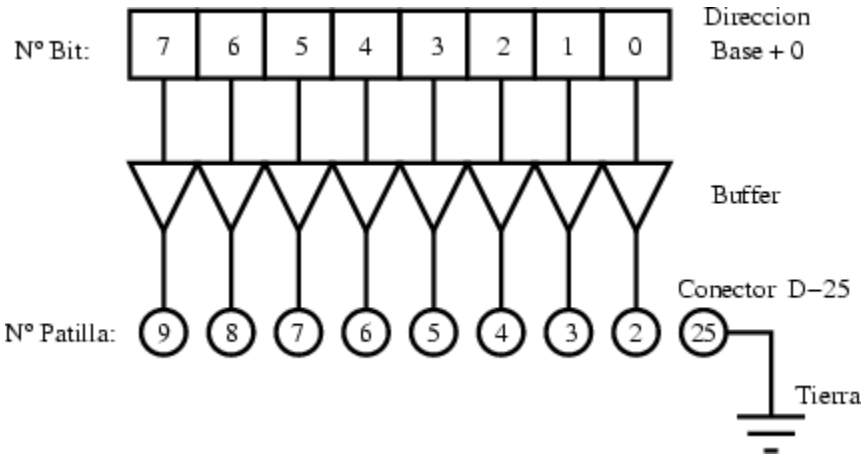


Figura. Esquema del primer byte del puerto paralelo y su conexión con el patillaje al conector exterior hembra D-25.

El driver "puertopar": inicio del módulo

La función init_module anterior del módulo "memoria" habrá que modificarla sustituyendo la reserva de memoria RAM por la reserva de la dirección de memoria del puerto paralelo, es decir, la 0x378. Para ello utilizaremos la función de chequeo de disponibilidad de la región de memoria, check_region, y la función de reserva de una región de memoria para este dispositivo, request_region. Ambas tienen como argumentos la dirección base de la región de memoria y su longitud. La función request_region además admite una cadena de caracteres que define el módulo.

```
<<puertopar modificacion init module>>=
/* Registrando puerto */
port = check_region(0x378, 1);
if (port) {
  printk("<1>puertopar: no puedo reservar 0x378\n");
  result = port;
  goto fallo;
}
request_region(0x378, 1, "puertopar");
```

El driver "puertopar": eliminando el módulo

Será muy parecida al módulo "memoria" pero sustituyendo la liberación de memoria por la eliminación de la reserva de memoria del puerto paralelo. Esto se realiza con la función release_region, la cual tiene los mismos argumentos que check_region.

```
<<puertopar modificacion cleanup module>>=
/* Liberando puerto */
if (!port) {
```

```
    release_region(0x378,1);
  }
```

El driver "puertopar": leyendo del dispositivo

En este caso tenemos que añadir la verdadera lectura del dispositivo para luego pasar este dato al espacio de usuario. La función inb consigue este resultado, admitiendo como argumento la dirección del puerto y devolviendo como resultado el contenido del puerto.

```
  <<puertopar inport>>=
/* Leyendo del puerto */
puertopar_buffer = inb(0x378);
```

La Tabla, equivalente a la Tabla 2, nos muestra esta nueva función.

Eventos	Funciones del kernel
Leer datos	inb
Escribir datos	

Tabla. Eventos de los drivers y sus funciones asociadas entre el espacio de kernel y el dispositivo hardware

El driver "puertopar": escribiendo al dispositivo

De nuevo tenemos que añadir la escritura al dispositivo para luego pasar este dato al espacio de usuario. La función outb consigue este resultado, admitiendo como argumento el contenido a escribir en el puerto y su dirección.

```
  <<puertopar outport>>=
/* Escribiendo al puerto */
outb(puertopar_buffer,0x378);
```

La Tabla , nos muestra esta nueva función.

Eventos	Funciones del kernel
Leer datos	inb
Escribir datos	outb

Tabla. Eventos de los drivers y sus funciones asociadas entre el espacio de kernel y el dispositivo hardware

El driver "puertopar" al completo

Procederemos ahora a ver todo el código del módulo "puertopar". En general habrá que cambiar en todo el código, respecto al módulo memoria.c, las palabras "memoria" por "puertopar". El driver final queda como sigue:

```
  <<puertopar.c>>=
<<puertopar inicio>>
<<puertopar init module>>
<<puertopar cleanup module>>
<<puertopar open>>
<<puertopar release>>
<<puertopar read>>
<<puertopar write>>
```

Sección inicial

En la parte inicial del driver utilizaremos un número mayor diferente, el 61, cambiaremos la variable global memoria_buffer por port e incluiremos con #include los ficheros ioport.h y io.h.

```
   <<puertopar inicio>>=

/* Definiciones e includes necesarios para los drivers */
#define MODULE
#define __KERNEL__
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h> /* printk() */
#include <linux/malloc.h> /* kmalloc() */
#include <linux/fs.h> /* everything... */
#include <linux/errno.h> /* error codes */
#include <linux/types.h> /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h> /* O_ACCMODE */
#include <linux/ioport.h>
#include <asm/system.h> /* cli(), *_flags */
#include <asm/uaccess.h> /* copy_from/to_user */
#include <asm/io.h> /* inb, outb */

/* Declaracion de funciones de puertopar.c */
int puertopar_open(struct inode *inode, struct file *filp);
int puertopar_release(struct inode *inode, struct file *filp);
ssize_t puertopar_read(struct file *filp, char *buf,
  size_t count, loff_t *f_pos);
ssize_t puertopar_write(struct file *filp, char *buf,
  size_t count, loff_t *f_pos);
void cleanup_module(void);

/* Estructura que declara las funciones tipicas */
/* de acceso a ficheros */
struct file_operations puertopar_fops = {
```

```
  read: puertopar_read,
  write: puertopar_write,
  open: puertopar_open,
  release: puertopar_release
};

/* Variables globales del driver */
/* Numero mayor */
int puertopar_major = 61;

/* Variable de control para la reserva */
/* de memoria del puerto paralelo*/
int port;
```

Inicio del módulo

En la rutina de inicio del módulo incluiremos la reserva de la dirección de memoria del puerto paralelo como describimos antes.

```
   <<puertopar init module>>=
int init_module(void) {
  int result;

  /* Registrando dispositivo */
  result = register_chrdev(puertopar_major, "puertopar",
      &puertopar_fops);
  if (result < 0) {
    printk(
      "<1>puertopar: no puedo obtener numero mayor %d\n",
      puertopar_major);
    return result;
  }

  <<puertopar modificacion init module>>

  printk("<1>Insertando modulo\n");
  return 0;

  fallo:
    cleanup_module();
    return result;
}
```

Eliminación del módulo

La rutina de eliminación del módulo incluirá las modificaciones antes reseñadas.

```
   <<puertopar cleanup module>>=
void cleanup_module(void) {
```

```
  /* Liberamos numero mayor */
  unregister_chrdev(puertopar_major, "memoria");

  <<puertopar modificacion cleanup module>>

  printk("<1>Quitando modulo\n");
}
```

Abriendo el dispositivo como fichero

Esta rutina es idéntica a la del driver "memoria".

```
   <<puertopar open>>=
int puertopar_open(struct inode *inode, struct file *filp) {
  /* Aumentamos la cuenta de uso */
  MOD_INC_USE_COUNT;

  /* Exito */
  return 0;

}
```

Cerrando el dispositivo como fichero

De nuevo la similitud es exacta.

```
   <<puertopar release>>=
int puertopar_release(struct inode *inode, struct file *filp) {

  /* Decrementamos la cuenta de uso */
  MOD_DEC_USE_COUNT;

  /* Exito */
  return 0;
}
```

Leyendo del dispositivo

La función de lectura es análoga a la del "memoria" con las consiguientes modificaciones de lectura del puerto del dispositivo.

```
   <<puertopar read>>=
ssize_t puertopar_read(struct file *filp, char *buf,
  size_t count, loff_t *f_pos) {

  /* Buffer para leer el dispositivo */
  char puertopar_buffer;
```

```
  <<puertopar inport>>

  /* Transferimos datos al espacio de usuario */
  copy_to_user(buf,&puertopar_buffer,1);

  /* Cambiamos posición de lectura segun convenga */
  if (*f_pos == 0) {
    *f_pos+=1;
    return 1;
  } else {
    return 0;
  }
}
```

Escribiendo al dispositivo

Es igual que el del "memoria" pero añadiendo la escritura al puerto del dispositivo.

```
   <<puertopar write>>=
ssize_t puertopar_write( struct file *filp, char *buf,
  size_t count, loff_t *f_pos) {

  char *tmp;

  /* Buffer para leer el dispositivo */
  char puertopar_buffer;

  tmp=buf+count-1;
  copy_from_user(&puertopar_buffer,tmp,1);

  <<puertopar outport>>

  return 1;
}
```

Entonces el archivo “puertopar.c”, se escribe asi:

/*puertopar.c*/

```
#define MODULE
#define __KERNEL__
//#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h> /* printk() */
//#include <linux/malloc.h> /* kmalloc() */
#include <linux/fs.h> /* everything... */
#include <linux/errno.h> /* error codes */
#include <linux/types.h> /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h> /* O_ACCMODE */
#include <linux/ioport.h>
#include <asm/system.h> /* cli(), *_flags */
#include <asm/uaccess.h> /* copy_from/to_user */
#include <asm/io.h> /* inb, outb */

/* Declaracion de funciones de puertopar.c */
```

```
int puertopar_open(struct inode *inode, struct file *filp);
int puertopar_release(struct inode *inode, struct file *filp);
ssize_t puertopar_read(struct file *filp, char *buf,
  size_t count, loff_t *f_pos);
ssize_t puertopar_write(struct file *filp, char *buf,
  size_t count, loff_t *f_pos);
void cleanup_module(void);

/* Estructura que declara las funciones tipicas */
/* de acceso a ficheros */
struct file_operations puertopar_fops = {
  read: puertopar_read,
  write: puertopar_write,
  open: puertopar_open,
  release: puertopar_release
};

/* Variables globales del driver */
/* Numero mayor */
int puertopar_major = 61;

/* Variable de control para la reserva */
/* de memoria del puerto paralelo*/
int port;


int init_module(void) {
  int result;

  /* Registrando dispositivo */
  result = register_chrdev(puertopar_major, "puertopar",
      &puertopar_fops);
  if (result < 0) {
    printk(
      "<1>puertopar: no puedo obtener numero mayor %d\n",
      puertopar_major);
    return result;
  }

    /* Registrando puerto */
    port = check_region(0x378, 1);
    if (port) {
      printk("<1>puertopar: no puedo reservar 0x378\n");
      result = port;
      goto fallo;
    }
    request_region(0x378, 1, "puertopar");


  printk("<1>Insertando modulo\n");
  return 0;

  fallo:
    cleanup_module();
    return result;
}

void cleanup_module(void) {

  /* Liberamos numero mayor */
  unregister_chrdev(puertopar_major, "memoria");

    /* Liberando puerto */
    if (!port) {
      release_region(0x378,1);
```

```
    }


  printk("<1>Quitando modulo\n");
}

int puertopar_open(struct inode *inode, struct file *filp) {
  /* Aumentamos la cuenta de uso */
//  MOD_INC_USE_COUNT;

  /* Exito */
  return 0;

}

int puertopar_release(struct inode *inode, struct file *filp) {

  /* Decrementamos la cuenta de uso */
//  MOD_DEC_USE_COUNT;

  /* Exito */
  return 0;
}

ssize_t puertopar_read(struct file *filp, char *buf,
  size_t count, loff_t *f_pos) {

  /* Buffer para leer el dispositivo */
  char puertopar_buffer;

  /* Leyendo del puerto */
  puertopar_buffer = inb(0x378);


  /* Transferimos datos al espacio de usuario */
  copy_to_user(buf,&puertopar_buffer,1);

  /* Cambiamos posiciÃ³n de lectura segun convenga */
  if (*f_pos == 0) {
    *f_pos+=1;
    return 1;
  } else {
    return 0;
  }
}

ssize_t puertopar_write( struct file *filp, char *buf,
  size_t count, loff_t *f_pos) {

  char *tmp;

  /* Buffer para leer el dispositivo */
  char puertopar_buffer;

  tmp=buf+count-1;
  copy_from_user(&puertopar_buffer,tmp,1);

  /* Escribiendo al puerto */
  outb(puertopar_buffer,0x378);


  return 1;
}
```

Y este programa lo compilamos con el archivo "**Makefile**" con el siguiente contenido:

```
KVERSION = $(shell uname -r)
obj-m = puertopar.o
all:
	make -C /lib/modules/$(KVERSION)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(KVERSION)/build M=$(PWD) clean
```

LEDs para comprobar el uso del puerto paralelo

Vamos a realizar en esta sección un "hardware" para poder visualizar el estado del puerto paralelo con unos simples LEDs.

ADVERTENCIA: Conectar dispositivos al puerto paralelo puede dañar su ordenador. Asegurése de que su cuerpo está conectado a una buena toma de tierra y su ordenador apagado al conectar un dispositivo. Cualquier problema que surja es totalmente responsabilidad suya.

El circuito a construir aparece en la Fig. Como referencia bibliográfica se puede consultar Zoller (1997).

Para utilizarlo, primero se debe comprobar que todas las conexiones del montaje son correctas. Posteriormente se apaga el PC y se conecta el módulo al puerto paralelo. Luego se arranca el PC y se desinstalan todos los módulos habituales relacionados con el puerto paralelo, como por ejemplo, lp, parport, parport_pc, etc. Si no existe el dispositivo puertopar, se crea, como root, con el comando

```
# mknod /dev/puertopar c 61 0
```

y se le otorgan permisos de escritura y lectura universales

```
# chmod 666 /dev/puertopar
```

Se instala el módulo que hemos realizado, puertopar, pudiendo comprobar que efectivamente reserva la posición de puertos de entrada/salida 0x378 mediante el comando

```
$ cat /proc/ioports
```

Para encender los LEDs y verificar que el sistema funciona ejecutamos el comando

```
$ echo -n A > /dev/puertopar
```

que debe tener como consecuencia que se encienda el LED cero y el seis, estando apagados todos los demás.

Podemos ver el estado del puerto paralelo con el comando

$ cat /dev/puertopar

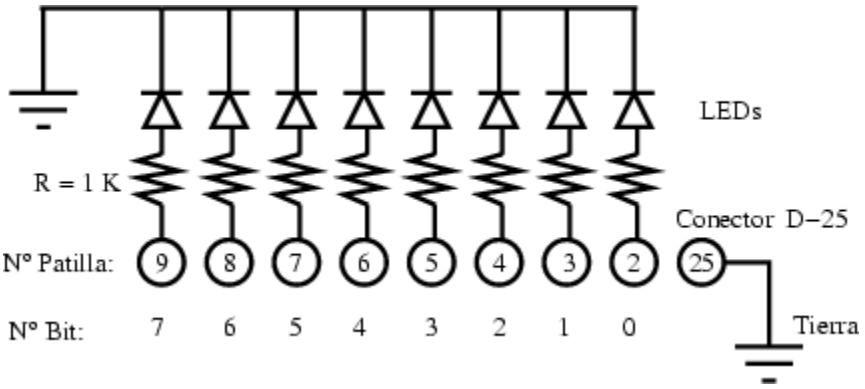


Figura. Esquema electrónico de la matriz de LEDs para monitorizar el puerto paralelo.

Aplicación final: luces centelleantes

Finalmente realizaremos una bonita aplicación que consiste en que se enciendan sucesivamente los LEDs en secuencia. Para ello realizaremos un programa en el espacio de usuario (el normal) con el que escribiremos al dispositivo /dev/puertopar un sólo bit de forma sucesiva.

```
#include <stdio.h>
#include <unistd.h>


int main() {
  unsigned char byte,dummy;
  FILE * PUERTOPAR;

  /* Abrimos el dispositivo puertopar */
  PUERTOPAR=fopen("/dev/puertopar","w");
  /* Eliminamos el buffer del fichero */
  setvbuf(PUERTOPAR,&dummy,_IONBF,1);

  /* Iniciamos la variable a uno */
  byte=1;

  /* Realizamos el bucle indefinido */
  while (1) {
    /* Escribimos al puerto paralelo */
    /* para encender un LED */
    printf("Byte vale %d\n",byte);
    fwrite(&byte,1,1,PUERTOPAR);
    sleep(1);

    /* Actualizamos el valor de byte */
    byte<<=1;
    if (byte == 0) byte = 1;
  }
```

```
fclose(PUERTOPAR);
    }
```

Lo compilamos de la forma habitual

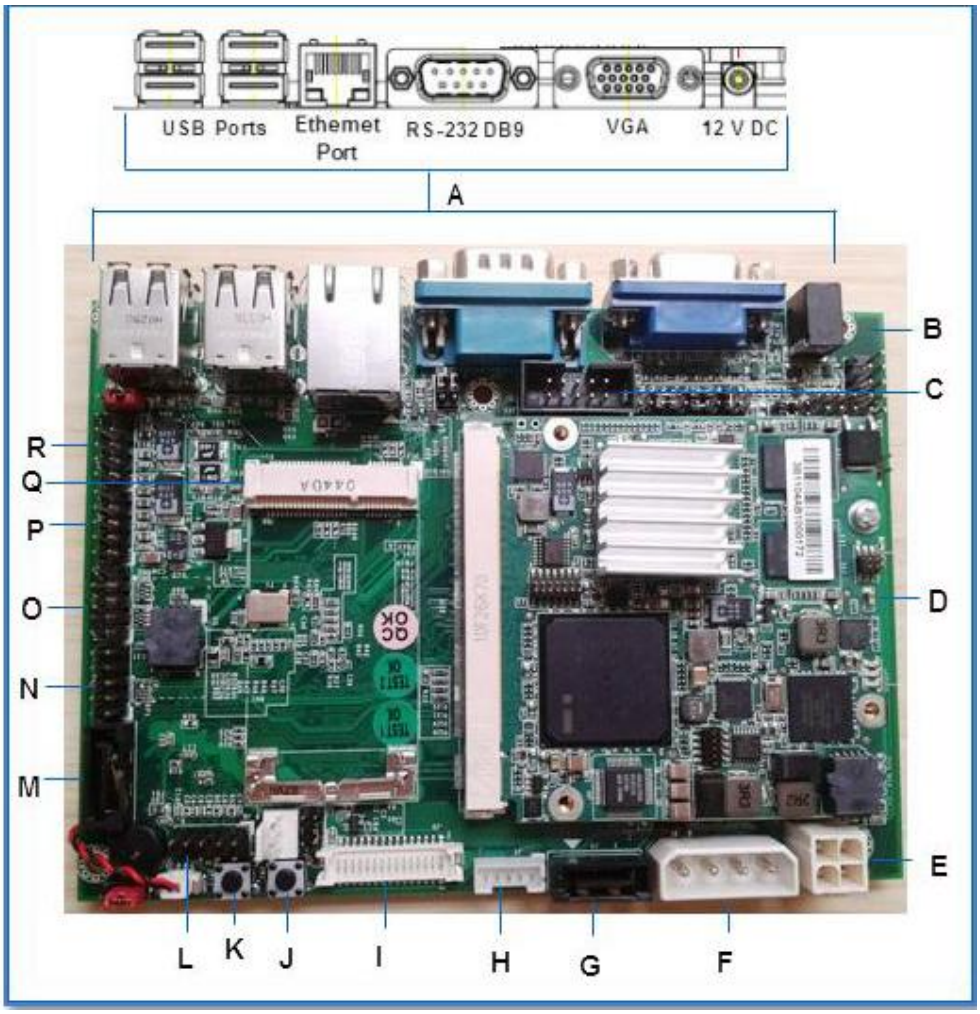
$ gcc -o luces luces.c

y lo ejecutamos con el comando

$ luces

¡Las luces se encenderán una por una secuencialmente!

Componentes Sistema SYS940X-01



Componentes del sistema

Label	Description	Label	Description
A	Back panel connectors	B	ECX Q7 carrier card
C	RS232 header	D	IFC940x-01 board
E	Auxiliary power connector	F	SATA power connector
G	SATA DATA connector 0	H	LVDS back light connector
I	LVDS connector	J	Restart switch
K	Power switch	L	Audio header
M	SATA DATA connector 1	N	GPIO header
O	LPC debug port	P	USB header 0
Q	Mini PCIex1 connector	R	USB header 1

Factor de forma

El SYS940X-01 (146 milímetros [pulgadas] 5.7 x 102 milímetros [4,0 pulgadas]).

Procesador

El SYS940X-01 incluye el procesador Intel ® Atom ™ E6xx Procesador. El procesador está soldado a la placa.

Memoria del sistema

El sistema SYS940X-01 soporta hasta 1GB de memoria DDR2 800 MHz. La memoria del sistema está soldado a la placa y no es actualizable campo.

Chipset

El sistema SYS940X-01 incluye el procesador Intel ® plataforma Controller Hub EG20T PCH

Display

El sistema SYS940X-01 soporta conectores LVDS y VGA de pantalla de la siguiente manera:

- El puerto LVDS interfaz soporta 112MHz de un solo canal a 24bpp. La resolución máxima admitida es de 1366 x 768
- SDVO a VGA controlador CH7317A está presente en el sistema, que proporciona una interfaz VGA

Audio

El sistema SYS940X-01 es compatible con Intel ® High Definicion de Audio a través de un micrófono integrado / header auriculares y zumbador interno.

Posibilidades de ampliación

El sistema SYS940X-01 soporta un mini x1 ranura PCI Express. Puede ser utilizado para la conexión inalámbrica o de otros módulos de expansión compatibles.

Almacenamiento

El sistema SYS940X-01 soporta dos puertos SATA 2.0 3 GB / s por ranura y una ranura para tarjetas SD.

Interfaces para periféricos

El sistema SYS940X-01 soporta las interfaces periféricas siguientes.

- Dos puertos RS-232: un conector DB9 (seleccionable como RS-422/485) y un puerto RS-232 a bordo de cabecera
- Cuatro puertos USB 2.0 Host
- Un puerto USB 2.0 Host onboard header
- Una Header SPI

Puertos de depuración

El sistema SYS940X-01 admite la depuración del sistema utilizando onboard LPC header.

BIOS

El SYS940X-01 sistema BIOS compatible con los siguientes Kits y del gestor de arranque de Desarrollo:

¬ Phoenix
¬ Intel BLDK1
¬ Intel BLDK2

LAN

El sistema SYS940X-01 incluye un Gigabit Ethernet (1000BASE-TX) a través de un puerto RJ-45.

Power

El sistema SYS940X-01 funciona con una externa de +12 V, fuente de alimentación 1.5A

Gestión de energía

El sistema SYS940X-01 admite la configuración avanzada e interfaz de energía (ACPI).

Utilizando una externa de +12 V, fuente de alimentación 1.5A.

Seguridad

- UL 60950-1, 2 ª edición, 27/3/2007 (Equipos de Tecnología de la Información - Seguridad - Parte 1: Requisitos generales).
- CSA C22.2 No. 60950-1-07, 2 ª edición, 2007-03 (Equipos de Tecnología de la Información - Seguridad - Parte 1: Requisitos generales).

Compatibilidad con sistemas operativos

Phoenix BIOS compatible con el sistema operativo siguiente

- Microsoft Windows XP SP3
- Fedora 11 (Linux Kernel versión 2.6.29)
- Fedora 14 (Linux Kernel versión 6.2.35)
- Ubuntu 11.04 (Linux Kernel versión 6.2.38)
- Meego 1.0 (Linux Kernel versión 2.6.33)
- Meego 1.1 (Linux Kernel versión 6.2.35)
- Debian 6.0.2 (versión del núcleo 2.6.32)

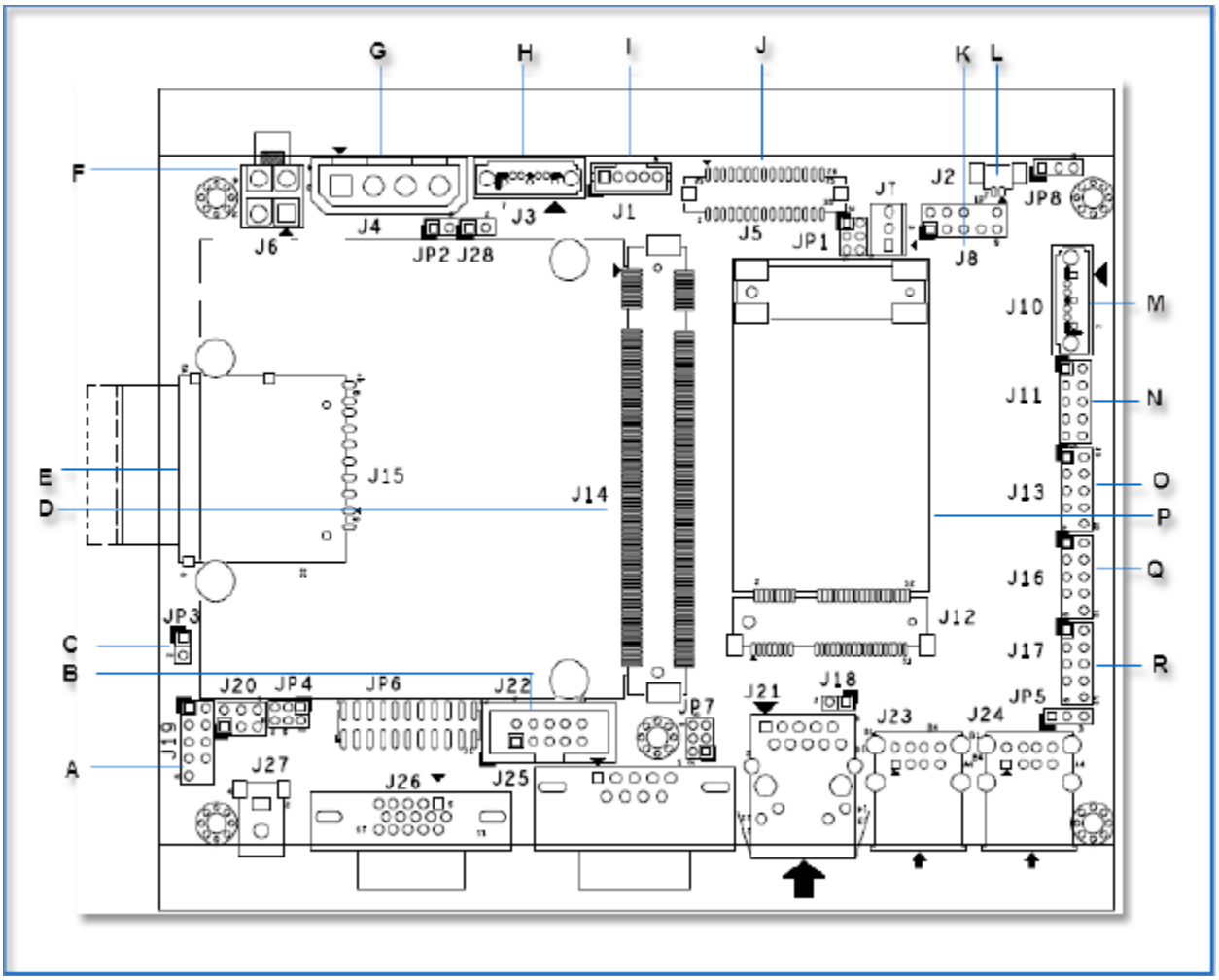
Drivers y Herramientas

Descarga de drivers para Windows y Linux desde http://www.inforcecomputing.com/products.html

Conexión a los conectores internos y conectores

La Figura muestra la ubicación de los conectores internos de la junta y los encabezados, la Tabla describe los conectores internos y los registros identificados en la Figura.

Header internos y conectores



Item	Description	Item	Description
A	Front panel control header	B	RS232 header
C	Auto power header	D	Q7 connector
E	SD cord slot	F	Auxiliary power connector
G	SATA power connector	H	SATA data connector 0
I	LVDS back light connector	J	LVDS connector
K	Audio header	L	RTC battery connector
M	SATA data connector 1	N	GPIO header
O	LPC debug port	P	Mini PCI connector
Q	USB header 0	R	USB header 1

Conectores y encabezados internos

Las Tablas siguientes enumeran los nombres de las señales de los conectores y soportes de conexión.

J22 COM2 Serial Port BOX Header

Pin	Signal Name	Pin	Signal Name
1	DCD#2	2	DSR#2
3	RXD#2	4	RTS#2
5	TXD#2	6	CTS#2
7	DTR#2	8	2RI#
9	GND	10	NO PIN

J15 SD Card Slot

Pin	Signal Name
1	DAT3
2	CMD_RSP
3	GND
4	VCC
5	CLK
6	GND
7	DAT0
8	DAT1
9	DAT2
Sa	WP3
Sb	CD#
Sc	C#_COM

J4 Power Connector

Pin	Signal Name	Pin	Signal Name
1	Ground	2	Ground
3	+12 V	4	+12 V

J3/J10 SATA Interface Connector

Pin	Signal Name	Signal Name
1	GND	GND
2	TX+	TX+
3	TX-	TX-
4	GND	GND
5	RX-	RX-
6	RX+	RX+
7	GND	GND

J1 LVDS and Back Light connector

Pin	Signal Name
1	Back light enable
2	GND
3	+12V
4	Back light control
5	+5V

J8 Audio Header

Pin	Signal Name
1.	CN_MIC_R
2	AGND
3	CN_MIC-L
4	AGND
5	CN_LINOUT-R
6	NC
7	VCC
8	NO PIN
9	CN_LINOUT-L
10	NC

J2 RTC Battery Connector

Pin	Signal Name
1	Positive
2	Negative

J11 GPIO header

Pin	Signal Name	Pin	Signal Name
1	LPC GP30	2	LPC GP31
3	LPC GP32	4	LPC GP33
5	LPC GP34	6	LPC GP35
7	LPC GP36	8	LPC GP37
9	GND	10	VCC

J16 USB Header (Only USB2.0 Device)

Pin	Signal Name	Pin	Signal Name
1	+5V	2	+5V
3	USBD6-	4	NC
5	USBD6+	6	NC
7	GND	8	GND
9	NO PIN	10	NC

J17 Client USB Header (Only USB2.0 Device)

Pin	Signal Name	Pin	Signal Name
1	+5V	2	NC
3	USBD1-	4	NC
5	USBD1+	6	NC
7	GND	8	NC
9	NO PIN	10	NC

J21 RJ45 LAN Port

Pin	Signal Name
1	L1_MDI+0
2	L1_MDI-0
3	L1_MDI+1
4	L1_MDI-1
5	VCTREF_GBEO_CT
6	GND
7	L1_MDI+2
8	L1_MDI-2
9	L1_MDI+3
10	L1_MDI-3
11	LAN1_ACT-
12	LAN1Link#
13	Lan1_link1000-
14	Lan1_link100-

J23 Dual Port USB

Pin	Signal Name	Pin	Signal Name
A1	+5V	B1	+5V
A2	USBD3-	B2	USBD2-
A3	USBD3+	B3	USBD2+
A4	GND	B4	GND

J24 Dual Port USB

Pin	Signal Name	Pin	Signal Name
A1	+5V	B1	+5V
A2	USBD5-	B2	USBD4-
A3	USBD5+	B3	USBD4+
A4	GND	B4	GND

J25 RS-232 DB9

Pin	Signal Name	Pin	Signal Name
1	DCD	2	RXD
3	TXD	4	DTR
5	GND	6	DSR
7	RTS	8	CTS
9	RI	10	NO PIN

J26 D-SUB15 VGA Connector

Pin	Signal Name	Pin	Signal Name
1	RED	2	GREEN
3	BLUE	4	ID0
5	GND	6	GND
7	GDN	8	GND
9	NC	10	GND
11	ID1	12	DDCDATA
13	HSYNC	14	VSYNC
15	DDCCLK		NO PIN

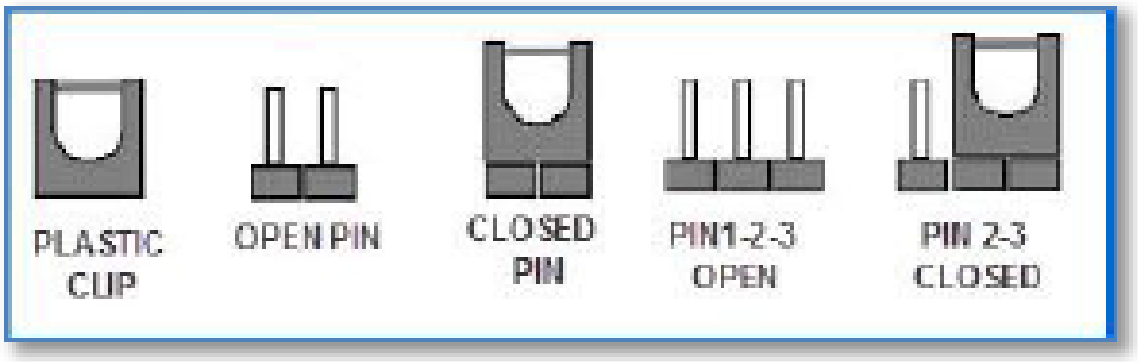
J27 Power Jack 12V

Pin	Signal Name
2	V12CON_IN
3	GND
4	GND

Configuracion de Jumpers

Un jumper es un puente de metal utilizado para cerrar un circuito eléctrico. Se compone de dos o tres clavijas de metal y un pequeño clip de metal (a menudo protegidos por una cubierta de plástico) que se desliza sobre las pins para conectarlos.

Configuración de pines de los jumpers



Cierre el pasador clip de plástico para activar o desactivar la función de cabecera y el conector.

Para cambiar el nivel de tensión de pantalla LVDS utiliza esta configuración del puente como se muestra

JP1 LVDS

Pin Closed	Function
1-3	+3.3V VDD (Default)
5-3	+5V VDD
4-3	+12V VDD

Para activar el botón automático de energía utiliza esta configuración del jumper como se muestra en la siguiente tabla. Por defecto, el encendido automático está activado

JP3 Enable Auto Power

Pin Closed	Function
1-2	Enable auto power on (Default)

Para configurar el puerto USB como huésped o cliente utiliza esta configuración del jumper como se muestra en la siguiente tabla

JP5 USB Host or Client

Pin Closed	Function
1-2	Host (Default)
2-3	Client

Para configurar el puerto serie COM1 como RS-232 / RS-422 / RS-485 utiliza esta configuración del jumper como se muestra en la siguiente tabla

JP6 COM1 Function Control

Pin Closed	Function
5-6, 9-11, 10-12, 15-17, 16-18	RS-232 (Default)
3-4, 7-9, 8-10, 13-15, 14-16, 21-22	RS-422
1-2, 7-9, 8-10, 19-20	RS-485

Para cambiar LVDS mostrar de nuevo la tensión de luz utiliza esta configuración del jumper como se muestra en la siguiente tabla

JP7 Back Light Enable Signal Set

Pin Closed	Function
1-3, 2-4	5V, Active high (Default)
1-3, 4-6	12V, Active high
3-5, 2-4	5V, Active low
3-5, 4-6	12V, Active low

Para restablecer el contenido CMOS utiliza esta configuración de un jumper como se muestra en la tabla siguiente

JP8 RTC Power Connector

Pin Closed	Function
1-2	Normal Operation (Default)
2-3	Clear CMOS Contents

Puerto 1 de Uso general de E / S (**Game Port & MIDI Port)**

SYMBOL	PIN	I/O	FUNCTION
MSI GP20	119	INtu I/OD_{12t}	MIDI serial data input .(Default) General purpose I/O port 2 bit 0.
MSO IRQIN0	120	O8c INc	MIDI serial data output. (Default) Alternate Function input: Interrupt channel input.
GPSA2 GP17	121	INcsu I/OD_{12csu}	Active-low, Joystick I switch input 2. This pin has an internal pull-up resistor. (Default) General purpose I/O port 1 bit 7.
GPSB2 GP16	122	INcsu I/OD_{12csu}	Active-low, Joystick II switch input 2. This pin has an internal pull-up resistor. (Default) General purpose I/O port 1 bit 6.
GPY1 GP15	123	I/OD_{12csd} I/OD_{12cs}	Joystick I timer pin. This pin connects to Y positioning variable resistors for the Joystick. (Default) General purpose I/O port 1 bit 5.
GPY2 GP14 P16	124	I/OD_{12csd} I/OD_{12cs}	Joystick II timer pin. This pin connects to Y positioning variable resistors for the Joystick. (Default) General purpose I/O port 1 bit 4. Alternate Function Output: KBC P16 I/O port.
GPX2 GP13 P15	125	I/OD_{12csd} I/OD_{12cs}	Joystick II timer pin. This pin connects to X positioning variable resistors for the Joystick. (Default) General purpose I/O port 1 bit 3. Alternate Function Output: KBC P15 I/O port.
GPX1 GP12 P14	126	I/OD_{12csd} I/OD_{12cs}	Joystick I timer pin. This pin connects to X positioning variable resistors for the Joystick. (Default) General purpose I/O port 1 bit 2. Alternate Function Output: KBC P14 I/O port.
GPSB1 GP11 P13	127	INcsu I/OD_{12csu}	Active-low, Joystick II switch input 1. (Default) General purpose I/O port 1 bit 1. Alternate Function Output: KBC P13 I/O port.
GPSA1 GP10 P12	128	INcsu I/OD_{12csu}	Active-low, Joystick I switch input 1. (Default) General purpose I/O port 1 bit 0. Alternate Function Output: KBC P12 I/O port.

Puerto 2 de Uso general de E / S (fuente de alimentación es Vcc)

SYMBOL	PIN	I/O	FUNCTION
GP20	119	I/OD_{12t}	General purpose I/O port 2 bit 0.
GP21	92	I/OD_{12t}	General purpose I/O port 2 bit 1.
GP22	91	I/OD_{12t}	General purpose I/O port 2 bit 2.
GP23	90	I/OD_{24t}	General purpose I/O port 2 bit 3.
GP24	89	I/OD_{12t}	General purpose I/O port 2 bit 4.
GP25	88	I/OD_{12t}	General purpose I/O port 2 bit 5.
GP26	87	I/OD_{12t}	General purpose I/O port 2 bit 6.
GP27	2	I/OD_{24t}	General purpose I/O port 2 bit 7.

Dispositivo Lógico 8 (GPIO Puerto 2)

CR30 (GP20-GP27 0x00 predeterminado)

- Bit 7 - 1: Reservado.
- Bit 0 = 1 Activar GPIO2.
  = 0 GPIO2 está inactivo.

CRF0 (GP20-GP27 E / S de registro de selección. Predeterminado 0xFF)

- Cuando se pone a '1 ', correspondiente puerto GPIO se programa como un puerto de entrada.
- Cuando se pone a '0 ', correspondiente puerto GPIO se programa como un puerto de salida.

CRF1 (GP20-GP27 registro de datos. Predeterminado 0x00)

- Si un puerto está programado para ser un puerto de salida, entonces su respectivo bit puede ser leído / escrito.
- Si un puerto está programado para que sea un puerto de entrada, entonces su bit respectiva sólo puede leer.

CRF2 (GP20-GP27 inversión registro. 0x00 predeterminado)

- Cuando se pone a '1 ', el valor del puerto de entrada / salida está invertida.
- Cuando se establece en '0 ', el valor del puerto de entrada / salida es el mismo que en el registro de datos.

CRF3 (0x00 predeterminado)

- Bit 7 - 4: Estos bits seleccionan recurso IRQ para IRQIN1.
- Bit 3 - 0: Estos bits seleccionan recurso IRQ para IRQIN0.

CRF4 (Reservado)

CRF5 (registro de modo PLED. Predeterminado 0x00)

- Bit 7-6: seleccione el modo de PLED
  - = 00 pin Power LED tri-establecido.
  - = 01 Power LED pin es conducía bajo.
  - = 10 Power LED pin es un pulso 1 Hz alternancia con el ciclo de trabajo del 50
  - = 11 Power LED pin es un pulso 1/4Hz palanca con un ciclo de trabajo del 50.

- Bit 5-4: Reservado
- Bit 3: seleccione WDTO modo de recuento.
  - = 0 segundos
  - = 1 minuto

- Bit 2: Habilitar el flanco ascendente de restablecimiento del teclado (P20) para forzar Tiempo de espera de eventos.
  - = 0 Desactivar
  - = 1 Habilitar

- Bit 1-0: Reservado

CRF6 (0x00 predeterminado)

Watch Dog Timer Tiempo de espera de valor. Escribir un valor distinto de cero en este registro hace que el contador se carga el valor del contador que hay que vigilar perro y comienza a contar hacia abajo. Si el bit 7 y bit 6 se establecen, cualquier interrupción del ratón o del teclado evento de interrupción también hará que la recarga de los previamente cargado valor distinto de cero a tener en contra del perro e iniciar la cuenta atrás. La lectura de este registro devuelve el valor actual del contador en lugar de Watch Dog Watch Dog Timer valor de tiempo de espera.

Bit 7 - 0 = 0x00 Tiempo de espera Desactivar
0x01 = Tiempo de espera se produce después de 1 segundo / minuto
0x02 = Tiempo de espera se produce después de 2 segundos / minutos
0x03 = Tiempo de espera se produce después de 3 segundos / minutos
.....
= 0xFF tiempo de espera se produce después de 255 segundos / minutos

CRF7 (0x00 predeterminado)

- Bit 7: Reset del ratón interrupción Activar o Desactivar
  - = 1 Watch Dog Timer se restablece a una interrupción del ratón
  - = 0 Watch Dog Timer no se ve afectada por la interrupción del ratón
- Bit 6: Restablecer interrupción de teclado Activar o Desactivar
  - = 1 Watch Dog Timer se restablece a una interrupción de teclado
  - = 0 Watch Dog Timer no se ve afectada por la interrupción del teclado
- Bit 5: Watch Dog Timer Fuerza Tiempo de espera, escriba sólo
  - = 1 Watch Dog Timer Fuerza de tiempo de evento, este bit es auto-limpieza.
- Bit 4: Watch Dog Timer de estado, R / W
  - = 1 Watch Dog Timer de tiempo de espera se produjo
  - = 0 Watch Dog Timer contar
- Bit 3 -0: Estos bits seleccionan recurso IRQ para Watch Dog. Configuración de 2 selecciona SMI.

