



# Práctica Final

LIN - Curso 2016-2017





# Contenido

## 1 Introducción

## 2 Variante 1

## 3 Variante 2

- Uso de Blinkstick Strip en Android-x86

## 4 Entrega de la práctica



# Contenido

## 1 Introducción

## 2 Variante 1

## 3 Variante 2

- Uso de Blinkstick Strip en Android-x86

## 4 Entrega de la práctica



# Práctica Final

- Existen 2 variantes de la práctica final
  - Cada estudiante del mismo grupo tiene asociada una variante distinta
  - El listado con las variantes asignadas se encuentra en el Campus Virtual



# Práctica final: Aspectos comunes

## Cada variante consta de 2 partes

- **(Parte A)** Realizar modificaciones en una práctica y probarla sobre la MV de Debian
- **(Parte B)** Probar la funcionalidad del código realizado en la parte A sobre MV de Android-x86
  - Exige crear también un programa de usuario en C que interactúe con módulo del kernel desarrollado en la parte A, y que funcione tanto en Linux como en Android
  - Recomendable depurar el programa de usuario en Linux y luego llevarlo a Android
  - **La máquina virtual de Android lleva un kernel de 32 bits**
    - El código del kernel (fork de Linux v3.10.52) puede consultarse a través de <https://srcxref.dacya.ucm.es>





# Contenido

## 1 Introducción

## 2 Variante 1

## 3 Variante 2

- Uso de Blinkstick Strip en Android-x86

## 4 Entrega de la práctica



# Práctica Final (Variante 1)

## Parte A (sobre MV de Debian)

- 1 Modificar el módulo `fifoproc` (Parte B - Práctica 3) para que gestione dos “FIFOs” (entradas `/proc`)
- 2 Desarrollar un programa de usuario de línea de comandos (en C) que emule el comportamiento de un chat sencillo usando los dos FIFOs gestionados por el módulo creado

## Parte B (sobre MV de Android-x86)

- Mostrar al profesor en el laboratorio el funcionamiento del módulo de la parte A de la práctica y del chat sobre la máquina virtual de Android-x86
- El binario de Android-x86 para el programa `chat` se puede construir desde la MV de Debian con el siguiente comando:

```
$ gcc -m32 -static -lpthread chat.c -o chat-android
```





# Parte A: Módulo del kernel

## Requisitos

- Al cargar el módulo, éste creará dos entradas /proc (/proc/fifo0 y /proc/fifo1) que se comporten como ficheros FIFO
  - Esas entradas sustituyen a la entrada /proc que existía en el módulo original
- Cada entrada /proc nueva tendrá asociada una estructura “FIFO” privada (a definir por el alumno)
  - Las operaciones sobre estas entradas tendrán el mismo comportamiento que /proc/modfifo aunque con distinta implementación
  - Algunas variables globales existentes en el módulo original serán ahora miembros de una estructura que representa al “objeto” FIFO
- Al descargar el módulo, todas las entradas /proc creadas deben destruirse y la memoria asociada a las estructuras asociadas (FIFOs privados) deben liberarse





# Parte A: Módulo del kernel

## Recomendaciones

- Es aconsejable permitir al usuario que pueda especificar cuántas entradas /proc/fifoX se crean cuando el módulo se carga
  - Esto se puede implementar fácilmente añadiendo parámetros al módulo del kernel
    - Consultar ejemplo hello5.c de la Práctica 1



# Parte A: Pistas

## Asociar datos de uso privado (`private_data`) a entrada /proc

- Al crear una entrada /proc con la función `proc_create_data()` es posible asociar cualquier tipo de datos privados a la entrada /proc
  - `private_data` puede ser un puntero a un entero, un puntero a una estructura, una cadena de caracteres,...

```
struct proc_dir_entry *proc_create_data(const char *name,
                                         umode_t mode,
                                         struct proc_dir_entry *parent,
                                         const struct file_operations *ops,
                                         void *private_data);
```

- Desde el código de las callbacks de una entrada /proc podemos recuperar el campo `private_data` a partir del parámetro `struct file*` de la siguiente forma:

```
ssize_t my_read_callback (struct file *filp, char __user *buf, size_t len,
                         loff_t *off) {
    mi_tipo_de_datos* private_data=(mi_tipo_de_datos*)PDE_DATA(filp->f_inode);
    ....
}
```



## Parte A: Pistas (cont.)

### Asociar datos de uso privado (`private_data`) a entrada /proc

- Este mecanismo permite usar una misma función callback para distintas entradas /proc
  - Misma operación, pero sobre distintos datos ...



