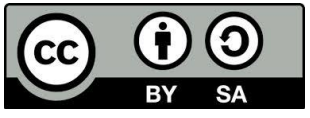
Práctica Final

LIN - Curso 2019-2020



J.C. Sáez
High Performance Computing Division ArTeCs Group
Universidad Complutense de Madrid

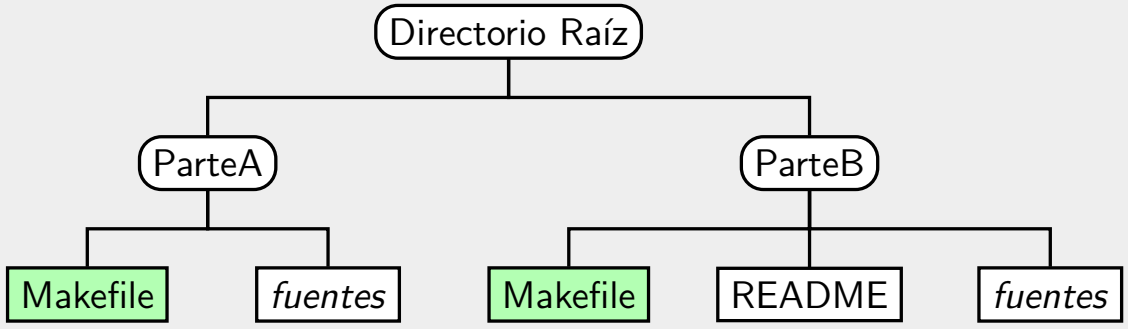
Práctica Final

- 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**

Entrega de la práctica

- Entrega a través del Campus Virtual
  - Entrega INDIVIDUAL hasta el 24 de enero a las 12:00
  - No se permiten entregas tardías
- Defensa individual obligatoria el 24 de enero en Lab 3
  - Defensa variante 1: 13:00h
  - Defensa variante 2: 14:20h
- 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)



Práctica Final

Parte A (sobre MV de Debian)

- **Variante 1**: Extender módulo del kernel de la Práctica 4A para que gestione múltiples listas enlazadas de enteros
- **Variante 2**: Extender módulo del kernel de la Práctica 4B para que gestione múltiples "FIFOs"

Parte B (sobre MV de Android-x86)

1. Hacer funcionar el módulo de la parte A en MV de Android-x86
2. Crear programa multihilo en C que interactúe con el módulo del kernel desarrollado
   - Recomendable depurar el programa de usuario en Linux y luego probarlo sobre Android

Práctica Final

Parte A (Ambas variantes)

- Al cargar el módulo, éste creará un directorio `$PROC_DIRECTORY` en el sistema de ficheros /proc
  - Variante 1: PROC_DIRECTORY=`/proc/list`
  - Variante 2: PROC_DIRECTORY=`/proc/fifo`
- También se crearán dos entradas /proc en `$PROC_DIRECTORY`:
  1. `"default"`: sustituye a la entrada que existía en el módulo original
     - Mismo comportamiento aunque distinta implementación
     - Tendrá asociada una lista enlazada o FIFO (depende de variante)
  2. `"control"`: permite crear/destruir entradas /proc en $PROC_DIRECTORY con misma semántica que la entrada "default" pero con sus propias estructuras asociadas (lista enlazada o fifo privados)
     - Crear entrada:
       ```
       $ echo create my_entry > $PROC_DIRECTORY/control
       ```
     - Eliminar entrada existente (como "default")
       ```
       $ echo delete my_entry > $PROC_DIRECTORY/control
       ```

Práctica Final

Requisitos Parte A

- Cada vez que se cree una entrada /proc escribiendo en la entrada de control, se creará también una estructura privada asociada a la entrada (lista enlazada o FIFO)
  - Algunas variables globales existentes en el módulo original serán ahora miembros de una estructura a definir en el módulo
  - La entrada default tendrá asociada también su propia estructura privada
  - Las callbacks de la entrada "default" así como aquellas de las entradas creadas dinámicamente se implementarán con las mismas funciones
    - Mismas callbacks pero sobre distintos datos
    - No se permite replicar código
- Al eliminar una entrada, la memoria asociada a la estructura correspondiente debe liberarse

Práctica Final

Requisitos Parte A (Cont.)