Puerto 3 de Uso general de E / S (Fuente de energía es VSB)

SYMBOL	PIN	I/O	FUNCTION
GP30 SLP_SX#	73	I/OD_{12t} INts	General purpose I/O port 3 bit 0. Chpset suspend C status input.
GP31 PWRCTL#	72	I/OD_{12t} O_{12}	General purpose I/O port 3 bit 1. This pin generates the PWRCTL# signal while the power failure. (Default)
GP32 PWROK	71	I/OD_{12t} OD_{12}	General purpose I/O port 3 bit 2. This pin generates the PWROK signal while the VCC come in. (Default)
GP33 RSMRST#	70	I/OD_{12t} OD_{12}	General purpose I/O port 3 bit 3. This pin generates the RSMRST signal while the VSB come in. (Default)
GP34 CIRRX#	69	I/OD_{12t} INts	General purpose I/O port 3 bit 4. Consumer IR receiving input. This pin can Wake-Up system from $S5_{cold}$. (Default)
GP35 SUSLED	64	I/OD_{24t} O_{24}	General purpose I/O port 3 bit 5. Suspend LED output, it can program to flash when suspend state. This function can work without VCC. (Default)

POWER PINS

SYMBOL	PIN	FUNCTION
VCC	12, 48, 77, 114	+5V power supply for the digital circuitry.
VSB	61	+5V stand-by power supply for the digital circuitry.
VCC3V	28	+3.3V power supply for driving 3V on host interface.
AVCC	97	Analog VCC input. Internally supplier to all analog circuitry.
AGND	93	Internally connected to all analog circuitry. The ground reference for all analog inputs..
VSS	20, 55, 86, 117	Ground.

visión de conjunto

El principal objetivo es mostrar que los circuitos lógicos digitales pueden ser combinadas con estado-of-the-art plataformas informáticas integradas para crear sistemas integrados que contienen componentes personalizados lógica. Los laboratorios están basados en la plataforma Intel Atom, que es ampliamente utiliza el procesador integrado que puede ejecutar un sistema operativo regular. Las tareas de laboratorio consisten en dos partes:

- Parte I: 7-Segment Contador
- Parte II: Iterativo 32-Bit Adder

Se espera completar el laboratorio de forma individual. I y II Partes del laboratorio no debe tomar más de alrededor de dos horas en completarse.

Equipos para laboratorio

Para completar el laboratorio asignación, tendrá una "caja de laboratorio" que contiene el procesador Atom, una placa, y piezas varias. Una vez que tenga una caja, utilice una de las estaciones de trabajo donde se puede conectar el procesador a un monitor, teclado y ratón. Si lo prefiere, también puede tomar la caja a un lugar diferente. en que caso, usted tendrá que proporcionar el equipo necesario (monitor, cables, etc.) Si transportar la caja, por favor llévelo con cuidado para evitar que su contenido se agita a su alrededor.

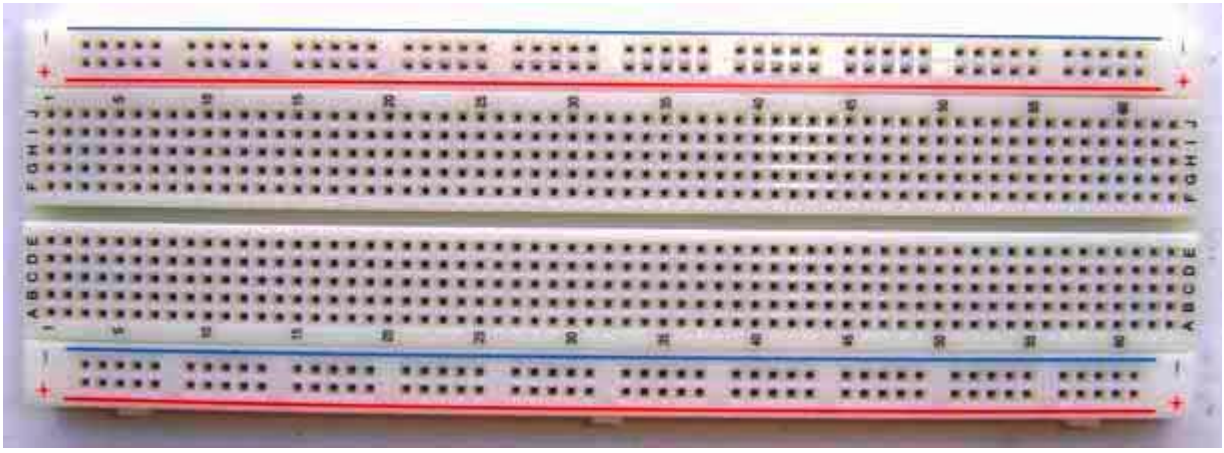
Background

LO siguiente proporciona algunos antecedentes para la asignación de laboratorio. Pasen por alto si usted está familiarizado con este material.

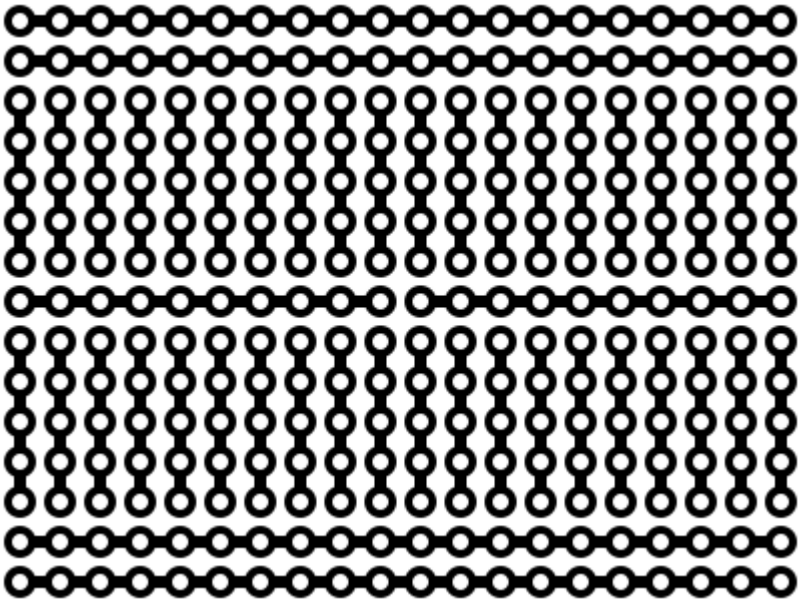
Breadboard

El protoboard o «breadboard» en inglés es un tablero con orificios conectados eléctricamente entre si, habitualmente siguiendo patrones de líneas, en el cual se pueden insertar componentes electrónicos y cables para el armado y prototipado de circuitos electrónicos y sistemas similares. Está hecho de dos materiales, un aislante, generalmente un plástico, y un conductor que conecta los diversos orificios entre si. Uno de sus usos principales es la creación y comprobación de prototipos de circuitos electrónicos antes de llegar a la impresión mecánica del circuito electrónico en sistemas de producción comercial.

Un Protoboard puede ser utilizado para los circuitos de prototipos electrónicos. La disposición de una placa típica se compone de dos zonas, llamadas tiras. Uno de ellos es la Regleta de bornes, que está en el medio de una placa. La otra es la tira del bus, que está en el lado y se compone de dos columnas (una para suelo y uno para voltaje de suministro).



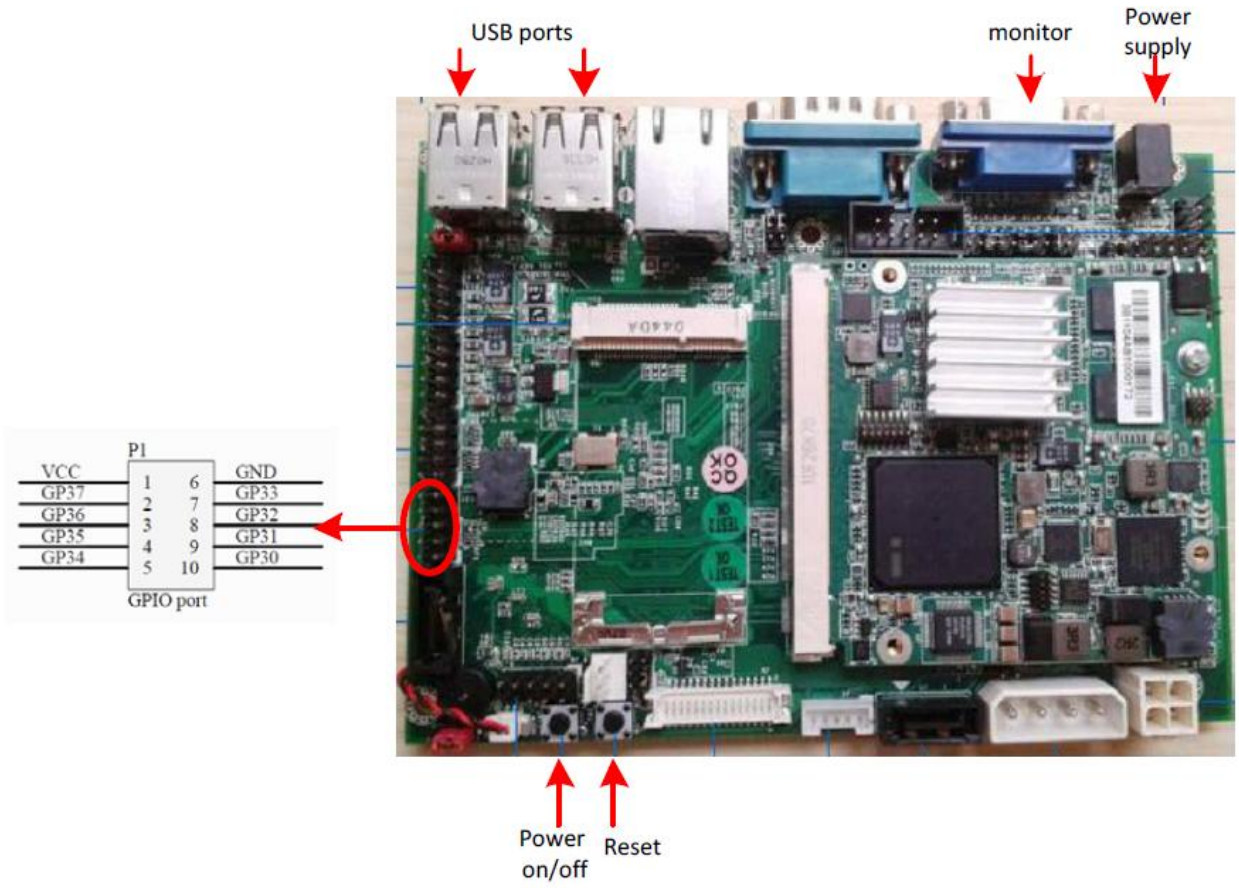
Las conexiones internas de las tiras se muestran en la siguiente figura



Patrón típico de disposición de las láminas
de material conductor en una placa de pruebas.

Atom Board

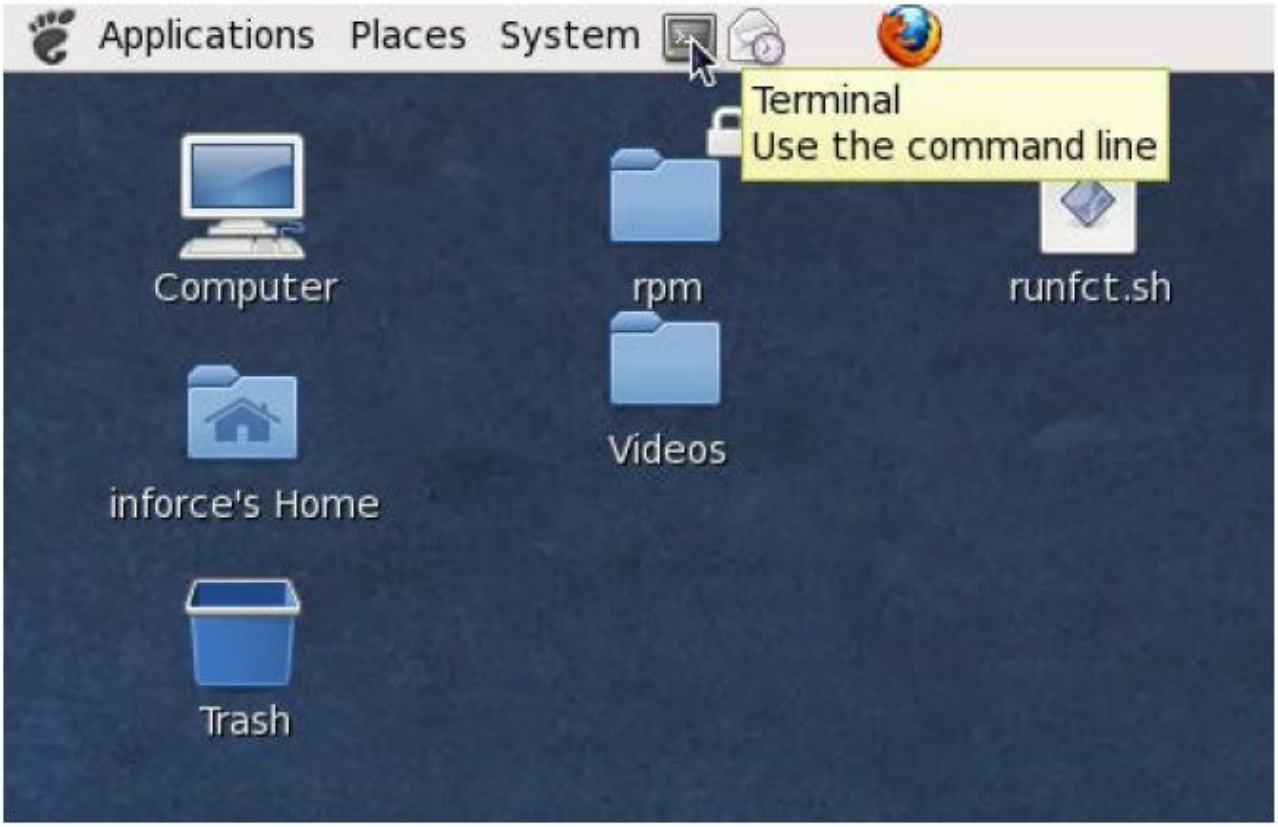
La siguiente figura muestra la placa Intel Atom que estamos utilizando para esta tarea. la bordo es un sistema completo con puertos procesador, memoria, disco, de entrada y salida, etc La ilustración muestra donde se puede conectar el monitor, el teclado, el ratón y potencia.



Para conectar nuestro circuito en el tablero para el sistema Atom, utilizamos el propósito general entrada / salida (GPIO) pasadores que se encuentran en el tablero como se muestra en la figura. Para conectar los cables a estos pines, utilice el cable especial incluido en la caja del laboratorio.

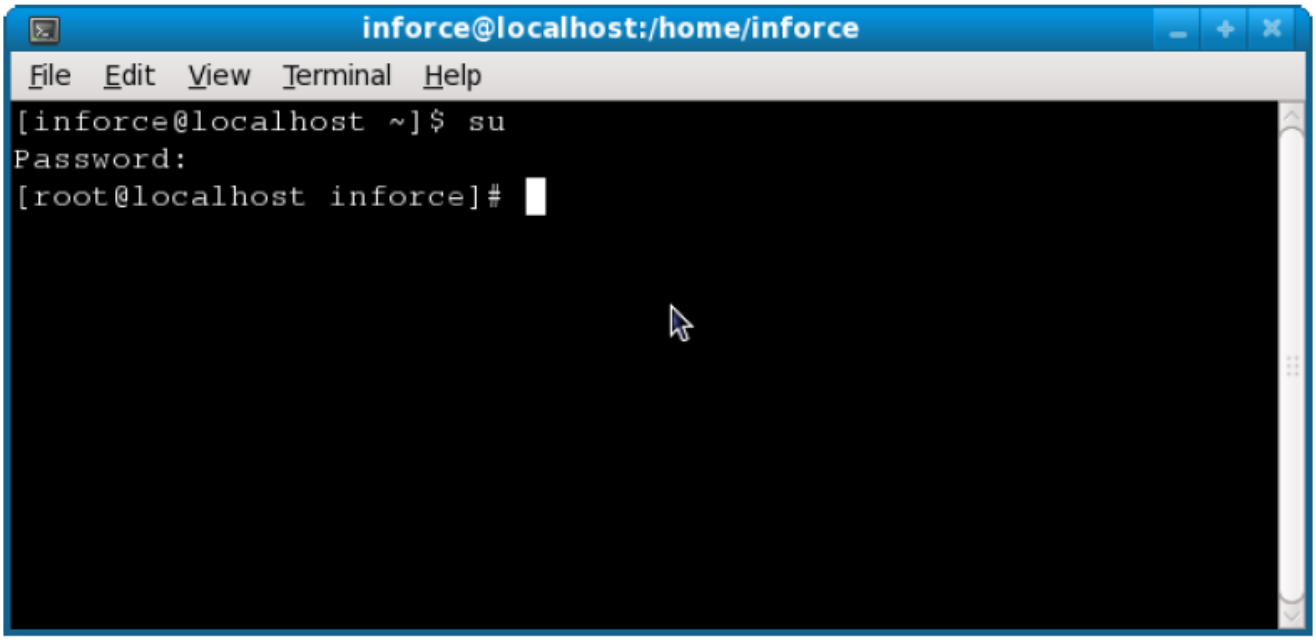
Si bien el sistema Atom está apagado, conectar los cables 10 al puerto GPIO se muestra en la figura anterior. La disposición de pines se muestra en la figura. Siempre asegúrese de que usted no lo hace accidentalmente corto Pins VCC y GND! Si lo hace, podría dañar la placa.

Después de conectar el monitor, un teclado USB y un ratón a la board Atom, enchufe el cable de alimentación y pulse el botón de encendido para arrancar el tablero. Al encender el sistema Atom, debe arrancar un sistema operativo Linux. Espere hasta que aparezca la interfaz de entrada y registro en (nombre de usuario: reforzar y contraseña: inforce123). Después de iniciar sesión, haga clic en el icono Terminal para entrar en la línea de comandos, como se muestra en la siguiente figura. ld dañar la placa.



En el terminal, escriba el siguiente comando:

su



La línea de comandos le pedirá una contraseña, que es inforce123. Ahora ya está listo para ejecutar los programas de la placa Atom. Por ejemplo, el siguiente comando se ejecute el programa en contra de la Parte I de la asignación:

./counter

Y el siguiente comando se ejecute el programa sumador para la Parte II de la asignación:

./adder

Si desea interrumpir un programa en ejecución, pulse Ctrl-C.

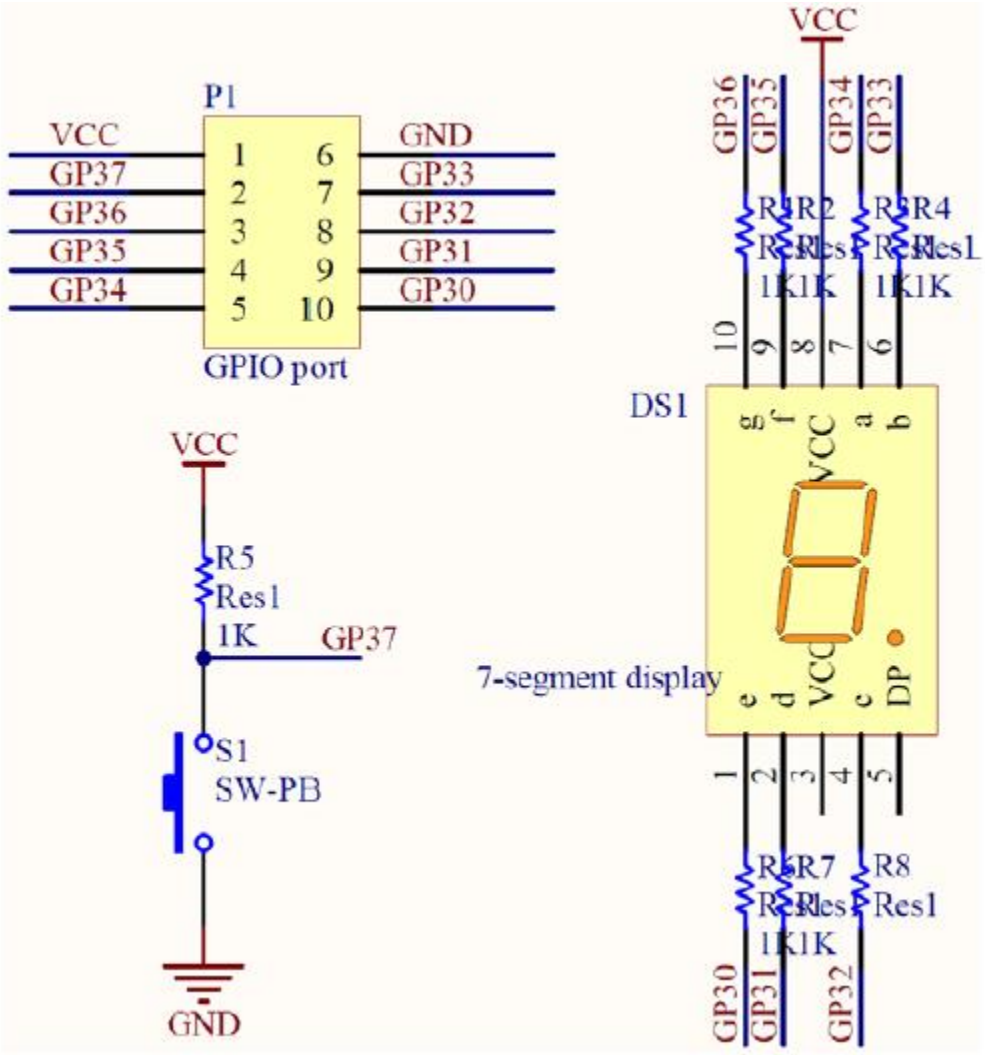
Parte I: 7-Segment Contador

En este experimento, se le conecta un LED de 7 segmentos de visualización para el sistema Atom. la software en el ordenador activará los pines de entrada / salida de modo que la pantalla contará 0 a 9. Hay un botón en el tablero que se utiliza para iniciar el conteo.

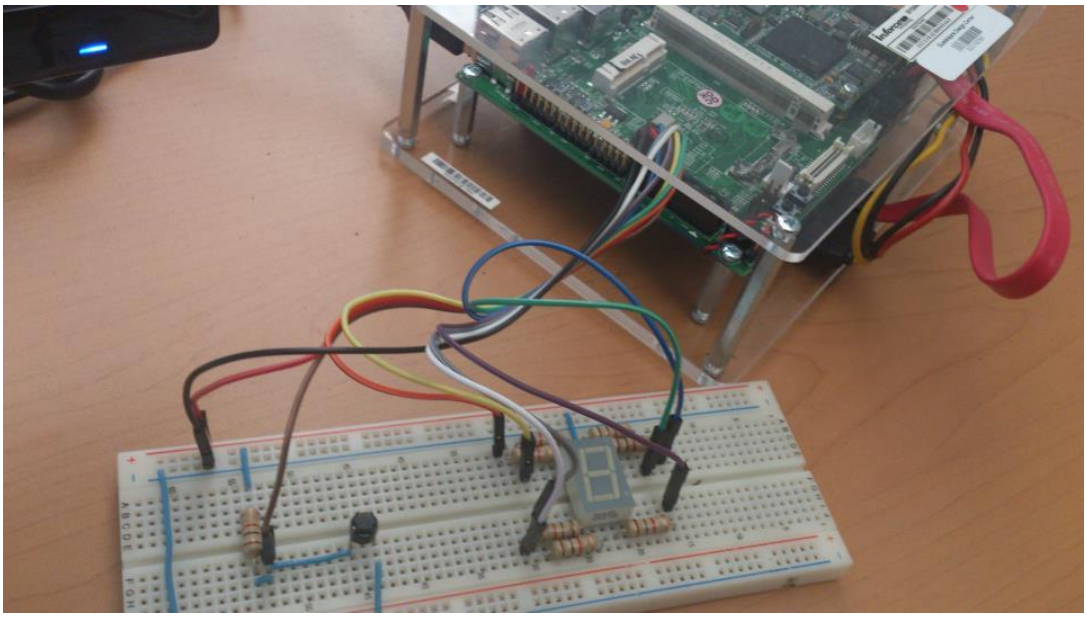
Los componentes necesarios son:

• 1 placa de pruebas
• 1 caja de cables,
• 1 pantalla de siete segmentos
• 8 1kW resistencias
• 1 botón

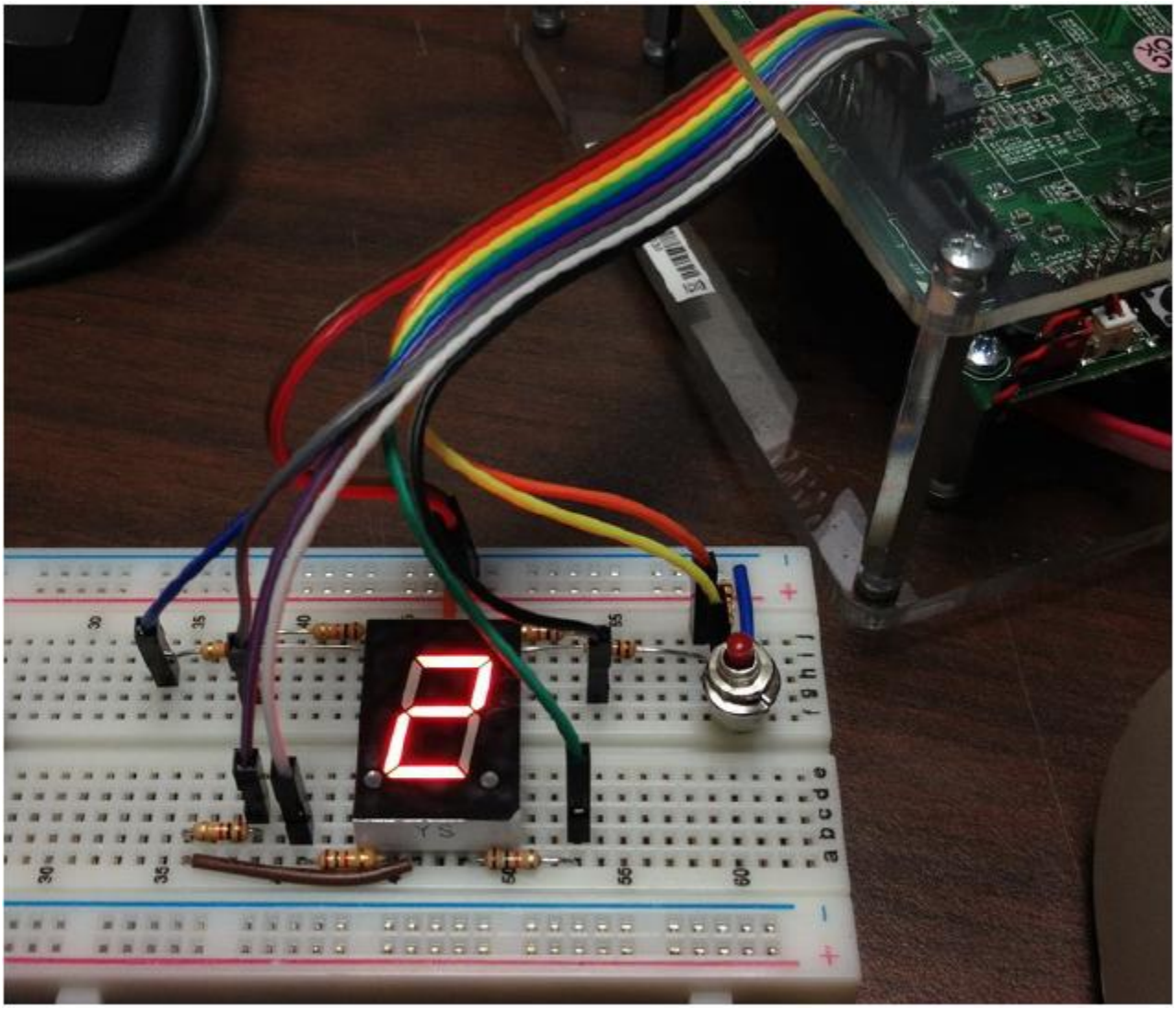
Coloque los dispositivos en la protoboard, alambre para arriba de acuerdo con el siguiente esquema:



Por favor recordar que los LEDs siempre tiene que estar conectado en línea con una resistencia para evitar daños en el LED!



Una vez terminado el cableado, la protoboard debería ser algo como esto:



A continuación, iniciar el contador de programa en la tarjeta Atom escribiendo el siguiente comando:

./counter

La pantalla debe mostrar 0 ahora. Una vez que pulse el botón, el conteo se inicia.

Asegúrese de que todos los números se muestran correctamente.

Para obtener crédito por este trabajo de laboratorio, presentar dos fotografías en un archivo zip en proymoodle:

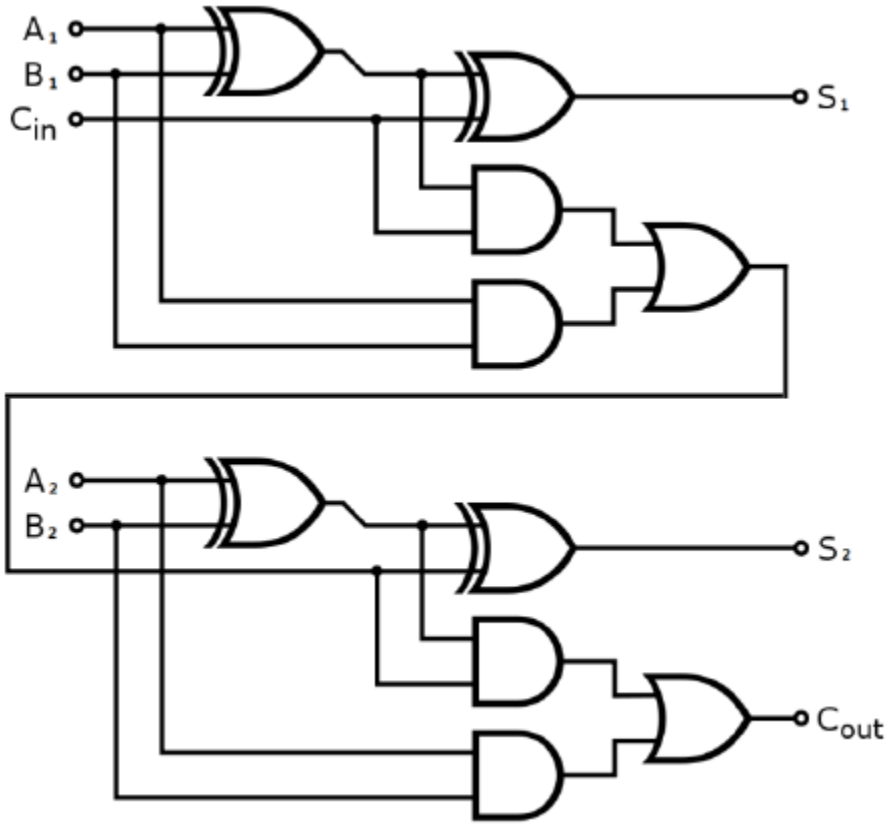
• 1 imagen que muestra un primer plano de tu protoboard en funcionamiento
• 1 imagen que muestra a usted, su placa, y la placa de Atom

Parte II: Iterativo 32-Bit Adder

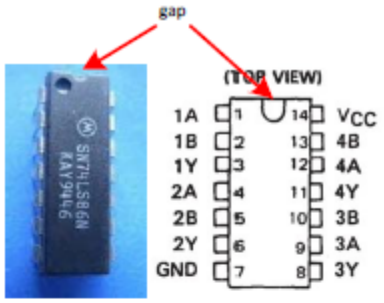
En este experimento, se construirá un sumador de dos bits que se utiliza varias veces para realizar un 32-bit Además. Este es un buen ejemplo de la construcción de su propia función de hardware (por ejemplo, 2-bit de suma) y usarlo inteligentemente a través de software para llevar a cabo una tarea más grande (es decir, la adición de 32-bit números).

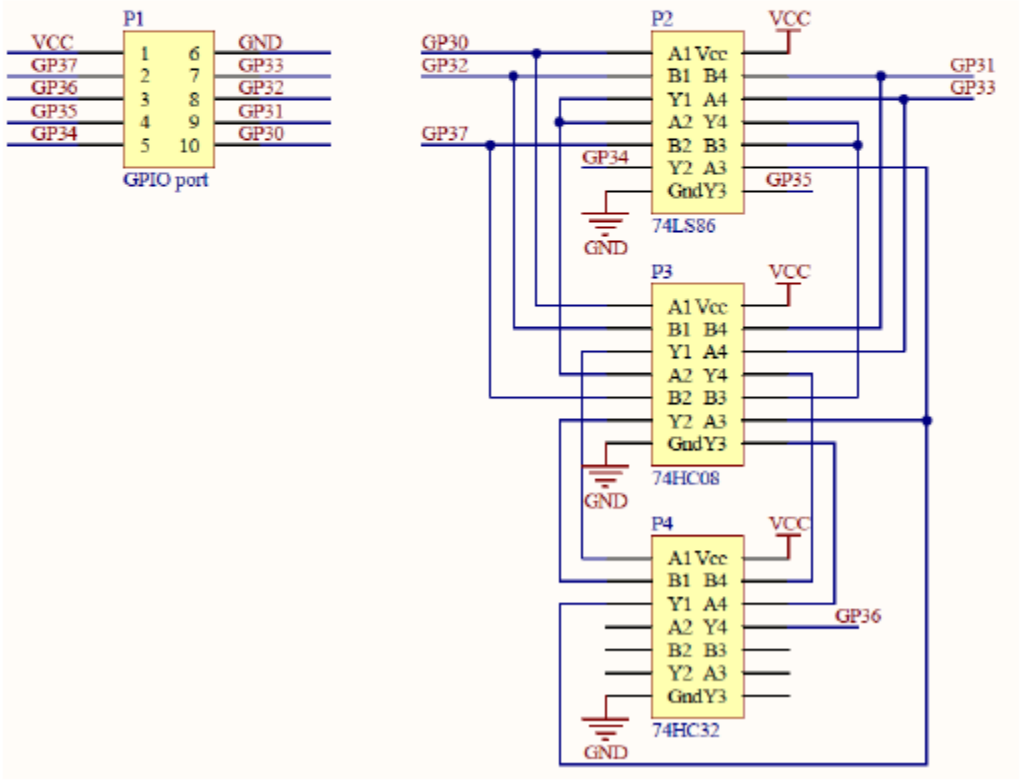
Componentes necesarios:
• 1 placa de pruebas
• 1 caja de cables
• 1 74LS86 (4 puertas XOR)
• 1 74HC08 (4 puertas AND)
• 1 74HC32 (4 puertas OR)

Puede implementar el circuito sumador de dos bits con 3 chips de la siguiente manera:

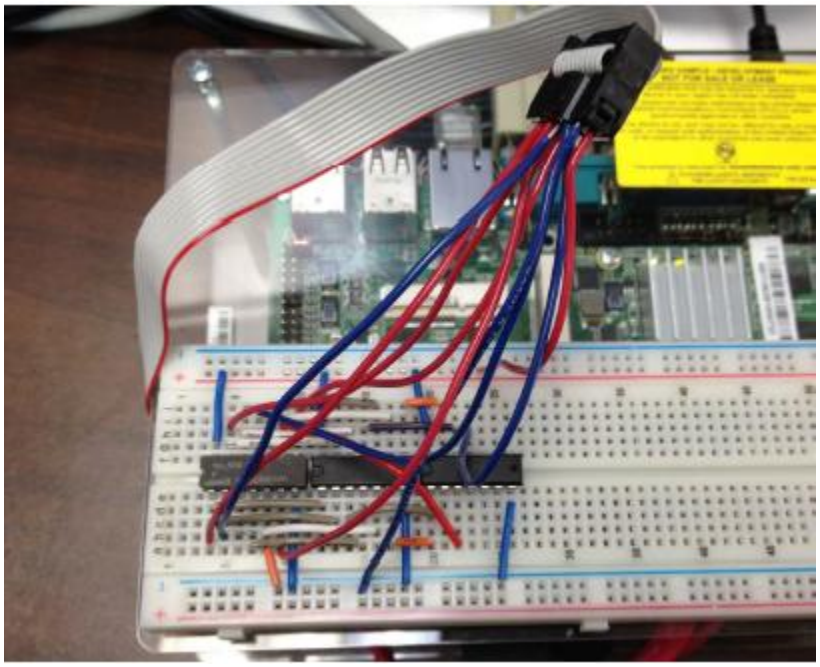


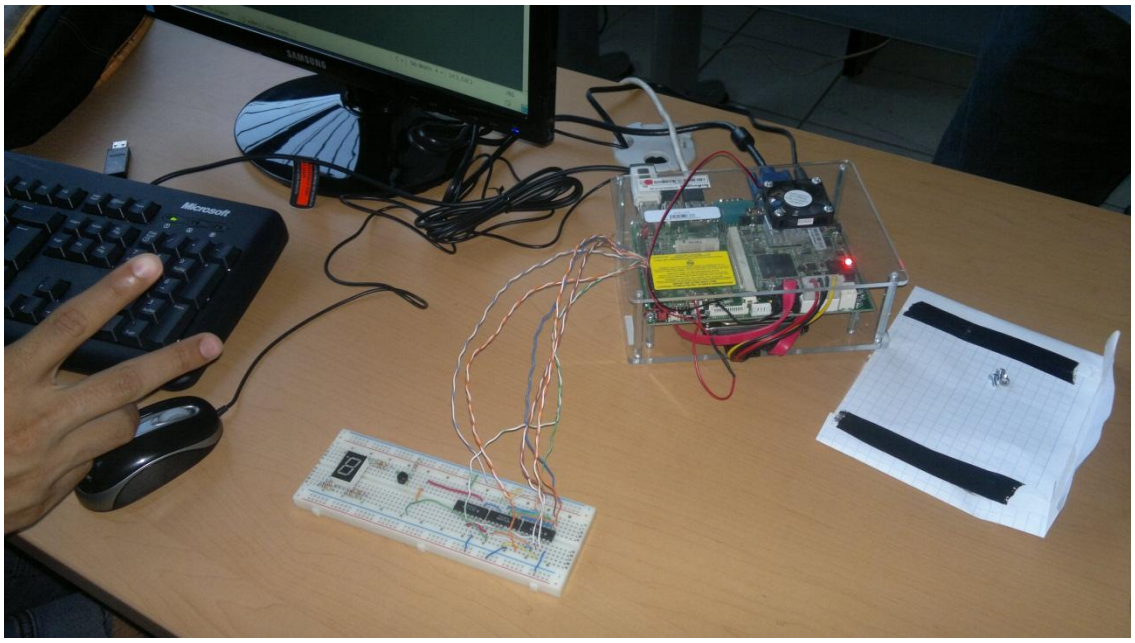
Coloque los dispositivos en la protoboard, alambre para arriba de acuerdo con el esquema siguiente. Asegúrese de colocar los chips en la orientación correcta. Nota existe una brecha que indica la parte superior del paquete.