# Parte A: Chat

- Implementar el programa de usuario `chat.c` que se comporte como un chat sencillo
- Modo de uso

```
./chat <usuario> <ruta-fifo-envío> <ruta-fifo-recepción>
```

- `usuario`: Cadena de caracteres arbitraria que identifica usuario que se “conecta” al chat
- El programa usa un FIFO para enviar mensajes al otro usuario del chat y otro FIFO para recibir los mensajes
- Para poder usar el chat será necesario lanzar dos instancias de dicho programa en distintos terminales del siguiente modo
  - Terminal 1: \$ ./chat <usuario1> <ruta-fifo1> <ruta-fifo2>
  - Terminal 2: \$ ./chat <usuario2> <ruta-fifo2> <ruta-fifo1>



# Parte A: Chat

- El programa chat creará dos hilos (emisor y receptor) usando `pthread_create()`

## 1 Hilo emisor:

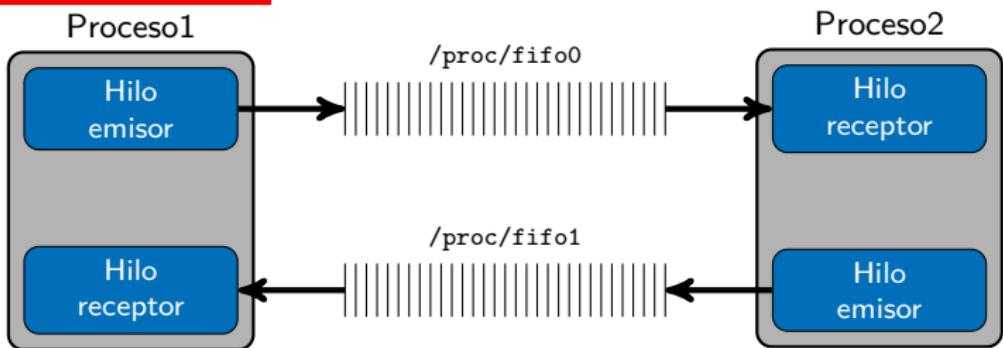
- Lee líneas por la entrada estandar (espera entrada usuario)
- Al leer una línea, envía un mensaje por el primer FIFO especificado en línea de comando

## 2 Hilo receptor:

- Permanece bloqueado hasta que se reciba un mensaje por el segundo FIFO especificado en línea de comando
- Al recibir mensaje, lo procesa y si procede muestra información por pantalla



# Chat: Ejemplo de ejecución



```
$ ./chat Fulanito /proc/fifo0 /proc/fifo1
Conexión de recepción establecida!!
Conexión de envío establecida!!
>Hola Menganito
>¿Estas por ahí?
>
Menganito dice: Sí, Menganito, aquí ando

Menganito dice: compilando el kernel. Tengo para rato...

>bueno, hasta luego entonces
> Menganito dice: ciao
[Ctrl+D]
$
```

```
$ ./chat Menganito /proc/fifo1 /proc/fifo0
Conexión de envío establecida!!
Conexión de recepción establecida!!
>
Fulanito dice: Hola Menganito

Fulanito dice: ¿Estas por ahí?

> Sí, Menganito, aquí ando
> compilando el kernel. Tengo para rato...
>
Fulanito dice: bueno, hasta luego entonces
>ciao
>Conexión finalizada por Fulanito!!
$
```





# Parte A: Chat (Implementación)

- Los datos transferidos a través de cada “FIFO” se representarán mediante el tipo de datos struct `chat_message`

```
#define MAX_CHARS_MSG 128

typedef enum {
    NORMAL_MSG, /* Mensaje para transferir lineas de la conversacion entre
                  ambos usuarios del chat */
    USERNAME_MSG, /* Tipo de mensaje reservado para enviar el nombre de
                   usuario al otro extremo*/
    END_MSG /* Tipo de mensaje que se envía por el FIFO cuando un extremo
             finaliza la comunicación */
}message_type_t;

struct chat_message{
    char contenido[MAX_CHARS_MSG]; //Cadena de caracteres (acabada en '\0')
    message_type_t type;
};
```



# Parte A: Chat (Implementación)

## Comportamiento hilo emisor de mensajes

- 1 Abre FIFO de envío en modo escritura
- 2 Envía nombre de usuario (especificado en linea de comandos) al otro extremo a través de FIFO de envío usando un mensaje de tipo USERNAME\_MSG
- 3 Lee líneas de la entrada estándar una a una y las envía encapsuladas en el campo contenido del mensaje (NORMAL\_MESSAGE) al otro extremo por el FIFO
  - El procesamiento finaliza cuando usuario teclea CTRL+D (EOF - *fin de fichero*)
  - Al detectar EOF en entrada estándar, enviará END\_MESSAGE al otro extremo
- 4 Cerrar FIFO de envío y salir