- El módulo del kernel exportará dos parámetros, a establecer en tiempo de carga:
  1. `max_entries`: número máximo de entradas que pueden existir de forma simultánea (sin contar la entrada `control`).
     - El módulo debe denegar la creación de un número de entradas por encima de ese máximo, devolviendo un código de error
     - Valor por defecto: 4
  2. `max_size`: tamaño máximo de listas (V1) o buffers circulares (V2)
     - Para lista, especificado en núm. elementos. Valor por defecto=10
     - Para buffer, especificado en bytes. Valor por defecto=64
- El módulo no debe poder cargarse (error en init_module()) si el usuario establece valores no permitidos en los parámetros
- Para más detalles sobre parámetros en módulos, repasar Ejercicio 3 de la práctica 1 o la información de este enlace

Práctica Final

Requisitos Parte A (Cont.)

- Se ha de mantener una estructura de datos (p. ej., lista enlazada) para llevar la cuenta de las entradas /proc y estructuras privadas creadas dinámicamente
- Al descargar el módulo, todas las entradas /proc creadas (incluido el directorio) deben destruirse y la memoria asociada a las estructuras asociadas debe liberarse
- La implementación del módulo del kernel debe ser SMP-safe
  - Se valorará la eficiencia/calidad de la implementación
  - Nota: `remove_proc_entry()` es una función bloqueante
    - Bloquea al flujo del kernel invocador hasta que la entrada /proc deja de estar en uso (en el caso de que lo esté)

Parte A (V1): Ejemplo de ejecución

Terminal

```
kernel@debian:~/Final/ParteAV1$ sudo insmod list.ko max_entries=4
kernel@debian:~/Final/ParteAV1$ ls /proc/list
control  default
kernel@debian:~/Final/ParteAV1$ cat /proc/list/default
kernel@debian:~/Final/ParteAV1$ echo add 10 >  /proc/list/default
kernel@debian:~/Final/ParteAV1$ cat /proc/list/default
10
kernel@debian:~/Final/ParteAV1$ echo add 4 >  /proc/list/default
kernel@debian:~/Final/ParteAV1$ echo add 4 >  /proc/list/default
kernel@debian:~/Final/ParteAV1$ cat /proc/list/default
10
4
4
kernel@debian:~/Final/ParteAV1$ echo add 2 >  /proc/list/default
kernel@debian:~/Final/ParteAV1$ echo add 5 >  /proc/list/default
kernel@debian:~/Final/ParteAV1$ cat /proc/list/default
10
4
4
2
5
```

Parte A (V1): Ejemplo de ejecución (cont.)

Terminal

```
kernel@debian:~/Final/ParteAV1$ echo create list0 > /proc/list/control
kernel@debian:~/Final/ParteAV1$ echo create list1 > /proc/list/control
kernel@debian:~/Final/ParteAV1$ cat /proc/list/list0
kernel@debian:~/Final/ParteAV1$ cat /proc/list/list1
kernel@debian:~/Final/ParteAV1$ for i in `seq 0 5`
> do
> echo add $i > /proc/list/list0
> done
kernel@debian:~/Final/ParteAV1$ echo add 1 > /proc/list/list1
kernel@debian:~/Final/ParteAV1$ cat /proc/list/list0
0
1
2
3
4
5
kernel@debian:~/Final/ParteAV1$ cat /proc/list/list1
1
kernel@debian:~/Final/ParteAV1$ cat /proc/list/default
10
4
4
2
5
```

Parte A (V1): Ejemplo de ejecución (cont.)