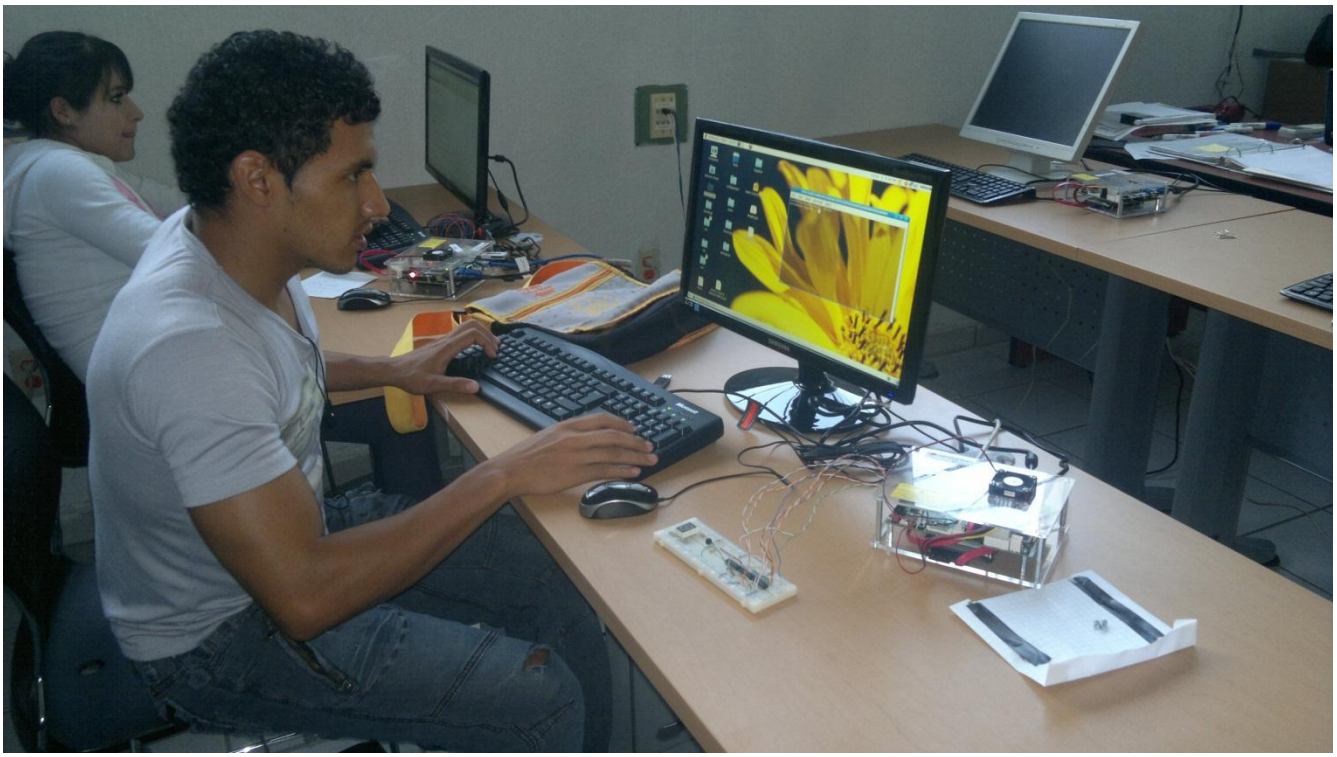


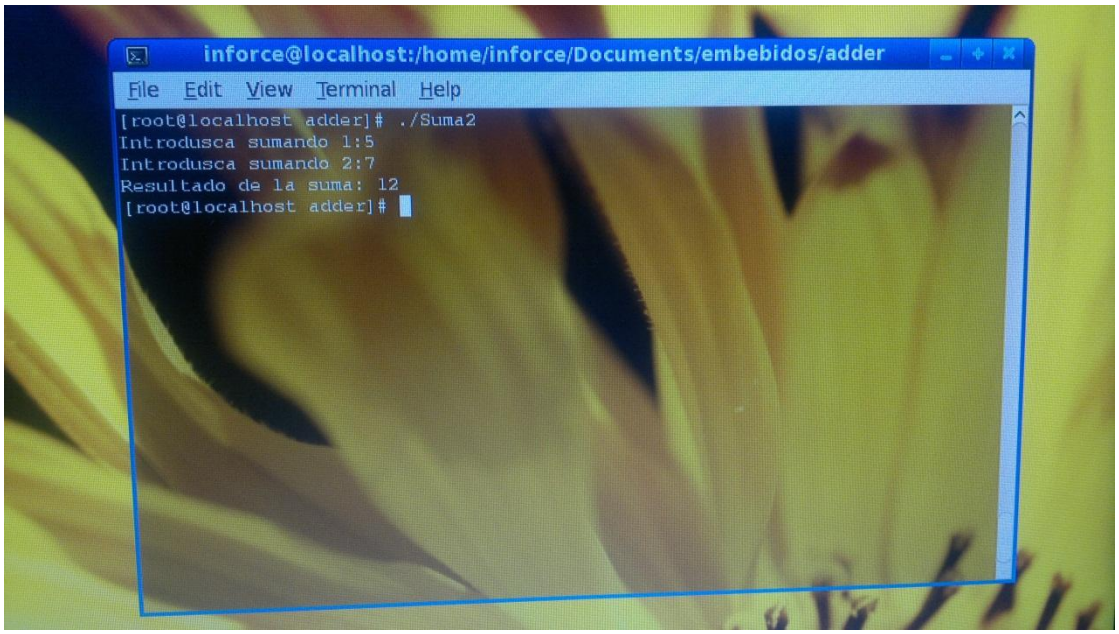


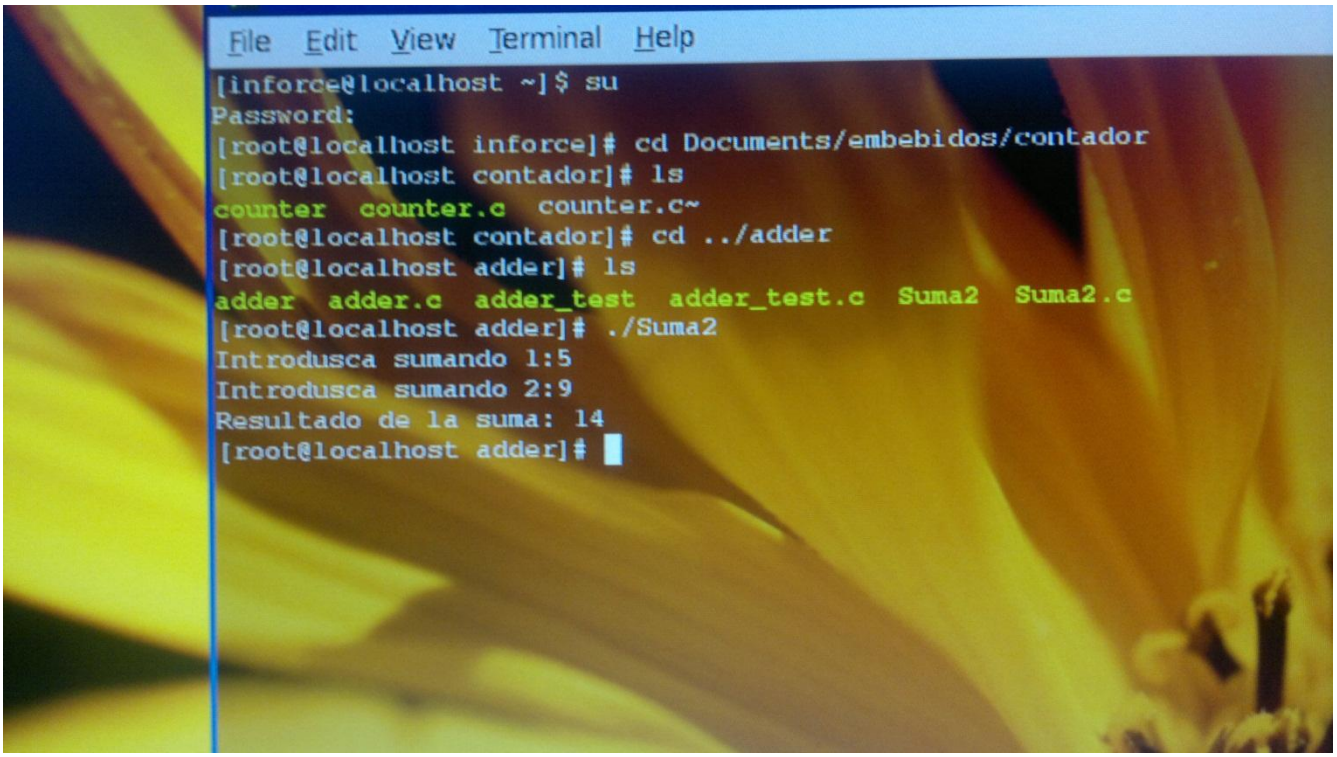
Una vez terminado el cableado, la protoboard debería ser algo como esto:




SAMSUNG
Microsoft




inforce@localhost:/home/inforce/Documents/embebidos/adder
File Edit View Terminal Help
[root@localhost adder]# ./Suma2
Introdusca sumando 1:5
Introdusca sumando 2:7
Resultado de la suma: 12
[root@localhost adder]#



d

A continuación, inicie el programa de prueba de la tarjeta Atom escribiendo el siguiente comando en la línea de comandos:

./adder_test

Este programa genera una serie de insumos para el sumador y comprueba si los resultados son correctos. Si el circuito sumador funciona correctamente, la salida de este programa debería tener este aspecto:

```
[root@localhost inforce]# ./adder_test
0+0=0
0+1=1
0+2=2
0+3=3
1+0=1
1+1=2
1+2=3
1+3=4
2+0=2
2+1=3
2+2=4
2+3=5
3+0=3
3+1=4
3+2=5
3+3=6
0+0+1=1
0+1+1=2
0+2+1=3
0+3+1=4
1+0+1=2
1+1+1=3
1+2+1=4
1+3+1=5
2+0+1=3
2+1+1=4
2+2+1=5
```

Si las ecuaciones de salida son incorrectas, el circuito no funciona correctamente. Revise los cables y chips conectados.

Después de pasar el programa de prueba, ejecute otro programa:

./adder

Este programa permite la entrada 2 enteros (valor debe estar entre 0 y 2147483647, ambos inclusive), y añadirlos con el circuito sumador. El resultado debería ser algo así:

[root@localhost] inforce# ./adder

```
[root@localhost inforce]# ./adder
This program calculates a+b, press Ctrl-C to exit
Input a:
1000
Input b:
42352
1000+42352=43352
```

Usted puede examinar con varios números para ver si el sumador funciona correctamente. Para obtener crédito por este trabajo de laboratorio, presentar dos fotografías en un archivo zip en proymoodle:

• 1 imagen que muestra un primer plano de tu protoboard
• 1 imagen que muestra a usted, su placa, la placa de Atom, y el monitor de funcionamiento el

programa sumador

Programas

counter.c

```
#include <stdio.h>
#include <sys/io.h>

#define w83627e 0x2e
#define w83627f 0x2f


unsigned char codec[10]={0x40,0x73,0x24,0x21,0x13,0x09,0x08,0x63,0x00,0x01};

void initreg() {

    //printf("Enter the Extended Function Mode\n");
          outb(0x87,w83627e);
          outb(0x87,w83627e);


    //printf("Select multi function pin as GPIO\n");
           outb(0x2C,w83627e);
           outb(0x02,w83627f);


   // printf("select logic device GPIO 2-5\n");
          outb (0x07,w83627e);
          outb(0x09,w83627f);

   //printf("activate GPIO 3, 0b00000010=0x02\n");
          outb(0x30,w83627e);
          outb(0x02,w83627f);

  //printf("set GPIO 30-36 as output, GPIO 37 as Input, 0b10000000\n");
        outb(0xF0,w83627e);
        outb(0x80,w83627f);


   //printf("set GPIO 30-37 inversion register as not inverted,0b00000000\n");
          outb(0xF2,w83627e);
          outb(0x00,w83627f);
}

void exitreg(){
```

```
printf("exit extended function mode \n");
      outb(0xAA,w83627e);
}

 void writeByte (unsigned char a){
     initreg();
              outb(0xF1,w83627e);
              outb(a,w83627f);
              exitreg();
}

unsigned char readbyte(){

  initreg();
           unsigned char a;
           outb(0xF1,w83627e);
           a=inb(w83627f);
           exitreg();
          return a;
}


int main (void)
{

  unsigned char a,b,c=0;
  iopl(3);
     printf( "This program drives an led counter, press Ctrl-c to exit\n");

while(1)
{
   //a=0x01:button is up, a=0x00: button is down

a=readbyte()&0x80;
//b is used as jitter filter

b=(b>>1)|a;

//if (b==0xFE)
//printf("button up\n");
if(b==0x1)
//printf("Button down\n");

c+=1;
if(c>9)
c=0;

printf("%d", c);
```

```
writeByte(codec[c]);
usleep(5000);
}

  return 0;
}
```

adder .c

```
#include<stdio.h>
#include<sys/io.h>

#define w83627e 0x2e
#define w83627f 0x2f

void initreg()
{
  //printf("Enter the Extended Function Mode\n");
          outb(0x87,w83627e);
          outb(0x87,w83627e);
  //printf("Select multi function pin as GPIO\n");
          outb(0x2C,w83627e);
          outb(0x02,w83627f);
  //printf("Select logic device GPIO 2-5\n");
          outb(0x07,w83627e);
          outb(0x09,w83627f);
  //printf("Activate GPIO 3, 0b00000010=0x02\n");
          outb(0x30,w83627e);
          outb(0x02,w83627f);
  //printf("Set GPIO 30-33,37 as output, GPIO 34-36 as input, 0b01110000\n");
          outb(0xF0,w83627e);
          outb(0x70,w83627f);
  //printf("Set GPIO 30-37 inversion register as not inverted, 0b00000000\n");
          outb(0xF2,w83627e);
          outb(0x00,w83627f);
}

void exitreg(){
  //printf("Exit Extended Function Mode\n");
          outb(0xAA,w83627e);
}

void writebyte(unsigned char a){
  initreg();
          outb(0xF1,w83627e);
          outb(a,w83627f);
```

```
        exitreg();
}

unsigned char readbyte(){
 initreg();
        unsigned char a;
        outb(0xF1,w83627e);
        a=inb(w83627f);
        exitreg();
        return a;
}

int main(void)
{
 unsigned char a_2bit,b_2bit,sum_2bit,outbyte,inbyte,inputbuffer[256],i,c_in;
 unsigned int a_32bit,b_32bit,sum_32bit;
 iopl(3);
 /*
 for(a_2bit=0x00;a_2bit<0x04;a_2bit++)
 {
  for(b_2bit=0x00;b_2bit<0x04;b_2bit++){
   outbyte=((b_2bit << 2)|a_2bit|0x80)&0x8F;
   writebyte(outbyte);
   usleep(100000);
   inbyte=readbyte();
   sum_8bit=(inbyte&0x70)>>4;
   printf("outbyte=%x,inbyte=%x,%d+%d+1=%d\n",outbyte,inbyte,a_2bit,b_2bit,sum_8bit);
  }
 }*/
 printf("This program calculates a+b, press Ctrl-C to exit\n");
 while(1){
  printf("Input a:\n");
        scanf("%s",inputbuffer);
        a_32bit=atoi(inputbuffer);
                printf("Input b:\n");
                scanf("%s",inputbuffer);
        b_32bit=atoi(inputbuffer);
        c_in=0x00;
        sum_32bit=0x00000000;
        for (i=0;i<=30;i+=2){
                        a_2bit=(unsigned char)((a_32bit >> i) & 0x00000003);
                        b_2bit=(unsigned char)((b_32bit >> i) & 0x00000003);
                outbyte=(b_2bit << 2)|a_2bit|c_in;
                writebyte(outbyte);
                usleep(100000);
                inbyte=readbyte();
                c_in=(inbyte<<1)&0x80;
                sum_2bit=(inbyte&0x30) >> 4;
```

```
                sum_32bit=sum_32bit|(sum_2bit<<i);
        }
    printf("%u+%u=%u\n\n",a_32bit,b_32bit,sum_32bit);
    if (c_in!=0x00){
        printf("Carry overflow!\n");
    }
  }
  return 0;
}
```

adder_test .c

```
#include<stdio.h>
#include<sys/io.h>

#define w83627e 0x2e
#define w83627f 0x2f

void initreg()
{
  //printf("Enter the Extended Function Mode\n");
        outb(0x87,w83627e);
        outb(0x87,w83627e);
  //printf("Select multi function pin as GPIO\n");
        outb(0x2C,w83627e);
        outb(0x02,w83627f);
  //printf("Select logic device GPIO 2-5\n");
        outb(0x07,w83627e);
        outb(0x09,w83627f);
  //printf("Activate GPIO 3, 0b00000010=0x02\n");
        outb(0x30,w83627e);
        outb(0x02,w83627f);
  //printf("Set GPIO 30-33,37 as output, GPIO 34-36 as input, 0b01110000\n");
        outb(0xF0,w83627e);
        outb(0x70,w83627f);
  //printf("Set GPIO 30-37 inversion register as not inverted, 0b00000000\n");
        outb(0xF2,w83627e);
        outb(0x00,w83627f);
}

void exitreg(){
  //printf("Exit Extended Function Mode\n");
        outb(0xAA,w83627e);
}

void writebyte(unsigned char a){
  initreg();
        outb(0xF1,w83627e);
        outb(a,w83627f);
        exitreg();
}

unsigned char readbyte(){
  initreg();
```

```
        unsigned char a;
        outb(0xF1,w83627e);
        a=inb(w83627f);
        exitreg();
        return a;
}

int main(void)
{
 unsigned char a_2bit,b_2bit,sum_2bit,outbyte,inbyte,inputbuffer[256],i,c_in;
 iopl(3);

 for(a_2bit=0x00;a_2bit<0x04;a_2bit++)
 {
  for(b_2bit=0x00;b_2bit<0x04;b_2bit++){
   outbyte=((b_2bit << 2)|a_2bit|0x00)&0x8F;
   writebyte(outbyte);
   usleep(100000);
   inbyte=readbyte();
   sum_2bit=(inbyte&0x70)>>4;
   printf("%d+%d=%d\n",a_2bit,b_2bit,sum_2bit);
  }
 }
 for(a_2bit=0x00;a_2bit<0x04;a_2bit++)
 {
  for(b_2bit=0x00;b_2bit<0x04;b_2bit++){
   outbyte=((b_2bit << 2)|a_2bit|0x80)&0x8F;
   writebyte(outbyte);
   usleep(100000);
   inbyte=readbyte();
   sum_2bit=(inbyte&0x70)>>4;
   printf("%d+%d+1=%d\n",a_2bit,b_2bit,sum_2bit);
  }
 }
 return 0;
}
```

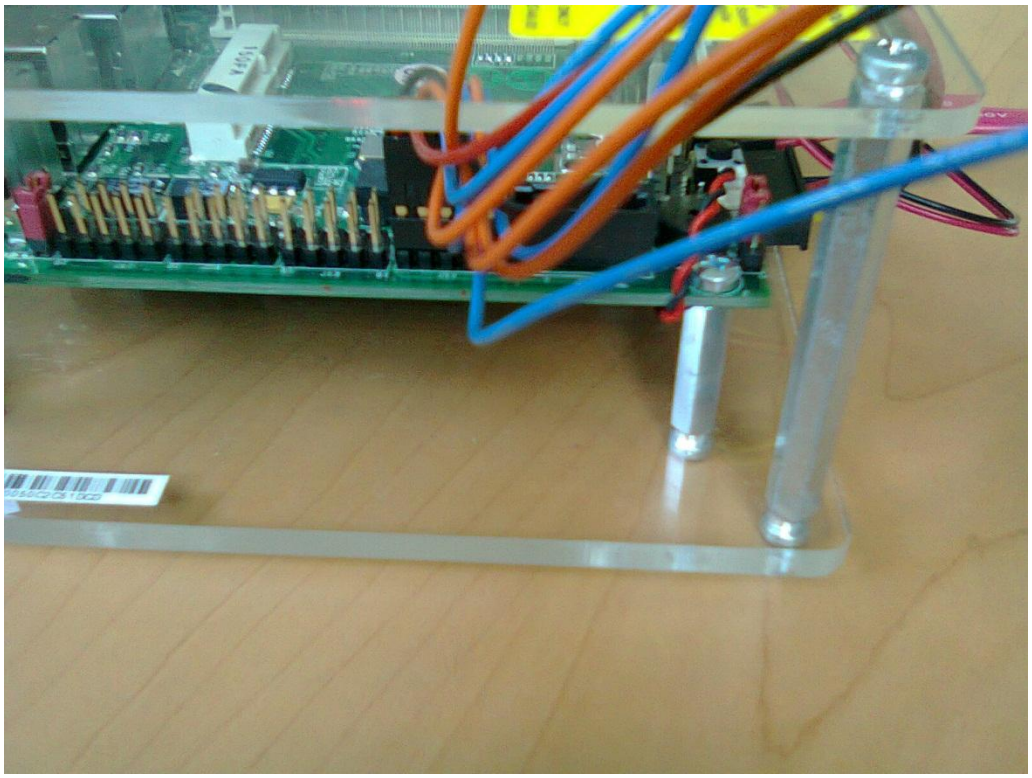
Rotabit

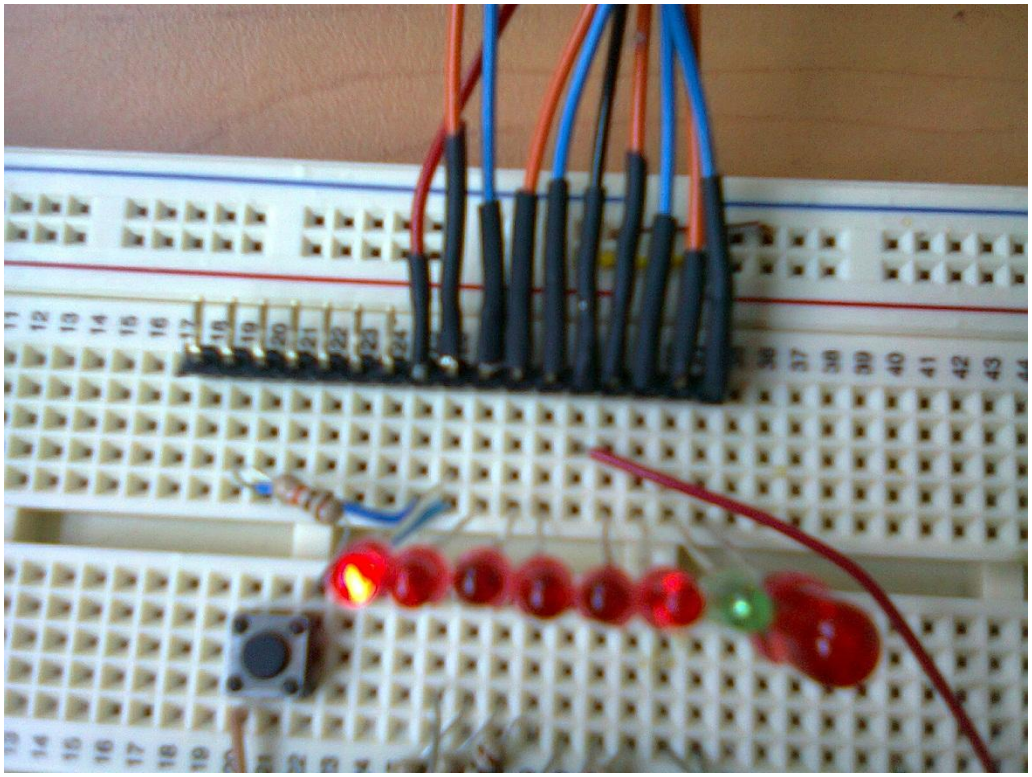
Se podrá visualizar el bit del puerto asignado, desplazándose por el puerto de lado a lado a través de una serie de LEDs conectados al ATOM. En este ejemplo se hará el programa en ATOM.

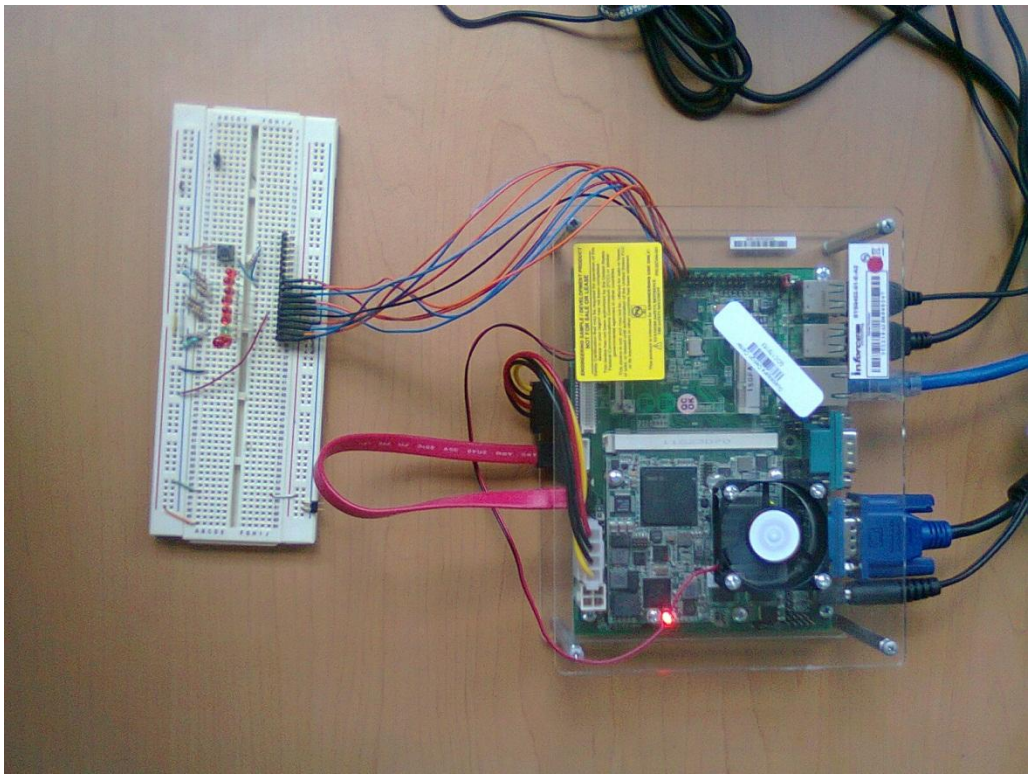
```
#include<stdio.h>
#include<sys/io.h>
#include<time.h>


#define w83627e 0x2e
#define w83627f 0x2f

void init_reg()
{
	// Entra en el modo de funciÃ³n extendida
	outb(0x87, w83627e);
	outb(0x87, w83627e);

	// Selecciona el pin multifunciones como GPIO
	outb(0x2C, w83627e);
	outb(0x02, w83627f);

	// Selecciona el dispositivo GPIO (3)
	outb(0x07, w83627e);
	outb(0x09, w83627f);

	// Activa el GPIO3
	outb(0x30, w83627e);
	outb(0x02, w83627f);

	// Configura los GPIO 30-36 como salida, GPIO37 como entrada
	outb(0xF0, w83627e);
	outb(0x00, w83627f);

	// COnfigura GPIO30-37 como no invertido
	outb(0xF2, w83627e);
	outb(0x00, w83627f);

}

void exit_reg()
{
```

```
	outb(0xAA, w83627e);
}

void write_byte(unsigned char a)
{
	init_reg();
	outb(0xF1, w83627e);
	outb(a, w83627f);
	exit_reg();
}

unsigned char read_byte()
{

	unsigned char a;
	init_reg();
	outb(0xF1, w83627e);
	a = inb(w83627f);
	exit_reg();
	return a;

}

int main(void)
{
	unsigned char i=0,a=1;
iopl(3);

	//read_byte()&0x80;
	while(1){
	for(i=0; i<7; i++){
	write_byte(a);
	usleep(1000000);
	a=(a<<1);}
	a=1;}
	//write_byte(0x00);

}
```

Cuando montemos los LED's en el protoboard, nosotros podremos ver el circuito como se ve en las siguientes imágenes.







El montado de los LED's en se usa también para los siguientes programas

Mete un dato y lo saca por un led

```
#include<stdio.h>
#include<sys/io.h>
#include<time.h>


#define w83627e 0x2e
#define w83627f 0x2f

void init_reg()
{
        // Entra en el modo de funciÃ³n extendida
        outb(0x87, w83627e);
        outb(0x87, w83627e);

        // Selecciona el pin multifunciones como GPIO
        outb(0x2C, w83627e);
        outb(0x02, w83627f);

        // Selecciona el dispositivo GPIO (3)
        outb(0x07, w83627e);
```

```
        outb(0x09, w83627f);

        // Activa el GPIO3
        outb(0x30, w83627e);
        outb(0x02, w83627f);

        // Configura los GPIO 30-36 como salida, GPIO37 como entrada
        outb(0xF0, w83627e);
        outb(0x80, w83627f);

        // COnfigura GPIO30-37 como no invertido
        outb(0xF2, w83627e);
        outb(0x00, w83627f);

}

void exit_reg()
{
        outb(0xAA, w83627e);
}

void write_byte(unsigned char a)
{
        init_reg();
        outb(0xF1, w83627e);
        outb(a, w83627f);
        exit_reg();
}

unsigned char read_byte()
{

        unsigned char a;
        init_reg();
        outb(0xF1, w83627e);
        a = inb(w83627f);
        exit_reg();
        return a;

}

int main(void)
{
        unsigned char i=0,a=1;
iopl(3);

        //read_byte()&0x80;
        while(1)
```

```
        {
        a = read_byte();
        if(a&0x80) write_byte(5);
        else write_byte(3);
        }
        //write_byte(0x00);

}
```

Le pulsaciones de un push bottom

```
#include<stdio.h>
#include<sys/io.h>
#include<time.h>


#define w83627e 0x2e
#define w83627f 0x2f

void init_reg()
{
        // Entra en el modo de funciÃ³n extendida
        outb(0x87, w83627e);
        outb(0x87, w83627e);

        // Selecciona el pin multifunciones como GPIO
        outb(0x2C, w83627e);
        outb(0x02, w83627f);

        // Selecciona el dispositivo GPIO (3)
        outb(0x07, w83627e);
        outb(0x09, w83627f);

        // Activa el GPIO3
        outb(0x30, w83627e);
        outb(0x02, w83627f);

        // Configura los GPIO 30-36 como salida, GPIO37 como entrada
        outb(0xF0, w83627e);
        outb(0x80, w83627f);

        // COnfigura GPIO30-37 como no invertido
        outb(0xF2, w83627e);
        outb(0x00, w83627f);

}
```

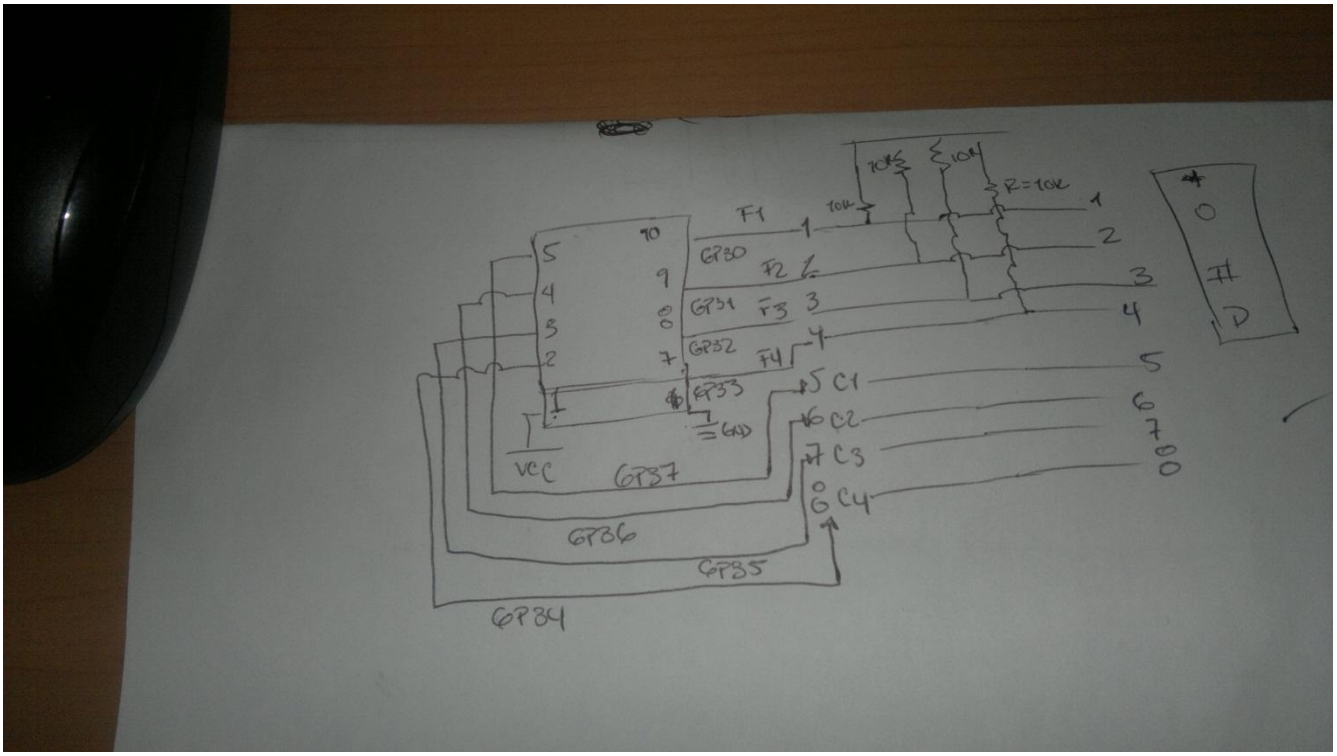
```
void exit_reg()
{
	outb(0xAA, w83627e);
}

void write_byte(unsigned char a)
{
	init_reg();
	outb(0xF1, w83627e);
	outb(a, w83627f);
	exit_reg();
}

unsigned char read_byte()
{

	unsigned char a;
	init_reg();
	outb(0xF1, w83627e);
	a = inb(w83627f);
	exit_reg();
	return a;

}

int main(void)
{
	unsigned char a=1;
	int i=0;
iopl(3);

	//read_byte()&0x80;
	while(1)
	{

	a = read_byte();
	if(a&0x80) {}
	else{
		write_byte(5);
		printf("Numero de personas: %i \n",i);
		i=i+1;

		}
	}
	//write_byte(0x00);

}
```

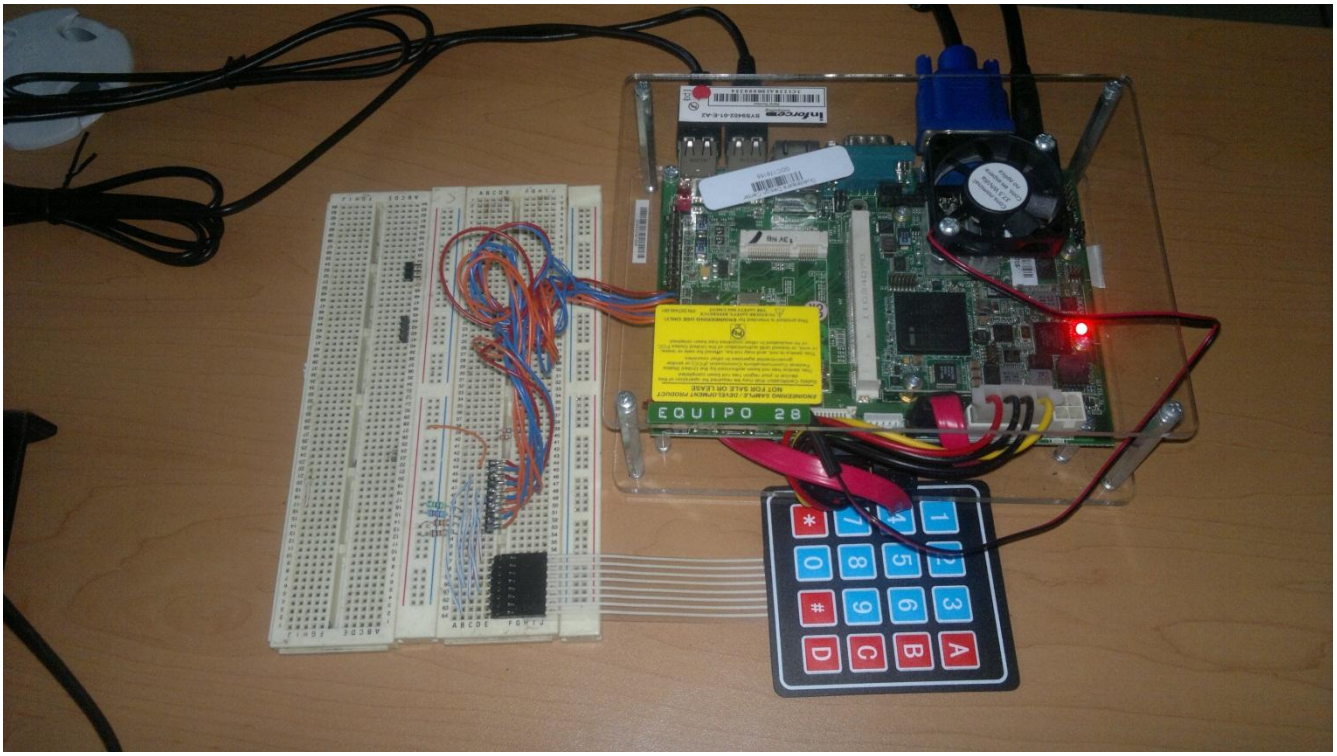
Mete pulsaciones de un push bottom, y corrige los rebotes.

```
#include<stdio.h>
#include<sys/io.h>
#include<time.h>


#define w83627e 0x2e
#define w83627f 0x2f

void init_reg()
{
	// Entra en el modo de funciÃ³n extendida
	outb(0x87, w83627e);
	outb(0x87, w83627e);

	// Selecciona el pin multifunciones como GPIO
	outb(0x2C, w83627e);
	outb(0x02, w83627f);

	// Selecciona el dispositivo GPIO (3)
	outb(0x07, w83627e);
	outb(0x09, w83627f);

	// Activa el GPIO3
	outb(0x30, w83627e);
	outb(0x02, w83627f);

	// Configura los GPIO 30-36 como salida, GPIO37 como entrada
	outb(0xF0, w83627e);
	outb(0x80, w83627f);

	// COnfigura GPIO30-37 como no invertido
	outb(0xF2, w83627e);
	outb(0x00, w83627f);

}

void exit_reg()
{
	outb(0xAA, w83627e);
}

void write_byte(unsigned char a)
{
	init_reg();
```

```
	outb(0xF1, w83627e);
	outb(a, w83627f);
	exit_reg();
}

unsigned char read_byte()
{

	unsigned char a;
	init_reg();
	outb(0xF1, w83627e);
	a = inb(w83627f);
	exit_reg();
	return a;

}

int main(void)
{
	unsigned char a=1;
	int i=0;
iopl(3);

	//read_byte()&0x80;
	while(1)
	{

	a = read_byte();
	if((a&0x80)==0)
		{

		usleep(500);
		a = read_byte();
			if((a&0x80)==0)
			{	write_byte(5);
				printf("Numero de personas: %i \n", i);
				i=i+1;

			}
			usleep(500000);
		}

	else write_byte(0);
	}
	//write_byte(0x00);

}
```

Teclado


F1
F2
F3
F4
C1
C2
C3
C4
GP30
GP31
GP32
GP33
GP34
GP35
GP36
GP37
VCC
GND
R=10K


EQUIPO 28


```
#include<stdio.h>
#include<sys/io.h>
#include<time.h>
#include<unistd.h>


#define w83627e 0x2e
#define w83627f 0x2f

void init_reg()
{
	// Entra en el modo de funciÃ³n extendida
	outb(0x87, w83627e);
	outb(0x87, w83627e);

	// Selecciona el pin multifunciones como GPIO
	outb(0x2C, w83627e);
```

```
	outb(0x02, w83627f);

	// Selecciona el dispositivo GPIO (3)
	outb(0x07, w83627e);
	outb(0x09, w83627f);

	// Activa el GPIO3
	outb(0x30, w83627e); //Indica que registro se va a modificar
	outb(0x02, w83627f); // El valor que tendra el registro

	// Configuracion de las entras/salidas del GP3. Si la bandera vale 0, se configura como salida, si vale 1 se configura como entrada.
	outb(0xF0, w83627e);
	outb(0x0F, w83627f);  //  0x0F corresponde a binario a 0b00001111   se configura el nibble alto como salida y en nibble bajo como entrada.

	// COnfigura GPIO30-37 como no invertido
	outb(0xF2, w83627e);
	outb(0x00, w83627f);

}

void exit_reg()
{
	outb(0xAA, w83627e);
}

void write_byte(unsigned char a)
{
	init_reg();
	outb(0xF1, w83627e);
	outb(a, w83627f);
	exit_reg();
}
```

```
unsigned char read_byte()
{

	unsigned char a;
	init_reg();
	outb(0xF1, w83627e);
	a = inb(w83627f);
	exit_reg();
	return a;

}

/*
 0111 - 7
 1001 - 9
 1010 - A
 1011 - B
 1100 - C
 1101 - D
 1110 - E
 1111 - F

*/

unsigned char teclado (void)
{
	unsigned char b=0x0F;
	while(1)
	{
		//printf("AQUI ESTOY\n");
		//fflush(stdout);
		//usleep(1000000);
```

```
write_byte(0xE0);    //// Escritura de la primer columna
usleep(1000);
b = read_byte();
(b = b&0x0F);
if (b!=0x0F){b+=0xE0;break;}

//////////////////
write_byte(0xD0);
usleep(1000);
b = read_byte();
(b = b&0x0F);
if (b!=0x0F){b+=0xD0;break;}



//////////////////
write_byte(0xB0);
usleep(1000);
b = read_byte();
(b = b&0x0F);
if (b!=0x0F){b+=0xB0;break;}


//////////////////
write_byte(0x70);
usleep(1000);
b = read_byte();
(b = b&0x0F);
if (b!=0x0F){b+=0x70;break;}


}
return b;
```

```
}
void lee_teclado(void)
{
	unsigned char b;
	b = teclado();
	//printf("%i", b);
	//fflush(stdout);
	switch(b)
	{
		case 0xEE: printf("UNO\n"); fflush(stdout); break;
		case 0xED: printf("CUATRO\n"); fflush(stdout); break;
		case 0xEB: printf("SIETE\n"); fflush(stdout); break;
		case 0xE7: printf("***\n"); fflush(stdout); break;

		case 0xDE: printf("DOS\n"); fflush(stdout); break;
		case 0xDD: printf("CINCO\n"); fflush(stdout); break;
		case 0xDB: printf("OCHO\n"); fflush(stdout); break;
		case 0xD7: printf("CERO\n"); fflush(stdout); break;

		case 0xBE: printf("TRES\n"); fflush(stdout); break;
		case 0xBD: printf("SEIS\n"); fflush(stdout); break;
		case 0xBB: printf("NUEVE\n"); fflush(stdout); break;
		case 0xB7: printf("GATO\n"); fflush(stdout); break;

		case 0x7E: printf("A\n"); fflush(stdout); break;
		case 0x7D: printf("B\n"); fflush(stdout); break;
		case 0x7B: printf("C\n"); fflush(stdout); break;
		case 0x77: printf("D\n"); fflush(stdout); break;
		default: printf("OTRO"); fflush(stdout); break;
	}
}
int main(void)
{
```

```
unsigned char a=1,b=0x0F;
int i=0;
iopl(3);
        while(1){
        lee_teclado();
        usleep(1000000);
        }

}
```

Libreria GTK

`

Para comenzar nuestra introducción a GTK, vamos a empezar con el programa más simple posible. Este programa creará una ventana de 200x200 píxeles y no hay manera de salir, sino para ser sacrificados utilizando el shell.

```
/* example-start base base.c */

#include <gtk/gtk.h>

int main( int   argc,
          char *argv[] )
{
    GtkWidget *window;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_widget_show  (window);

    gtk_main ();

    return(0);
}
```

/* example-end */

Puede compilar el programa anterior con cc o gcc usando:

linux01@jaelegz07-ubuntu:~/Documentos/programas/viernes23$ **cc base.c -o base $(pkg-config**

gtk+-2.0 --cflags --libs)
linux01@jaelegz07-ubuntu:~/Documentos/programas/viernes23$ **gcc base.c -o base $(pkg-config gtk+-2.0 --cflags --libs)**
y puede ejecutarlo de la siguiente forma:
linux01@jaelegz07-ubuntu:~/Documentos/programas/viernes23$ ./base

El significado de las opciones de compilación inusuales se explica a continuación en la Compilación de Hello World . Todos los programas incluyen, por supuesto, gtk / gtk.h que declara las variables, funciones, estructuras, etc, que se utilizarán en la aplicación GTK. La siguiente línea: `gtk_init (& argc, & argv);` llama a la función gtk_init (gint * argc, argv gchar ***), que se llama en todas las aplicaciones GTK. Esto crea un par de cosas para nosotros como el mapa por defecto visual y el color y luego procede a llamar gdk_init (gint * argc, argv gchar ***). Esta función inicializa la biblioteca para su uso, se configuran los controladores por defecto de la señal, y comprueba los argumentos pasados a la aplicación en la línea de comandos, en busca de uno de los siguientes:

- `--gtk-module`
- `--g-fatal-warnings`
- `--gtk-debug`
- `--gtk-no-debug`
- `--gdk-debug`
- `--gdk-no-debug`
- `--display`
- `--sync`
- `--no-xshm`
- `--name`
- `--class` Se elimina estos de la lista de argumentos, dejando cualquier cosa que no reconoce a su aplicación para analizar o ignorar. Esto crea un conjunto de argumentos estándar aceptado por todas las aplicaciones GTK.

Las siguientes dos líneas de código crear y mostrar una ventana.

```
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_widget_show (ventana);
```

El `GTK_WINDOW_TOPLEVEL` argumento especifica que queremos que la ventana de someterse decoración gestor de ventanas y colocación. En lugar de crear una ventana de tamaño 0x0, una ventana sin hijos está establecida en 200x200 por defecto por lo que aún puede manipular. El gtk_widget_show () permite GTK saber que estamos terminado de configurar los atributos de este widget, y que puede mostrarlo. La última línea entra en el bucle de procesamiento principal de GTK.

```
gtk_main ();
```

gtk_main () es otra llamada se verá en todas las aplicaciones GTK. Cuando el control llega a este punto, GTK va a dormir esperando eventos X (como pulsaciones de teclas o botones), tiempos de espera, o notificaciones de archivos IO que se produzca. En nuestro ejemplo simple, sin embargo, los eventos son ignorados.