# Parte A: Chat (Implementación)

## Comportamiento hilo receptor de mensajes

- 1 Abre FIFO de recepción en modo lectura
- 2 Procesa mensaje de nombre de usuario (USERNAME\_MSG) leyendo del FIFO de recepción
- 3 Procesa resto de mensajes hasta fin de fichero en el FIFO de recepción, error de lectura o recepción de END\_MESSAGE
  - Al recibir un mensaje normal, imprime el campo content por pantalla con el siguiente formato:  
`<nombre_usuario_emisor> dice: <contenido>`
- 4 Cierra FIFO de recepción y sale



# Contenido

**1** Introducción

**2** Variante 1

**3** Variante 2

- Uso de Blinkstick Strip en Android-x86

**4** Entrega de la práctica



# Parte A (Variante 2)

- Ampliar la funcionalidad del driver del dispositivo Blinkstick Strip (Práctica 2 - Parte C) incorporando dos nuevos modos de funcionamiento
  - 1 Modo binario**
  - 2 Modo contador binario módulo N**
- Los nuevos modos permitirán gestionar de forma conjunta los LEDs del Blinkstick y los del teclado estándar desde el driver
- *Convenio de representación de números binarios de 11 bits con los LEDs:*
  - bits 10-3 → LEDs 7-0 del dispositivo Blinkstick Strip
  - bit 2 → *Num Lock*
  - bit 1 → *Caps Lock*
  - bit 0 → *Scroll Lock*

# Parte A (Variante 2)

## 2 nuevos modos

### 1 Modo binario

- En este modo el driver mostrará la representación binaria de un número decimal (<2048) usando los leds:
  - `sudo echo value 510 > /dev/usb/blinkstick0`



### 2 Modo contador binario módulo N

- El contador soportará las funciones *start*, *pause*, *resume* y *stop*
  - `sudo echo <acción> > /dev/usb/blinkstick0`
- La cuenta se actualizará mediante un temporizador del kernel (frecuencia configurable) y se mostrará en binario usando los leds según el convenio establecido



# Parte A (Contador binario módulo N)





# Parte A (Variante 2)

## Semántica funciones modo contador

- start: Empieza la cuenta del contador módulo N (arranca el temporizador del kernel)
- pause: Para temporalmente la cuenta del contador módulo N
- resume: El contador continuará contando desde el punto en el que se quedó antes de pausarse
- stop: Parar la cuenta del contador (detiene el temporizador del kernel) y apaga todos los LEDs



# Parte A (Variante 2)

- Los nuevos modos de funcionamiento exigen incorporar 3 parámetros configurables en el módulo
  - 1 color: Color en formato HTML (hexadecimal) a utilizar (valor por defecto: 0x101500)
  - 2 timer\_period: Periodo del temporizador del kernel (en ms) que se usará en el *Modo contador binario módulo N* (Valor por defecto: 500)
  - 3 max\_val: Máximo valor (N-1) que se usará durante la cuenta en el *Modo contador binario módulo N* (Valor por defecto: 2047)
- El driver permitirá ahora cambiar y consultar valor de parámetros de configuración via /dev/usb/blinkstick0
  - definir *read callback*

```
kernel@debian:~$ echo timer_period 300 > /dev/usb/blinkstick0
kernel@debian:~$ cat /dev/usb/blinkstick0
color=0x101500
max_val=2047
timer_period=300
```



# Parte A (Variante 2)

## Restricciones de implementación

- La implementación del módulo del kernel debe ser SMP-safe
  - Se valorará la eficiencia/calidad de la implementación
- El módulo del kernel no debe poder descargarse con rmmod mientras el driver esté en modo contador
  - Incrementar contador de referencias del módulo en *start* y decrementarlo en *stop*
- Mientras el modo contador esté activo el driver no debe permitir fijar el valor de los LEDs al usuario de forma manual

```
kernel@debian:~$ echo start > /dev/usb/blinkstick0
kernel@debian:~$ echo 1:0x110000,2:0x000022 > /dev/usb/blinkstick0
bash: echo: error de escritura: Dispositivo o recurso ocupado
kernel@debian:~$ echo stop > /dev/usb/blinkstick0
kernel@debian:~$ echo 1:0x110000,2:0x000022 > /dev/usb/blinkstick0
kernel@debian:~$
```