Terminal

```
kernel@debian:~/Final/ParteAV1$ echo delete no_existe > /proc/list/control
bash: echo: error de escritura: No existe el fichero o el directorio
kernel@debian:~/Final/ParteAV1$ echo delete default > /proc/list/control
kernel@debian:~/Final/ParteAV1$ ls /proc/list/
control  list0  list1
kernel@debian:~/Final/ParteAV1$ sudo rmmod list.ko
kernel@debian:~/Final/ParteAV1$
```

Parte A (V2): Ejemplo de ejecución

terminal 1

```
kernel@debian:~/Final/ParteAV2$ sudo insmod fifo.ko max_entries=3
kernel@debian:~/Final/ParteAV2$ ls /proc/fifo/
control  default
kernel@debian:~/Final/ParteAV2$ cd ~/FifoTest/
kernel@debian:~/FifoTest$ ./fifotest -f /proc/fifo/default -s < test.txt
kernel@debian:~/FifoTest$
```

terminal 2

```
kernel@debian:~/FifoTest$ ./fifotest -f /proc/fifo/default -r
En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo
que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y
galgo corredor. Una olla de algo más vaca que carnero, salpicón las más noches,
duelos y quebrantos los sábados, lentejas los viernes, algún palomino de añadidura
los domingos, consumían las tres partes de su hacienda. El resto della concluían
sayo de velarte, calzas de velludo para las fiestas con sus pantuflos de lo mismo,
los días de entre semana se honraba con su vellori de lo más fino. Tenía en su casa
una ama que pasaba de los cuarenta, y una sobrina que no llegaba a los veinte, y
un mozo de campo y plaza, que así ensillaba el rocín como tomaba la podadera...
kernel@debian:~/FifoTest$
```

Parte A (V2): Ejemplo de ejecución (cont.)

terminal 1

```
kernel@debian:~/FifoTest$ echo create fifoA > /proc/fifo/control
kernel@debian:~/FifoTest$ echo create fifoB > /proc/fifo/control
kernel@debian:~/FifoTest$ ls /proc/fifo
control  default  fifoA  fifoB
kernel@debian:~/FifoTest$ ./fifotest -f /proc/fifo/fifoA -s
Esto va por fifoA
kernel@debian:~/FifoTest$
```

terminal 2

```
kernel@debian:~/FifoTest$ ./fifotest -f /proc/fifo/fifoA -r
Esto va por fifoA
kernel@debian:~/FifoTest$
```

terminal 3

```
kernel@debian:~/FifoTest$ ./fifotest -f /proc/fifo/fifoB -s
Esto va por fifoB
kernel@debian:~/FifoTest$
```

terminal 4

```
kernel@debian:~/FifoTest$ ./fifotest -f /proc/fifo/fifoB -r
Esto va por fifoB
kernel@debian:~/FifoTest$
```

Parte A (V2): Ejemplo de ejecución (cont.)

terminal

```
kernel@debian:~/Final/ParteAV2$ echo delete no_existe > /proc/fifo/control
bash: echo: error de escritura: No existe el fichero o el directorio
kernel@debian:~/Final/ParteAV2$ echo delete default > /proc/fifo/control
kernel@debian:~/Final/ParteAV2$ ls /proc/fifo
control  fifoA  fifoB
kernel@debian:~/Final/ParteAV2$ echo delete fifoB > /proc/fifo/control
kernel@debian:~/Final/ParteAV2$ ls /proc/fifo
control  fifoA
kernel@debian:~/Final/ParteAV2$ sudo rmmod fifo
```

Parte A: Pistas

Creación y destrucción de directorios en /proc

- La función `proc_mkdir()` permite crear un directorio `name` en el sistema de ficheros /proc
  - Retorna descriptor del directorio creado

```
struct proc_dir_entry *proc_mkdir(const char *name,
                struct proc_dir_entry *parent);
```

- El descriptor devuelto se pasa como tercer parámetro de la función `proc_create()` o `proc_create_data()` para crear una entrada /proc que "cuelge" del directorio creado
- Para destruir un directorio dentro de /proc es preciso utilizar `remove_proc_entry()`
  - El directorio ha de estar vacío para que la eliminación tenga éxito
- Se proporciona módulo de ejemplo `clipboard_dir.c` que crea entrada `/proc/test/clipboard`

Parte A: Pistas (cont.)