Hello World en GTK

Ahora, para un programa con un control (un botón). Es el mundo clásico saludo a la GTK

```
/* example-start helloworld helloworld.c */

#include <gtk/gtk.h>

/* This is a callback function. The data arguments are ignored
 * in this example. More on callbacks below. */
void hello( GtkWidget *widget,
            gpointer   data )
{
    g_print ("Hello World\n");
}

gint delete_event( GtkWidget *widget,
                   GdkEvent  *event,
                   gpointer   data )
{
    /* If you return FALSE in the "delete_event" signal handler,
     * GTK will emit the "destroy" signal. Returning TRUE means
     * you don't want the window to be destroyed.
     * This is useful for popping up 'are you sure you want to quit?'
     * type dialogs. */

    g_print ("delete event occurred\n");

    /* Change TRUE to FALSE and the main window will be destroyed with
     * a "delete_event". */

    return(TRUE);
}

/* Another callback */
void destroy( GtkWidget *widget,
              gpointer   data )
{
    gtk_main_quit();
}

int main( int   argc,
          char *argv[] )
{
    /* GtkWidget is the storage type for widgets */
    GtkWidget *window;
    GtkWidget *button;

    /* This is called in all GTK applications. Arguments are parsed
     * from the command line and are returned to the application. */
    gtk_init(&argc, &argv);

    /* create a new window */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* When the window is given the "delete_event" signal (this is given
     * by the window manager, usually by the "close" option, or on the
     * titlebar), we ask it to call the delete_event () function
     * as defined above. The data passed to the callback
     * function is NULL and is ignored in the callback function. */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (delete_event), NULL);
```

```
    /* Here we connect the "destroy" event to a signal handler.
     * This event occurs when we call gtk_widget_destroy() on the window,
     * or if we return FALSE in the "delete_event" callback. */
    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        GTK_SIGNAL_FUNC (destroy), NULL);

    /* Sets the border width of the window. */
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    /* Creates a new button with the label "Hello World". */
    button = gtk_button_new_with_label ("Hello World");

    /* When the button receives the "clicked" signal, it will call the
     * function hello() passing it NULL as its argument.  The hello()
     * function is defined above. */
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (hello), NULL);

    /* This will cause the window to be destroyed by calling
     * gtk_widget_destroy(window) when "clicked".  Again, the destroy
     * signal could come from here, or the window manager. */
    gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                               GTK_SIGNAL_FUNC (gtk_widget_destroy),
                               GTK_OBJECT (window));

    /* This packs the button into the window (a gtk container). */
    gtk_container_add (GTK_CONTAINER (window), button);

    /* The final step is to display this newly created widget. */
    gtk_widget_show (button);

    /* and the window */
    gtk_widget_show (window);

    /* All GTK applications must have a gtk_main(). Control ends here
     * and waits for an event to occur (like a key press or
     * mouse event). */
    gtk_main ();

    return(0);
}
/* example-end */
```

Compilando Hello World

Para compilar utilizar:

linux01@jaelegz07-ubuntu:~/Documentos/programas/viernes23$ **cc** `helloworld`**.c -o** `helloworld`
$(pkg-config gtk+-2.0 --cflags --libs)
linux01@jaelegz07-ubuntu:~/Documentos/programas/viernes23$ **gcc** `helloworld`**.c -o** `helloworld`
$(pkg-config gtk+-2.0 --cflags –libs)
y lo ejecutas:
linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./helloworld

Se utiliza el programa `gtk-config` , que viene con GTK. Este programa "sabe" lo que

se necesitan modificadores del compilador para compilar programas que usan GTK. `gtk-config --cflags` dará salida a una lista de directorios de inclusión para el compilador que debe buscar en y `gtk-config --libs` dará salida a la lista de bibliotecas para el compilador para enlazar con los directorios y de encontrarlos pulg En el ejemplo anterior podrían haber sido combinados en una sola instancia, como `` `gtk-config --cflags --libs` `` . Tenga en cuenta que el tipo de comilla simple utilizado en el comando de compilación anterior es significativo. Las bibliotecas que están por lo general relacionadas en son: La librería GTK (-lgtk), la librería de widgets, con base en la parte superior de GDK. La biblioteca GDK (-LGDK), la envoltura de Xlib. La biblioteca GModule (-lgmodule), que se utiliza para cargar las extensiones de tiempo de ejecución. La biblioteca Glib (-lglib), que contiene funciones misceláneas; g_print solo () se utiliza en este ejemplo en particular. GTK se construye en la parte superior de glib para que siempre requerirá esta biblioteca. GLib Vea la sección de GLib para más detalles. La biblioteca Xlib (-lX11) que es utilizado por GDK. La biblioteca Xext (-lXext). Esta contiene código para pixmaps de memoria compartida y otras extensiones X. La biblioteca matemática (-lm). Esto es usado por GTK para diversos fines

Teoría de señales y respuestas

Antes de ver en detalle en *helloworld,* hablaremos de señales y respuestas. GTK es un evento impulsado conjunto de herramientas, lo que significa que se dormirá en gtk_main hasta que ocurre un evento y se pasa el control a la función apropiada. Este paso de control se realiza con la idea de "señales". (Obsérvese que estas señales no son las mismas que las señales de sistema Unix, y no se implementan utilizando ellos, aunque la terminología es casi idéntico.) Cuando se produce un evento, tal como la prensa de un botón del ratón, la señal apropiada será " emitida "por el aparato que se presionó. Así es como GTK hace la mayor parte de su trabajo útil. Hay señales que todos los widgets heredan, como "destroy", y hay señales que son widget específico, como "activarse" en un botón de activación. Para que un botón realice una acción, hemos creado un manejador de señales para capturar estas señales y llame a la función apropiada. Esto se hace mediante el uso de una función, tales como: `gtk_signal_connect gint (GtkObject objeto *,`

```
                          gchar nombre *,
                          GtkSignalFunc func,
                          gpointer func_data);
```

donde el primer argumento es el widget que se emite la señal, y el segundo el nombre de la señal que se desea capturar. La tercera es la función que desea ser llamado cuando es capturado, y el cuarto, los datos que desea haber pasado a esta función. La función especificada en el tercer argumento se llama una "función de devolución de llamada", y por lo general debe ser de la forma `vacío callback_func (GtkWidget * widget,`

```
                    gpointer callback_data);
```

donde el primer argumento será un puntero al widget que emitió la señal, y el segundo un puntero a los datos dados como último argumento en la gtk_signal_connect () como se muestra arriba. Tenga en cuenta que el formulario de arriba para obtener la declaración de la señal llamada de retorno es sólo una guía general, ya que algunas señales específicas de widgets generan diferentes parámetros de llamada.

Por ejemplo, la CList "select_row" señal proporciona tanto fila y parámetros de la columna. Otra llamada utilizada en el ejemplo *HelloWorld,* es: `gint gtk_signal_connect_object (* GtkObject objeto,`
`gchar nombre *,`
`GtkSignalFunc func,`
`GtkObject * slot_object);` gtk_signal_connect_object () es el mismo que gtk_signal_connect (), excepto que la función de devolución de llamada sólo usa un argumento, un puntero a un objeto GTK. Así que cuando se utiliza esta función para conectar señales, la devolución de llamada debe ser de la forma `vacío callback_func (GtkObject objeto *);` donde el objetivo es por lo general un widget. Por lo general, no configure callbacks para gtk_signal_connect_object sin embargo. Por lo general se utiliza para llamar a una función GTK que acepte un control único o un objeto como argumento, como es el caso de nuestro ejemplo *HelloWorld.* El propósito de tener dos funciones para conectar señales es simplemente para permitir que las devoluciones de llamada a tener un número diferente de argumentos. Muchas de las funciones de la biblioteca GTK aceptar sólo un puntero GtkWidget solo como argumento, por lo que desea utilizar el gtk_signal_connect_object () para ellos, mientras que para sus funciones, puede ser necesario contar con datos adicionales suministrados a las devoluciones de llamada.

Eventos

Además del mecanismo de señales descrito anteriormente, hay un conjunto de *eventos* que reflejan el mecanismo de eventos X. Devoluciones de llamada también puede estar unido a estos eventos. Estos eventos son: evento button_press_event button_release_event motion_notify_event delete_event destroy_event expose_event key_press_event key_release_event enter_notify_event leave_notify_event configure_event focus_in_event focus_out_event map_event unmap_event property_notify_event selection_clear_event selection_request_event selection_notify_event

- proximity_in_event proximity_out_event drag_begin_event drag_request_event drag_end_event drop_enter_event drop_leave_event drop_data_available_event other_event

Con el fin de conectar una función de devolución de llamada a uno de estos eventos, se utiliza la función gtk_signal_connect, como se describió anteriormente, usando uno de los nombres de eventos anteriores como el `name` parámetro. La función de devolución de llamada para eventos tiene una forma ligeramente diferente a la de las señales: `vacío callback_func (GtkWidget * widget,`
`* GdkEvent evento,`
`gpointer callback_data);` GdkEvent C es una `union` estructura cuyo tipo dependerá de cuál de los eventos anteriores ha ocurrido. Para que podamos decir qué evento se ha emitido cada una de las posibles alternativas tiene un `type` parámetro que refleja el evento que se emitan. Los otros componentes de la estructura de eventos dependerá del tipo de evento. Los valores posibles para el tipo son:

```
GDK_NOTHING
GDK_DELETE
GDK_DESTROY
GDK_EXPOSE
GDK_MOTION_NOTIFY
GDK_BUTTON_PRESS
GDK_2BUTTON_PRESS
GDK_3BUTTON_PRESS
GDK_BUTTON_RELEASE
GDK_KEY_PRESS
```

```
GDK_KEY_RELEASE
GDK_ENTER_NOTIFY
GDK_LEAVE_NOTIFY
GDK_FOCUS_CHANGE
GDK_CONFIGURE
GDK_MAP
GDK_UNMAP
GDK_PROPERTY_NOTIFY
GDK_SELECTION_CLEAR
GDK_SELECTION_REQUEST
GDK_SELECTION_NOTIFY
GDK_PROXIMITY_IN
GDK_PROXIMITY_OUT
GDK_DRAG_BEGIN
GDK_DRAG_REQUEST
GDK_DROP_ENTER
GDK_DROP_LEAVE
GDK_DROP_DATA_AVAIL
GDK_CLIENT_EVENT
GDK_VISIBILITY_NOTIFY
GDK_NO_EXPOSE
```

`GDK_OTHER_EVENT / * desuso, utilice los filtros lugar / *` Por lo tanto, para conectar una función de devolución de llamada a uno de estos eventos que usaría algo como: `gtk_signal_connect (GTK_OBJECT (botón), "button_press_event"`
`GTK_SIGNAL_FUNC (button_press_callback),`
`NULL);` Esto supone que `button` es un control Button. Ahora, cuando el ratón está sobre el botón y el botón del ratón pulsado, la función `button_press_callback` será llamado. Esta función puede ser declarado como: `estático button_press_callback gint (GtkWidget * widget,`
`* GdkEventButton evento,`
`gpointer de datos);` Tenga en cuenta que podemos declarar el segundo argumento como tipo `GdkEventButton` que sabemos qué tipo de evento se producirá para esta función a ser llamada. El valor devuelto por esta función indica si el evento debe ser propagado aún más por el mecanismo de manejo de eventos GTK. Volviendo TRUE indica que el evento ha sido manipulado, y que no debe propagar aún más. Volviendo FALSO continúa la gestión de eventos normal.

Advanced Event and Signal Handling

For details on the GdkEvent data types, see the appendix entitled GDK Event Types . paso a paso por Hello World

Ahora que ya conocemos la teoría detrás de esto, vamos a aclarar a pie a través del programa de ejemplo *HelloWorld.* Esta es la función callback que se llamará cuando el botón es "clickeado". Ignoramos tanto el control y los datos de este ejemplo, pero no es difícil de hacer cosas con ellos. En el ejemplo siguiente se utiliza el argumento de datos que nos diga qué botón se ha presionado.

```
void hello( GtkWidget *widget,
            gpointer   data )
{
```

```
    g_print ("Hello World\n");
}
```

La devolución de llamada siguiente es un poco especial. El "delete_event" se produce cuando el gestor de ventanas envía este evento a la aplicación. Tenemos una opción aquí en cuanto a qué hacer con estos eventos. Podemos ignorar, hacer algún tipo de respuesta, o simplemente salir de la aplicación. El valor que se devuelva en esta devolución de llamada le permite a GTK saber qué medidas tomar. Si devolvemos TRUE, lo dejamos saber que no queremos tener la señal "destroy" emitida, manteniendo nuestra aplicación en ejecución. Al devolver FALSE, le pedimos que "destruir" a ser emitidas, que a su vez llama a nuestra "destruir" manejador de señales.

```
gint delete_event( GtkWidget *widget,
                   GdkEvent  *event,
                   gpointer   data )
{
    g_print ("delete event occurred\n");

    return (TRUE);
}
```

Aquí hay otra función de devolución de llamada que hace que el programa se cierre llamando gtk_main_quit (). Esta función indica a GTK que debe salir de la gtk_main cuando se devuelve el control a la misma.

```
void destroy( GtkWidget *widget,
              gpointer   data )
{
    gtk_main_quit ();
}
```

Supongo que usted sabe acerca de la función main () ... Sí, como con otras aplicaciones, todas las aplicaciones GTK también tendrá uno de estos.

```
int main( int   argc,
          char *argv[] )
{
```

La siguiente parte declara punteros a una estructura de tipo GtkWidget. Estos se utilizan a continuación para crear una ventana y un botón.

```
    GtkWidget *window;
    GtkWidget *button;
```

Aquí está nuestra gtk_init nuevo. Al igual que antes, esto inicializa el conjunto de herramientas, y analiza los argumentos que aparecen en la línea de comandos. Cualquier argumento que reconoce desde la línea de co-

mando, se elimina de la lista, y modifica argc y argv para que se vea como si nunca hubieran existido, lo que permite su aplicación para analizar los argumentos restantes.

```
gtk_init (&argc, &argv);
```
Crear una nueva ventana. Esto es bastante sencillo. La memoria se asigna para la estructura GtkWidget * ventana por lo que ahora apunta a una estructura válida. Se establece una nueva ventana, pero no se muestra hasta que nosotros llamamos gtk_widget_show (ventana), cerca del final de nuestro programa.

```
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

He aquí dos ejemplos de cómo conectar un manejador de señal a un objeto, en este caso, la ventana. En este caso, las señales "delete_event" y "destruir" han sido detenidos. La primera se emite cuando se utiliza el gestor de ventanas para matar a la ventana, o cuando usamos la gtk_widget_destroy () llamar pasando el control de ventana como el objeto de destruir. El segundo se emite cuando, en el "delete_event" handler, nos devolverá FALSE. El `GTK_OBJECT` y `GTK_SIGNAL_FUNC` son macros que realizan tipo de fundición y comprobar por nosotros, así como ayudar a la legibilidad del código.

```
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (delete_event), NULL);
    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        GTK_SIGNAL_FUNC (destroy), NULL);
```
La siguiente función se utiliza para establecer un atributo de un objeto contenedor. Esto sólo establece la ventana por lo que tiene un área en blanco a lo largo del interior de la misma 10 píxeles de ancho donde no los widgets irá. Y de nuevo, `GTK_CONTAINER` es una macro para realizar la conversión de tipos.

```
  gtk_container_set_border_width (GTK_CONTAINER (window), 10);
```
Esta llamada crea un nuevo botón. Se asigna espacio para una estructura GtkWidget nuevo en la memoria, la inicializa, y hace que el punto de puntero botón a la misma. Contará con la etiqueta "Hello World" en él cuando se muestran.

```
button = gtk_button_new_with_label ("Hello World");
```

En este caso, tomamos este botón y hacer que haga algo útil. Adjuntamos un manejador de señales para lo que cuando se emite la señal "clicked", nuestra función hello () es llamado. Los datos son ignorados, por lo que basta con pasar NULL a la función de devolución de llamada hello (). Obviamente, la señal "clicked" se emite cuando hacemos clic en el botón con el puntero del ratón.

```
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (hello), NULL);
```

También vamos a usar este botón para salir de nuestro programa. Esto ilustrará cómo la señal "destroy" puede venir de cualquier gestor de ventanas, o nuestro programa. Al pulsar el botón "clicked", igual que antes, se llama la primera función hello () de devolución de llamada, y luego éste en el orden en que se crean. Usted puede tener funciones de devolución de llamada tantas como sea necesario, y todos serán ejecutados en el orden en que se conectan. Debido a que el gtk_widget_destroy () acepta sólo un widget * GtkWidget como argumento, se utiliza el gtk_signal_connect_object () de aquí en lugar de gtk_signal_connect recta ().

```
    gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                               GTK_SIGNAL_FUNC (gtk_widget_destroy),
                               GTK_OBJECT (window));
```

Packing Widgets Esta es una llamada de colocación, que se explicará en profundidad más adelante en Reproductores de embalaje . Sin embargo, es bastante fácil de entender. Simplemente le dice GTK que el botón se va a colocar en la ventana donde se mostrará. Tenga en cuenta que un contenedor GTK sólo puede contener un widget. Existen otros widgets, que se describen posteriormente, que están diseñados para los widgets de presentación múltiple de varias maneras.

`gtk_container_add (GTK_CONTAINER (window), button);` Ahora tenemos todo configurado como queremos que sea. Con todos los manejadores de señales en el lugar, y el botón situado en la ventana donde tiene que estar, pedimos a GTK para "mostrar" a los widgets en la pantalla. El widget ventana se muestra pasada por lo que la ventana entera aparezca de una vez en lugar de ver la ventana emergente, y luego la forma de botón dentro de ella. Aunque con un ejemplo tan simple, que nunca lo note.

```
    gtk_widget_show (button);

    gtk_widget_show (window);
```

Y por supuesto, que nosotros llamamos gtk_main () que espera a los acontecimientos por venir desde el servidor X y pedirá a los widgets para emitir señales cuando estos eventos venideros.

`gtk_main ();` Y la rentabilidad final. Control vuelve aquí después de gtk_quit () es llamado.

`return (0);` Ahora, cuando haga clic en el botón del ratón sobre un botón GTK, el control emitirá una señal "clicked". Para que podamos utilizar esta información, nuestro programa se configura un controlador de señales para capturar la señal, lo que a la función que nuestra elección. En nuestro ejemplo, al pulsar el botón que hemos creado es "clickeado", la función hello () se llama con un argumento

NULL, y luego el siguiente manejador para esta señal se llama. Esto llama a la función gtk_widget_destroy (), pasándole el widget ventana como su argumento, destruyendo el widget ventana. Esto hace que la ventana emita la señal "destroy", que está atrapado, y nos llama la destroy () la función de devolución de llamada, que simplemente sale de GTK. Otro curso de los acontecimientos es utilizar el gestor de ventanas para matar a la ventana, lo que hará que el "delete_event", que se emite. Para ello, será nuestro "delete_event" manejador. Si devolvemos TRUE aquí, la ventana se quedará como está y no pasará nada. Volviendo false hará que GTK emita la señal "destroy" que desde luego llama la "destrucción" de devolución de llamada, salir de GTK.

Moving On Data Types Tipos de datos

Hay algunas cosas que usted probablemente ha notado en los ejemplos anteriores que necesitan explicación. El gint, gchar, etc, que se ve son typedefs a int y char, respectivamente, que forman parte del sistema de GLlib. Esto se hace para moverse por esa dependencia desagradable en el tamaño de los tipos de datos simples al hacer cálculos. Un buen ejemplo es "gint32", que se typedef'd en un entero de 32 bits para cualquier plataforma dada, ya sea el alfa de 64 bits, o el i386 de 32 bits. Los typedefs son muy sencillas e intuitivas. Todos ellos están definidos en glib / glib.h (que se incluye en gtk.h). También te darás cuenta GTK capacidad de utilizar GtkWidget cuando la función requiere un objeto. GTK es un diseño orientado a objetos, y un widget es un objeto.

More on Signal Handlers Más sobre manejadores de señales

Vamos a echar otro vistazo a la declaración gtk_signal_connect.

```
gint gtk_signal_connect( GtkObject *object,
                         gchar *name,
                         GtkSignalFunc func,
                         gpointer func_data );
```

Tenga en cuenta el valor de retorno gint? Esta es una etiqueta que identifica su función de devolución de llamada. Como se mencionó anteriormente, es posible que como muchas devoluciones de llamada por señal y por objeto que estime oportuno, y cada uno será ejecutado a su vez, en el orden que se adjunta.

Esta etiqueta le permite eliminar esta devolución de llamada desde la lista mediante:

```
vacío gtk_signal_disconnect (* GtkObject objeto,
                             gint id);
```

Por lo tanto, al aprobar en el widget que desea eliminar el controlador de, y la etiqueta devuelto por una de las funciones signal_connect, puede desconectar un manejador de señales. También puede deshabilitar temporalmente manejadores de señales con el gtk_signal_handler_block () y familia gtk_signal_handler_unblock () de funciones. `void gtk_signal_handler_block(`

```
GtkObject *object,
                               guint      handler_id );
```

```
void gtk_signal_handler_block_by_func( GtkObject     *object,
                                       GtkSignalFunc  func,
                                       gpointer       data );

void gtk_signal_handler_block_by_data( GtkObject *object,
                                       gpointer   data );

void gtk_signal_handler_unblock( GtkObject *object,
                                 guint      handler_id );

void gtk_signal_handler_unblock_by_func( GtkObject     *object,
                                         GtkSignalFunc  func,
                                         gpointer       data );

void gtk_signal_handler_unblock_by_data( GtkObject *object,
                                         gpointer   data);
```

Un Hola Mundo Mejorado

Echemos un vistazo a un *helloworld* ligeramente mejorado con mejores ejemplos de devoluciones de llamada. Esto también nos introducirá a nuestro siguiente tema, embalaje widgets.

```
/* example-start helloworld2 helloworld2.c */

#include <gtk/gtk.h>

/* Our new improved callback.  The data passed to this function
 * is printed to stdout. */
void callback( GtkWidget *widget,
               gpointer   data )
{
    g_print ("Hello again - %s was pressed\n", (char *) data);
}

/* another callback */
gint delete_event( GtkWidget *widget,
                   GdkEvent  *event,
                   gpointer   data )
{
    gtk_main_quit();
    return(FALSE);
}

int main( int   argc,
          char *argv[] )
{
    /* GtkWidget is the storage type for widgets */
    GtkWidget *window;
    GtkWidget *button;
    GtkWidget *box1;

    /* This is called in all GTK applications. Arguments are parsed
     * from the command line and are returned to the application. */
```

```
    gtk_init (&argc, &argv);

    /* Create a new window */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* This is a new call, which just sets the title of our
     * new window to "Hello Buttons!" */
    gtk_window_set_title (GTK_WINDOW (window), "Hello Buttons!");

    /* Here we just set a handler for delete_event that immediately
     * exits GTK. */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (delete_event), NULL);

    /* Sets the border width of the window. */
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    /* We create a box to pack widgets into.  This is described in detail
     * in the "packing" section. The box is not really visible, it
     * is just used as a tool to arrange widgets. */
    box1 = gtk_hbox_new(FALSE, 0);

    /* Put the box into the main window. */
    gtk_container_add (GTK_CONTAINER (window), box1);

    /* Creates a new button with the label "Button 1". */
    button = gtk_button_new_with_label ("Button 1");

    /* Now when the button is clicked, we call the "callback" function
     * with a pointer to "button 1" as its argument */
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (callback), (gpointer) "button 1");

    /* Instead of gtk_container_add, we pack this button into the invisible
     * box, which has been packed into the window. */
    gtk_box_pack_start(GTK_BOX(box1), button, TRUE, TRUE, 0);

    /* Always remember this step, this tells GTK that our preparation for
     * this button is complete, and it can now be displayed. */
    gtk_widget_show(button);

    /* Do these same steps again to create a second button */
    button = gtk_button_new_with_label ("Button 2");

    /* Call the same callback function with a different argument,
     * passing a pointer to "button 2" instead. */
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (callback), (gpointer) "button 2");

    gtk_box_pack_start(GTK_BOX(box1), button, TRUE, TRUE, 0);

    /* The order in which we show the buttons is not really important, but I
     * recommend showing the window last, so it all pops up at once. */
    gtk_widget_show(button);

    gtk_widget_show(box1);

    gtk_widget_show (window);
```

```
    /* Rest in gtk_main and wait for the fun to begin! */
    gtk_main ();

    return(0);
}
/* example-end */
```

Compile el programa con los mismos argumentos que vinculan como nuestro primer ejemplo. Se dará cuenta de este tiempo no hay manera fácil de salir del programa, usted tiene que utilizar el gestor de ventanas o la línea de comandos para acabar con él. Un buen ejercicio para el lector sería insertar un tercer "Quit" botón que salir del programa. También podría jugar con las opciones para gtk_box_pack_start () al leer la siguiente sección. Trate de cambiar el tamaño de la ventana y observar el comportamiento. Así como una nota al margen, hay otra utilidad para definir gtk_window_new () - `GTK_WINDOW_DIALOG` . Esta interactúa con el gestor de ventanas un poco diferente y se debe utilizar para las ventanas transitorias.

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ **gcc helloworld2.c -o helloworld2 $(pkg-config gtk+-2.0 --cflags --libs)**

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ **./helloworld2**

Aplicaciones de embalaje

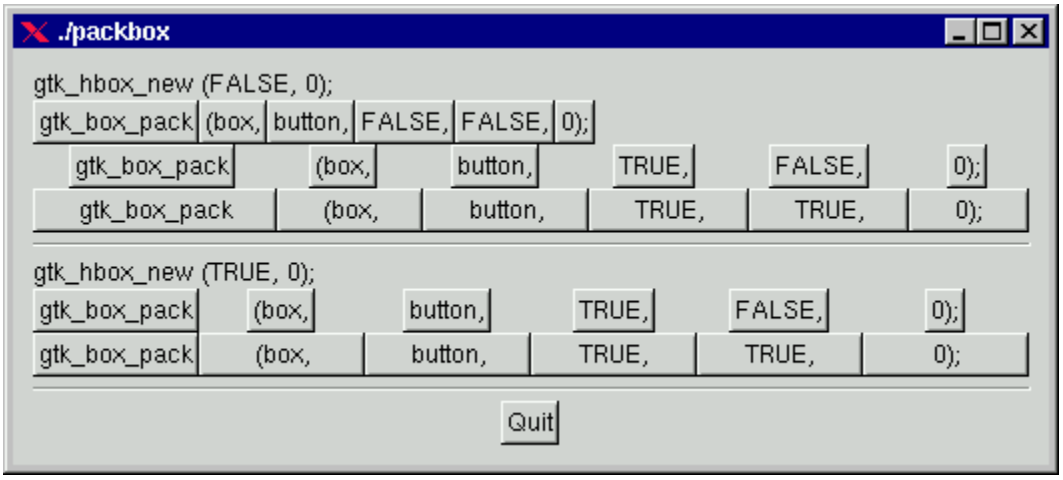
Al crear una aplicación, tendrá que poner más de un control dentro de una ventana. Nuestro ejemplo *HelloWorld* primero sólo se utiliza un widget por lo que simplemente podría utilizar una llamada gtk_container_add a "empaquetar" el control dentro de la ventana. Pero cuando se quiere poner más de un widget en una ventana, ¿cómo controlar dónde se sitúa ese widget? Aquí es donde entra en juego embalaje

Theory of Packing BoxesTeoría de Cajas de embalaje

La mayoría de embalaje se realiza mediante la creación de cuadros como en el ejemplo anterior. Estos son los contenedores invisibles de widgets que podemos embalar nuestros widgets en la que se presenta en dos formas, una caja horizontal, y una caja vertical. Al embalar los widgets en una caja horizontal, los objetos se insertan horizontalmente de izquierda a derecha o de derecha a izquierda, dependiendo de la llamada utilizado. En una caja vertical, los widgets se empaquetan de arriba hacia abajo o viceversa. Puede utilizar cualquier combinación de cajas dentro de cajas o al lado de otros para crear el efecto deseado. Para crear una nueva caja horizontal, se utiliza una llamada a gtk_hbox_new (), y para las cajas verticales, gtk_vbox_new (). El gtk_box_pack_start () y funciones gtk_box_pack_end () se utiliza para situar obje-

tos en el interior de estos recipientes. El gtk_box_pack_start () la función se iniciará en la parte superior y trabaje su manera abajo en una vbox, y el paquete de izquierda a derecha en una hbox. gtk_box_pack_end () hará lo contrario, el embalaje de abajo hacia arriba en una vbox, y de derecha a izquierda en una hbox. El uso de estas funciones nos permite justificar a la derecha o izquierda justificar nuestros widgets y se pueden mezclar de cualquier manera para lograr el efecto deseado. Usaremos gtk_box_pack_start () en la mayoría de nuestros ejemplos. Un objeto puede ser otro contenedor o un widget. De hecho, muchos widgets son contenedores propios, incluyendo el botón, pero por lo general sólo se utiliza una etiqueta en el interior de un botón. Mediante el uso de estas llamadas, GTK sabe donde desea colocar los widgets por lo que puede hacer el cambio de tamaño automático y otras cosas ingeniosas. También hay un número de opciones en cuanto a cómo los widgets deben ser embalados. Como se puede imaginar, este método nos da un un poco de flexibilidad en la colocación y la creación de widgets.

Las Cajas en detalle Gracias a esta flexibilidad, cajas de embalaje en GTK puede ser confuso al principio. Hay un montón de opciones, y no es inmediatamente obvio cómo encaja todo. En el extremo, sin embargo, existen básicamente cinco estilos diferentes.



Cada línea contiene una caja horizontal (hbox) con varios botones. La llamada a gtk_box_pack es una abreviación de la llamada a empacar cada uno de los botones en la hbox. Cada uno de los botones se embala en la hbox del mismo modo (es decir, los mismos argumentos que en el gtk_box_pack_start () función). Esta es la declaración de la función gtk_box_pack_start.

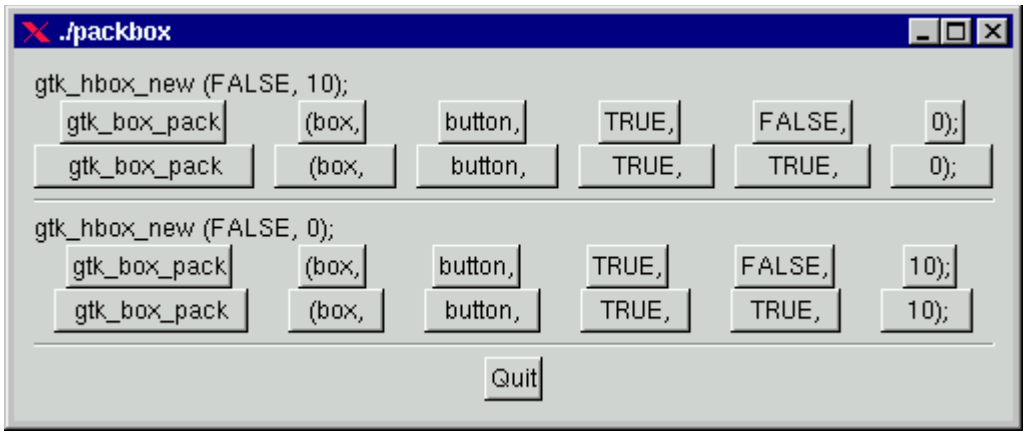
```
void gtk_box_pack_start( GtkBox    *box,
                         GtkWidget *child,
                         gint       expand,
                         gint       fill,
                         gint       padding );
```

El primer argumento es la caja que está envasado en el objeto, el segundo es el objeto. Los objetos serán todos los botones, por ahora, así que vamos a estar empaquetando botones dentro de las cajas. El argumento para ampliar gtk_box_pack_start () y gtk_box_pack_end () controla

si los controles se presentan en el cuadro para llenar todo el espacio extra en la caja para la caja se expande para llenar el área asignada a ella (TRUE), o la caja se encoge para adaptarse a los widgets (FALSO). Configuración de ampliar a FALSE le permitirá hacer justificación derecha e izquierda de los widgets. De lo contrario, todos se expanda para encajar en la caja, y el mismo efecto puede lograrse mediante el uso de sólo uno de gtk_box_pack_start o gtk_box_pack_end. El argumento de relleno para el control de funciones gtk_box_pack si el espacio extra se asigna a los propios objetos (TRUE), o como acolchado adicional en la caja alrededor de estos objetos (FALSO). Sólo tiene efecto si el argumento expand también es TRUE. Al crear una nueva caja, la función es la siguiente:

```
GtkWidget *gtk_hbox_new (gint homogeneous,
                         gint spacing);
```

El argumento homogénea a gtk_hbox_new (y lo mismo para gtk_vbox_new) controla si cada objeto en la caja tiene el mismo tamaño (es decir, el mismo ancho en una hbox, o la misma altura en una vbox). Si se establece, las rutinas gtk_box_pack funcionan esencialmente como si la `expand` argumento se convirtió en siempre. ¿Cuál es la diferencia entre el espaciamiento (establecido cuando la caja se crea) y el relleno (juego cuando los elementos se embalan)? El espaciado se añade entre objetos, y el relleno se añade a cada lado de un objeto. La siguiente figura debería hacerlo más claro:



Aquí está el código utilizado para crear las imágenes de arriba. Lo he comentado bastante fuertes así que espero que no tendrá ningún problema después de eso. Compilar usted mismo y jugar con él.

Programa demostración de empaquetamiento

```
/* example-start packbox packbox.c */

#include <stdio.h>
#include <stdlib.h>
#include "gtk/gtk.h"

gint delete_event( GtkWidget *widget,
                   GdkEvent  *event,
                   gpointer   data )
{
```

```
    gtk_main_quit();
    return(FALSE);
}

/* Make a new hbox filled with button-labels. Arguments for the
 * variables we're interested are passed in to this function.
 * We do not show the box, but do show everything inside. */
GtkWidget *make_box( gint homogeneous,
                     gint spacing,
                     gint expand,
                     gint fill,
                     gint padding )
{
    GtkWidget *box;
    GtkWidget *button;
    char padstr[80];

    /* Create a new hbox with the appropriate homogeneous
     * and spacing settings */
    box = gtk_hbox_new (homogeneous, spacing);

    /* Create a series of buttons with the appropriate settings */
    button = gtk_button_new_with_label ("gtk_box_pack");
    gtk_box_pack_start (GTK_BOX (box), button, expand, fill, padding);
    gtk_widget_show (button);

    button = gtk_button_new_with_label ("(box,");
    gtk_box_pack_start (GTK_BOX (box), button, expand, fill, padding);
    gtk_widget_show (button);

    button = gtk_button_new_with_label ("button,");
    gtk_box_pack_start (GTK_BOX (box), button, expand, fill, padding);
    gtk_widget_show (button);

    /* Create a button with the label depending on the value of
     * expand. */
    if (expand == TRUE)
            button = gtk_button_new_with_label ("TRUE,");
    else
            button = gtk_button_new_with_label ("FALSE,");

    gtk_box_pack_start (GTK_BOX (box), button, expand, fill, padding);
    gtk_widget_show (button);

    /* This is the same as the button creation for "expand"
     * above, but uses the shorthand form. */
    button = gtk_button_new_with_label (fill ? "TRUE," : "FALSE,");
    gtk_box_pack_start (GTK_BOX (box), button, expand, fill, padding);
    gtk_widget_show (button);

    sprintf (padstr, "%d);", padding);

    button = gtk_button_new_with_label (padstr);
    gtk_box_pack_start (GTK_BOX (box), button, expand, fill, padding);
    gtk_widget_show (button);

    return box;
}
```

```
int main( int   argc,
          char *argv[])
{
    GtkWidget *window;
    GtkWidget *button;
    GtkWidget *box1;
    GtkWidget *box2;
    GtkWidget *separator;
    GtkWidget *label;
    GtkWidget *quitbox;
    int which;

    /* Our init, don't forget this! :) */
    gtk_init (&argc, &argv);

    if (argc != 2) {
        fprintf (stderr, "usage: packbox num, where num is 1, 2, or 3.\n");
        /* This just does cleanup in GTK and exits with an exit status of 1.
*/
        gtk_exit (1);
    }

    which = atoi (argv[1]);

    /* Create our window */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* You should always remember to connect the delete_event signal
     * to the main window. This is very important for proper intuitive
     * behavior */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (delete_event), NULL);
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    /* We create a vertical box (vbox) to pack the horizontal boxes into.
     * This allows us to stack the horizontal boxes filled with buttons one
     * on top of the other in this vbox. */
    box1 = gtk_vbox_new (FALSE, 0);

    /* which example to show. These correspond to the pictures above. */
    switch (which) {
    case 1:
        /* create a new label. */
        label = gtk_label_new ("gtk_hbox_new (FALSE, 0);");

        /* Align the label to the left side.  We'll discuss this function and
         * others in the section on Widget Attributes. */
        gtk_misc_set_alignment (GTK_MISC (label), 0, 0);

        /* Pack the label into the vertical box (vbox box1).  Remember that
         * widgets added to a vbox will be packed one on top of the other in
         * order. */
        gtk_box_pack_start (GTK_BOX (box1), label, FALSE, FALSE, 0);

        /* Show the label */
        gtk_widget_show (label);
```

```
        /* Call our make box function - homogeneous = FALSE, spacing = 0,
         * expand = FALSE, fill = FALSE, padding = 0 */
        box2 = make_box (FALSE, 0, FALSE, FALSE, 0);
        gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, FALSE, 0);
        gtk_widget_show (box2);

        /* Call our make box function - homogeneous = FALSE, spacing = 0,
         * expand = TRUE, fill = FALSE, padding = 0 */
        box2 = make_box (FALSE, 0, TRUE, FALSE, 0);
        gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, FALSE, 0);
        gtk_widget_show (box2);

        /* Args are: homogeneous, spacing, expand, fill, padding */
        box2 = make_box (FALSE, 0, TRUE, TRUE, 0);
        gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, FALSE, 0);
        gtk_widget_show (box2);

        /* Creates a separator, we'll learn more about these later,
         * but they are quite simple. */
        separator = gtk_hseparator_new ();

        /* Pack the separator into the vbox. Remember each of these
         * widgets is being packed into a vbox, so they'll be stacked
         * vertically. */
        gtk_box_pack_start (GTK_BOX (box1), separator, FALSE, TRUE, 5);
        gtk_widget_show (separator);

        /* Create another new label, and show it. */
        label = gtk_label_new ("gtk_hbox_new (TRUE, 0);");
        gtk_misc_set_alignment (GTK_MISC (label), 0, 0);
        gtk_box_pack_start (GTK_BOX (box1), label, FALSE, FALSE, 0);
        gtk_widget_show (label);

        /* Args are: homogeneous, spacing, expand, fill, padding */
        box2 = make_box (TRUE, 0, TRUE, FALSE, 0);
        gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, FALSE, 0);
        gtk_widget_show (box2);

        /* Args are: homogeneous, spacing, expand, fill, padding */
        box2 = make_box (TRUE, 0, TRUE, TRUE, 0);
        gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, FALSE, 0);
        gtk_widget_show (box2);

        /* Another new separator. */
        separator = gtk_hseparator_new ();
        /* The last 3 arguments to gtk_box_pack_start are:
         * expand, fill, padding. */
        gtk_box_pack_start (GTK_BOX (box1), separator, FALSE, TRUE, 5);
        gtk_widget_show (separator);

        break;

    case 2:

        /* Create a new label, remember box1 is a vbox as created
         * near the beginning of main() */
        label = gtk_label_new ("gtk_hbox_new (FALSE, 10);");
        gtk_misc_set_alignment (GTK_MISC (label), 0, 0);
```

```
        gtk_box_pack_start (GTK_BOX (box1), label, FALSE, FALSE, 0);
        gtk_widget_show (label);

        /* Args are: homogeneous, spacing, expand, fill, padding */
        box2 = make_box (FALSE, 10, TRUE, FALSE, 0);
        gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, FALSE, 0);
        gtk_widget_show (box2);

        /* Args are: homogeneous, spacing, expand, fill, padding */
        box2 = make_box (FALSE, 10, TRUE, TRUE, 0);
        gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, FALSE, 0);
        gtk_widget_show (box2);

        separator = gtk_hseparator_new ();
        /* The last 3 arguments to gtk_box_pack_start are:
         * expand, fill, padding. */
        gtk_box_pack_start (GTK_BOX (box1), separator, FALSE, TRUE, 5);
        gtk_widget_show (separator);

        label = gtk_label_new ("gtk_hbox_new (FALSE, 0);");
        gtk_misc_set_alignment (GTK_MISC (label), 0, 0);
        gtk_box_pack_start (GTK_BOX (box1), label, FALSE, FALSE, 0);
        gtk_widget_show (label);

        /* Args are: homogeneous, spacing, expand, fill, padding */
        box2 = make_box (FALSE, 0, TRUE, FALSE, 10);
        gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, FALSE, 0);
        gtk_widget_show (box2);

        /* Args are: homogeneous, spacing, expand, fill, padding */
        box2 = make_box (FALSE, 0, TRUE, TRUE, 10);
        gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, FALSE, 0);
        gtk_widget_show (box2);

        separator = gtk_hseparator_new ();
        /* The last 3 arguments to gtk_box_pack_start are: expand, fill, pad-
ding. */
        gtk_box_pack_start (GTK_BOX (box1), separator, FALSE, TRUE, 5);
        gtk_widget_show (separator);
        break;

    case 3:

        /* This demonstrates the ability to use gtk_box_pack_end() to
         * right justify widgets. First, we create a new box as before. */
        box2 = make_box (FALSE, 0, FALSE, FALSE, 0);

        /* Create the label that will be put at the end. */
        label = gtk_label_new ("end");
        /* Pack it using gtk_box_pack_end(), so it is put on the right
         * side of the hbox created in the make_box() call. */
        gtk_box_pack_end (GTK_BOX (box2), label, FALSE, FALSE, 0);
        /* Show the label. */
        gtk_widget_show (label);

        /* Pack box2 into box1 (the vbox remember ? :) */
        gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, FALSE, 0);
        gtk_widget_show (box2);
```

```
        /* A separator for the bottom. */
        separator = gtk_hseparator_new ();
        /* This explicitly sets the separator to 400 pixels wide by 5 pixels
         * high. This is so the hbox we created will also be 400 pixels wide,
         * and the "end" label will be separated from the other labels in the
         * hbox. Otherwise, all the widgets in the hbox would be packed as
         * close together as possible. */
        gtk_widget_set_usize (separator, 400, 5);
        /* pack the separator into the vbox (box1) created near the start
         * of main() */
        gtk_box_pack_start (GTK_BOX (box1), separator, FALSE, TRUE, 5);
        gtk_widget_show (separator);
    }

    /* Create another new hbox.. remember we can use as many as we need! */
    quitbox = gtk_hbox_new (FALSE, 0);

    /* Our quit button. */
    button = gtk_button_new_with_label ("Quit");

    /* Setup the signal to terminate the program when the button is clicked
*/
    gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                               GTK_SIGNAL_FUNC (gtk_main_quit),
                               GTK_OBJECT (window));
    /* Pack the button into the quitbox.
     * The last 3 arguments to gtk_box_pack_start are:
     * expand, fill, padding. */
    gtk_box_pack_start (GTK_BOX (quitbox), button, TRUE, FALSE, 0);
    /* pack the quitbox into the vbox (box1) */
    gtk_box_pack_start (GTK_BOX (box1), quitbox, FALSE, FALSE, 0);

    /* Pack the vbox (box1) which now contains all our widgets, into the
     * main window. */
    gtk_container_add (GTK_CONTAINER (window), box1);

    /* And show everything left */
    gtk_widget_show (button);
    gtk_widget_show (quitbox);

    gtk_widget_show (box1);
    /* Showing the window last so everything pops up at once. */
    gtk_widget_show (window);

    /* And of course, our main function. */
    gtk_main ();

    /* Control returns here when gtk_main_quit() is called, but not when
     * gtk_exit is used. */

    return(0);
}
/* example-end */


linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc packbox.c -o packbox
$(pkg-config gtk+-2.0 --cflags –libs)
```

```
linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./packbox 3
```

Uso de tablas embalaje

Vamos a echar un vistazo a otra forma de embalaje - Tables. Estos pueden ser muy útiles en ciertas situaciones. Uso de tablas, creamos una cuadrícula que podemos colocar widgets pulg Los widgets pueden tomar tantas casillas como indique. La primera cosa que ver, por supuesto, es la función gtk_table_new:

```
GtkWidget *gtk_table_new( gint rows,
                          gint columns,
                          gint homogeneous );
```

El primer argumento es el número de filas que se hacen en la mesa, mientras que el segundo, obviamente, es el número de columnas. El argumento homogeneous tiene que ver con la forma en cajas de la tabla están dimensionados. Si homogeneous es TRUE, las casillas de la tabla se cambia de tamaño para el tamaño de la más grande de widget en la mesa. Si homogeneous es FALSE, el tamaño de una caja de mesa es dictada por el más alto widget en su misma fila, y la más amplia widget en su columna. Las filas y columnas se disponen de 0 a n, donde n es el número especificado en la llamada a gtk_table_new. Por lo tanto, si especifica las filas y columnas = 2 = 2, la disposición sería algo como esto:

```
    0            1            2  0 + ---------- + ---------- +
    |            |            |  1 + ---------- + ---------- +
    |            |            |  2 + ---------- + ---------- +
```

Tenga en cuenta que el sistema de coordenadas se inicia en la esquina superior izquierda. Para colocar un control en un cuadro, utilice la siguiente función:

```
void gtk_table_attach( GtkTable  *table,
                       GtkWidget *child,
                       gint       left_attach,
                       gint       right attach,
                       gint       top_attach,
                       gint       bottom_attach,
                       gint       xoptions,
                       gint       yoptions,
                       gint       xpadding,
                       gint       ypadding );
```

El primer argumento ("tabla") es la tabla que hemos creado y el segundo ("hijo"), el widget que desea colocar en la mesa. Los argumentos izquierdo y derecho adjuntar

especificar dónde colocar el widget, y cuántas cajas de usar. Si quieres un botón en la entrada de la tabla inferior derecha de nuestra tabla 2x2, y quiere que se llene esa entrada solamente, left_attach sería = 1, right_attach = 2, top_attach = 1, bottom_attach = 2. Ahora, si usted quiere un widget para ocupar toda la fila superior de nuestra tabla 2x2, tendrá que utilizar left_attach = 0, right_attach = 2, top_attach = 0, bottom_attach = 1. Los xoptions y yoptions se utilizan para especificar las opciones de embalaje y puede ser OR'ed bit a bit entre sí para permitir múltiples opciones.

Estas opciones son las siguientes: `GTK_FILL` - Si la caja de la tabla es mayor que el widget, y `GTK_FILL` se especifica, el widget se expandirá para usar todo el espacio disponible. `GTK_SHRINK` - Si el widget tabla se asignó menos espacio que se solicitó (por lo general por el cambio de tamaño de la ventana de usuario), entonces los widgets normalmente sólo se separó de la parte inferior de la ventana y desaparecer. Si `GTK_SHRINK` se especifica, los widgets se reducirán con la mesa. `GTK_EXPAND` - Esto hará que la tabla se expanda para usar todo el espacio restante en la ventana. El relleno es igual que en las cajas, la creación de una zona libre alrededor del widget especificado en píxeles. gtk_table_attach () tiene un montón de opciones. Por lo tanto, hay un atajo:

```
void gtk_table_attach_defaults( GtkTable  *table,
                                GtkWidget *widget,
                                gint       left_attach,
                                gint       right_attach,
                                gint       top_attach,
                                gint       bottom_attach ); El X e Y para las
```

opciones por defecto `GTK_FILL | GTK_EXPAND` , y X e Y son relleno a 0. El resto de los argumentos son idénticos a la función anterior. También tenemos gtk_table_set_row_spacing () y gtk_table_set_col_spacing (). Estos lugares espacio entre las filas de la fila o columna especificada.