# Parte A (Variante 2)

## Restricciones de implementación (cont.)

- Desde la función del temporizador del kernel o es posible establecer el color de leds del dispositivo Blinkstick Strip directamente
  - **Problema:** La función `usb_control_msg()` es bloqueante
  - **Solución:** emplear tarea diferida (workqueues) solo cuando sea necesario cambiar el estado de los leds dispositivo Blinkstick Strip
  - Si los bits 3-11 del número actual difieren con respecto al del número anterior de la cuenta ⇒ planificar tarea diferida que establezca el valor de **TODOS** los LEDs correctamente (incluidos los del teclado)
  - En caso contrario, solo es necesario actualizar el estado de los LEDs del teclado (desde la propia función del timer)





# Parte B (Variante 2)

## Parte B

- 1** Mostrar al profesor en el laboratorio el funcionamiento del módulo de la parte A de la práctica sobre la máquina virtual de Android-x86
- 2** Desarrollar un programa de usuario ledtest.c para Android-x86 que interactúe mediante llamadas al sistema con el módulo del kernel desarrollado en la parte A
  - Adjuntar fichero README en la entrega con breve descripción sobre funcionalidad del programa desarrollado
  - El binario de Android-x86 para el programa ledtest se puede construir desde la MV de Debian con el siguiente comando:

```
$ gcc -m32 -static -g -Wall ledtest.c -o ledtest
```





# Programa parte B

## Objetivo

- El objetivo del programa es demostrar que los servicios que exporta el módulo del kernel funcionan correctamente
  - Queda a elección del alumno el tipo de procesamiento que realice el programa de usuario
  - No se permite usar `system()` ni `exec()`

## En esta parte se valorarán los siguientes aspectos:

- 1 Efectividad y completitud de los casos de uso/prueba del módulo llevados a cabo por el programa
- 2 Originalidad de la propuesta
- 3 Nivel de complejidad del programa C para Android-x86



# Blinkstick Strip en Android-x86

## Consideraciones generales

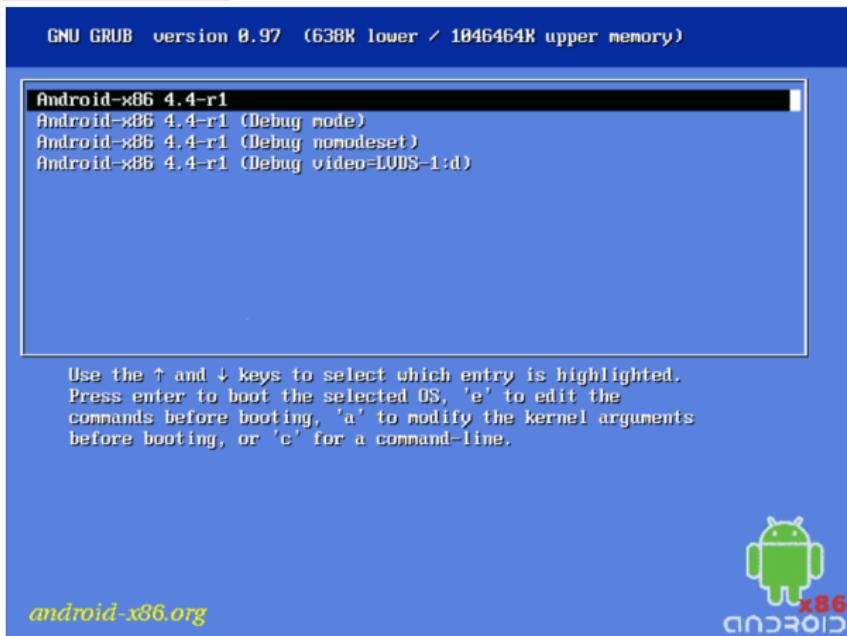
- En Android-x86 no es posible descargar el driver usbhid, porque no está compilado como fichero .ko (es parte del kernel)
  - Para que el driver usbhid ignore el dispositivo es preciso incluir una opción especial en la configuración de arranque del kernel (GRUB)
    - usbhid.quirks=0x20A0:0x41E5:0x0004
    - Tutorial sobre esto en las siguientes transparencias
- Al cargar driver de Blinkstick Strip en Android-x86 el fichero /dev/usb/blinkstick0 no se crea automáticamente porque en Android no hay udev
  - Solución: Crear el fichero de dispositivo manualmente desde adb shell con mknod, asignando el par (180,0) ⇒ (major,minor)

```
root@x86:/ $ mkdir /dev/usb  
root@x86:/ $ mknod /dev/usb/blinkstick0 c 180 0
```