```
void gtk_table_set_row_spacing( GtkTable *table,
                                gint      row,
                                gint      spacing );

y

void gtk_table_set_col_spacing ( GtkTable *table,
                                 gint      column,
                                 gint      spacing );
```

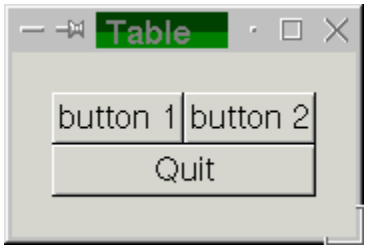
Tenga en cuenta que para las columnas, el espacio va a la derecha de la columna, y para las filas, el espacio está por debajo de la fila. También puede establecer una separación uniforme de todas las filas y / o columnas con:

```
void gtk_table_set_row_spacings( GtkTable *table,
                            gint spacing );  Y,
void gtk_table_set_col_spacings( GtkTable *table,
                                 gint      spacing );
```

Tenga en cuenta que con estas llamadas, la última fila y la última columna no reciben ninguna separación.

Ejemplo de Empaquetado con Tablas

Aquí hacemos una ventana con tres botones en una tabla de 2x2. Los dos primeros botones se coloca en la fila superior. Un tercer botón, dejar de fumar, se coloca en la fila inferior, que abarca ambas columnas. Lo que significa que debería ser algo como esto:



Aquí está el código fuente:

```
/* example-start table table.c */

#include <gtk/gtk.h>

/* Our callback.
 * The data passed to this function is printed to stdout */
void callback( GtkWidget *widget,
               gpointer   data )
{
    g_print ("Hello again - %s was pressed\n", (char *) data);
}

/* This callback quits the program */
gint delete_event( GtkWidget *widget,
                   GdkEvent  *event,
                   gpointer   data )
{
    gtk_main_quit ();
    return(FALSE);
}

int main( int   argc,
          char *argv[] )
{
    GtkWidget *window;
    GtkWidget *button;
    GtkWidget *table;

    gtk_init (&argc, &argv);

    /* Create a new window */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* Set the window title */
    gtk_window_set_title (GTK_WINDOW (window), "Table");
```

```
    /* Set a handler for delete_event that immediately
     * exits GTK. */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (delete_event), NULL);

    /* Sets the border width of the window. */
    gtk_container_set_border_width (GTK_CONTAINER (window), 20);

    /* Create a 2x2 table */
    table = gtk_table_new (2, 2, TRUE);

    /* Put the table in the main window */
    gtk_container_add (GTK_CONTAINER (window), table);

    /* Create first button */
    button = gtk_button_new_with_label ("button 1");

    /* When the button is clicked, we call the "callback" function
     * with a pointer to "button 1" as its argument */
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
              GTK_SIGNAL_FUNC (callback), (gpointer) "button 1");


    /* Insert button 1 into the upper left quadrant of the table */
    gtk_table_attach_defaults (GTK_TABLE(table), button, 0, 1, 0, 1);

    gtk_widget_show (button);

    /* Create second button */

    button = gtk_button_new_with_label ("button 2");

    /* When the button is clicked, we call the "callback" function
     * with a pointer to "button 2" as its argument */
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
              GTK_SIGNAL_FUNC (callback), (gpointer) "button 2");
    /* Insert button 2 into the upper right quadrant of the table */
    gtk_table_attach_defaults (GTK_TABLE(table), button, 1, 2, 0, 1);

    gtk_widget_show (button);

    /* Create "Quit" button */
    button = gtk_button_new_with_label ("Quit");

    /* When the button is clicked, we call the "delete_event" function
     * and the program exits */
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (delete_event), NULL);

    /* Insert the quit button into the both
     * lower quadrants of the table */
    gtk_table_attach_defaults (GTK_TABLE(table), button, 0, 2, 1, 2);

    gtk_widget_show (button);

    gtk_widget_show (table);
    gtk_widget_show (window);
```

```
    gtk_main ();

    return 0;
}
/* example-end */


linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc table.c -o table $(pkg-
config gtk+-2.0 --cflags --libs)

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./table
```

Widget general

Los pasos generales para crear un widget en GTK son las siguientes: gtk_ * _new - una de varias funciones para crear un nuevo widget. Todos estos son detallados en esta sección. Conecte todas las señales y eventos que deseen utilizar a los controladores apropiados. Establezca los atributos del widget. Embale el widget en un contenedor utilizando la llamada apropiada, tal como gtk_container_add () o gtk_box_pack_start (). gtk_widget_show () del widget. gtk_widget_show () le permite a GTK saber que estamos terminado de configurar los atributos del widget, y está listo para ser mostrado. También puede utilizar gtk_widget_hide para hacerlo desaparecer. El orden en que se muestran los widgets no es importante, pero te sugiero que muestra la última ventana para que la ventana entera aparezca de una vez en lugar de ver los widgets individuales van a plantear en la pantalla a medida que se forman. Los hijos de un widget (una ventana es también un widget) no se mostrará hasta que la propia ventana se muestra con el gtk_widget_show () función.

Casting Fundición

Se dará cuenta de la marcha en GTK que utiliza un sistema de conversión de tipos. Esto siempre se hace uso de macros que tanto la prueba de la habilidad de lanzar el tema dado, y llevar a cabo el reparto. Entre los más comunes se pueden ver son:

```
  GTK_WIDGET(widget)
  GTK_OBJECT(object)
  GTK_SIGNAL_FUNC(function)
  GTK_CONTAINER(container)
  GTK_WINDOW(window)
  GTK_BOX(box)
```

Estos son usados para emitir argumentos en las funciones. Las verás en los ejemplos, y por lo general se puede decir cuándo se deben utilizar sólo con ver la declaración de la función. Como se puede ver más abajo en la jerarquía de clases, todos

los GtkWidgets se derivan de la clase base Object. Esto significa que puede utilizar un widget en cualquier lugar de la función pide un objeto - sólo tiene que utilizar el `GTK_OBJECT()` macro. Por ejemplo:

```
gtk_signal_connect( GTK_OBJECT(button), "clicked",
                    GTK_SIGNAL_FUNC(callback_function), callback_data);
```

Esto pone el botón en un objeto, y ofrece un reparto para el puntero de función a la devolución de llamada. Muchos widgets son también contenedores. Si nos fijamos en la jerarquía de clases más adelante, te darás cuenta de que muchos widgets derivan de la clase Container. Cualquiera de estos controles puede ser utilizado con la `GTK_CONTAINER` macro para pasar a las funciones que piden contenedores. Por desgracia, estas macros no son ampliamente cubierto en el tutorial, pero yo recomiendo echar un vistazo a través de los archivos de cabecera GTK. Puede ser muy educativo. De hecho, no es difícil de aprender cómo funciona un aparato con sólo mirar a las declaraciones de función.

Jerarquía Widget

Para su referencia, aquí está el árbol de jerarquía de clases utilizada para implementar widgets.

```
  GtkObject
   + GtkWidget
   | + GtkMisc
   | | + GtkLabel
   | | | + GtkAccelLabel
   | | | `GtkTipsQuery
   | | + GtkArrow
   | | + GtkImage
   | | `GtkPixmap
   | + GtkContainer
   | | + GtkBin
   | | | + GtkAlignment
   | | | + GtkFrame
   | | | | `GtkAspectFrame
   | | | + GtkButton
   | | | | + GtkToggleButton
   | | | | | `GtkCheckButton
   | | | | | `GtkRadioButton
   | | | | `GtkOptionMenu
   | | | + GtkItem
   | | | | + GtkMenuItem
   | | | | | + GtkCheckMenuItem
   | | | | | | `GtkRadioMenuItem
   | | | | | `GtkTearoffMenuItem
   | | | | + GtkListItem
   | | | | `GtkTreeItem
   | | | + GtkWindow
```

```
    | | | | + GtkColorSelectionDialog
    | | | | + GtkDialog
    | | | | | `GtkInputDialog
    | | | | + GtkDrawWindow
    | | | | + GtkFileSelection
    | | | | + GtkFontSelectionDialog
    | | | | `GtkPlug
    | | | + GtkEventBox
    | | | + GtkHandleBox
    | | | + GtkScrolledWindow
    | | | `GtkViewport
    | | + GtkBox
    | | | + GtkButtonBox
    | | | | + GtkHButtonBox
    | | | | `GtkVButtonBox
    | | | + GtkVBox
    | | | | + GtkColorSelection
    | | | | `GtkGammaCurve
    | | | `GtkHBox
    | | | + GtkCombo
    | | | `GtkStatusbar
    | | + GtkCList
    | | | `GtkCTree
    | | + GtkFixed
    | | + GtkNotebook
    | | | `GtkFontSelection
    | | + GtkPaned
    | | | + GtkHPaned
    | | | `GtkVPaned
    | | + GtkLayout
    | | + GtkList
    | | + GtkMenuShell
    | | | + GtkMenuBar
    | | | `GtkMenu
    | | + GtkPacker
    | | + GtkSocket
    | | + GtkTable
    | | + GtkToolbar
    | | `GtkTree
    | + GtkCalendar
    | + GtkDrawingArea
    | | `GtkCurve
    | + GtkEditable
    | | + GtkEntry
    | | | `GtkSpinButton
    | | `GtkText
    | + GtkRuler
    | | + GtkHRuler
    | | `GtkVRuler
    | + GtkRange
    | | + GtkScale
    | | | + GtkHScale
    | | | `GtkVScale
    | | `GtkScrollbar
    | | + GtkHScrollbar
    | | `GtkVScrollbar
    | + GtkSeparator
    | | + GtkHSeparator
```

```
   | | `GtkVSeparator
   | + GtkPreview
   | `GtkProgress
   | `GtkProgressBar
   + GtkData
   | + GtkAdjustment
   | `GtkTooltips
   `GtkItemFactory
```

Aplicaciones sin ventanas

Los widgets siguientes no tienen asociada una ventana. Si desea capturar los eventos, usted tendrá que usar el EventBox.

EventBox

Vea la sección sobre la EventBox widget. `GtkAlignment GtkArrow GtkBin GtkBox GtkImage GtkItem GtkLabel GtkPixmap GtkScrolledWindow GtkSeparator GtkTable GtkAspectFrame GtkFrame GtkVBox GtkHBox GtkVSeparator GtkHSeparator` Vamos a continuar nuestra exploración de GTK mediante el examen de cada widget, a su vez, la creación de algunas funciones simples para mostrarlas. Otra buena fuente es el programa testgtk.c que viene con GTK. Se puede encontrar en gtk / testgtk.c.

The Button Widget Normal Buttons Botones normales

Hemos visto casi todo lo que hay para ver el widget de botón. Es bastante simple. Sin embargo, hay dos formas de crear un botón. Usted puede utilizar el gtk_button_new_with_label () para crear un botón con una etiqueta o utilice gtk_button_new () para crear un botón en blanco. Es entonces depende de usted para empacar una etiqueta o un pixmap en este nuevo botón. Para ello, cree una nueva caja, y luego empacar sus objetos en este cuadro con el gtk_box_pack_start de costumbre, y luego usar gtk_container_add para empacar la caja en el botón. He aquí un ejemplo del uso de gtk_button_new para crear un botón con una imagen y una etiqueta en ella. He roto el código para crear un cuadro del resto para que pueda utilizar en sus programas. Hay otros ejemplos de la utilización de mapas de píxeles más adelante en el tutorial.

```
/* example-start buttons buttons.c */

#include <gtk/gtk.h>

/* Create a new hbox with an image and a label packed into it
 * and return the box. */

GtkWidget *xpm_label_box( GtkWidget *parent,
                          gchar     *xpm_filename,
                          gchar     *label_text )
{
```

```
    GtkWidget *box1;
    GtkWidget *label;
    GtkWidget *pixmapwid;
    GdkPixmap *pixmap;
    GdkBitmap *mask;
    GtkStyle *style;

    /* Create box for xpm and label */
    box1 = gtk_hbox_new (FALSE, 0);
    gtk_container_set_border_width (GTK_CONTAINER (box1), 2);

    /* Get the style of the button to get the
     * background color. */
    style = gtk_widget_get_style(parent);

    /* Now on to the xpm stuff */
    pixmap = gdk_pixmap_create_from_xpm (parent->window, &mask,
                                         &style->bg[GTK_STATE_NORMAL],
                                         xpm_filename);
    pixmapwid = gtk_pixmap_new (pixmap, mask);

    /* Create a label for the button */
    label = gtk_label_new (label_text);

    /* Pack the pixmap and label into the box */
    gtk_box_pack_start (GTK_BOX (box1),
                        pixmapwid, FALSE, FALSE, 3);

    gtk_box_pack_start (GTK_BOX (box1), label, FALSE, FALSE, 3);

    gtk_widget_show(pixmapwid);
    gtk_widget_show(label);

    return(box1);
}

/* Our usual callback function */
void callback( GtkWidget *widget,
               gpointer   data )
{
    g_print ("Hello again - %s was pressed\n", (char *) data);
}


int main( int   argc,
          char *argv[] )
{
    /* GtkWidget is the storage type for widgets */
    GtkWidget *window;
    GtkWidget *button;
    GtkWidget *box1;

    gtk_init (&argc, &argv);

    /* Create a new window */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    gtk_window_set_title (GTK_WINDOW (window), "Pixmap'd Buttons!");
```

```
    /* It's a good idea to do this for all windows. */
    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        GTK_SIGNAL_FUNC (gtk_exit), NULL);

    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (gtk_exit), NULL);

    /* Sets the border width of the window. */
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_realize(window);

    /* Create a new button */
    button = gtk_button_new ();

    /* Connect the "clicked" signal of the button to our callback */
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (callback), (gpointer) "cool but-
ton");

    /* This calls our box creating function */
    box1 = xpm_label_box(window, "info.xpm", "cool button");

    /* Pack and show all our widgets */
    gtk_widget_show(box1);

    gtk_container_add (GTK_CONTAINER (button), box1);

    gtk_widget_show(button);

    gtk_container_add (GTK_CONTAINER (window), button);

    gtk_widget_show (window);

    /* Rest in gtk_main and wait for the fun to begin! */
    gtk_main ();

    return(0);
}
/* example-end */

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc buttons.c -o buttons
$(pkg-config gtk+-2.0 --cflags --libs)

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./buttons
```

La función xpm_label_box podría ser usada para empaquetar xpm y etiquetas en cualquier widget que puede ser un contenedor. Observe en `xpm_label_box` cómo se hace un llamado a `gtk_widget_get_style` . Cada widget tiene un "estilo", que consta de los colores frontal y de fondo para una variedad de situaciones, la selección de fuentes y datos de otros gráficos relacionados con un widget. Estos valores de estilo están en mora en cada widget, y son requeridos por muchas llamadas a funciones GDK, como `gdk_pixmap_create_from_xpm` , que aquí se le da el "normal" color de fondo.

GTK's rc files

Los datos del estilo de los widgets pueden personalizarse usando los archivos rc de GTK + . Observe también la llamada a `gtk_widget_realize` después de ajustar el ancho de la ventana de la frontera. Esta función utiliza GDK para crear las ventanas X relacionado con el widget. La función se activa automáticamente cuando se invoca `gtk_widget_show` para un control, por lo que no se ha demostrado en los ejemplos anteriores. Pero el llamado a `gdk_pixmap_create_from_xpm` requiere que su `window` argumento se refiere a una ventana X real, por lo que es necesario para realizar el control antes de esta llamada GDK. Los widget Button tiene las siguientes señales: `pressed` - se emite cuando se presiona el botón del puntero dentro Widget de botón `released` - se emite cuando el botón del puntero se libera dentro Widget de botón `clicked` - se emite cuando se presiona el botón del puntero y luego puesto en libertad dentro Widget de botón `enter` - se emite cuando el puntero entra Widget de botón `leave` - se emite cuando el puntero sale de control Botón

Botones con Estado

Botones de alternar se derivan de los botones normales y son muy similares, excepto que siempre estará en uno de dos estados, alternado con un clic. Pueden estar deprimido, y al hacer clic de nuevo, se abrirá de nuevo. Haga clic de nuevo, y se abrirá hacia abajo. Botones de alternar son la base para los botones de verificación y botones de radio, por lo tanto, muchas de las llamadas usadas para botones de selección son heredados por la radio y los botones de control. Voy a destacar este hecho cuando llegamos a ellos. Creación de un botón de activación nuevo:

```
GtkWidget *gtk_toggle_button_new( void );

GtkWidget *gtk_toggle_button_new_with_label( gchar *label );
```

Como se puede imaginar, estas funcionan igual que el widget de botón normal llama. La primera crea un botón de activación en blanco, y el segundo, un botón con un widget de etiqueta ya en ella. Para recuperar el estado del widget toggle, incluyendo radio y botones de activación, utilizamos una construcción como se muestra en el ejemplo a continuación. Se comprueba el estado de la palanca, mediante el acceso al `active` campo de la estructura del widget de palanca, después del primer uso del `GTK_TOGGLE_BUTTON` macro del puntero de control dentro de un puntero Toggle Widget. La señal que nos interesa emiten los botones biestado (el botón de alternar, botón de verificación y botón de exclusión mútua) es el " toggle" de la señal. Para comprobar el estado de estos botones, configurar un manejador de señales para captar la señal cambia, y acceder a la estructura para determinar su estado. La devolución de llamada se verá algo como:

```
void toggle_button_callback (GtkWidget *widget, gpointer data)
{
    if (GTK_TOGGLE_BUTTON (widget)->active)
    {
        /* If control reaches here, the toggle button is down */
```

```
    } else {

        /* If control reaches here, the toggle button is up */
    }
}
```

Para forzar el estado de un botón de alternar, y sus hijos, la radio y los botones de activación, utilice esta función:

```
void gtk_toggle_button_set_active( GtkToggleButton *toggle_button,
                                   gint             state );
```

La llamada anterior se puede utilizar para establecer el estado del botón de conmutación, y sus niños la radio y los botones de activación. Pasando en su botón creado como primer argumento, y un TRUE o FALSE para el argumento segundo estado para especificar si debe ser hacia abajo (depresión) o arriba (liberado). Por defecto está para arriba, o FALSE. Tenga en cuenta que cuando se utiliza el gtk_toggle_button_set_active () función, y se cambia el estado de hecho, hace que la señal "clicked" que se emitirá desde el botón.

`void gtk_toggle_button_toggled (GtkToggleButton *toggle_button);` Esto simplemente cambia el botón y emite la señal "toggled".

botones de activación

Compruebe botones heredan muchas propiedades y funciones de los botones de la palanca arriba, pero mira un poco diferente. En lugar de ser botones con texto dentro de ellos, son pequeños cuadrados con el texto a la derecha de ellos. Estos se utilizan a menudo para alternar las opciones de encendido y apagado en aplicaciones. Las dos funciones de creación son similares a las de los botones normales.

```
GtkWidget *gtk_check_button_new( void );

GtkWidget *gtk_check_button_new_with_label ( gchar *label );
```

La función new_with_label crea un botón con una etiqueta de verificación junto a él. Comprobación del estado del botón de comprobación es idéntica a la del botón de conmutación.

Botones de opción

Los botones de radio son similares a comprobar los botones excepto que se agrupan de manera que sólo uno puede ser seleccionado / deprimido a la vez. Esto es bueno para los lugares en su aplicación en la que usted necesita para seleccionar de una lista de opciones. Creación de un nuevo botón de radio se realiza con una de estas llamadas:

```
GtkWidget *gtk_radio_button_new( GSList *group );

GtkWidget *gtk_radio_button_new_with_label( GSList *group,
                                            gchar  *label );
```

Se dará cuenta el argumento extra para estas llamadas. Requieren de un grupo para llevar a cabo sus deberes adecuadamente. La primera llamada a gtk_radio_button_new_with_label o gtk_radio_button_new_with_label debe pasar NULL como primer argumento. A continuación, cree un grupo mediante: `GSList *gtk_radio_button_group( GtkRadioButton *radio_button );`

Lo importante a recordar es que gtk_radio_button_group debe llamarse para cada botón nuevo agregado al grupo, con el botón anterior pasa como un argumento. El resultado se pasa a la siguiente llamada a gtk_radio_button_new o gtk_radio_button_new_with_label. Esto permite que una cadena de botones que se establezcan. El siguiente ejemplo debería dejar esto claro. Puede acortar ligeramente esta utilizando la sintaxis siguiente, lo que elimina la necesidad de una variable para almacenar la lista de botones. Esta forma es utilizada en el ejemplo para crear el tercer botón:

```
button2 = gtk_radio_button_new_with_label(
                gtk_radio_button_group (GTK_RADIO_BUTTON (button1)),
                "button2");
```

También es una buena idea establecer explícitamente qué botón debe ser presionado el botón por defecto con:

```
void gtk_toggle_button_set_active( GtkToggleButton *toggle_button,
                                   gint             state );
```

Esto se describe en la sección de botones de conmutación, y funciona exactamente de la misma manera. Una vez que los botones de radio se agrupan, sólo uno de los grupos puede estar activo a la vez. Si el usuario hace clic en un botón de opción, y luego en otro, el primer botón de primero emitirá una señal "toggled" (para informar a estar inactivo), y luego el segundo emitirá su "alternar" de la señal (para reportar cada activo) . En el ejemplo siguiente, se crea un grupo de botones de radio con tres botones.

```
/* example-start radiobuttons radiobuttons.c */

#include <gtk/gtk.h>
#include <glib.h>

gint close_application( GtkWidget *widget,
                        GdkEvent  *event,
                        gpointer   data )
{
  gtk_main_quit();
  return(FALSE);
}

int main( int   argc,
          char *argv[] )
{
    GtkWidget *window = NULL;
    GtkWidget *box1;
    GtkWidget *box2;
    GtkWidget *button;
    GtkWidget *separator;
    GSList *group;

    gtk_init(&argc,&argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC(close_application),
                        NULL);

    gtk_window_set_title (GTK_WINDOW (window), "radio buttons");
    gtk_container_set_border_width (GTK_CONTAINER (window), 0);

    box1 = gtk_vbox_new (FALSE, 0);
    gtk_container_add (GTK_CONTAINER (window), box1);
    gtk_widget_show (box1);

    box2 = gtk_vbox_new (FALSE, 10);
    gtk_container_set_border_width (GTK_CONTAINER (box2), 10);
    gtk_box_pack_start (GTK_BOX (box1), box2, TRUE, TRUE, 0);
    gtk_widget_show (box2);

    button = gtk_radio_button_new_with_label (NULL, "button1");
    gtk_box_pack_start (GTK_BOX (box2), button, TRUE, TRUE, 0);
    gtk_widget_show (button);

    group = gtk_radio_button_group (GTK_RADIO_BUTTON (button));
    button = gtk_radio_button_new_with_label(group, "button2");
    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (button), TRUE);
    gtk_box_pack_start (GTK_BOX (box2), button, TRUE, TRUE, 0);
    gtk_widget_show (button);

    button = gtk_radio_button_new_with_label(
                 gtk_radio_button_group (GTK_RADIO_BUTTON (button)),
                 "button3");
    gtk_box_pack_start (GTK_BOX (box2), button, TRUE, TRUE, 0);
    gtk_widget_show (button);
```

```
    separator = gtk_hseparator_new ();
    gtk_box_pack_start (GTK_BOX (box1), separator, FALSE, TRUE, 0);
    gtk_widget_show (separator);

    box2 = gtk_vbox_new (FALSE, 10);
    gtk_container_set_border_width (GTK_CONTAINER (box2), 10);
    gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, TRUE, 0);
    gtk_widget_show (box2);

    button = gtk_button_new_with_label ("close");
    gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                               GTK_SIGNAL_FUNC(close_application),
                               GTK_OBJECT (window));
    gtk_box_pack_start (GTK_BOX (box2), button, TRUE, TRUE, 0);
    GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);
    gtk_widget_grab_default (button);
    gtk_widget_show (button);
    gtk_widget_show (window);

    gtk_main();

    return(0);
}
/* example-end */

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc radiobuttons.c -o
radiobuttons $(pkg-config gtk+-2.0 --cflags --libs)

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./radiobuttons
```

Ajustes Range Widgets

GTK tiene varios widgets que pueden ser ajustados visualmente por el usuario usando el ratón o el teclado, tales como los controles de rango, que se describen en el rango Reproductores sección. También hay unos widgets que pueden ser ajustados parcialmente un área más grande de datos, como el widget de texto y el control de vista. Obviamente, la aplicación debe ser capaz de reaccionar a los cambios que realice el usuario en los controles de rango. Una forma de hacerlo sería que cada widget emitir su propio tipo de señal cuando sus cambios de ajuste, y, o bien pasar el nuevo valor en el manejador de la señal, o requerir que se vea el interior de la estructura de datos del widget para determinar el valor. Pero usted también puede querer conectar los ajustes de varios widgets juntos, por lo que un ajuste ajusta los demás. El ejemplo más obvio de esto es conectar una barra de desplazamiento para un paneo de visualización o un área de desplazamiento de texto. Si cada widget tiene su propia manera de establecer u obtener el valor de ajuste, entonces el programador tenga que escribir sus propios manejadores de señales para traducir entre la salida de la señal de un widget, y la "entrada" de otro la función de ajuste de ajuste. GTK resuelve este problema mediante el objeto de ajuste, que no es un widget, sino una forma para los widgets para almacenar y traspasar información de una forma abstracta y flexible. El uso más obvio de ajuste consiste en almacenar los paráme-

tros de configuración y los valores de los controles de rango, tales como barras de desplazamiento y los controles de escala. Sin embargo, dado que los ajustes se derivan de Object, que tienen algunos poderes especiales más allá de las estructuras de datos normales. Lo más importante es que pueden emitir señales, al igual que los widgets, estas señales se pueden utilizar no sólo para permitir que su programa para reaccionar a la entrada del usuario en controles ajustables, pero también para propagar los valores de ajuste de forma transparente entre controles ajustables.

Creación de un Ajuste

Muchos de los widgets que utilizan objetos de ajuste de hacerlo automáticamente, pero algunos casos se muestra en los ejemplos posteriores donde usted puede necesitar para crear uno usted mismo. Para crear un ajuste usando:

```
GtkObject *gtk_adjustment_new( gfloat value,
                               gfloat lower,
                               gfloat upper,
                               gfloat step_increment,
                               gfloat page_increment,
                               gfloat page_size );
```

El `value` argumento es el valor inicial que desea dar al ajuste, por lo general corresponde a la posición más alta o más a la izquierda de un control ajustable. El `lower` argumento especifica el valor más bajo que el ajuste puede contener. El `step_increment` argumento especifica el "más pequeño" de los dos incrementos en la que el usuario puede cambiar el valor, mientras que el `page_increment` es la "más grande" uno. El `page_size` argumento suele corresponder de alguna manera a la zona visible de un control desplazable. El `upper` argumento se utiliza para representar la parte inferior derecha más o más coordenadas en el niño un control desplazable de. Por lo tanto, *no* siempre es el número más grande que `value` puede tomar, desde el `page_size` de widgets, es generalmente distinto de cero.

Utilización de Ajustes de la manera fácil

Los controles ajustables pueden dividirse en aquellos que usan y requieren unidades específicas para estos valores y aquellos que los tratan como números arbitrarios. El grupo que trata los valores como números arbitrarios incluye los controles de rango (barras de desplazamiento y escalas, el widget de la barra de progreso, y el control botones giratorios). Estos widgets son todos los aparatos que normalmente se "ajustan" directamente por el usuario mediante el ratón o el teclado. Se tratará los `lower` y `upper` valores de un ajuste como un rango dentro del cual el usuario puede manipular el ajuste de `value` . Por defecto, sólo se modificará el `value` de un ajuste. El otro grupo incluye el widget de texto, el control de vista, el widget compuesto lista, y el widget ventana desplazado. Todos estos widgets de utilizar valores de píxel para sus ajustes. Estos son también todos los widgets que normalmente se "ajustan" indirectamente usando barras de desplazamiento. Si bien todos los controles que usan

ajustes pueden crear sus propios ajustes o usar los que suministran, generalmente querrá dejar esta categoría particular de widgets crear sus propios ajustes. Por lo general, no tendrán en cuenta todos los valores excepto el `value` sí mismo en todos los ajustes que les dan, pero los resultados son, en general, indefinido (es decir, que tendrás que leer el código fuente para averiguar, y puede ser diferente de un widget a). Ahora, usted probablemente está pensando, ya que los widgets de texto y las vistas insisten en establecer todo menos el `value` de los ajustes, mientras que las barras de desplazamiento *sólo* tocará el ajuste de `value` , si se *comparte* un objeto ajuste entre una barra de desplazamiento y un control de texto, la manipulación de la barra de desplazamiento automágicamente ajustar el widget de texto? Por supuesto que sí! Sólo de esta manera:

```
  /* creates its own adjustments */
  text = gtk_text_new (NULL, NULL);
  /* uses the newly-created adjustment for the scrollbar as well */
  vscrollbar = gtk_vscrollbar_new (GTK_TEXT(text)->vadj);
```

Internos de ajuste

Ok, usted dice, eso es bueno, pero lo que si quiero crear mis propios controladores para responder cuando el usuario ajusta un control de rango o un botón de número, y cómo puedo obtener el valor del ajuste en estos manejadores? Para responder a estas preguntas y más, vamos a empezar por echar un vistazo a `struct _GtkAdjustment` misma:

```
struct _GtkAdjustment
{
  GtkData data;

  gfloat lower;
  gfloat upper;
  gfloat value;
  gfloat step_increment;
  gfloat page_increment;
  gfloat page_size;
};
```

Lo primero que debes saber es que no hay ninguna macro práctico-excelente o función de acceso para obtener el `value` de un ajuste, por lo que tendrás que (horror de horrores) les gusta un *verdadero* programador C. No se preocupe - la `GTK_ADJUSTMENT (Object)` macro se ejecuta en tiempo de comprobación de tipos (como lo hacen todos los GTK conversión de tipos de macros, en realidad). Puesto que, cuando se establece el `value` de un ajuste, por lo general, quiere que el cambio se refleje en todos los widgets que utiliza este ajuste, GTK proporciona esta función de confort para hacer esto:

```
void gtk_adjustment_set_value( GtkAdjustment *adjustment,
                               gfloat         value );
```

Como se mencionó anteriormente, el ajuste es una subclase del objeto al igual que todos los widgets diferentes, y por lo tanto es capaz de emitir señales. Esto es, por supuesto, ¿por qué actualizaciones se automágicamente cuando se comparte un objeto ajuste entre una barra de desplazamiento y otro control ajustable; todos los widgets ajustables co-

nectan manejadores de señales para su ajuste de `value_changed` señal, al igual que el programa. Aquí está la definición de esta señal en `struct _GtkAdjustmentClass` :

```
void (* value_changed) (GtkAdjustment *adjustment);
```

Los varios widgets que usan el objeto Adjustment emitirán esta señal en un ajuste cada vez que cambia su valor. Esto sucede tanto cuando la entrada de usuario hace que el control deslizante para desplazarse en un control de rango, así como cuando el programa explícitamente cambia el valor con `gtk_adjustment_set_value()` . Así, por ejemplo, si usted tiene un control de escala, y usted desea cambiar la rotación de una imagen cada vez que su valor cambia, debe crear una devolución de llamada de esta manera: `void cb_rotate_picture (GtkAdjustment *adj, GtkWidget *picture)`

```
{
  set_picture_rotation (picture, adj->value);
...
```

y conéctelo al ajuste del control de escala de la siguiente manera:

```
gtk_signal_connect (GTK_OBJECT (adj), "value_changed",
                    GTK_SIGNAL_FUNC (cb_rotate_picture), picture);
```

¿Qué pasa cuando un control reconfigura los `upper` o `lower` campos de su ajuste, por ejemplo cuando un usuario añade más texto en un widget de texto? En este caso, se emite la `changed` señal, que se ve así:

```
void (* changed)       (GtkAdjustment *adjustment);
```

Los controles de rango típicamente conectar un controlador a esta señal, que cambia su apariencia para reflejar el cambio - por ejemplo, el tamaño de la corredera en una barra de desplazamiento aumentará o disminuirá en proporción inversa a la diferencia entre los `lower` y `upper` valores de su ajuste. Probablemente nunca tendrá que conectar un controlador a esta señal, a menos que usted está escribiendo un nuevo tipo de control de rango. Sin embargo, si cambia alguno de los valores de un ajuste directamente, debe emitir esta señal para volver a configurar los widgets de lo que lo están utilizando, así:

```
gtk_signal_emit_by_name (GTK_OBJECT (adjustment), "changed");
```

Ahora vayan y ajustar!

Controles de Rango ´

La categoría de los controles de rango incluye el widget omnipresente barra de desplazamiento y el menos común widget "escala". Aunque estos dos tipos de controles se utiliza generalmente para fines diferentes, que son bastante similares en función y la aplicación. Todos los controles de rango comparten un conjunto común de elementos gráficos, cada uno de los cuales tiene su propia ventana X y recibe eventos. Todas ellas contienen una "depresión" y un "slider" (lo que a veces se llama una "rueda" en otros entornos GUI). Arrastrando el deslizador con el puntero se mueve hacia atrás y adelante dentro de la cubeta, mientras hace clic en los avances valle de la corredera hacia la ubicación del clic, ya sea completamente, o por una cantidad designada, dependiendo de qué botón del ratón es usado.

Adjustments

Como se mencionó en ajustes anteriores, todos los controles de rango están asociados con un objeto de ajuste, de la que se calcula la longitud de la corredera y su posición dentro de la cubeta. Cuando el usuario manipula el deslizador, el control de rango cambiará el valor del ajuste.

Aplicaciones Scrollbar

Estos son sus estándares, run-of-the-mill barras de desplazamiento. Estos deben ser utilizados sólo para desplazarse otro control, como una lista, un cuadro de texto o una ventana (y por lo general es más fácil utilizar el widget de la ventana desplazan por la mayoría de los casos). Para otros fines, deberá utilizar los controles de escala, ya que son más amigables y más completo más. Hay tipos separados para barras de desplazamiento horizontal y vertical. Realmente no hay mucho que decir acerca de estos. Se crean las siguientes funciones, definidas en `<gtk/gtkhscrollbar.h>` y `<gtk/gtkvscrollbar.h>`: `GtkWidget *gtk_hscrollbar_new( GtkAdjustment *adjustment );`

```
GtkWidget *gtk_vscrollbar_new( GtkAdjustment *adjustment );
```

y eso es todo (si no me crees, mira en los archivos de cabecera!). El `adjustment` argumento puede ser un puntero a un ajuste existente, o NULL, en cuyo caso se creará para usted. Especificación de NULL en realidad podría ser útil en este caso, si desea pasar el ajuste recién creado a la función constructora de algún otro widget que lo configurará para usted, como un widget de texto.

Controles de Escala

Los controles de escala se utilizan para permitir al usuario seleccionar y manipular visualmente un valor dentro de un rango específico. Es posible que desee utilizar un control de escala, por ejemplo, para ajustar el nivel de ampliación en una vista previa ampliada de una imagen, o para controlar el brillo de un color, o para especificar el

número de minutos de inactividad antes de que el salvapantallas se apodera de la pantalla .

Creación de un Control de Escala

Al igual que con las barras de desplazamiento, hay muchos tipos diferentes de widgets para widgets de escala horizontal y vertical. (La mayoría de los programadores parecen favorecer a los widgets de escala horizontal.) Ya que trabajan esencialmente de la misma manera, no hay necesidad de tratarlos por separado aquí. Las siguientes funciones, definidas en `<gtk/gtkvscale.h>` y `<gtk/gtkhscale.h>` , crear widgets de escala vertical y horizontal, respectivamente:

```
GtkWidget *gtk_vscale_new( GtkAdjustment *adjustment );

GtkWidget *gtk_hscale_new( GtkAdjustment *adjustment );
```

El `adjustment` argumento puede ser un ajuste que ya se ha creado con `gtk_adjustment_new()` , o `NULL` , en cuyo caso, un ajuste anónimo se crea con todos sus valores establece en `0.0` (que no es muy útil en este caso). Con el fin de evitar confundir a ti mismo, es probable que desee para crear su ajuste con un `page_size` de `0.0` por lo que su `upper` valor en realidad corresponde al valor más alto que el usuario puede seleccionar.

Adjustments

(Si usted *ya* está completamente confundido, lea la sección sobre Ajustes de nuevo para obtener una explicación de qué es exactamente ajustes hacer y cómo crearlos y manipularlos.)

Funciones y Señales (bueno, las funciones, por lo menos)

Los widgets de escala pueden mostrar su valor actual como un número al lado del canal. El comportamiento por defecto es mostrar el valor, pero puede cambiar esto con la siguiente función:

```
void gtk_scale_set_draw_value( GtkScale *scale,
                               gint      draw_value );
```

Como ya habrán adivinado, `draw_value` o es `TRUE` o `FALSE` , con consecuencias predecibles para cualquiera de ellos. El valor mostrado por un control de escala se redondea a un decimal por defecto, al igual que el `value` de campo en su GtkAdjustment. Usted puede cambiar esto con:

```
void gtk_scale_set_digits( GtkScale *scale,
                           gint     digits );
```

donde `digits` es el número de decimales que desee. Puede ajustar `digits` para lo que quieras, pero no más de 13 lugares decimales en realidad se dibujará en la pan-

talla. Finalmente, el valor se puede dibujar en diferentes posiciones relativas al canal:

```
void gtk_scale_set_value_pos( GtkScale        *scale,
                              GtkPositionType  pos );
```

El argumento `pos` es de tipo `GtkPositionType` , que se define en `<gtk/gtkenums.h>` , y puede tomar uno de los valores siguientes:

```
   GTK_POS_LEFT
   GTK_POS_RIGHT
   GTK_POS_TOP
   GTK_POS_BOTTOM
```

Si coloca el valor en el "lado" del canal (por ejemplo, en la parte superior o inferior de un control de escala horizontal), entonces seguirá el control deslizante hacia arriba y hacia abajo del canal. Todas las funciones anteriores están definidos en `<gtk/gtkscale.h>` . Los archivos de cabecera para todos los widgets GTK se incluyen automáticamente cuando se incluye `<gtk/gtk.h>` . Sin embargo, usted debe buscar en los archivos de cabecera de todos los widgets que le interesan,

Funciones de rango común

La clase de widget Range es bastante complicada internamente, pero, como todos los "clase base" widgets, la mayor parte de su complejidad sólo es interesante si quieres hackear en él. Además, casi todas las funciones y señales que define sólo se ha usado en la escritura de los widgets derivados. Hay, sin embargo, algunas funciones útiles que se definen en `<gtk/gtkrange.h>` y funcionará en todos los controles de rango. Configuración de la Actualización de Políticas La "política de actualización" de un widget rango define en qué puntos durante la interacción con el usuario que va a cambiar el `value` campo de su Adjustment y emitirá la señal "value_changed" en este ajuste. Las políticas de actualización, que se definen en `<gtk/gtkenums.h>` como el tipo de `enum GtkUpdateType` , son: GTK_UPDATE_POLICY_CONTINUOUS - Este es el valor predeterminado. El "value_changed" señal se emite de forma continua, es decir, cada vez que el cursor se desplaza por la más mínima cantidad. GTK_UPDATE_POLICY_DISCONTINUOUS - El "value_changed" señal sólo se emite cuando el regulador ha dejado de moverse y el usuario ha soltado el botón del ratón. GTK_UPDATE_POLICY_DELAYED - La señal "value_changed" se emite cuando el usuario suelta el botón del ratón, o si el control deslizante deja de moverse durante un corto período de tiempo. La política de actualización de un control de rango se puede ajustar por colada usando el `GTK_RANGE (Widget)` macro y pasarla a esta función:

```
void gtk_range_set_update_policy( GtkRange      *range,
```

```
                                    GtkUpdateType  policy) ;
```

Obtener y establecer ajustes

Obtener y establecer el ajuste de un control de rango "sobre la marcha" se hace, como era previsible, con:

```
GtkAdjustment* gtk_range_get_adjustment( GtkRange *range );

void gtk_range_set_adjustment( GtkRange      *range,
                               GtkAdjustment *adjustment );
```

`gtk_range_get_adjustment()` devuelve un puntero a la adaptación a la que `range` está conectado. `gtk_range_set_adjustment()` no hace absolutamente nada si se le pasa el ajuste que `range` ya está utilizando, independientemente de si ha cambiado alguno de sus campos o no. Si se le pasa un nuevo ajuste, lo hará unreference la anterior si es que existe (posiblemente destruyendo), conecte las señales apropiadas a la nueva, y llamar a la función privada `gtk_range_adjustment_changed()` , el cual (o por lo menos, se supone a ...) volver a calcular el tamaño y / o posición de la corredera y volver a dibujar, si es necesario. Como se mencionó en la sección de ajustes, si desea volver a utilizar el mismo ajuste, al modificar sus valores directamente, se debe emitir la señal "changed" en ella, así: `gtk_signal_emit_by_name (GTK_OBJECT (adjustment), "changed");`

Key and Mouse bindings

Todos los controles de rango de GTK reaccionan a clics de ratón más o menos de la misma manera. Al hacer clic en el botón-1 en el canal hará que su ajuste de `page_increment` que se añade o se resta de su `value` , y la corredera se mueva en consecuencia. Al hacer clic en el botón del ratón-2 en el canal va a saltar el cursor hasta el punto en que se ha hecho clic en el botón. Al hacer clic en un botón en una barra de desplazamiento de flechas hará que su valor de ajuste para cambiar `step_increment` a la vez. Se puede tomar un poco de tiempo para acostumbrarse, pero por defecto, barras de desplazamiento, así como los controles de escala puede tomar el foco del teclado en GTK. Si crees que tus usuarios encontrarán esta muy confuso, siempre se puede desactivar esto desarmar el `GTK_CAN_FOCUS` bandera en la barra de desplazamiento, así:

```
GTK_WIDGET_UNSET_FLAGS (scrollbar, GTK_CAN_FOCUS);
```

Las asociaciones de teclas (que son, por supuesto, sólo se activa cuando el control tiene el foco) son ligeramente diferentes entre los controles de rango horizontal y vertical, por razones obvias. Tampoco son exactamente lo mismo para los controles de escala como lo son para las barras de desplazamiento, por razones un poco menos obvias (posiblemente pa-

ra evitar la confusión entre las claves de barras de desplazamiento horizontales y verticales en desplazar las ventanas, donde ambos operan en la misma área).

Aplicaciones verticales de rango

Todos los controles de rango verticales pueden ser operados con el arriba y abajo las teclas de flecha, así como con el `Page Up` y `Page Down` teclas. Las flechas mover el control deslizante hacia arriba y hacia abajo por `step_increment` , mientras que `Page Up` y `Page Down` moverlo `page_increment` . El usuario también puede mover el deslizador todo el camino a un extremo o el otro de la cubeta mediante el teclado. Con el widget VScale, esto se hace con el `Home` y `End` teclas, mientras que con el widget VScrollBar, esto se hace tecleando `Control-Page Up` y `Control-Page Down` .

Aplicaciones horizontales Gama

Las teclas de flecha izquierda y derecha funcionan como es de esperar en estos widgets, moviendo el deslizador hacia atrás y hacia adelante por `step_increment` . El `Home` y `End` teclas mueven el cursor a los extremos de la canaleta. Por el widget hscale, moviendo el deslizador `page_increment` se realiza con `Control-Left` y `Control-Right` , mientras que para HScrollbar, se hace con `Control-Home` y `Control-End` .

Ejemplo

Este ejemplo es una versión modificada del "controles de la cocina" de prueba desde `testgtk.c` . Básicamente se pone una ventana con tres controles de rango todos conectados al mismo ajuste, y un par de controles para ajustar algunos de los parámetros mencionados anteriormente y en la sección de ajustes, para que pueda ver cómo afectan a la forma en que estos aparatos para trabajar el usuario.

```
/* example-start rangewidgets rangewidgets.c */

#include <gtk/gtk.h>

GtkWidget *hscale, *vscale;

void cb_pos_menu_select( GtkWidget       *item,
                         GtkPositionType  pos )
{
    /* Set the value position on both scale widgets */
    gtk_scale_set_value_pos (GTK_SCALE (hscale), pos);
    gtk_scale_set_value_pos (GTK_SCALE (vscale), pos);
}

void cb_update_menu_select( GtkWidget     *item,
                            GtkUpdateType  policy )
{
    /* Set the update policy for both scale widgets */
    gtk_range_set_update_policy (GTK_RANGE (hscale), policy);
```

```
    gtk_range_set_update_policy (GTK_RANGE (vscale), policy);
}

void cb_digits_scale( GtkAdjustment *adj )
{
    /* Set the number of decimal places to which adj->value is rounded */
    gtk_scale_set_digits (GTK_SCALE (hscale), (gint) adj->value);
    gtk_scale_set_digits (GTK_SCALE (vscale), (gint) adj->value);
}

void cb_page_size( GtkAdjustment *get,
                   GtkAdjustment *set )
{
    /* Set the page size and page increment size of the sample
     * adjustment to the value specified by the "Page Size" scale */
    set->page_size = get->value;
    set->page_increment = get->value;
    /* Now emit the "changed" signal to reconfigure all the widgets that
     * are attached to this adjustment */
    gtk_signal_emit_by_name (GTK_OBJECT (set), "changed");
}

void cb_draw_value( GtkToggleButton *button )
{
    /* Turn the value display on the scale widgets off or on depending
     *  on the state of the checkbutton */
    gtk_scale_set_draw_value (GTK_SCALE (hscale), button->active);
    gtk_scale_set_draw_value (GTK_SCALE (vscale), button->active);
}

/* Convenience functions */

GtkWidget *make_menu_item( gchar         *name,
                           GtkSignalFunc  callback,
                           gpointer       data )
{
    GtkWidget *item;

    item = gtk_menu_item_new_with_label (name);
    gtk_signal_connect (GTK_OBJECT (item), "activate",
                        callback, data);
    gtk_widget_show (item);

    return(item);
}

void scale_set_default_values( GtkScale *scale )
{
    gtk_range_set_update_policy (GTK_RANGE (scale),
                                 GTK_UPDATE_CONTINUOUS);
    gtk_scale_set_digits (scale, 1);
    gtk_scale_set_value_pos (scale, GTK_POS_TOP);
    gtk_scale_set_draw_value (scale, TRUE);
}

/* makes the sample window */

void create_range_controls( void )
```

```
{
    GtkWidget *window;
    GtkWidget *box1, *box2, *box3;
    GtkWidget *button;
    GtkWidget *scrollbar;
    GtkWidget *separator;
    GtkWidget *opt, *menu, *item;
    GtkWidget *label;
    GtkWidget *scale;
    GtkObject *adj1, *adj2;

    /* Standard window-creating stuff */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        GTK_SIGNAL_FUNC(gtk_main_quit),
                        NULL);
    gtk_window_set_title (GTK_WINDOW (window), "range controls");

    box1 = gtk_vbox_new (FALSE, 0);
    gtk_container_add (GTK_CONTAINER (window), box1);
    gtk_widget_show (box1);

    box2 = gtk_hbox_new (FALSE, 10);
    gtk_container_set_border_width (GTK_CONTAINER (box2), 10);
    gtk_box_pack_start (GTK_BOX (box1), box2, TRUE, TRUE, 0);
    gtk_widget_show (box2);

    /* value, lower, upper, step_increment, page_increment, page_size */
    /* Note that the page_size value only makes a difference for
     * scrollbar widgets, and the highest value you'll get is actually
     * (upper - page_size). */
    adj1 = gtk_adjustment_new (0.0, 0.0, 101.0, 0.1, 1.0, 1.0);

    vscale = gtk_vscale_new (GTK_ADJUSTMENT (adj1));
    scale_set_default_values (GTK_SCALE (vscale));
    gtk_box_pack_start (GTK_BOX (box2), vscale, TRUE, TRUE, 0);
    gtk_widget_show (vscale);

    box3 = gtk_vbox_new (FALSE, 10);
    gtk_box_pack_start (GTK_BOX (box2), box3, TRUE, TRUE, 0);
    gtk_widget_show (box3);

    /* Reuse the same adjustment */
    hscale = gtk_hscale_new (GTK_ADJUSTMENT (adj1));
    gtk_widget_set_usize (GTK_WIDGET (hscale), 200, 30);
    scale_set_default_values (GTK_SCALE (hscale));
    gtk_box_pack_start (GTK_BOX (box3), hscale, TRUE, TRUE, 0);
    gtk_widget_show (hscale);

    /* Reuse the same adjustment again */
    scrollbar = gtk_hscrollbar_new (GTK_ADJUSTMENT (adj1));
    /* Notice how this causes the scales to always be updated
     * continuously when the scrollbar is moved */
    gtk_range_set_update_policy (GTK_RANGE (scrollbar),
                                 GTK_UPDATE_CONTINUOUS);
    gtk_box_pack_start (GTK_BOX (box3), scrollbar, TRUE, TRUE, 0);
    gtk_widget_show (scrollbar);
```

```
    box2 = gtk_hbox_new (FALSE, 10);
    gtk_container_set_border_width (GTK_CONTAINER (box2), 10);
    gtk_box_pack_start (GTK_BOX (box1), box2, TRUE, TRUE, 0);
    gtk_widget_show (box2);

    /* A checkbutton to control whether the value is displayed or not */
    button = gtk_check_button_new_with_label("Display value on scale widg-
ets");
    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (button), TRUE);
    gtk_signal_connect (GTK_OBJECT (button), "toggled",
                        GTK_SIGNAL_FUNC(cb_draw_value), NULL);
    gtk_box_pack_start (GTK_BOX (box2), button, TRUE, TRUE, 0);
    gtk_widget_show (button);

    box2 = gtk_hbox_new (FALSE, 10);
    gtk_container_set_border_width (GTK_CONTAINER (box2), 10);

    /* An option menu to change the position of the value */
    label = gtk_label_new ("Scale Value Position:");
    gtk_box_pack_start (GTK_BOX (box2), label, FALSE, FALSE, 0);
    gtk_widget_show (label);

    opt = gtk_option_menu_new();
    menu = gtk_menu_new();

    item = make_menu_item ("Top",
                           GTK_SIGNAL_FUNC(cb_pos_menu_select),
                           GINT_TO_POINTER (GTK_POS_TOP));
    gtk_menu_append (GTK_MENU (menu), item);

    item = make_menu_item ("Bottom", GTK_SIGNAL_FUNC (cb_pos_menu_select),
                           GINT_TO_POINTER (GTK_POS_BOTTOM));
    gtk_menu_append (GTK_MENU (menu), item);

    item = make_menu_item ("Left", GTK_SIGNAL_FUNC (cb_pos_menu_select),
                           GINT_TO_POINTER (GTK_POS_LEFT));
    gtk_menu_append (GTK_MENU (menu), item);

    item = make_menu_item ("Right", GTK_SIGNAL_FUNC (cb_pos_menu_select),
                            GINT_TO_POINTER (GTK_POS_RIGHT));
    gtk_menu_append (GTK_MENU (menu), item);

    gtk_option_menu_set_menu (GTK_OPTION_MENU (opt), menu);
    gtk_box_pack_start (GTK_BOX (box2), opt, TRUE, TRUE, 0);
    gtk_widget_show (opt);

    gtk_box_pack_start (GTK_BOX (box1), box2, TRUE, TRUE, 0);
    gtk_widget_show (box2);

    box2 = gtk_hbox_new (FALSE, 10);
    gtk_container_set_border_width (GTK_CONTAINER (box2), 10);

    /* Yet another option menu, this time for the update policy of the
     * scale widgets */
    label = gtk_label_new ("Scale Update Policy:");
    gtk_box_pack_start (GTK_BOX (box2), label, FALSE, FALSE, 0);
    gtk_widget_show (label);
```

```
    opt = gtk_option_menu_new();
    menu = gtk_menu_new();

    item = make_menu_item ("Continuous",
                           GTK_SIGNAL_FUNC (cb_update_menu_select),
                           GINT_TO_POINTER (GTK_UPDATE_CONTINUOUS));
    gtk_menu_append (GTK_MENU (menu), item);

    item = make_menu_item ("Discontinuous",
                            GTK_SIGNAL_FUNC (cb_update_menu_select),
                            GINT_TO_POINTER (GTK_UPDATE_DISCONTINUOUS));
    gtk_menu_append (GTK_MENU (menu), item);

    item = make_menu_item ("Delayed",
                           GTK_SIGNAL_FUNC (cb_update_menu_select),
                           GINT_TO_POINTER (GTK_UPDATE_DELAYED));
    gtk_menu_append (GTK_MENU (menu), item);

    gtk_option_menu_set_menu (GTK_OPTION_MENU (opt), menu);
    gtk_box_pack_start (GTK_BOX (box2), opt, TRUE, TRUE, 0);
    gtk_widget_show (opt);

    gtk_box_pack_start (GTK_BOX (box1), box2, TRUE, TRUE, 0);
    gtk_widget_show (box2);

    box2 = gtk_hbox_new (FALSE, 10);
    gtk_container_set_border_width (GTK_CONTAINER (box2), 10);

    /* An HScale widget for adjusting the number of digits on the
     * sample scales. */
    label = gtk_label_new ("Scale Digits:");
    gtk_box_pack_start (GTK_BOX (box2), label, FALSE, FALSE, 0);
    gtk_widget_show (label);

    adj2 = gtk_adjustment_new (1.0, 0.0, 5.0, 1.0, 1.0, 0.0);
    gtk_signal_connect (GTK_OBJECT (adj2), "value_changed",
                        GTK_SIGNAL_FUNC (cb_digits_scale), NULL);
    scale = gtk_hscale_new (GTK_ADJUSTMENT (adj2));
    gtk_scale_set_digits (GTK_SCALE (scale), 0);
    gtk_box_pack_start (GTK_BOX (box2), scale, TRUE, TRUE, 0);
    gtk_widget_show (scale);

    gtk_box_pack_start (GTK_BOX (box1), box2, TRUE, TRUE, 0);
    gtk_widget_show (box2);

    box2 = gtk_hbox_new (FALSE, 10);
    gtk_container_set_border_width (GTK_CONTAINER (box2), 10);

    /* And, one last HScale widget for adjusting the page size of the
     * scrollbar. */
    label = gtk_label_new ("Scrollbar Page Size:");
    gtk_box_pack_start (GTK_BOX (box2), label, FALSE, FALSE, 0);
    gtk_widget_show (label);

    adj2 = gtk_adjustment_new (1.0, 1.0, 101.0, 1.0, 1.0, 0.0);
    gtk_signal_connect (GTK_OBJECT (adj2), "value_changed",
                        GTK_SIGNAL_FUNC (cb_page_size), adj1);
    scale = gtk_hscale_new (GTK_ADJUSTMENT (adj2));
```

```
    gtk_scale_set_digits (GTK_SCALE (scale), 0);
    gtk_box_pack_start (GTK_BOX (box2), scale, TRUE, TRUE, 0);
    gtk_widget_show (scale);

    gtk_box_pack_start (GTK_BOX (box1), box2, TRUE, TRUE, 0);
    gtk_widget_show (box2);

    separator = gtk_hseparator_new ();
    gtk_box_pack_start (GTK_BOX (box1), separator, FALSE, TRUE, 0);
    gtk_widget_show (separator);

    box2 = gtk_vbox_new (FALSE, 10);
    gtk_container_set_border_width (GTK_CONTAINER (box2), 10);
    gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, TRUE, 0);
    gtk_widget_show (box2);

    button = gtk_button_new_with_label ("Quit");
    gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                               GTK_SIGNAL_FUNC(gtk_main_quit),
                               NULL);
    gtk_box_pack_start (GTK_BOX (box2), button, TRUE, TRUE, 0);
    GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);
    gtk_widget_grab_default (button);
    gtk_widget_show (button);

    gtk_widget_show (window);
}

int main( int   argc,
          char *argv[] )
{
    gtk_init(&argc, &argv);

    create_range_controls();

    gtk_main();

    return(0);
}

/* example-end */

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc rangewidgets.c -o
rangewidgets $(pkg-config gtk+-2.0 --cflags --libs)

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./rangewidgets
```

Usted se dará cuenta de que el programa no llama `gtk_signal_connect` para el "delete_event", pero sólo para la señal "destroy". Esto todavía llevará a cabo la función deseada, ya que una no controlada "delete_event" dará lugar a una señal "destroy" está dando a la ventana.

Miscelánea Labels Etiquetas

Las etiquetas se utilizan mucho en GTK, y son relativamente simples. No pueden emitir señales, ya que no tienen una ventana X asociada.

EventBox

Si usted necesita para capturar señales, o hacer recortes, colóquelo dentro de un EventBox widget o un control Button. Para crear una nueva etiqueta, utilice:

```
GtkWidget *gtk_label_new( char *str );
```

El único argumento es la cadena que desea la etiqueta para mostrar. Para cambiar el texto de la etiqueta después de la creación, el uso de la función:

```
void gtk_label_set_text( GtkLabel *label,
                         char     *str );
```

El primer argumento es la etiqueta que ha creado anteriormente (emitirse en el `GTK_LABEL()` macro), y la segunda es la nueva cadena. El espacio necesario para la nueva cadena se ajustará automática-mente si es necesario. Puede producir etiquetas multilíneas poniendo saltos de línea en la cadena de la etiqueta. Para recuperar la cadena actual, utilice:

```
void gtk_label_get( GtkLabel  *label,
                    char     **str );
```

El primer argumento es la etiqueta que ha creado, y el segundo, el cambio de la cadena. No liberar la cadena de retorno, ya que se utiliza internamente por GTK. El texto de la etiqueta se puede justificar usando:

```
void gtk_label_set_justify( GtkLabel         *label,
                            GtkJustification  jtype );
```

Los valores para `jtype` son:

```
GTK_JUSTIFY_LEFT
GTK_JUSTIFY_RIGHT
GTK_JUSTIFY_CENTER (the default)
GTK_JUSTIFY_FILL
```

El widget de etiqueta también es capaz de envolver la línea de texto automáticamente. Esto se puede activar usando:

```
void gtk_label_set_line_wrap (GtkLabel *label,
                              gboolean  wrap);
```

La `wrap` argumento toma un valor VERDADERO o FALSO. Si desea que su etiqueta subrayada, entonces se puede establecer un patrón en la etiqueta:

```
void       gtk_label_set_pattern   (GtkLabel          *label,
                                    const gchar       *pattern);
```

El argumento patrón indica cómo el subrayado debe mirar. Consiste en una serie de subrayado y espacios. Un guión indica que el carácter correspondiente en la etiqueta debe ser subrayado. Por ejemplo, la cadena "__ __"

hincapié en los dos primeros caracteres y los caracteres octavo y noveno. A continuación se muestra un pequeño ejemplo para ilustrar estas funciones. Este ejemplo hace uso del control Frame para

demostrar mejor los estilos de etiqueta.

Frame

Puede ignorar esto por ahora como el Frame widget se explicará más adelante.

```
/* example-start label label.c */

#include <gtk/gtk.h>

int main( int   argc,
          char *argv[] )
{
  static GtkWidget *window = NULL;
  GtkWidget *hbox;
  GtkWidget *vbox;
  GtkWidget *frame;
  GtkWidget *label;

  /* Initialise GTK */
  gtk_init(&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_signal_connect (GTK_OBJECT (window), "destroy",
                      GTK_SIGNAL_FUNC(gtk_main_quit),
                      NULL);

  gtk_window_set_title (GTK_WINDOW (window), "Label");
  vbox = gtk_vbox_new (FALSE, 5);
  hbox = gtk_hbox_new (FALSE, 5);
  gtk_container_add (GTK_CONTAINER (window), hbox);
  gtk_box_pack_start (GTK_BOX (hbox), vbox, FALSE, FALSE, 0);
  gtk_container_set_border_width (GTK_CONTAINER (window), 5);

  frame = gtk_frame_new ("Normal Label");
  label = gtk_label_new ("This is a Normal label");
  gtk_container_add (GTK_CONTAINER (frame), label);
  gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

  frame = gtk_frame_new ("Multi-line Label");
  label = gtk_label_new ("This is a Multi-line label.\nSecond line\n" \
                         "Third line");
  gtk container add (GTK CONTAINER (frame), label);
  gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

  frame = gtk_frame_new ("Left Justified Label");
  label = gtk_label_new ("This is a Left-Justified\n" \
                         "Multi-line label.\nThird      line");
  gtk_label_set_justify (GTK_LABEL (label), GTK_JUSTIFY_LEFT);
  gtk_container_add (GTK_CONTAINER (frame), label);
  gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

  frame = gtk_frame_new ("Right Justified Label");
  label = gtk_label_new ("This is a Right-Justified\nMulti-line label.\n" \
                         "Fourth line, (j/k)");
  gtk_label_set_justify (GTK_LABEL (label), GTK_JUSTIFY_RIGHT);
```

```
  gtk_container_add (GTK_CONTAINER (frame), label);
  gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

  vbox = gtk_vbox_new (FALSE, 5);
  gtk_box_pack_start (GTK_BOX (hbox), vbox, FALSE, FALSE, 0);
  frame = gtk_frame_new ("Line wrapped label");
  label = gtk_label_new ("This is an example of a line-wrapped label.  It " \
                         "should not be taking up the entire             " /*
big space to test spacing */\
                         "width allocated to it, but automatically " \
                         "wraps the words to fit.  " \
                         "The time has come, for all good men, to come to " \
                         "the aid of their party.  " \
                         "The sixth sheik's six sheep's sick.\n" \
                         "     It supports multiple paragraphs correctly, " \
                         "and  correctly   adds "\
                         "many          extra  spaces. ");
  gtk_label_set_line_wrap (GTK_LABEL (label), TRUE);
  gtk_container_add (GTK_CONTAINER (frame), label);
  gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

  frame = gtk_frame_new ("Filled, wrapped label");
  label = gtk_label_new ("This is an example of a line-wrapped, filled label.
" \
                         "It should be taking "\
                         "up the entire              width allocated to it.
" \
                         "Here is a sentence to prove "\
                         "my point.  Here is another sentence. "\
                         "Here comes the sun, do de do de do.\n"\
                         "    This is a new paragraph.\n"\
                         "    This is another newer, longer, better " \
                         "paragraph.  It is coming to an end, "\
                         "unfortunately.");
  gtk_label_set_justify (GTK_LABEL (label), GTK_JUSTIFY_FILL);
  gtk_label_set_line_wrap (GTK_LABEL (label), TRUE);
  gtk_container_add (GTK_CONTAINER (frame), label);
  gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

  frame = gtk_frame_new ("Underlined label");
  label = gtk_label_new ("This label is underlined!\n"
                         "This one is underlined in quite a funky fashion");
  gtk_label_set_justify (GTK_LABEL (label), GTK_JUSTIFY_LEFT);
  gtk_label_set_pattern (GTK_LABEL (label),
                         "_____________________________ _ _________ _ ______
__ ________ ___");
  gtk_container_add (GTK_CONTAINER (frame), label);
  gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

  gtk_widget_show_all (window);

  gtk_main ();

  return(0);
}
/* example-end */
```

```
linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc label.c -o label $(pkg-
config gtk+-2.0 --cflags –libs)
linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./label
```

Arrows Los widget Flecha dibuja una punta de flecha, se enfrenta en un número de direcciones posibles y que tiene un número de estilos posibles. Puede ser muy útil cuando se coloca sobre un botón en muchas aplicaciones. Al igual que el widget Label, emite ninguna señal. Sólo hay dos funciones para manipular un widget Arrow:

```
GtkWidget *gtk_arrow_new( GtkArrowType   arrow_type,
                          GtkShadowType  shadow_type );

void gtk_arrow_set( GtkArrow      *arrow,
                    GtkArrowType   arrow_type,
                    GtkShadowType  shadow_type );
```

El primero crea un widget flecha nuevo con el tipo indicado y apariencia. El segundo permite que estos valores sean alterados retrospectivamente. El `arrow_type` argumento puede tomar uno de los siguientes valores:

```
  GTK_ARROW_UP
  GTK_ARROW_DOWN
  GTK_ARROW_LEFT
  GTK_ARROW_RIGHT
```

Estos valores obviamente indican la dirección en la que la flecha apuntará. El `shadow_type` argumento puede tomar uno de estos valores:

```
  GTK_SHADOW_IN
  GTK_SHADOW_OUT (the default)
  GTK_SHADOW_ETCHED_IN
  GTK_SHADOW_ETCHED_OUT
   GTK_SHADOW_ETCHED_OUT
```

He aquí un pequeño ejemplo para ilustrar su uso.

```
/* example-start arrow arrow.c */

#include <gtk/gtk.h>

/* Create an Arrow widget with the specified parameters
 * and pack it into a button */
GtkWidget *create_arrow_button( GtkArrowType  arrow_type,
                                GtkShadowType shadow_type )
{
  GtkWidget *button;
  GtkWidget *arrow;

  button = gtk_button_new();
  arrow = gtk_arrow_new (arrow_type, shadow_type);

  gtk_container_add (GTK_CONTAINER (button), arrow);
```

```
  gtk_widget_show(button);
  gtk_widget_show(arrow);

  return(button);
}

int main( int   argc,
          char *argv[] )
{
  /* GtkWidget is the storage type for widgets */
  GtkWidget *window;
  GtkWidget *button;
  GtkWidget *box;

  /* Initialize the toolkit */
  gtk_init (&argc, &argv);

  /* Create a new window */
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

  gtk_window_set_title (GTK_WINDOW (window), "Arrow Buttons");

  /* It's a good idea to do this for all windows. */
  gtk_signal_connect (GTK_OBJECT (window), "destroy",
                      GTK_SIGNAL_FUNC (gtk_main_quit), NULL);

  /* Sets the border width of the window. */
  gtk_container_set_border_width (GTK_CONTAINER (window), 10);

  /* Create a box to hold the arrows/buttons */
  box = gtk_hbox_new (FALSE, 0);
  gtk_container_set_border_width (GTK_CONTAINER (box), 2);
  gtk_container_add (GTK_CONTAINER (window), box);

  /* Pack and show all our widgets */
  gtk_widget_show(box);

  button = create_arrow_button(GTK_ARROW_UP, GTK_SHADOW_IN);
  gtk_box_pack_start (GTK_BOX (box), button, FALSE, FALSE, 3);

  button = create_arrow_button(GTK_ARROW_DOWN, GTK_SHADOW_OUT);
  gtk_box_pack_start (GTK_BOX (box), button, FALSE, FALSE, 3);

  button = create_arrow_button(GTK_ARROW_LEFT, GTK_SHADOW_ETCHED_IN);
  gtk_box_pack_start (GTK_BOX (box), button, FALSE, FALSE, 3);

  button = create_arrow_button(GTK_ARROW_RIGHT, GTK_SHADOW_ETCHED_OUT);
  gtk_box_pack_start (GTK_BOX (box), button, FALSE, FALSE, 3);

  gtk_widget_show (window);

  /* Rest in gtk_main and wait for the fun to begin! */
  gtk_main ();

  return(0);
}
/* example-end */
```

```
linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc arrow.c -o arrow $(pkg-
config gtk+-2.0 --cflags --libs)

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./arrow
```

El objeto Tooltips

Estas son pequeñas cadenas de texto que aparecen cuando se deja el puntero sobre un botón u otro control durante unos segundos. Son fáciles de usar, por lo que me limitaré a explicar sin dar un ejemplo. Si quieres ver algo de código, eche un vistazo al programa testgtk.c distribuye con GTK. Los controles que no reciben eventos (widgets que no tienen su propia ventana) no funcionará con las pistas. La primera llamada que va a utilizar crea una nueva descripción. Sólo necesita hacer esto una vez para un conjunto de información sobre herramientas como los `GtkTooltips` objeto Esta función devuelve se puede utilizar para crear múltiples pistas.

```
GtkTooltips *gtk_tooltips_new( void );
```
Una vez que haya creado una nueva descripción, y el widget que desea utilizarlo en adelante, sólo tiene que utilizar esta llamada para establecer:

```
void gtk_tooltips_set_tip( GtkTooltips *tooltips,
                           GtkWidget   *widget,
                           const gchar *tip text,
                           const gchar *tip_private );
```
El primer argumento es la descripción que ya ha creado, seguido por el widget que desea que esta información sobre herramientas emergente para, y el texto que quieres que diga. El último argumento es una cadena de texto que puede ser utilizado como un identificador cuando se usa para aplicar GtkTipsQuery ayuda sensible al contexto. Por el momento, se puede establecer en NULL. He aquí un pequeño ejemplo:

```
GtkTooltips *tooltips;
GtkWidget *button;
.
.
.
tooltips = gtk_tooltips_new ();
button = gtk_button_new_with_label ("button 1");
.
.
.
```

gtk_tooltips_set_tip (tooltips, button, "This is button 1", NULL);

Hay otras llamadas que se pueden utilizar con las pistas. Me limitaré a enumerar con una breve descripción de lo que hacen.

```
void gtk_tooltips_enable( GtkTooltips *tooltips );
```

Habilitar un conjunto de información sobre herramientas desactivada.

```
void gtk_tooltips_disable( GtkTooltips *tooltips );
```

Desactivar un conjunto de pistas activadas.

```
void gtk_tooltips_set_delay( GtkTooltips *tooltips,
                             gint         delay );
```

Establece los milisegundos que tiene que estar el puntero sobre el control antes de la pista aparezca. El valor por defecto es de 500 milisegundos (medio segundo).

```
void gtk_tooltips_set_colors( GtkTooltips *tooltips,
                              GdkColor    *background,
                              GdkColor    *foreground );
```

Establecer el primer plano y color de fondo de la información sobre herramientas. Y eso es todas las funciones asociadas con las pistas. Más de lo que nunca querrá saber.

Barras de Progreso

Las barras de progreso se utilizan para mostrar el estado de una operación. Ellos son bastante fáciles de usar, como se puede ver en los ejemplos siguientes. Pero primero vamos a empezar con las llamadas a crear una nueva barra de progreso. Hay dos formas de crear una barra de progreso, uno sencillo que no tiene argumentos, y uno que toma un objeto de ajuste como argumento. Si se utiliza la primera, la barra de progreso crea su propio objeto de ajuste.

```
GtkWidget *gtk_progress_bar_new( void );

GtkWidget *gtk_progress_bar_new_with_adjustment( GtkAdjustment *adjustment );
```

El segundo método tiene la ventaja de que se puede utilizar el objeto de ajuste para especificar los parámetros de rango propios de la barra de progreso. El ajuste de un objeto de progreso se puede cambiar dinámicamente usando:

```
void gtk_progress_set_adjustment( GtkProgress   *progress,
                                  GtkAdjustment *adjustment );
```

Ahora que la barra de progreso se ha creado lo podemos usar.

```
void gtk_progress_bar_update( GtkProgressBar *pbar,
                              gfloat          percentage );
```

El primer argumento es la barra que se quiere operar, y el segundo argumento es la cantidad "completa", es decir, la cantidad ha sido llenada la barra de 0-100%. Esto se pasa a la función como un número real comprendido entre 0 y 1. GTK v1.2 ha añadido nuevas funciones a la barra de progreso que le permite visualizar el valor de diferentes maneras, y para informar al usuario de su valor actual y su rango. Una barra de progreso puede ser fijado a uno de una serie de orientaciones utilizando la función

```
void gtk_progress_bar_set_orientation( GtkProgressBar *pbar,
                                       GtkProgressBarOrientation orientation
);
```

La `orientation` argumento puede tomar uno de los valores siguientes para indicar la dirección en que se mueve la barra de progreso

```
  GTK_PROGRESS_LEFT_TO_RIGHT
  GTK_PROGRESS_RIGHT_TO_LEFT
  GTK_PROGRESS_BOTTOM_TO_TOP
  GTK_PROGRESS_TOP_TO_BOTTOM
```

Cuando se utiliza como una medida de hasta qué punto un proceso ha progresado, el ProgressBar puede ser configurado para mostrar su valor en un modo continuo o discreto. En el modo continuo, la barra de progreso se actualiza para cada valor. En el modo discreto, la barra de progreso se actualiza en un número de bloques discretos. El número de bloques también es configurable. El estilo de una barra de progreso se puede establecer mediante la siguiente función.

```
void gtk_progress_bar_set_bar_style( GtkProgressBar      *pbar,
```

GtkProgressBarStyle style); El `style` parámetro puede tomar uno de dos valores:

```
  GTK_PROGRESS_CONTINUOUS
  GTK_PROGRESS_DISCRETE
```

El número de bloques discretos puede ser seleccionado llamando

```
void gtk_progress_bar_set_discrete_blocks( GtkProgressBar *pbar,
                                           guint           blocks );
```

Además de indicar la cantidad de progreso que se ha producido, la barra de progreso se puede ajustar para indicar simplemente que hay algo de actividad. Esto puede ser útil en situaciones donde el progreso no pueden ser medidos contra un rango de valo-

res. Modo de actividad no es afectada por el estilo de barra que se ha descrito anteriormente, y se anula. Este modo es VERDADERO o FALSO, y se selecciona por la siguiente función.

```
void gtk_progress_set_activity_mode( GtkProgress *progress,
                                     guint        activity_mode );
```

El tamaño del paso del indicador de actividad, y el número de bloques se fijan utilizando las siguientes funciones.

```
void gtk_progress_bar_set_activity_step( GtkProgressBar *pbar,
                                         guint           step );

void gtk_progress_bar_set_activity_blocks( GtkProgressBar *pbar,
                                           guint           blocks );
```

En el modo continuo, la barra de progreso también puede mostrar una cadena de texto configurable dentro de su canal, usando la siguiente función.

```
void gtk_progress_set_format_string( GtkProgress *progress,
                                     gchar       *format);
```

El `format` argumento es similar a aquella que se utiliza en un C `printf` comunicado. Las siguientes directrices pueden utilizarse dentro de la cadena de formato: %p - - porcentaje %v - valor %l - valor inferior del rango % u - valor superior del rango La visualización de esta cadena de texto se puede activar usando:

```
void gtk progress set show text( GtkProgress *progress,
                                 gint         show_text );
```

El `show_text` argumento es un valor booleano VERDADERO / FALSO. La aparición del texto puede ser modificado adicionalmente usando:

```
void gtk_progress_set_text_alignment( GtkProgress   *progress,
                                      gfloat         x_align,
                                      gfloat         y_align );
```

Los `x_align` y `y_align` argumentos toman valores entre 0,0 y 1,0. Sus valores indican la posición de la cadena de texto dentro de la cubeta. Los valores de 0,0 para ambos sería colocar la cadena en la esquina superior izquierda, y los valores de 0,5 (el valor predeterminado) centra el texto y los valores de 1,0 coloca el texto en la esquina inferior derecha. La configuración actual texto de un objeto de progreso se puede recuperar con la actual o un valor de ajuste especificado utilizando los siguientes dos funciones. La cadena de caracteres devuelto por estas funciones deben ser liberados por la aplicación (utilizando el g_free () función). Estas funciones devuelven la cadena con formato que se mostrará dentro de la cubeta.

```
gchar *gtk_progress_get_current_text( GtkProgress   *progress );

gchar *gtk_progress_get_text_from_value( GtkProgress *progress,
```

```
                                        gfloat       value ); Hay otra ma-
```

nera de cambiar el rango y el valor de un objeto de progreso usando la siguiente función:

```
void gtk_progress_configure( GtkProgress  *progress,
                             gfloat        value,
                             gfloat        min,
                             gfloat        max ); Esta función proporciona
```

una interfaz sencilla muy al alcance y el valor de un objeto de progreso. El resto de funciones se pueden utilizar para obtener y establecer el valor actual de un objeto progess en diversos tipos y formatos:

```
void gtk_progress_set_percentage( GtkProgress *progress,
                                  gfloat       percentage );

void gtk_progress_set_value( GtkProgress *progress,
                             gfloat       value );

gfloat gtk_progress_get_value( GtkProgress *progress );

gfloat gtk_progress_get_current_percentage( GtkProgress *progress );

gfloat gtk_progress_get_percentage_from_value( GtkProgress *progress,
                                               gfloat       value );
```

Estas funciones son bastante explica por sí mismo. La última función utiliza el ajuste de la del objeto especificado progess para calcular el valor del porcentaje del valor del rango dado.

Timeouts, I/O and Idle Functions

Barras de progreso se utiliza con tiempos de espera u otras funciones especiales (véase la sección sobre tiempos de espera, E / S y Funciones de Inactividad) para dar la ilusión de multitarea. Todas usan la función gtk_progress_bar_update de la misma manera. He aquí un ejemplo de la barra de progreso, actualizada usando tiempos de espera. Este código también muestra cómo restablecer la barra de progreso.