# Arranque kernel para Blinkstick (1/5)



1 Arrancar la MV de Android-X86

2 Teclear “e” en GRUB (editar config. de arranque)





# Arranque kernel para Blinkstick (2/5)

```
GNU GRUB version 0.97 (638K lower / 1046464K upper memory)

kernel /android-4.4-r1/kernel quiet root=/dev/ram0 androidboot.hardware=BLINKSTICK
initrd /android-4.4-r1/initrd.img

Use the ↑ and ↓ keys to select which entry is highlighted.
Press 'b' to boot, 'e' to edit the selected command in the
boot sequence, 'c' for a command-line, 'o' to open a new line
after ('O' for before) the selected line, 'd' to remove the
selected line, or escape to go back to the main menu.

    android-x86.org
```



Teclear "e" de nuevo (Edición de línea de configuración)



# Arranque kernel para Blinkstick (3/5)

```
[ Minimal BASH-like line editing is supported. For the first word, TAB lists possible command completions. Anywhere else TAB lists the possible completions of a device/filename. ESC at any time cancels. ENTER at any time accepts your changes.]
```

```
<dboot.hardware=android_x86 video=-16 SRC=/android-4.4-r1>
```

[android-x86.org](http://android-x86.org)



*Insertar al final la siguiente línea de configuración:*

```
usbhid.quirks=0x20A0:0x41E5:0x0004
```



# Arranque kernel para Blinkstick (4/5)

## ■ Advertencia: Teclado en inglés

- “=” está en tecla de “¡” (exclamación)
- “:” se obtiene con SHIFT+Ñ

```
[ Minimal BASH-like line editing is supported. For the first word, TAB
 lists possible command completions. Anywhere else TAB lists the possible
 completions of a device/filename. ESC at any time cancels. ENTER
 at any time accepts your changes.]<dboot.hardware=android_x86 video=-16 SRC=/android-4.4-r1 usbhid.quirks=0x20A0>
```

```
[ Minimal BASH-like line editing is supported. For the first word, TAB
 lists possible command completions. Anywhere else TAB lists the possible
 completions of a device/filename. ESC at any time cancels. ENTER
 at any time accepts your changes.]<rkss=0x20A0:0x41E5:0x0004>
```

Pulsar ENTER (Aceptar cambios)



# Arranque kernel para Blinkstick (5/5)

```
GNU GRUB version 0.97 (638K lower / 1046464K upper memory)

kernel /android-4.4-r1/kernel quiet root=/dev/ram0 androidboot.hardware=BLINKSTICK
initrd /android-4.4-r1/initrd.img

Use the ↑ and ↓ keys to select which entry is highlighted.
Press 'b' to boot, 'e' to edit the selected command in the
boot sequence, 'c' for a command-line, 'o' to open a new line
after ('O' for before) the selected line, 'd' to remove the
selected line, or escape to go back to the main menu.

android-x86.org
```



Teclear "b" (Arrancar el kernel)



# Contenido

## 1 Introducción

## 2 Variante 1

## 3 Variante 2

- Uso de Blinkstick Strip en Android-x86

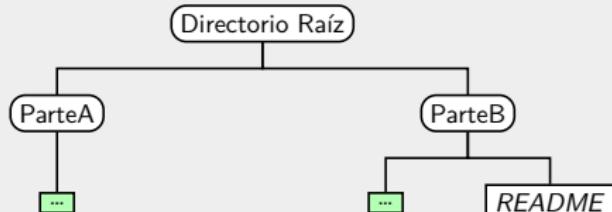
## 4 Entrega de la práctica



# Entrega de la práctica

- Entrega a través del Campus Virtual
  - Entrega INDIVIDUAL hasta el 1 de febrero a las 9:00h
  - No se permiten entregas tardías
- Defensa individual obligatoria el 1 de febrero en Lab 3
  - Defensa variante 1: 9:30h
  - Defensa variante 2: 11:00h
- Esta práctica es “opcional” (Máximo 2.5 puntos+)
  - Se permiten “entregas parciales” (p. ej., sólo un apartado)

## Estructura entrega (en un fichero comprimido .tar.gz o .zip)





# Licencia

LIN - Práctica Final  
Versión 0.5

©J.C. Sáez

*This work is licensed under the Creative Commons **Attribution-Share Alike 3.0 Spain License**. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/es/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.*

*Esta obra está bajo una licencia **Reconocimiento-Compartir Bajo La Misma Licencia 3.0 España de Creative Commons**. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-sa/3.0/es/> o envíe una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.*

Este documento (o uno muy similar) está disponible en  
<https://cv4.ucm.es/moodle/course/view.php?id=75410>