```
/* example-start progressbar progressbar.c */

#include <gtk/gtk.h>

typedef struct _ProgressData {
    GtkWidget *window;
    GtkWidget *pbar;
    int timer;
} ProgressData;

/* Update the value of the progress bar so that we get
 * some movement */
gint progress_timeout( gpointer data )
{
    gfloat new_val;
    GtkAdjustment *adj;
```

```
    /* Calculate the value of the progress bar using the
     * value range set in the adjustment object */

    new_val = gtk_progress_get_value( GTK_PROGRESS(data) ) + 1;

    adj = GTK_PROGRESS (data)->adjustment;
    if (new_val > adj->upper)
      new_val = adj->lower;

    /* Set the new value */
    gtk_progress_set_value (GTK_PROGRESS (data), new_val);

    /* As this is a timeout function, return TRUE so that it
     * continues to get called */
    return(TRUE);
}

/* Callback that toggles the text display within the progress
 * bar trough */
void toggle_show_text( GtkWidget    *widget,
                       ProgressData *pdata )
{
    gtk_progress_set_show_text (GTK_PROGRESS (pdata->pbar),
                                GTK_TOGGLE_BUTTON (widget)->active);
}

/* Callback that toggles the activity mode of the progress
 * bar */
void toggle_activity_mode( GtkWidget    *widget,
                           ProgressData *pdata )
{
    gtk_progress_set_activity_mode (GTK_PROGRESS (pdata->pbar),
                                    GTK_TOGGLE_BUTTON (widget)->active);
}

/* Callback that toggles the continuous mode of the progress
 * bar */
void set_continuous_mode( GtkWidget    *widget,
                          ProgressData *pdata )
{
    gtk_progress_bar_set_bar_style (GTK_PROGRESS_BAR (pdata->pbar),
                                    GTK_PROGRESS_CONTINUOUS);
}

/* Callback that toggles the discrete mode of the progress
 * bar */
void set_discrete_mode( GtkWidget    *widget,
                        ProgressData *pdata )
{
    gtk_progress_bar_set_bar_style (GTK_PROGRESS_BAR (pdata->pbar),
                                    GTK_PROGRESS_DISCRETE);
}

/* Clean up allocated memory and remove the timer */
void destroy_progress( GtkWidget     *widget,
                       ProgressData *pdata)
{
```

```
    gtk_timeout_remove (pdata->timer);
    pdata->timer = 0;
    pdata->window = NULL;
    g_free(pdata);
    gtk_main_quit();
}

int main( int   argc,
          char *argv[])
{
    ProgressData *pdata;
    GtkWidget *align;
    GtkWidget *separator;
    GtkWidget *table;
    GtkAdjustment *adj;
    GtkWidget *button;
    GtkWidget *check;
    GtkWidget *vbox;

    gtk_init (&argc, &argv);

    /* Allocate memory for the data that is passwd to the callbacks */
    pdata = g_malloc( sizeof(ProgressData) );

    pdata->window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_policy (GTK_WINDOW (pdata->window), FALSE, FALSE, TRUE);

    gtk_signal_connect (GTK_OBJECT (pdata->window), "destroy",
                        GTK_SIGNAL_FUNC (destroy_progress),
                        pdata);
    gtk_window_set_title (GTK_WINDOW (pdata->window), "GtkProgressBar");
    gtk_container_set_border_width (GTK_CONTAINER (pdata->window), 0);

    vbox = gtk_vbox_new (FALSE, 5);
    gtk_container_set_border_width (GTK_CONTAINER (vbox), 10);
    gtk_container_add (GTK_CONTAINER (pdata->window), vbox);
    gtk_widget_show(vbox);

    /* Create a centering alignment object */
    align = gtk_alignment_new (0.5, 0.5, 0, 0);
    gtk_box_pack_start (GTK_BOX (vbox), align, FALSE, FALSE, 5);
    gtk_widget_show(align);

    /* Create a Adjusment object to hold the range of the
     * progress bar */
    adj = (GtkAdjustment *) gtk_adjustment_new (0, 1, 150, 0, 0, 0);

    /* Create the GtkProgressBar using the adjustment */
    pdata->pbar = gtk_progress_bar_new_with_adjustment (adj);

    /* Set the format of the string that can be displayed in the
     * trough of the progress bar:
     * %p - percentage
     * %v - value
     * %l - lower range value
     * %u - upper range value */
    gtk_progress_set_format_string (GTK_PROGRESS (pdata->pbar),
                                    "%v from [%l-%u] (=%p%%)");
```

```
    gtk_container_add (GTK_CONTAINER (align), pdata->pbar);
    gtk_widget_show(pdata->pbar);

    /* Add a timer callback to update the value of the progress bar */
    pdata->timer = gtk_timeout_add (100, progress_timeout, pdata->pbar);

    separator = gtk_hseparator_new ();
    gtk_box_pack_start (GTK_BOX (vbox), separator, FALSE, FALSE, 0);
    gtk_widget_show(separator);

    /* rows, columns, homogeneous */
    table = gtk_table_new (2, 3, FALSE);
    gtk_box_pack_start (GTK_BOX (vbox), table, FALSE, TRUE, 0);
    gtk_widget_show(table);

    /* Add a check button to select displaying of the trough text */
    check = gtk_check_button_new_with_label ("Show text");
    gtk_table_attach (GTK_TABLE (table), check, 0, 1, 0, 1,
                      GTK_EXPAND | GTK_FILL, GTK_EXPAND | GTK_FILL,
                      5, 5);
    gtk_signal_connect (GTK_OBJECT (check), "clicked",
                        GTK_SIGNAL_FUNC (toggle_show_text),
                        pdata);
    gtk_widget_show(check);

    /* Add a check button to toggle activity mode */
    check = gtk_check_button_new_with_label ("Activity mode");
    gtk_table_attach (GTK_TABLE (table), check, 0, 1, 1, 2,
                      GTK_EXPAND | GTK_FILL, GTK_EXPAND | GTK_FILL,
                      5, 5);
    gtk_signal_connect (GTK_OBJECT (check), "clicked",
                        GTK_SIGNAL_FUNC (toggle_activity_mode),
                        pdata);
    gtk_widget_show(check);

    separator = gtk_vseparator_new ();
    gtk_table_attach (GTK_TABLE (table), separator, 1, 2, 0, 2,
                      GTK_EXPAND | GTK_FILL, GTK_EXPAND | GTK_FILL,
                      5, 5);
    gtk_widget_show(separator);

    /* Add a radio button to select continuous display mode */
    button = gtk_radio_button_new_with_label (NULL, "Continuous");
    gtk_table_attach (GTK_TABLE (table), button, 2, 3, 0, 1,
                      GTK_EXPAND | GTK_FILL, GTK_EXPAND | GTK_FILL,
                      5, 5);
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (set_continuous_mode),
                        pdata);
    gtk_widget_show (button);

    /* Add a radio button to select discrete display mode */
    button = gtk_radio_button_new_with_label(
               gtk_radio_button_group (GTK_RADIO_BUTTON (button)),
               "Discrete");
    gtk_table_attach (GTK_TABLE (table), button, 2, 3, 1, 2,
                      GTK_EXPAND | GTK_FILL, GTK_EXPAND | GTK_FILL,
                      5, 5);
```

```
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (set_discrete_mode),
                        pdata);
    gtk_widget_show (button);

    separator = gtk_hseparator_new ();
    gtk_box_pack_start (GTK_BOX (vbox), separator, FALSE, FALSE, 0);
    gtk_widget_show(separator);

    /* Add a button to exit the program */
    button = gtk_button_new_with_label ("close");
    gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                               (GtkSignalFunc) gtk_widget_destroy,
                               GTK_OBJECT (pdata->window));
    gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);

    /* This makes it so the button is the default. */
    GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);

    /* This grabs this button to be the default button. Simply hitting
     * the "Enter" key will cause this button to activate. */
    gtk_widget_grab_default (button);
    gtk_widget_show(button);

    gtk_widget_show (pdata->window);

    gtk_main ();

    return(0);
}
/* example-end */

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc progressbar.c -o
progressbar $(pkg-config gtk+-2.0 --cflags --libs)

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./progressbar
```

Cuadros de diálogo

El widget de diálogo es muy simple, y es en realidad sólo una ventana con algunas cosas ya preempaquetadas para usted. La estructura de un cuadro de diálogo es:

```
struct GtkDialog
{
      GtkWindow window;

      GtkWidget *vbox;
      GtkWidget *action_area;
};
```

Así que ya ves, simplemente crea una ventana, y se empaqueta una vbox, que contiene un separador y una hbox llamada "action_area". El widget de diálogo se puede utilizar para mensajes emergentes para el usuario, y otras tareas similares. Es realmente básico, y sólo hay una función para el cuadro de diálogo, que es:

`GtkWidget *gtk_dialog_new( void );` Así que para crear un nuevo cuadro de diálogo, utilice,

```
GtkWidget *window;
window = gtk_dialog_new ();
```

Esto creará el cuadro de diálogo, y ahora depende de usted para utilizarlo. Usted podría empaquetar un botón en el área de acción al hacer algo como esto:

```
button = ...
gtk_box_pack_start (GTK_BOX (GTK_DIALOG (window)->action_area),
                    button, TRUE, TRUE, 0);
gtk_widget_show (button);
```

Y se podría agregar al área vbox por el embalaje, por ejemplo, una etiqueta en él, pruebe algo como esto:

```
label = gtk_label_new ("Dialogs are groovy");
gtk_box_pack_start (GTK_BOX (GTK_DIALOG (window)->vbox),
                    label, TRUE, TRUE, 0);
gtk_widget_show (label);
```

A modo de ejemplo, en el cuadro de diálogo, se puede poner dos botones en el área de acción, un botón Cancelar y un botón Aceptar, y una etiqueta en la vbox, pidiendo al usuario una pregunta o dar un error, etc A continuación, puede adjuntar una señal diferente a cada uno de los botones y realizar la operación que el usuario selecciona. Si la simple funcionalidad proporcionada por las cajas por defecto verticales y horizontales en las dos áreas no le da suficiente control para su aplicación, entonces usted puede simplemente empacar otro widget diseño en las casillas correspondientes. Por ejemplo, usted puede empacar una tabla en la caja vertical.

pixmaps

Pixmaps son estructuras de datos que contienen dibujos. Estas imágenes se pueden utilizar en varios lugares, pero más comúnmente como iconos en el escritorio de X, o como cursores. Un pixmap que sólo tiene dos colores se llama un mapa de bits, y hay unas cuantas rutinas adicionales para el manejo de este caso especial común. Para entender pixmaps, ayudaría a entender cómo funciona el sistema de ventanas X funciona. En X, las aplicaciones no es necesario que se ejecuta en el mismo equipo que está interactuando con el usuario. En su lugar, las diversas aplicaciones, llamados "clientes", se comunican con un programa que muestra los gráficos y maneja el teclado y el ratón. Este programa, que interactúa directamente con el usuario que se llama un "servidor de pantalla" o "servidor X". Dado que la comunicación puede tener lugar a través de una red, es importante tener alguna información con el servidor X. Mapas de píxeles, por ejemplo, se almacenan en la memoria del servidor de X. Esto significa que una vez que los valores pixmap se establecen, no hay que seguir transmitida a través de la red; ». Mostrar pixmap número XYZ aquí" y no se envía un comando a Incluso si usted no está usando X con GTK actualmente, usando construcciones como Pixmaps hará que sus programas funcionan aceptablemente bajo X. Para usar pixmaps en GTK, primero tenemos que construir una estructura GdkPix-

map utilizando rutinas de GDK. Pixmaps se pueden crear a partir de datos in-memory, o de la lectura de datos desde un archivo. Vamos a ir a través de cada una de las llamadas para crear un pixmap.

```
GdkPixmap *gdk_bitmap_create_from_data( GdkWindow *window,
                                        gchar     *data,
                                        gint       width,
                                        gint       height );
```

Esta rutina se utiliza para crear un mapa de píxeles de un solo plano (2 colores) de datos en la memoria. Cada bit de los datos representa si el píxel está apagado o encendido. Anchura y altura en píxeles. El puntero GdkWindow es la ventana actual, ya que los recursos de un pixmap sólo tienen sentido en el contexto de la pantalla donde se va a mostrar.

```
GdkPixmap *gdk_pixmap_create_from_data( GdkWindow *window,
                                        gchar     *data,
                                        gint       width,
                                        gint       height,
                                        gint       depth,
                                        GdkColor  *fg,
                                        GdkColor  *bg );
```

Esto se utiliza para crear un mapa de pixels de la profundidad (número de colores) de los datos de mapa de bits especificado. `fg` y `bg` son el primer plano y el color de fondo a utilizar.

```
GdkPixmap *gdk_pixmap_create_from_xpm( GdkWindow   *window,
                                       GdkBitmap  **mask,
                                       GdkColor    *transparent_color,
                                       const gchar *filename );
```

Formato XPM es una representación legible de pixmap para el sistema X Window. Es ampliamente utilizado y existen muchos programas están disponibles para la creación de archivos de imagen en este formato. El archivo especificado por filename debe contener una imagen en ese formato y se carga en la estructura pixmap. La máscara especifica qué bits de la son opacos. Todos los demás bits se colorean con el color especificado por transparent_color. Un ejemplo de usar este sigue a continuación.

```
GdkPixmap *gdk_pixmap_create_from_xpm_d( GdkWindow  *window,
                                         GdkBitmap **mask,
                                         GdkColor   *transparent_color,
                                         gchar     **data );
```

Las imágenes pequeñas se pueden incorporar en un programa como datos en el formato XPM. Un pixmap se crea con estos datos, en lugar de leer desde un archivo. Un ejemplo de tales datos es

```
/* XPM */
static const char * xpm_data[] = {
"16 16 3 1",
```

```
"       c None",
".      c #000000000000",
"X      c #FFFFFFFFFFFF",
"                ",
"   ......       ",
"   .XXX.X.      ",
"   .XXX.XX.     ",
"   .XXX.XXX.    ",
"   .XXX.....    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .........    ",
"                ",
"                "};
```

Cuando haya terminado de usar un pixmap y sin posibilidades de volver a utilizarlo de nuevo pronto, es una buena idea para liberar el recurso mediante gdk_pixmap_unref (). Pixmaps debe considerarse como un recurso precioso, ya que ocupan memoria en el proceso del usuario final X servidor. A pesar de que el cliente de X se escribe se puede ejecutar en un poderoso "servidor" equipo, el usuario puede ejecutar el servidor X en un ordenador personal pequeño. Una vez que hemos creado un mapa de píxeles, lo podemos visualizar como un widget GTK. Debemos crear un pixmap en GTK widget a contener el pixmap GDK. Esto se hace utilizando

```
GtkWidget *gtk_pixmap_new( GdkPixmap *pixmap,
                           GdkBitmap *mask );
```

Los otros pixmap llamadas widgets son

```
guint gtk_pixmap_get_type( void );

void  gtk_pixmap_set( GtkPixmap  *pixmap,
                      GdkPixmap  *val,
                      GdkBitmap  *mask );

void  gtk_pixmap_get( GtkPixmap  *pixmap,
                      GdkPixmap **val,
                      GdkBitmap **mask);
```

La función gtk_pixmap_set se utiliza para cambiar el mapa de píxeles que el widget está gestionando. Val es el pixmap GDK creado usando. El siguiente es un ejemplo del uso de un mapa de píxeles en un botón.

```
/* example-start pixmap pixmap.c */

#include <gtk/gtk.h>


/* XPM data of Open-File icon */
static const char * xpm_data[] = {
```

```
"16 16 3 1",
"       c None",
".      c #000000000000",
"X      c #FFFFFFFFFFFF",
"                ",
"   ......       ",
"   .XXX.X.      ",
"   .XXX.XX.     ",
"   .XXX.XXX.    ",
"   .XXX.....    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .........    ",
"                ",
"                "};


/* when invoked (via signal delete_event), terminates the application.
 */
gint close_application( GtkWidget *widget,
                        GdkEvent  *event,
                        gpointer   data )
{
    gtk_main_quit();
    return(FALSE);
}


/* is invoked when the button is clicked.  It just prints a message.
 */
void button_clicked( GtkWidget *widget,
                     gpointer   data ) {
    g_print( "button clicked\n" );
}

int main( int   argc,
          char *argv[] )
{
    /* GtkWidget is the storage type for widgets */
    GtkWidget *window, *pixmapwid, *button;
    GdkPixmap *pixmap;
    GdkBitmap *mask;
    GtkStyle *style;

    /* create the main window, and attach delete_event signal to terminating
       the application */
    gtk_init( &argc, &argv );
    window = gtk_window_new( GTK_WINDOW_TOPLEVEL );
    gtk_signal_connect( GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (close_application), NULL );
    gtk_container_set_border_width( GTK_CONTAINER (window), 10 );
    gtk_widget_show( window );
```

```
    /* now for the pixmap from gdk */
    style = gtk_widget_get_style( window );
    pixmap = gdk_pixmap_create_from_xpm_d( window->window,  &mask,
                                           &style->bg[GTK_STATE_NORMAL],
                                           (gchar **)xpm_data );

    /* a pixmap widget to contain the pixmap */
    pixmapwid = gtk_pixmap_new( pixmap, mask );
    gtk_widget_show( pixmapwid );

    /* a button to contain the pixmap widget */
    button = gtk_button_new();
    gtk_container_add( GTK_CONTAINER(button), pixmapwid );
    gtk_container_add( GTK_CONTAINER(window), button );
    gtk_widget_show( button );

    gtk_signal_connect( GTK_OBJECT(button), "clicked",
                        GTK_SIGNAL_FUNC(button_clicked), NULL );

    /* show the window */
    gtk_main ();

    return 0;
}
/* example-end */

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc pixmap.c -o pixmap $(pkg-
config gtk+-2.0 --cflags –libs)
linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./pixmap
```

Para cargar un archivo desde un archivo de datos llamado XPM icon0.xpm en el directorio actual, se habría creado el pixmap así

```
    /* load a pixmap from a file */
    pixmap = gdk_pixmap_create_from_xpm( window->window, &mask,
                                         &style->bg[GTK_STATE_NORMAL],
                                         "./icon0.xpm" );
    pixmapwid = gtk_pixmap_new( pixmap, mask );
    gtk_widget_show( pixmapwid );
    gtk_container_add( GTK_CONTAINER(window), pixmapwid );
```

Una desventaja de los pixmaps es que el objeto mostrado siempre es rectangular, independientemente de la imagen. Nos gustaría crear escritorios y aplicaciones con iconos que tienen formas más naturales. Por ejemplo, para una interfaz de juego, nos gustaría tener botones redondos para empujar. La manera de hacer esto es usar ventanas en forma. Una ventana en forma es simplemente un pixmap donde los píxeles de fondo son transparentes. De esta manera, cuando la imagen de fondo es de varios colores, no sobreescribirlo con una rectangular y que no encaja frontera de nuestro icono. El siguiente ejemplo muestra una imagen carretilla llena en el escritorio.

```
/* example-start wheelbarrow wheelbarrow.c */

#include <gtk/gtk.h>

/* XPM */
static char * WheelbarrowFull_xpm[] = {
"48 48 64 1",
"       c None",
".      c #DF7DCF3CC71B",
"X      c #965875D669A6",
"o      c #71C671C671C6",
"O      c #A699A289A699",
"+      c #965892489658",
"@      c #8E38410330C2",
"#      c #D75C7DF769A6",
"$      c #F7DECF3CC71B",
"%      c #96588A288E38",
"&      c #A69992489E79",
"*      c #8E3886178E38",
"=      c #104008200820",
"-      c #596510401040",
";      c #C71B30C230C2",
":      c #C71B9A699658",
">      c #618561856185",
",      c #20811C712081",
"<      c #104000000000",
"1      c #861720812081",
"2      c #DF7D4D344103",
"3      c #79E769A671C6",
"4      c #861782078617",
"5      c #41033CF34103",
"6      c #000000000000",
"7      c #49241C711040",
"8      c #492445144924",
"9      c #082008200820",
"0      c #69A618611861",
"q      c #B6DA71C65144",
"w      c #410330C238E3",
"e      c #CF3CBAEAB6DA",
"r      c #71C6451430C2",
"t      c #EFBEDB6CD75C",
"y      c #28A208200820",
"u      c #186110401040",
"i      c #596528A21861",
"p      c #71C661855965",
"a      c #A69996589658",
"s      c #30C228A230C2",
"d      c #BEFBA289AEBA",
"f      c #596545145144",
"g      c #30C230C230C2",
"h      c #8E3882078617",
"j      c #208118612081",
"k      c #38E30C300820",
"l      c #30C2208128A2",
"z      c #38E328A238E3",
"x      c #514438E34924",
"c      c #618555555965",
"v      c #30C2208130C2",
```

```
"b      c #38E328A230C2",
"n      c #28A228A228A2",
"m      c #41032CB228A2",
"M      c #104010401040",
"N      c #492438E34103",
"B      c #28A2208128A2",
"V      c #A699596538E3",
"C      c #30C21C711040",
"Z      c #30C218611040",
"A      c #965865955965",
"S      c #618534D32081",
"D      c #38E31C711040",
"F      c #082000000820",
"                                          ",
"          .XoO                            ",
"         +@#$%o&                          ",
"         *=-;#::o+                        ",
"           >,<12#:34                      ",
"             45671#:X3                    ",
"               +89<02qwo                  ",
"e*                >,67;ro                 ",
"ty>                 459@>+&&              ",
"$2u+                  ><ipas8*            ",
"%$;=*                *3:.Xa.dfg>           ",
"Oh$;ya              *3d.a8j,Xe.d3g8+        ",
" Oh$;ka           *3d$a8lz,,xxc:.e3g54      ",
"  Oh$;kO        *pd$%svbzz,sxxxxfX..&wn>    ",
"   Oh$@mO    *3dthwlsslszjzxxxxxxx3:td8M4   ",
"    Oh$@g& *3d$XNlvvvlllm,mNwxxxxxxxfa.:,B* ",
"     Oh$@,Od.czllllllzlmmqV@V#V@fxxxxxxxf:%j5& ",
"      Oh$1hd5lllslllCCZrV#r#:#2AxxxxxxxxxcdwM* ",
"       OXq6c.%8vvvllZZiqqApA:mq:Xxcpcxxxxxfdc9* ",
"        2r<6gde3bllZZrVi7S@SV77A::qApxxxxxxfdcM ",
"        :,q-6MN.dfmZZrrSS:#riirDSAX@Af5xxxxxfevo",
"         +A26jguXtAZZZC7iDiCCrVVii7Cmmmxxxxxx%3g",
"          *#16jszN..3DZZZZrCVSA2rZrV7Dmmwxxxx&en",
"           p2yFvzssXe:fCZZCiiD7iiZDiDSSZwwxx8e*>",
"           OA1<jzxwwc:$d%NDZZZZCCCZCCZZCmxxfd.B ",
"            3206Bwxxszx%et.eaAp77m77mmmf3&eeeg* ",
"             @26MvzxNzvlbwfpdettttttttttt.c,n&  ",
"             *;16=lsNwwNwgsvslbwwvccc3pcfu<o    ",
"              p;<69BvwwsszslllbBlllllllu<5+     ",
"              OS0y6FBlvvvzvzss,u=Blllj=54       ",
"               c1-699Blvllllllu7k96MMMg4         ",
"               *10y8n6Fjvllll8<166668           ",
"                S-kg+>666<M<996-y6n<8*          ",
"                p71=4 m69996kD8Z-66698&&        ",
"                &i0ycm6n4 ogk17,0<6666g         ",
"                 N-k-<>     >=01-kuu666>        ",
"                 ,6ky&      &46-10ul,66,        ",
"                 Ou0<>       o66y<ulw<66&       ",
"                  *kk5       >66By7=xu664       ",
"                   <<M4      466lj<Mxu66o       ",
"                   *>>       +66uv,zN666*       ",
"                             566,xxj669         ",
"                             4666FF666>         ",
"                              >966666M          ",
"                                oM6668+         ",
```

```
"                                   *4            ",
"                                                 ",
"                                                 "};


/* When invoked (via signal delete_event), terminates the application */
gint close_application( GtkWidget *widget,
                        GdkEvent  *event,
                        gpointer   data )
{
    gtk_main_quit();
    return(FALSE);
}

int main (int argc,
          char *argv[] )
{
    /* GtkWidget is the storage type for widgets */
    GtkWidget *window, *pixmap, *fixed;
    GdkPixmap *gdk_pixmap;
    GdkBitmap *mask;
    GtkStyle *style;
    GdkGC *gc;

    /* Create the main window, and attach delete_event signal to terminate
     * the application.  Note that the main window will not have a titlebar
     * since we're making it a popup. */
    gtk_init (&argc, &argv);
    window = gtk_window_new( GTK_WINDOW_POPUP );
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (close_application), NULL);
    gtk_widget_show (window);

    /* Now for the pixmap and the pixmap widget */
    style = gtk_widget_get_default_style();
    gc = style->black_gc;
    gdk_pixmap = gdk_pixmap_create_from_xpm_d( window->window, &mask,
                                               &style->bg[GTK_STATE_NORMAL],
                                               WheelbarrowFull_xpm );
    pixmap = gtk_pixmap_new( gdk_pixmap, mask );
    gtk_widget_show( pixmap );

    /* To display the pixmap, we use a fixed widget to place the pixmap */
    fixed = gtk_fixed_new();
    gtk_widget_set_usize( fixed, 200, 200 );
    gtk_fixed_put( GTK_FIXED(fixed), pixmap, 0, 0 );
    gtk_container_add( GTK_CONTAINER(window), fixed );
    gtk_widget_show( fixed );

    /* This masks out everything except for the image itself */
    gtk_widget_shape_combine_mask( window, mask, 0, 0 );

    /* show the window */
    gtk_widget_set_uposition( window, 20, 400 );
    gtk_widget_show( window );
    gtk_main ();

    return(0);
```

```
}
/* example-end */

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc wheelbarrow.c -o wheel-
barrow $(pkg-config gtk+-2.0 --cflags --libs)

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./wheelbarrow
```

Para hacer que la imagen sensible, podríamos fijar la pulsación del botón para que haga algo. Con las líneas siguientes se hacen el dibujo sensible a un botón del ratón se presiona lo que hace que la aplicación termine.

```
    gtk_widget_set_events( window,
                           gtk_widget_get_events( window ) |
                           GDK_BUTTON_PRESS_MASK );

   gtk_signal_connect( GTK_OBJECT(window), "button_press_event",
                       GTK_SIGNAL_FUNC(close_application), NULL );
```

Reglas

Las reglas son usadas para indicar la ubicación del puntero del ratón en una ventana dada. Una ventana puede tener una regla vertical que abarca todo el ancho y una regla horizontal a lo largo de la altura. Un pequeño indicador triangular en la regla muestra la ubicación exacta del puntero en relación con la regla.

Un gobernante debe ser creado. Las reglas horizontales y verticales se crean con

```
GtkWidget *gtk_hruler_new( void );    /* horizontal ruler */


GtkWidget *gtk_vruler_new( void );    /* vertical ruler   */
```

Una vez que un regla se crea, podemos definir la unidad de medida. Las unidades de medida para las reglas beGTK_PIXELS puede, GTK_INCHES o GTK_CENTIMETERS.
Esto se ajusta con

```
void gtk_ruler_set_metric( GtkRuler      *ruler,
                           GtkMetricType  metric );
```

La medida predeterminada es GTK_PIXELS.

```
gtk_ruler_set_metric( GTK_RULER(ruler), GTK_PIXELS );
```

Otras características importantes de un regla es cómo marca las unidades de escala y donde el indicador de posición se coloca inicialmente. Esto se fija la regla con

```
void gtk_ruler_set_range( GtkRuler *ruler,
                          gfloat    lower,
                          gfloat    upper,
                          gfloat    position,
                          gfloat    max_size );
```

Los argumentos inferior y superior de definir el alcance de la regla, y max_size es el número más alto que se mostrará. Posición define la posición inicial del indicador dentro de la regla.

Una regla vertical puede medir una ventana de 800 píxeles de ancho por lo tanto

```
gtk_ruler_set_range( GTK_RULER(vruler), 0, 800, 0, 800);
```

Las marcas que aparecen en la regla será de 0 a 800, con una periodicidad de 100 píxeles. Si por el contrario queremos que el gobernante en un rango de 7 a 16, que codificaría

```
gtk_ruler_set_range( GTK_RULER(vruler), 7, 16, 0, 20);
```

El indicador de la regla es un pequeño triángulo que indica la posición del puntero en relación con la regla. Si la regla se utiliza para seguir el puntero del ratón, la señal motion_notify_event debe estar conectado a la motion_notify_event de la regla. Para seguir todos los movimientos del ratón en un área de la ventana, usaríamos

```
#define EVENT_METHOD(i, x) GTK_WIDGET_CLASS(GTK_OBJECT(i)->klass)->x

    gtk_signal_connect_object( GTK_OBJECT(area), "motion_notify_event",
           (GtkSignalFunc)EVENT_METHOD(ruler, motion_notify_event),
           GTK_OBJECT(ruler) );
```

En el ejemplo siguiente, se crea un área de dibujo con una regla horizontal por encima de ella y una regla vertical a la izquierda de la misma. El tamaño del área de dibujo es de 600 pixeles de ancho por 400 píxeles de alto. La regla horizontal oscila entre 7 y 13 con una marca cada 100 píxeles, mientras que el vertical va desde 0 a 400 con una marca cada 100 píxeles. La ubicación del área de dibujo y las reglas se realiza mediante una tabla.

```
/* example-start rulers rulers.c */

#include <gtk/gtk.h>

#define EVENT_METHOD(i, x) GTK_WIDGET_CLASS(GTK_OBJECT(i)->klass)->x

#define XSIZE  600
#define YSIZE  400

/* This routine gets control when the close button is clicked */
```

```
gint close_application( GtkWidget *widget,
                        GdkEvent  *event,
                        gpointer   data )
{
    gtk_main_quit();
    return(FALSE);
}

/* The main routine */
int main( int   argc,
          char *argv[] ) {
    GtkWidget *window, *table, *area, *hrule, *vrule;

    /* Initialize GTK and create the main window */
    gtk_init( &argc, &argv );

    window = gtk_window_new( GTK_WINDOW_TOPLEVEL );
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
            GTK_SIGNAL_FUNC( close_application ), NULL);
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    /* Create a table for placing the ruler and the drawing area */
    table = gtk_table_new( 3, 2, FALSE );
    gtk_container_add( GTK_CONTAINER(window), table );

    area = gtk_drawing_area_new();
    gtk_drawing_area_size( (GtkDrawingArea *)area, XSIZE, YSIZE );
    gtk_table_attach( GTK_TABLE(table), area, 1, 2, 1, 2,
                      GTK_EXPAND|GTK_FILL, GTK_FILL, 0, 0 );
    gtk_widget_set_events( area, GDK_POINTER_MOTION_MASK |
                                 GDK_POINTER_MOTION_HINT_MASK );

    /* The horizontal ruler goes on top. As the mouse moves across the
     * drawing area, a motion_notify_event is passed to the
     * appropriate event handler for the ruler. */
    hrule = gtk_hruler_new();
    gtk_ruler_set_metric( GTK_RULER(hrule), GTK_PIXELS );
    gtk_ruler_set_range( GTK_RULER(hrule), 7, 13, 0, 20 );
    gtk_signal_connect_object( GTK_OBJECT(area), "motion_notify_event",
                               (GtkSignalFunc)EVENT_METHOD(hrule,
                                                           motion_notify_event),
                               GTK_OBJECT(hrule) );
    /*  GTK_WIDGET_CLASS(GTK_OBJECT(hrule)->klass)->motion_notify_event, */
    gtk_table_attach( GTK_TABLE(table), hrule, 1, 2, 0, 1,
                      GTK_EXPAND|GTK_SHRINK|GTK_FILL, GTK_FILL, 0, 0 );

    /* The vertical ruler goes on the left. As the mouse moves across
     * the drawing area, a motion_notify_event is passed to the
     * appropriate event handler for the ruler. */
    vrule = gtk_vruler_new();
    gtk_ruler_set_metric( GTK_RULER(vrule), GTK_PIXELS );
    gtk_ruler_set_range( GTK_RULER(vrule), 0, YSIZE, 10, YSIZE );
    gtk_signal_connect_object( GTK_OBJECT(area), "motion_notify_event",
                               (GtkSignalFunc)
                                  GTK_WIDGET_CLASS(GTK_OBJECT(vrule)->klass)-
>
                                                           motion_notify_event,
                               GTK_OBJECT(vrule) );
```

```
    gtk_table_attach( GTK_TABLE(table), vrule, 0, 1, 1, 2,
                      GTK_FILL, GTK_EXPAND|GTK_SHRINK|GTK_FILL, 0, 0 );

    /* Now show everything */
    gtk_widget_show( area );
    gtk_widget_show( hrule );
    gtk_widget_show( vrule );
    gtk_widget_show( table );
    gtk_widget_show( window );
    gtk_main();

    return(0);
}
```

/* example-end */

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc rulers.c -o rulers $(pkg-config gtk+-2.0 --cflags –libs)
rulers.c: In function ‘main’:

rulers.c:49: error: ‘GtkObject’ has no member named ‘klass’

rulers.c:63: error: ‘GtkObject’ has no member named ‘klass’

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./rulers

Nota: en correccion :(

Barras de estado

Estado son widgets usados para mostrar un mensaje de texto. Mantienen una pila de los mensajes inserta en ellos, por lo que al quitar el mensaje actual se volverá a mostrar el mensaje de texto anterior.

A fin de que las diferentes partes de una aplicación para utilizar la misma barra de estado para mostrar mensajes, las cuestiones statusbar widget de identificadores de contexto que se utilizan para identificar diferentes "usuarios". El mensaje en la parte superior de la pila es la que se muestra, no importa qué contexto es pulg Los mensajes se apilan en último en entrar, primero en salir el fin, no el fin identificador de contexto.

Una barra de estado se crea con una llamada a:

```
GtkWidget *gtk_statusbar_new( void );
```

Un identificador de contexto nuevo se solicita mediante una llamada a la función siguiente con una pequeña descripción textual del contexto:

```
guint gtk_statusbar_get_context_id( GtkStatusbar *statusbar,
                                    const gchar  *context_description );
```

Hay tres funciones que pueden operar el statusbar:

```
guint gtk_statusbar_push( GtkStatusbar *statusbar,
                          guint         context_id,
                          gchar        *text );

void gtk_statusbar_pop( GtkStatusbar *statusbar)
                        guint         context_id );

void gtk_statusbar_remove( GtkStatusbar *statusbar,
                           guint         context_id,
                           guint         message_id );
```

El gtk_statusbar_push primera, se utiliza para añadir un nuevo mensaje a la barra de estado. Devuelve un identificador de mensaje, que se puede pasar más tarde a la gtk_statusbar_remove función para eliminar el mensaje con el mensaje dado y los identificadores de contexto de la barra de estado de la pila.

La función gtk_statusbar_pop elimina el mensaje más alto en la pila con el identificador de contexto dado.

El ejemplo siguiente crea una barra de estado y dos botones, uno para meter un elemento en la barra de estado, y uno para sacar el último elemento de marcha atrás.

```
/* example-start statusbar statusbar.c */

#include <gtk/gtk.h>
#include <glib.h>

GtkWidget *status_bar;

void push_item( GtkWidget *widget,
                gpointer   data )
{
  static int count = 1;
  char buff[20];

  g_snprintf(buff, 20, "Item %d", count++);
  gtk_statusbar_push( GTK_STATUSBAR(status_bar), GPOINTER_TO_INT(data),
buff);

  return;
}

void pop_item( GtkWidget *widget,
               gpointer   data )
{
  gtk_statusbar_pop( GTK_STATUSBAR(status_bar), GPOINTER_TO_INT(data) );
  return;
```

```
}

int main( int   argc,
          char *argv[] )
{

    GtkWidget *window;
    GtkWidget *vbox;
    GtkWidget *button;

    gint context_id;

    gtk_init (&argc, &argv);

    /* create a new window */
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_widget_set_usize( GTK_WIDGET (window), 200, 100);
    gtk_window_set_title(GTK_WINDOW (window), "GTK Statusbar Example");
    gtk_signal_connect(GTK_OBJECT (window), "delete_event",
                       (GtkSignalFunc) gtk_exit, NULL);

    vbox = gtk_vbox_new(FALSE, 1);
    gtk_container_add(GTK_CONTAINER(window), vbox);
    gtk_widget_show(vbox);

    status_bar = gtk_statusbar_new();
    gtk_box_pack_start (GTK_BOX (vbox), status_bar, TRUE, TRUE, 0);
    gtk_widget_show (status_bar);

    context_id = gtk_statusbar_get_context_id(
                         GTK_STATUSBAR(status_bar), "Statusbar example");

    button = gtk_button_new_with_label("push item");
    gtk_signal_connect(GTK_OBJECT(button), "clicked",
        GTK_SIGNAL_FUNC (push_item), GINT_TO_POINTER(context_id) );
    gtk_box_pack_start(GTK_BOX(vbox), button, TRUE, TRUE, 2);
    gtk_widget_show(button);

    button = gtk_button_new_with_label("pop last item");
    gtk_signal_connect(GTK_OBJECT(button), "clicked",
        GTK_SIGNAL_FUNC (pop_item), GINT_TO_POINTER(context_id) );
    gtk_box_pack_start(GTK_BOX(vbox), button, TRUE, TRUE, 2);
    gtk_widget_show(button);

    /* always display the window as the last step so it all splashes on
     * the screen at once. */
    gtk_widget_show(window);

    gtk_main ();

    return 0;
}
/* example-end */

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc statusbar.c -o statusbar
$(pkg-config gtk+-2.0 --cflags --libs)
```

```
linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./statusbar
```

comentarios de texto

El widget Entry permite escribir texto y se muestra en un cuadro de texto de una sola línea. El texto puede ser ajustado con llamadas a funciones que permiten un nuevo texto para sustituir, anteponer o añadir los contenidos actuales del control Entry.

Existen dos funciones para crear widgets de entrada:

```
void gtk_entry_set_text( GtkEntry    *entry,
                         const gchar *text );

void gtk_entry_append_text( GtkEntry    *entry,
                            const gchar *text );

void gtk_entry_prepend_text( GtkEntry    *entry,
                             const gchar *text );
```

La primera sirve para crear un widget nueva entrada, mientras que el segundo crea una nueva entrada y establece un límite en la longitud del texto dentro de la entrada.

Hay varias funciones para modificar el texto que se encuentra actualmente en el widget.

El gtk_entry_set_text función establece el contenido del widget Entry, reemplazando el contenido actual. Las funciones gtk_entry_append_text gtk_entry_prepend_text y permitir el contenido actual que se preañadir.

La siguiente función permite que el punto de inserción actual que desea ajustar.

```
void gtk_entry_set_position( GtkEntry *entry,
                             gint      position );
```

El contenido de la entrada se puede recuperar mediante una llamada a la función siguiente. Esto es útil en las funciones de devolución de llamada se describen a continuación.

```
gchar *gtk_entry_get_text( GtkEntry *entry );
```

El valor devuelto por esta función es de uso interno y no debe ser liberado usando free () o g_free ()

Si no queremos que el contenido de la entrada para ser cambiado por alguien escribiendo en él, podemos cambiar su estado de edición.

```
void gtk_entry_set_editable( GtkEntry *entry,
                             gboolean  editable );
```

La función anterior nos permite cambiar el estado de edición del control de entrada pasando un valor true o false para el argumento editable.

Si utilizamos la entrada donde no queremos que el texto escrito sea visible, por ejemplo cuando una contraseña se introduce, se puede utilizar la siguiente función, que también tiene una bandera booleana.

```
void gtk_entry_set_visibility( GtkEntry *entry,
                               gboolean  visible );
```

Una región del texto se puede poner como seleccionada usando la siguiente función. Esta función se puede utilizar después de establecer un texto predeterminado en una entrada, lo que hace que sea fácil para el usuario para eliminarlo.

void gtk_entry_select_region(GtkEntry *entry,

```
                              gint      start,
                              gint      end );
```

Si queremos saber el momento en que el usuario introduzca texto, puede conectarse a la señal de activación o cambiado. Activate se produce cuando el usuario pulsa la tecla enter en el control Entry. Cambiado se produce cuando el texto cambia en absoluto, por ejemplo, para cada carácter introducido o retirado.

El código siguiente es un ejemplo del uso de un control Entry.

```
/* example-start entry entry.c */

#include <stdio.h>
#include <gtk/gtk.h>

void enter_callback( GtkWidget *widget,
                     GtkWidget *entry )
{
  gchar *entry_text;
  entry_text = gtk_entry_get_text(GTK_ENTRY(entry));
  printf("Entry contents: %s\n", entry_text);
```

```
}

void entry_toggle_editable( GtkWidget *checkbutton,
                            GtkWidget *entry )
{
  gtk_entry_set_editable(GTK_ENTRY(entry),
                         GTK_TOGGLE_BUTTON(checkbutton)->active);
}

void entry_toggle_visibility( GtkWidget *checkbutton,
                              GtkWidget *entry )
{
  gtk_entry_set_visibility(GTK_ENTRY(entry),
                         GTK_TOGGLE_BUTTON(checkbutton)->active);
}

int main( int   argc,
          char *argv[] )
{

    GtkWidget *window;
    GtkWidget *vbox, *hbox;
    GtkWidget *entry;
    GtkWidget *button;
    GtkWidget *check;

    gtk_init (&argc, &argv);

    /* create a new window */
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_widget_set_usize( GTK_WIDGET (window), 200, 100);
    gtk_window_set_title(GTK_WINDOW (window), "GTK Entry");
    gtk_signal_connect(GTK_OBJECT (window), "delete_event",
                       (GtkSignalFunc) gtk_exit, NULL);

    vbox = gtk_vbox_new (FALSE, 0);
    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show (vbox);

    entry = gtk_entry_new_with_max_length (50);
    gtk_signal_connect(GTK_OBJECT(entry), "activate",
                       GTK_SIGNAL_FUNC(enter_callback),
                       entry);
    gtk_entry_set_text (GTK_ENTRY (entry), "hello");
    gtk_entry_append_text (GTK_ENTRY (entry), " world");
    gtk_entry_select_region (GTK_ENTRY (entry),
                             0, GTK_ENTRY(entry)->text_length);
    gtk_box_pack_start (GTK_BOX (vbox), entry, TRUE, TRUE, 0);
    gtk_widget_show (entry);

    hbox = gtk_hbox_new (FALSE, 0);
    gtk_container_add (GTK_CONTAINER (vbox), hbox);
    gtk_widget_show (hbox);

    check = gtk_check_button_new_with_label("Editable");
    gtk_box_pack_start (GTK_BOX (hbox), check, TRUE, TRUE, 0);
    gtk_signal_connect (GTK_OBJECT(check), "toggled",
                        GTK_SIGNAL_FUNC(entry_toggle_editable), entry);
```

```
    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(check), TRUE);
    gtk_widget_show (check);

    check = gtk_check_button_new_with_label("Visible");
    gtk_box_pack_start (GTK_BOX (hbox), check, TRUE, TRUE, 0);
    gtk_signal_connect (GTK_OBJECT(check), "toggled",
                        GTK_SIGNAL_FUNC(entry_toggle_visibility), entry);
    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(check), TRUE);
    gtk_widget_show (check);

    button = gtk_button_new_with_label ("Close");
    gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                               GTK_SIGNAL_FUNC(gtk_exit),
                               GTK_OBJECT (window));
    gtk_box_pack_start (GTK_BOX (vbox), button, TRUE, TRUE, 0);
    GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);
    gtk_widget_grab_default (button);
    gtk_widget_show (button);

    gtk_widget_show(window);

    gtk_main();
    return(0);
}
/* example-end */

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc entry.c -o entry $(pkg-
config gtk+-2.0 --cflags –libs)
entry.c: In function ‘enter_callback’:

entry.c:10: warning: assignment discards qualifiers from pointer target type


Nota: :( aun tiene error...
```

Botones Aumentar

Los widget botón de giro se utiliza generalmente para permitir al usuario seleccionar un valor de un rango de valores numéricos. Consiste en un cuadro de entrada de texto con los botones de flecha hacia arriba y hacia abajo pegadas en los lados. Al seleccionar uno de los botones hace que el valor a "girar" hacia arriba y hacia abajo el rango de valores posibles. El cuadro de entrada también puede ser editado directamente para introducir un valor específico.

El botón de giro permite que el valor de tener cero o un número de cifras decimales y se incrementa / decrementa en pasos configurables. La acción de mantener pulsado uno de los botones opcionalmente provoca una aceleración del cambio en el valor de acuerdo con el tiempo que está deprimido.

El botón de giro utiliza un objeto de ajuste para contener información acerca de la gama de

valores que el botón de la vuelta puede tomar. Esto lo convierte en un widget de giro del botón de gran alcance.

Recordemos que un widget de ajuste se crea con la siguiente función, que ilustra la información que se tiene:

```
GtkObject *gtk_adjustment_new( gfloat value,
                               gfloat lower,
                               gfloat upper,
                               gfloat step_increment,
                               gfloat page_increment,
                               gfloat page_size );
```

Estos atributos de un ajuste se utilizan por el botón de giro de la siguiente manera:

Estos atributos de un ajuste se utilizan por el botón de giro de la siguiente manera:

- valor: valor inicial del botón de la vuelta
- inferior: valor inferior del rango
- superior: superior rango de valores
- step_increment: valor para aumentar / disminuir cuando se pulsa el botón 1 del ratón sobre un botón
- page_increment: valor para aumentar / disminuir cuando se pulsa el botón 2 del ratón sobre un botón
- page_size: unused

Además, el botón 3 se puede utilizar para saltar directamente a los valores superiores o inferiores cuando se utiliza para seleccionar uno de los botones. Veamos cómo crear un botón de Spin:

```
GtkWidget *gtk_spin_button_new( GtkAdjustment *adjustment,
                                gfloat         climb_rate,
                                guint          digits );
```

El argumento climb_rate tomar un valor entre 0,0 y 1,0 e indica la cantidad de aceleración que tiene el botón de girar. El argumento digits especifica el número de posiciones decimales en el que se muestran el valor.

Un botón de la vuelta puede ser reconfigurado después de la creación mediante la siguiente función:

```
void gtk_spin_button_configure( GtkSpinButton *spin_button,
                                GtkAdjustment *adjustment,
```

```
                                gfloat         climb_rate,
                                guint          digits );
```

El argumento spin_button especifica el widget de botón de número que se va a reconfigurar. Los otros argumentos son los especificados anteriormente.

El ajuste se puede establecer y recuperar forma independiente usando las dos funciones siguientes:

```
void gtk_spin_button_set_adjustment( GtkSpinButton  *spin_button,
                                     GtkAdjustment  *adjustment );
```

El número de decimales también se puede modificar mediante:

void gtk_spin_button_set_digits(GtkSpinButton *spin_button,

guint digits) ;

El valor que un botón de la vuelta está mostrando actualmente se puede cambiar mediante la siguiente función:

void gtk_spin_button_set_value(GtkSpinButton *spin_button,

gfloat value);

El valor actual de un botón de giro se puede recuperar como un punto flotante o un valor entero con las siguientes funciones:

gfloat gtk_spin_button_get_value_as_float(GtkSpinButton *spin_button);

gint gtk_spin_button_get_value_as_int(GtkSpinButton *spin_button);

Si desea modificar el valor de un valor de espín en relación a su valor actual, entonces la siguiente función se puede utilizar:

void gtk_spin_button_spin(GtkSpinButton *spin_button,

GtkSpinType direction,

gfloat increment);

El parámetro de dirección puede tomar uno de los siguientes valores:

GTK_SPIN_STEP_FORWARD

GTK_SPIN_STEP_BACKWARD

GTK_SPIN_PAGE_FORWARD

GTK_SPIN_PAGE_BACKWARD

GTK_SPIN_HOME

GTK_SPIN_END

GTK_SPIN_USER_DEFINED

Esta función empaqueta en un poco de funcionalidad, que voy a tratar de explicar con claridad. Muchos de estos ajustes, utilice los valores del objeto de ajuste que está asociado a un botón de Spin.

GTK_SPIN_STEP_FORWARD y GTK_SPIN_STEP_BACKWARD cambiar el valor del botón de la vuelta por la cantidad especificada por incremento, a menos incremento es igual a 0, en cuyo caso el valor se cambia por el valor de step_increment en theAdjustment.

GTK_SPIN_PAGE_FORWARD y GTK_SPIN_PAGE_BACKWARD simplemente alterar el valor del botón de girar por incremento.

GTK_SPIN_HOME establece el valor del botón de la vuelta a la parte inferior de la gama de ajustes.

GTK_SPIN_END establece el valor del botón de la vuelta a la parte superior de la gama de ajustes.

GTK_SPIN_USER_DEFINED simplemente altera el valor del botón de número en la cantidad especificada.

Nos alejamos de las funciones de establecer y retreving los atributos del rango del botón de la vuelta ahora, y pasar a las funciones que modifican la apariencia y el comportamiento del widget de botón de número en sí.

La primera de estas funciones se utiliza para limitar la caja de texto del botón de giro tal que sólo puede contener un valor numérico. Esto evita que el usuario escriba algo distinto de los valores numéricos en el cuadro de texto de un botón de Spin:

```
void gtk_spin_button_set_numeric( GtkSpinButton *spin_button,
                                  gboolean      numeric );
```

Puede establecer si un botón de Spin quede en los valores superior e inferior con la siguiente función:

```
void gtk_spin_button_set_wrap( GtkSpinButton *spin_button,
                               gboolean      wrap );
```

Puede configurar un botón de número para redondear el valor a la más cercana step_increment, que se encuentra dentro del objeto de ajuste se utiliza con el botón Spin. Esto se logra con la siguiente función:

```
void gtk_spin_button_set_snap_to_ticks( GtkSpinButton  *spin_button,
                                        gboolean       snap_to_ticks );
```

La política de actualización de un botón de giro se puede cambiar con la siguiente función:

```
void gtk_spin_button_set_update_policy( GtkSpinButton  *spin_button,
                                        GtkSpinButtonUpdatePolicy policy );
```

Los valores posibles de la política son o GTK_UPDATE_ALWAYS o GTK_UPDATE_IF_VALID.

Estas políticas afectan el comportamiento de un botón de la vuelta al analizar el texto insertado y sincroniza su valor con los valores del ajuste.

En el caso de GTK_UPDATE_IF_VALID el valor de botón de número sólo si se cambia el texto de entrada es un valor numérico que se encuentra dentro del rango especificado por el ajuste. De lo contrario el texto se restablece en el valor actual.

En caso de GTK_UPDATE_ALWAYS ignoramos los errores de conversión de texto en un valor numérico.

La apariencia de los botones que se utilizan en un botón de la vuelta se puede cambiar mediante la siguiente función:

```
void gtk_spin_button_set_shadow_type( GtkSpinButton *spin_button,
                                      GtkShadowType  shadow_type );
```

Como de costumbre, el shadow_type puede ser uno de:

GTK_SHADOW_IN

GTK_SHADOW_OUT

GTK_SHADOW_ETCHED_IN

GTK_SHADOW_ETCHED_OUT

Por último, usted puede solicitar explícitamente que un botón de la vuelta se actualice:

void gtk_spin_button_update(GtkSpinButton *spin_button);

Es hora de ejemplo de nuevo.

```
/* example-start spinbutton spinbutton.c */

#include <stdio.h>
#include <gtk/gtk.h>

static GtkWidget *spinner1;

void toggle_snap( GtkWidget     *widget,
                  GtkSpinButton *spin )
{
  gtk_spin_button_set_snap_to_ticks (spin, GTK_TOGGLE_BUTTON (widget)-
>active);
}

void toggle_numeric( GtkWidget *widget,
                     GtkSpinButton *spin )
{
  gtk_spin_button_set_numeric (spin, GTK_TOGGLE_BUTTON (widget)->active);
}

void change_digits( GtkWidget *widget,
                    GtkSpinButton *spin )
{
  gtk_spin_button_set_digits (GTK_SPIN_BUTTON (spinner1),
                              gtk_spin_button_get_value_as_int (spin));
}

void get_value( GtkWidget *widget,
                gpointer data )
{
  gchar buf[32];
  GtkLabel *label;
  GtkSpinButton *spin;

  spin = GTK_SPIN_BUTTON (spinner1);
  label = GTK_LABEL (gtk_object_get_user_data (GTK_OBJECT (widget)));
  if (GPOINTER TO INT (data) == 1)
    sprintf (buf, "%d", gtk_spin_button_get_value_as_int (spin));
  else
    sprintf (buf, "%0.*f", spin->digits,
```

```
              gtk_spin_button_get_value_as_float (spin));
  gtk_label_set_text (label, buf);
}


int main( int   argc,
          char *argv[] )
{
  GtkWidget *window;
  GtkWidget *frame;
  GtkWidget *hbox;
  GtkWidget *main_vbox;
  GtkWidget *vbox;
  GtkWidget *vbox2;
  GtkWidget *spinner2;
  GtkWidget *spinner;
  GtkWidget *button;
  GtkWidget *label;
  GtkWidget *val_label;
  GtkAdjustment *adj;

  /* Initialise GTK */
  gtk_init(&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

  gtk_signal_connect (GTK_OBJECT (window), "destroy",
                      GTK_SIGNAL_FUNC (gtk_main_quit),
                      NULL);

  gtk_window_set_title (GTK_WINDOW (window), "Spin Button");

  main_vbox = gtk_vbox_new (FALSE, 5);
  gtk_container_set_border_width (GTK_CONTAINER (main_vbox), 10);
  gtk_container_add (GTK_CONTAINER (window), main_vbox);

  frame = gtk_frame_new ("Not accelerated");
  gtk_box_pack_start (GTK_BOX (main_vbox), frame, TRUE, TRUE, 0);

  vbox = gtk_vbox_new (FALSE, 0);
  gtk_container_set_border_width (GTK_CONTAINER (vbox), 5);
  gtk_container_add (GTK_CONTAINER (frame), vbox);

  /* Day, month, year spinners */

  hbox = gtk_hbox_new (FALSE, 0);
  gtk_box_pack_start (GTK_BOX (vbox), hbox, TRUE, TRUE, 5);

  vbox2 = gtk_vbox_new (FALSE, 0);
  gtk_box_pack_start (GTK_BOX (hbox), vbox2, TRUE, TRUE, 5);

  label = gtk_label_new ("Day :");
  gtk_misc_set_alignment (GTK_MISC (label), 0, 0.5);
  gtk_box_pack_start (GTK_BOX (vbox2), label, FALSE, TRUE, 0);

  adj = (GtkAdjustment *) gtk_adjustment_new (1.0, 1.0, 31.0, 1.0,
                                              5.0, 0.0);
  spinner = gtk_spin_button_new (adj, 0, 0);
```

```
  gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (spinner), TRUE);
  gtk_spin_button_set_shadow_type (GTK_SPIN_BUTTON (spinner),
                                   GTK_SHADOW_OUT);
  gtk_box_pack_start (GTK_BOX (vbox2), spinner, FALSE, TRUE, 0);

  vbox2 = gtk_vbox_new (FALSE, 0);
  gtk_box_pack_start (GTK_BOX (hbox), vbox2, TRUE, TRUE, 5);

  label = gtk_label_new ("Month :");
  gtk_misc_set_alignment (GTK_MISC (label), 0, 0.5);
  gtk_box_pack_start (GTK_BOX (vbox2), label, FALSE, TRUE, 0);

  adj = (GtkAdjustment *) gtk_adjustment_new (1.0, 1.0, 12.0, 1.0,
                                              5.0, 0.0);
  spinner = gtk_spin_button_new (adj, 0, 0);
  gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (spinner), TRUE);
  gtk_spin_button_set_shadow_type (GTK_SPIN_BUTTON (spinner),
                                   GTK_SHADOW_ETCHED_IN);
  gtk_box_pack_start (GTK_BOX (vbox2), spinner, FALSE, TRUE, 0);

  vbox2 = gtk_vbox_new (FALSE, 0);
  gtk_box_pack_start (GTK_BOX (hbox), vbox2, TRUE, TRUE, 5);

  label = gtk_label_new ("Year :");
  gtk_misc_set_alignment (GTK_MISC (label), 0, 0.5);
  gtk_box_pack_start (GTK_BOX (vbox2), label, FALSE, TRUE, 0);

  adj = (GtkAdjustment *) gtk_adjustment_new (1998.0, 0.0, 2100.0,
                                              1.0, 100.0, 0.0);
  spinner = gtk_spin_button_new (adj, 0, 0);
  gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (spinner), FALSE);
  gtk_spin_button_set_shadow_type (GTK_SPIN_BUTTON (spinner),
                                   GTK_SHADOW_IN);
  gtk_widget_set_usize (spinner, 55, 0);
  gtk_box_pack_start (GTK_BOX (vbox2), spinner, FALSE, TRUE, 0);

  frame = gtk_frame_new ("Accelerated");
  gtk_box_pack_start (GTK_BOX (main_vbox), frame, TRUE, TRUE, 0);

  vbox = gtk_vbox_new (FALSE, 0);
  gtk_container_set_border_width (GTK_CONTAINER (vbox), 5);
  gtk_container_add (GTK_CONTAINER (frame), vbox);

  hbox = gtk_hbox_new (FALSE, 0);
  gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, TRUE, 5);

  vbox2 = gtk_vbox_new (FALSE, 0);
  gtk_box_pack_start (GTK_BOX (hbox), vbox2, TRUE, TRUE, 5);

  label = gtk_label_new ("Value :");
  gtk_misc_set_alignment (GTK_MISC (label), 0, 0.5);
  gtk_box_pack_start (GTK_BOX (vbox2), label, FALSE, TRUE, 0);

  adj = (GtkAdjustment *) gtk_adjustment_new (0.0, -10000.0, 10000.0,
                                              0.5, 100.0, 0.0);
  spinner1 = gtk_spin_button_new (adj, 1.0, 2);
  gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (spinner1), TRUE);
  gtk_widget_set_usize (spinner1, 100, 0);
```

```
  gtk_box_pack_start (GTK_BOX (vbox2), spinner1, FALSE, TRUE, 0);

  vbox2 = gtk_vbox_new (FALSE, 0);
  gtk_box_pack_start (GTK_BOX (hbox), vbox2, TRUE, TRUE, 5);

  label = gtk_label_new ("Digits :");
  gtk_misc_set_alignment (GTK_MISC (label), 0, 0.5);
  gtk_box_pack_start (GTK_BOX (vbox2), label, FALSE, TRUE, 0);

  adj = (GtkAdjustment *) gtk_adjustment_new (2, 1, 5, 1, 1, 0);
  spinner2 = gtk_spin_button_new (adj, 0.0, 0);
  gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (spinner2), TRUE);
  gtk_signal_connect (GTK_OBJECT (adj), "value_changed",
                      GTK_SIGNAL_FUNC (change_digits),
                      (gpointer) spinner2);
  gtk_box_pack_start (GTK_BOX (vbox2), spinner2, FALSE, TRUE, 0);

  hbox = gtk_hbox_new (FALSE, 0);
  gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, TRUE, 5);

  button = gtk_check_button_new_with_label ("Snap to 0.5-ticks");
  gtk_signal_connect (GTK_OBJECT (button), "clicked",
                      GTK_SIGNAL_FUNC (toggle_snap),
                      spinner1);
  gtk_box_pack_start (GTK_BOX (vbox), button, TRUE, TRUE, 0);
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (button), TRUE);

  button = gtk_check_button_new_with_label ("Numeric only input mode");
  gtk_signal_connect (GTK_OBJECT (button), "clicked",
                      GTK_SIGNAL_FUNC (toggle_numeric),
                      spinner1);
  gtk_box_pack_start (GTK_BOX (vbox), button, TRUE, TRUE, 0);
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (button), TRUE);

  val_label = gtk_label_new ("");

  hbox = gtk_hbox_new (FALSE, 0);
  gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, TRUE, 5);
  button = gtk_button_new_with_label ("Value as Int");
  gtk_object_set_user_data (GTK_OBJECT (button), val_label);
  gtk_signal_connect (GTK_OBJECT (button), "clicked",
                      GTK_SIGNAL_FUNC (get_value),
                      GINT_TO_POINTER (1));
  gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 5);

  button = gtk_button_new_with_label ("Value as Float");
  gtk_object_set_user_data (GTK_OBJECT (button), val_label);
  gtk_signal_connect (GTK_OBJECT (button), "clicked",
                      GTK_SIGNAL_FUNC (get_value),
                      GINT_TO_POINTER (2));
  gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 5);

  gtk_box_pack_start (GTK_BOX (vbox), val_label, TRUE, TRUE, 0);
  gtk_label_set_text (GTK_LABEL (val_label), "0");

  hbox = gtk_hbox_new (FALSE, 0);
  gtk_box_pack_start (GTK_BOX (main_vbox), hbox, FALSE, TRUE, 0);
```

```
  button = gtk_button_new_with_label ("Close");
  gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                             GTK_SIGNAL_FUNC (gtk_widget_destroy),
                             GTK_OBJECT (window));
  gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 5);

  gtk_widget_show_all (window);

  /* Enter the event loop */
  gtk_main ();

  return(0);
}
/* example-end */
```

:(creo que estoy chafiando aqui jejeje, aun no lo puedo compilar.

Selección de Color

El control de selección de color es, como cabe de esperar, un control para seleccionar colores interactivamente. Este control compuesto permite al usuario seleccionar un color manipulando triples RGB (Rojo, Verde, Azul) y HSV (Tono, Saturación, Valor). Esto se consigue ajustando valores simples con deslizadores o entradas, o haciendo clic en el color deseado en una rueda de tono-saturación y una barra de valor. Opcionalmente, la opacidad del color también se puede especificar.

El control de selección de color solo emite una señal por ahora, "color_changed", que se emite siempre que el color actual del control cambie, bien porque el usuario lo cambia o porque se especifique explícitamente a través del método `set_color`() .

Veamos lo que nos ofrece el control de selección de color. El control viene en dos sabores: `gtk.ColorSelection` y `gtk.ColorSelectionDialog`.

```
colorsel = gtk.ColorSelection()
```

Probablemente no se use este constructor directamente. Se crea un control `ColorSelection` huerfano que habría que emparentar uno mismo. El control `ColorSelection` hereda del control `VBox`.

```
colorseldlg = gtk.ColorSelectionDialog(title)
```

donde `title` (título) es una cadena usada para la barra de título del diálogo.

Este es el constructor más común del selector de color. Crea un `ColorSelectionDialog`. Éste consiste en un `Frame` que contiene un control `ColorSelection`, un `HSeparator` y un `HBox` con tres botones, Ok, Cancelar y Ayuda. Se pueden obtener estos botones accediendo a los atributos `ok_button`, `cancel_button` y `help_button` del `ColorSelectionDialog`, (por ejemplo, `colorseldlg.ok_button`). El control `ColorSelection` es accesible usando la variable `colorsel`:

```
colorsel = colorseldlg.colorsel
```

El control ColorSelection tiene unos cuantos métodos que cambian sus características o proporcionan acceso a la selección de color.

```
colorsel.set_has_opacity_control(has_opacity)
```

El control de selección de color permite ajustar la opacidad de un color (también conocida como el canal alfa). Esto esta desactivado por defecto. Llamando a este método con has_opacity igual TRUE activa la opacidad. De la misma forma, has_opacity igual a FALSE desactivará la opacidad.

```
colorsel.set_current_color(color)
colorsel.set_current_alpha(alpha)
```

Puedes poner el color actual explícitamente llamando al método set_current_color() con un GdkColor. La opacidad (canal alfa) se pone con el método set_current_alpha().El valor del canal alfa alpha debe estar entre 0 (completamente transparente) y 65636 (completamente opaco).

```
color = colorsel.get_current_color()
alpha = colorsel.get_current_alpha()
```

Cuando se tenga que mirar el color actual, típicamente al recibir la señal "color_changed", se pueden usar esos métodos.

```
/* example-start colorsel colorsel.c */

#include <glib.h>

#include <gdk/gdk.h>

#include <gtk/gtk.h>


GtkWidget *colorseldlg = NULL;

GtkWidget *drawingarea = NULL;


/* Color changed handler */
```

```
void color_changed_cb( GtkWidget        *widget,
                       GtkColorSelection *colorsel )
{
  gdouble color[3];
  GdkColor gdk_color;
  GdkColormap *colormap;

  /* Get drawingarea colormap */

  colormap = gdk_window_get_colormap (drawingarea->window);

  /* Get current color */

  gtk_color_selection_get_color (colorsel,color);

  /* Fit to a unsigned 16 bit integer (0..65535) and
   * insert into the GdkColor structure */

  gdk_color.red = (guint16)(color[0]*65535.0);
  gdk_color.green = (guint16)(color[1]*65535.0);
  gdk_color.blue = (guint16)(color[2]*65535.0);

  /* Allocate color */
```

```
  gdk_color_alloc (colormap, &gdk_color);


  /* Set window background color */


  gdk_window_set_background (drawingarea->window, &gdk_color);


  /* Clear window */


  gdk_window_clear (drawingarea->window);
}


/* Drawingarea event handler */


gint area_event( GtkWidget *widget,
                 GdkEvent  *event,
                 gpointer   client_data )
{
  gint handled = FALSE;

  GtkWidget *colorsel;
  /* Check if we've received a button pressed event */
  if (event->type == GDK_BUTTON_PRESS && colorseldlg == NULL)
    {
      /* Yes, we have an event and there's no colorseldlg yet! */
      handled = TRUE;
      /* Create color selection dialog */
      colorseldlg = gtk_color_selection_dialog_new("Select background color");
      /* Get the ColorSelection widget */
      colorsel = GTK_COLOR_SELECTION_DIALOG(colorseldlg)->colorsel;
      /* Connect to the "color_changed" signal, set the client-data
       * to the colorsel widget */
```

```
      gtk_signal_connect(GTK_OBJECT(colorsel), "color_changed",
        (GtkSignalFunc)color_changed_cb, (gpointer)colorsel);
     /* Show the dialog */
      gtk_widget_show(colorseldlg);
    }
  return handled;
}


/* Close down and exit handler */


gint destroy_window( GtkWidget *widget,

                     GdkEvent  *event,

                     gpointer   client_data )

{

  gtk_main_quit ();

  return(TRUE);

}


/* Main */


gint main( gint   argc,

           gchar *argv[] )

{

  GtkWidget *window;


  /* Initialize the toolkit, remove gtk-related commandline stuff */
```

```
 gtk_init (&argc,&argv);

 /* Create toplevel window, set title and policies */

 window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
 gtk_window_set_title (GTK_WINDOW(window), "Color selection test");
 gtk_window_set_policy (GTK_WINDOW(window), TRUE, TRUE, TRUE);

 /* Attach to the "delete" and "destroy" events so we can exit */

 gtk_signal_connect (GTK_OBJECT(window), "delete_event",
   (GtkSignalFunc)destroy_window, (gpointer)window);

 /* Create drawingarea, set size and catch button events */

 drawingarea = gtk_drawing_area_new ();

 gtk_drawing_area_size (GTK_DRAWING_AREA(drawingarea), 200, 200);

 gtk_widget_set_events (drawingarea, GDK_BUTTON_PRESS_MASK);

 gtk_signal_connect (GTK_OBJECT(drawingarea), "event",
   (GtkSignalFunc)area_event, (gpointer)drawingarea);
```

```
  /* Add drawingarea to window, then show them both */

  gtk_container_add (GTK_CONTAINER(window), drawingarea);

  gtk_widget_show (drawingarea);
  gtk_widget_show (window);

  /* Enter the gtk main loop (this never returns) */

  gtk_main ();

  /* Satisfy grumpy compilers */

  return(0);
}
/* example-end */
```

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ **gcc colorsel.c -o colorsel $(pkg-config gtk+-2.0 --cflags --libs)**

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ **./colorsel**

Selectores de Archivo

El control de selección de fichero es una forma rápida y fácil de mostrar una caja de diálogo de Fichero. Viene con botones Ok, Cancelar y Ayuda, por lo que es estupendo para ahorrase tiempo de programación.

Para crear una nueva caja de selección de fichero se usa:

```
filesel = gtk.FileSelection(title=None)
```

Para fijar el nombre de fichero, por ejemplo para mostrar un directorio específico, o establecer un fichero predeterminado, usa este método:

```
filesel.set_filename(filename)
```

Para obtener el nombre de fichero que el usuario ha escrito o seleccionado, se usa este método:

```
filename = filesel.get_filename()
```

También hay referencias a los controles contenidos en el control de selección de ficheros. Estos son los atributos:

```
filesel.dir_list         # lista de directorios
filesel.file_list        # lista de ficheros
filesel.selection_entry  # entrada de selección
filesel.selection_text   # texto de selección
filesel.main_vbox        # caja vertical principal
filesel.ok_button        # botón ok
filesel.cancel_button    # botón cancelar
filesel.help_button      # botón ayuda
filesel.history_pulldown # lista de historia
filesel.history_menu     # menú de historia
filesel.fileop_dialog # diálogo
filesel.fileop_entry # entrada
filesel.fileop_file # fichero
filesel.fileop_c_dir # cambio de directorio
filesel.fileop_del_file # borrar fichero
filesel.fileop_ren_file # renombrar fichero
filesel.button_area      # área de botones
filesel.action_area      # área de acción
```

Lo más probable es que se quieran usar los atributos `ok_button`, `cancel_button`, y `help_button` para conectar sus señales a las retrollamadas.

```
/* example-start filesel filesel.c */


#include <gtk/gtk.h>


/* Get the selected filename and print it to the console */

void file_ok_sel( GtkWidget        *w,
```

```
                GtkFileSelection *fs )
{
   g_print ("%s\n", gtk_file_selection_get_filename (GTK_FILE_SELECTION (fs)));
}

void destroy( GtkWidget *widget,
              gpointer   data )
{
   gtk_main_quit ();
}

int main( int   argc,
          char *argv[] )
{
   GtkWidget *filew;

   gtk_init (&argc, &argv);

   /* Create a new file selection widget */
   filew = gtk_file_selection_new ("File selection");

   gtk_signal_connect (GTK_OBJECT (filew), "destroy",
                       (GtkSignalFunc) destroy, &filew);
   /* Connect the ok_button to file_ok_sel function */
```

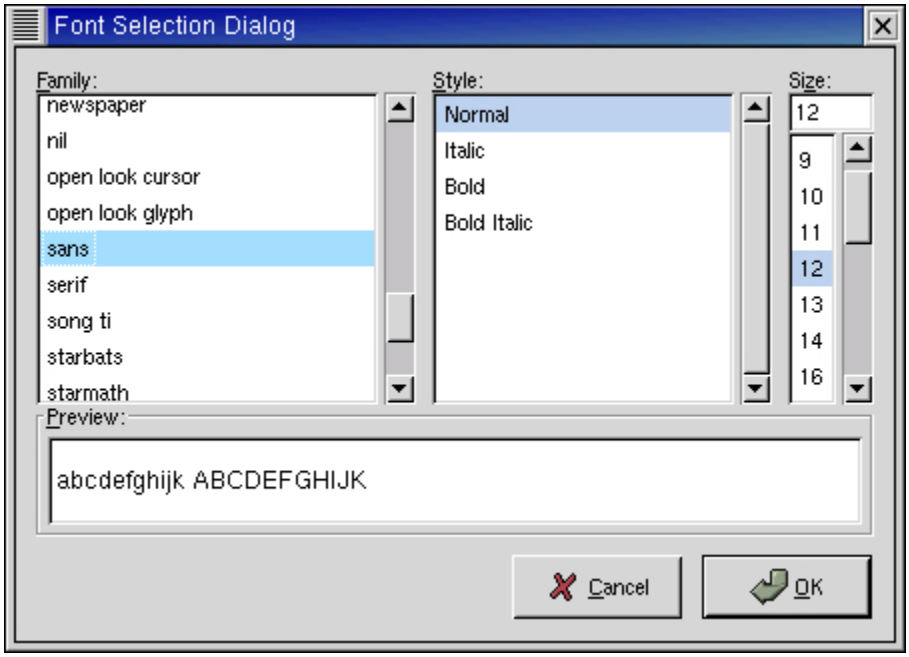
```
    gtk_signal_connect (GTK_OBJECT (GTK_FILE_SELECTION (filew)->ok_button),
                        "clicked", (GtkSignalFunc) file_ok_sel, filew );

    /* Connect the cancel_button to destroy the widget */
    gtk_signal_connect_object (GTK_OBJECT (GTK_FILE_SELECTION
                                           (filew)->cancel_button),
                               "clicked", (GtkSignalFunc) gtk_widget_destroy,
                               GTK_OBJECT (filew));

    /* Lets set the filename, as if this were a save dialog, and we are giving
     a default filename */
    gtk_file_selection_set_filename (GTK_FILE_SELECTION(filew),
                                     "penguin.png");

    gtk_widget_show(filew);
    gtk_main ();
    return 0;
}
/* example-end */
```

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ **gcc filesel.c -o filesel $(pkg-config gtk+-2.0 --cflags --libs)**

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ **./filesel**

Diálogo de Selección de Fuentes

El Diálogo de Selección de Fuentes permite al usuario seleccionar una fuente de forma interactiva. El diálogo contiene un control `FontSelection` y botones de OK y Cancelar. Un botón de Aplicar también está disponible en el diálogo, pero inicialmente está oculto. El Diálogo de Selección de Fuentes permite al usuario seleccionar una fuente de las fuentes de sistema disponibles (las mismas que se obtienen al usar **xlsfonts**).



Diálogo de Selección de Fuentes

El diálogo contiene un conjunto de tres fichas que proporcionan:

- una interfaz para seleccionar la fuente, el estilo y el tamaño
- información detallada sobre la fuente seleccionada
- una interfaz para el mecanismo de filtrado de fuente que restringe las fuentes disponibles para seleccionar

La función para crear un `FontSelectionDialog` es:

```
fontseldlg = gtk.FontSelectionDialog(title)
```

El `title` (título) es una cadena que se usará en el texto de la barra de título.

Una instancia de un Diálogo de Selección de Fuentes tiene varios atributos:

```
fontsel
main_vbox
```

```
action_area
ok_button
apply_button
cancel_button
```

El atributo `fontsel` es una referencia al control de selección de fuente. `main_vbox` es una referencia a la `gtk.VBox` que contiene el `fontsel` y el `action_area` en el diálogo. El atributo `action_area` es una referencia a la `gtk.HButtonBox` que contiene los botones OK, Aplicar y Cancelar. Los atributos `ok_button`, `cancel_button` y `apply_button` son referencias a los botones OK, Cancelar y Aplicar que se pueden usar para realizar las conexiones a las señales de los botones. La referencia `apply_button` también se puede usar para mostrar el botón Aplicar mediante el método `show`() .

Se puede fijar la fuente inicial que se mostrará en el diálogo usando el método:

```
fontseldlg.set_font_name(fontname)
```

El argumento `fontname` es el nombre de una fuente de sistema completo o parcialmente especificado. Por ejemplo:

```
fontseldlg.set_font_name('-adobe-courier-bold-*-*-*-*-120-*-*-*-*-*-*-*')
```

especifica una fuente inicial parcialmente.

El nombre de la fuente seleccionada se puede obtener con el método:

```
font_name = fontseldlg.get_font_name()
```

El Diálogo de Selección de Fuentes tiene un área de previsualización que muestra texto usando la fuente seleccionada. El texto que se usa en el área de previsualización se puede establecer con el método:

```
fontseldlg.set_preview_text(text)
```

El texto de previsualización se puede obtener con el método:

```
text = fontseldlg.get_preview_text()
```

/* example-start eventbox eventbox.c */

```
#include <gtk/gtk.h>


int main( int argc,
          char *argv[] )
{
    GtkWidget *window;
    GtkWidget *event_box;
    GtkWidget *label;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    gtk_window_set_title (GTK_WINDOW (window), "Event Box");

    gtk_signal_connect (GTK_OBJECT (window), "destroy",
```

```
                GTK_SIGNAL_FUNC (gtk_exit), NULL);


  gtk_container_set_border_width (GTK_CONTAINER (window), 10);


  /* Create an EventBox and add it to our toplevel window */


  event_box = gtk_event_box_new ();
  gtk_container_add (GTK_CONTAINER(window), event_box);
  gtk_widget_show (event_box);


  /* Create a long label */


  label = gtk_label_new ("Click here to quit, quit, quit, quit, quit");
  gtk_container_add (GTK_CONTAINER (event_box), label);
  gtk_widget_show (label);
```

```
    /* Clip it short. */

    gtk_widget_set_usize (label, 110, 20);



/* And bind an action to it */

    gtk_widget_set_events (event_box, GDK_BUTTON_PRESS_MASK);

    gtk_signal_connect (GTK_OBJECT(event_box), "button_press_event",

                        GTK_SIGNAL_FUNC (gtk_exit), NULL);



    /* Yet one more thing you need an X window for ... */



    gtk_widget_realize (event_box);

    gdk_window_set_cursor (event_box->window, gdk_cursor_new (GDK_HAND1));



    gtk_widget_show (window);



    gtk_main ();
```

```
    return(0);

}

/* example-end */
```

```
linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ gcc eventbox.c -o eventbox
$(pkg-config gtk+-2.0 --cflags --libs)

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ ./eventbox
```

Controles Contenedores

La Caja de Eventos (EventBox)

Algunos controles GTK no tienen ventanas X asociadas, por lo que simplemente se dibujan encima de sus padres. A causa de esto, no pueden recibir eventos y si se dimensionan incorrectamente, no pueden recortarse por lo que puedes ver partes mal, etc. Si se necesita más de estos controles, la `EventBox` (Caja de Eventos) es lo que se necesita.

A primera vista, el control `EventBox` puede aparecer completamente inútil. No dibuja nada en la pantalla y no responde a ningún evento. Sin embargo, tiene una función - proporciona una ventana X a sus controles hijos. Esto es importante ya que muchos controles GTK no tienen una ventana X asociada. No tener una ventana X ahorra memoria y mejora el rendimiento, pero también tiene inconvenientes. Un control sin ventana X no puede recibir eventos, no realiza ningun recortado sobre sus contenidos y no puede establecer su color de fondo. Aunque el nombre `EventBox` enfatiza la función de manejador de eventos, el control también se puede usar para recorte. (y más, mira el ejemplo más abajo).

Para crear un nuevo control `EventBox`, se usa:

```
event_box = gtk.EventBox()
```

Un control hijo se puede añadir a esta `event_box`:

```
event_box.add(widget)
```

```
/* example-start eventbox eventbox.c */


#include <gtk/gtk.h>


int main( int argc,
          char *argv[] )
{
    GtkWidget *window;
    GtkWidget *event_box;
    GtkWidget *label;


    gtk_init (&argc, &argv);


    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);


    gtk_window_set_title (GTK_WINDOW (window), "Event Box");
```

```
gtk_signal_connect (GTK_OBJECT (window), "destroy",

                    GTK_SIGNAL_FUNC (gtk_exit), NULL);



gtk_container_set_border_width (GTK_CONTAINER (window), 10);



/* Create an EventBox and add it to our toplevel window */



event_box = gtk_event_box_new ();

gtk_container_add (GTK_CONTAINER(window), event_box);

gtk_widget_show (event_box);



/* Create a long label */



label = gtk_label_new ("Click here to quit, quit, quit, quit, quit");

gtk_container_add (GTK_CONTAINER (event_box), label);
```

```
    gtk_widget_show (label);


    /* Clip it short. */

    gtk_widget_set_usize (label, 110, 20);


/* And bind an action to it */

    gtk_widget_set_events (event_box, GDK_BUTTON_PRESS_MASK);

    gtk_signal_connect (GTK_OBJECT(event_box), "button_press_event",

                        GTK_SIGNAL_FUNC (gtk_exit), NULL);


    /* Yet one more thing you need an X window for ... */


    gtk_widget_realize (event_box);

    gdk_window_set_cursor (event_box->window, gdk_cursor_new (GDK_HAND1));


    gtk_widget_show (window);
```

```
    gtk_main ();



    return(0);

}

/* example-end */
```

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ **gcc eventbox.c -o eventbox $(pkg-config gtk+-2.0 --cflags --libs)**

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ **./eventbox**

El control Alineador

El control `Alignment` (Alineador) permite colocar un control dentro de su ventana con una posición y un tamaño relativos al tamaño del propio control `Alignment`. Por ejemplo, puede ser útil para centrar un control dentro de la ventana.

Sólo hay dos llamadas asociadas al control `Alignment`:

```
alignment = gtk.Alignment(xalign=0.0, yalign=0.0, xscale=0.0, yscale=0.0)

alignment.set(xalign, yalign, xscale, yscale)
```

La función `gtk.Alignment`() crea un nuevo control `Alignment` con los parámetros especificados. El método `set`() permite alterar los parámetros de alineación de un control `Alignment` existente.

Los cuatro parámetros son números en coma flotante que pueden estar entre 0.0 y 1.0. Los argumentos `xalign` y `yalign` afectan a la posición del control dentro del Alignment. Las propiedades de alineación especifican la fracción de espacio *libre* que se colocará por encima o a la izquierda del control hijo. Sus valoren van de 0.0 (sin espacio *libre* por encima o a la izquierda del hijo) a 1.0 (todo espacio *libre* o a la izquierda del hijo). Naturalmente, si las dos propiedades de escala están puestas a 1.0, entonces las propiedades de alineación no tienen efecto, puesto que el control hijo se expandirá para llenar el espacio disponible.

Los argumentos `xscale` e `yscale` especifican la fracción de espacio *libre* absorbido por el control hijo. Los valores pueden variar desde 0.0 (el hijo no absorbe nada) hasta 1.0 (el hijo toma todo el espacio *libre*.

Un control hijo puede añadirse a este `Alignment` usando:

```
alignment.add(widget)
```

Contenedor Fijo (Fixed)

El contenedor `Fixed` (Fijo) permite situar controles en una posición fija dentro de su ventana, relativa a su esquina superior izquierda. La posición de los controles se puede cambiar dinámicamente.

Sólo hay dos llamadas asociadas al control fijo:

```
fixed = gtk.Fixed()

fixed.put(widget, x, y)

fixed.move(widget, x, y)
```

La función `gtk.Fixed`() permite crear un nuevo contenedor `Fixed`.

El método `put`() coloca al control en el contenedor fijo en la posición especificada por `x` e `y`.

El método `move`() te permite mover el control especificado a una nueva posición.

```
/* example-start fixed fixed.c */



#include <gtk/gtk.h>



/* I'm going to be lazy and use some global variables to

 * store the position of the widget within the fixed
```

```
 * container */

gint x=50;

gint y=50;



/* This callback function moves the button to a new position

 * in the Fixed container. */

void move_button( GtkWidget *widget,

                  GtkWidget *fixed )

{

  x = (x+30)%300;

  y = (y+50)%300;

  gtk_fixed_move( GTK_FIXED(fixed), widget, x, y);

}



int main( int   argc,

          char *argv[] )
```

```
{

  /* GtkWidget is the storage type for widgets */

  GtkWidget *window;

  GtkWidget *fixed;

  GtkWidget *button;

  gint i;



  /* Initialise GTK */

  gtk_init(&argc, &argv);



  /* Create a new window */

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

  gtk_window_set_title(GTK_WINDOW(window), "Fixed Container");



  /* Here we connect the "destroy" event to a signal handler */

  gtk_signal_connect (GTK_OBJECT (window), "destroy",
```

```
                GTK_SIGNAL_FUNC (gtk_main_quit), NULL);


 /* Sets the border width of the window. */

 gtk_container_set_border_width (GTK_CONTAINER (window), 10);


 /* Create a Fixed Container */

 fixed = gtk_fixed_new();

 gtk_container_add(GTK_CONTAINER(window), fixed);

 gtk_widget_show(fixed);


 for (i = 1 ; i <= 3 ; i++) {

   /* Creates a new button with the label "Press me" */

   button = gtk_button_new_with_label ("Press me");


   /* When the button receives the "clicked" signal, it will call the

    * function move_button() passing it the Fixed Container as its
```

```
     * argument. */

    gtk_signal_connect (GTK_OBJECT (button), "clicked",

                        GTK_SIGNAL_FUNC (move_button), fixed);



    /* This packs the button into the fixed containers window. */

    gtk_fixed_put (GTK_FIXED (fixed), button, i*50, i*50);



    /* The final step is to display this newly created widget. */

    gtk_widget_show (button);

  }



  /* Display the window */

  gtk_widget_show (window);



  /* Enter the event loop */

  gtk_main ();
```

```
  return(0);

}

/* example-end */
```

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ **gcc fixed.c -o fixed $(pkg-config gtk+-2.0 --cflags --libs)**

linux01@jaelegz07-ubuntu:~/Documentos/viernes8$ **./fixed**

Contenedor de Disposición (Layout)

El contenedor `Layout` (Disposición) es similar al contenedor `Fixed` (Fijo) excepto en que implementa un área de desplazamiento infinita (donde infinito es menor que 2^32). El sistema de ventanas X tiene una limitación que hace uqe las ventanas sólo puedan tener 32767 píxeles de ancho o alto. El contenedor `Layout` soluciona esta limitación haciendo algunos trucos exóticos mediante el uso de gravedad de ventana y de bit, para que se tenga un desplazamiento suave incluso cuando se tienen muchos controles en el área de desplazamiento.

Un contenedor `Layout` se crea con:

```
layout = gtk.Layout(hadjustment=None, vadjustment=None)
```

Como se puede ver, se puede especificar opcionalmente los objetos `Adjustment` que el control `Layout` usará para su desplazamiento. Si no se especifican los objetos `Adjustment`, se crearán unos nuevos.

Se pueden añadir y mover controles en el contenedor `Layout` usando los dos métodos siguientes:

```
layout.put(child_widget, x, y)

layout.move(child_widget, x, y)
```

El tamaño del contenedor `Layout` se puede establecer y consultar usando los siguientes métodos:

```
layout.set_size(width, height)
```

```
size = layout.get_size()
```

Los últimos cuatro métodos para los controles `Layout` widgets son para manipular los controles de ajuste horizontal y vertical:

```
hadj = layout.get_hadjustment()

vadj = layout.get_vadjustment()

layout.set_hadjustment(adjustment)

layout.set_vadjustment(adjustment)
```

Representar imágenes a pantalla completa con GTK+ en C (y romperles la pantalla del portátil)

Aunque muchos de mis proyectos con GTK+ son en C++, cuando es algo pequeño, es más corto hacerlo en C (aunque una vez que te acostumbras a los objetos, es poco intuitivo).

Lo que vamos a hacer es crear una ventana, introducir una imagen dentro redimensionada para ocupar la pantalla y poner la ventana a pantalla completa (ojo, no en todos los window managers funciona, pero en casi todos). Al final, lo que conseguimos es poner de fondo la imagen que ilustra el post en cualquier pantalla de portátil para conseguir dar un susto al dueño.

```
#include <gtk/gtk.h>

int main(int argc, char **argv)
{
GtkWindow *window;
GdkPixbuf *pixbuf;
GtkWidget *picture;
GdkScreen *screen;

gtk_init(&argc, &argv);

/* Creamos una ventana en el nivel superior */
window = GTK_WINDOW(gtk_window_new(GTK_WINDOW_TOPLEVEL));
/* Obtenemos la información de pantalla */
screen = gtk_window_get_screen(window);
/* Cargamos la imagen */
```

```
pixbuf = gdk_pixbuf_new_from_file_at_scale("CrackedScreen1440x900.jpg", /* nombre de fichero */
gdk_screen_get_width(screen), /* ancho de la pantalla */
gdk_screen_get_height(screen), /* alto de la pantalla */
FALSE, /* No respetar aspecto */
NULL);

/* Cuando cerremos la ventana, salimos de la aplicación */
gtk_signal_connect (GTK_OBJECT (window), "destroy",
GTK_SIGNAL_FUNC (gtk_main_quit), NULL);

/* Metemos la imagen dentro de la ventana */
picture=gtk_image_new_from_pixbuf(pixbuf);
gtk_container_add(GTK_CONTAINER(window), GTK_WIDGET(picture));

/* Mostramos todo */
gtk_widget_show_all(GTK_WIDGET(window));

/* Hacemos pantalla completa */
gtk_window_fullscreen(window);

/* Comenzamos */
gtk_main();

return 0;
}
```

Es importante ver cómo obtenemos los parámetros de la pantalla: una vez tenemos la ventana, podemos obtener la pantalla (screen) donde se encuentra, con ese elemento podemos acceder a sus dimensiones y pasárselas a la función de carga de la imagen para que la redimensione con esas características.

Al final, cuando tenemos todo preparado llamamos a gtk_window_fullscreen() pasándole la ventana actual. Si usáis GTKmm el método es Gdk::Window::fullscreen()

También es importante que, aunque sea para una tontería como esta, se capture la señal “destroy” ya que aunque se cierre la ventana, el programa seguirá en ejecución de forma indefinida, y no queremos eso.

Dibujar una senoidal

```
/*---test.c----*/

#include "gtk/gtk.h"
```

```
#include "glib.h"

#include "math.h"



void CerrarPrueba(GtkWidget *widget,gpointer *data)

{

gtk_main_quit();

}



void CrearSenhal(gint len, gfloat *vector)

{

gint i;

gfloat freq = 1;

gfloat dt = 0.01;

gfloat amp = 7.5;

for (i=0; i < len; i++)

{
```

```
*vector++=amp * sin (i*dt*2*3.14159*freq);

}

}

int main( int argc, char *argv[] )

{

gint veclen = 1000;

gfloat vector[veclen];



GtkWidget *window,*curva;

gtk_init (&argc, &argv);

window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

gtk_signal_connect( GTK_OBJECT(window),"destroy", GTK_SIGNAL_FUNC

(CerrarPrueba), NULL);

gtk_window_set_title ( GTK_WINDOW(window),"Mi primer grafica, Meguelo ;)");

gtk_window_set_default_size(GTK_WINDOW(window),500,100);

curva=gtk_curve_new();
```

```
gtk_curve_set_range (GTK_CURVE(curva),0,1000,-10,10);

CrearSenhal(veclen, vector);

gtk_container_add(GTK_CONTAINER(window),curva);

gtk_widget_show(curva);

gtk_widget_show(window);


gtk_curve_set_curve_type (GTK_CURVE(curva), GTK_CURVE_TYPE_FREE);

gtk_curve_set_vector (GTK_CURVE(curva),veclen,vector);

g_print("Grafica Meguelo\n");

gtk_main();

g_print("Hecho\n");

return 0;

}

[inforce@localhost viernes23]$ gcc test.c -o test $(pkg-config gtk+-2.0 --cflags --libs)
[inforce@localhost viernes23]$ ./test
```