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 ...

Práctica Final: Parte B

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
   - Para los alumnos que realicen la variante 2 de la práctica ...
     - El binario de Android-x86 para el programa `fifotest` se puede construir desde la MV de Debian con el siguiente comando:

       ```
       $ gcc -static -g -Wall fifotest.c -o fifotest
       ```

2. Desarrollar un programa de usuario multihilo en C para Android-x86 que interactúe mediante llamadas al sistema con el módulo del kernel desarrollado en la parte A
   - Compilación del programa para Android-x86 (p.ej., `user.c`)

     ```
     $ gcc -g -Wall -static -lpthread user.c -o user
     ```

Práctica Final: Parte B

- Como vimos en la clase del 4 de diciembre, se precisa tener el kernel Linux de la MV Android-x86 ya compilado en el *host* de desarrollo para poder compilar módulos del kernel compatibles con el sistema de la MV

Compilación módulos del kernel para Android desde MV Debian

- 2 alternativas
  1. Compilar kernel de Android dentro de la MV de Debian (siguiendo las instrucciones del tema "Kernel y Native Userspace") y usar ese kernel para generar fichero .ko del módulo
     - Puede requerir incrementar los recursos de la MV (Discos extra, Cores y RAM disponible)
     - Se valorará positivamente usar esta alternativa
  2. Utilizar un kernel ya compilado disponible en este enlace (tar.gz a extraer en el HOME de la MV)

```
$ cd ; tar xzvf ~/Descargas/android-oreo-kernel-built.tar.gz
```

Variante 1: programa parte B

Objetivo

- El objetivo del programa multihilo es demostrar que los servicios que exporta el módulo del kernel (p. ej.: entradas `/proc`) funcionan correctamente
  - Queda a elección del alumno el tipo de procesamiento que realice el programa de usuario
    - Adjuntar fichero `README` en la entrega con breve descripción sobre funcionalidad del programa desarrollado
  - 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

Variante 2 - Parte B: Implementación

- Desarrollar **programa de usuario que emule el comportamiento de un chat** rudimentario usando **2 FIFOs gestionados por el módulo del kernel desarrollado**
- Modo de uso:
  `./chat <NombreDeUsuario> <RutaFIFO1> <RutaFIFO2>`
- El programa chat creará dos hilos (emisor y receptor) usando `pthread_create()`
  1. Hilo emisor: gestiona primer FIFO especificado en línea de comando
  2. Hilo receptor: gestiona segundo FIFO especificado en línea de comando
- El "chat" se pondrá en marcha al lanzar dos instancias de dicho programa en distintos terminales

Variante 2 - Parte B: Implementación

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

```
#define MAX_CHARS_MSG 128

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

struct mensaje_chat{
     char contenido[MAX_CHARS_MSG]; //Cadena de caracteres (acabada en '\0)
     enum tipo_mensaje tipo;
};
```

Variante 2 - Parte B: Implementación

Comportamiento hilo emisor de mensajes

1. Abre FIFO de envío en modo escritura
2. Envía nombre de usuario (especificado en línea de comandos) al otro extremo a través de FIFO de envío usando un mensaje de tipo MENSAJE_NOMBRE
3. Lee líneas de la entrada estándar una a una (ej: vía `getline()`) y las envía encapsuladas en el campo `contenido` de MENSAJE_NORMAL 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 MENSAJE_FIN al otro extremo
4. Cerrar FIFO de envío y salir

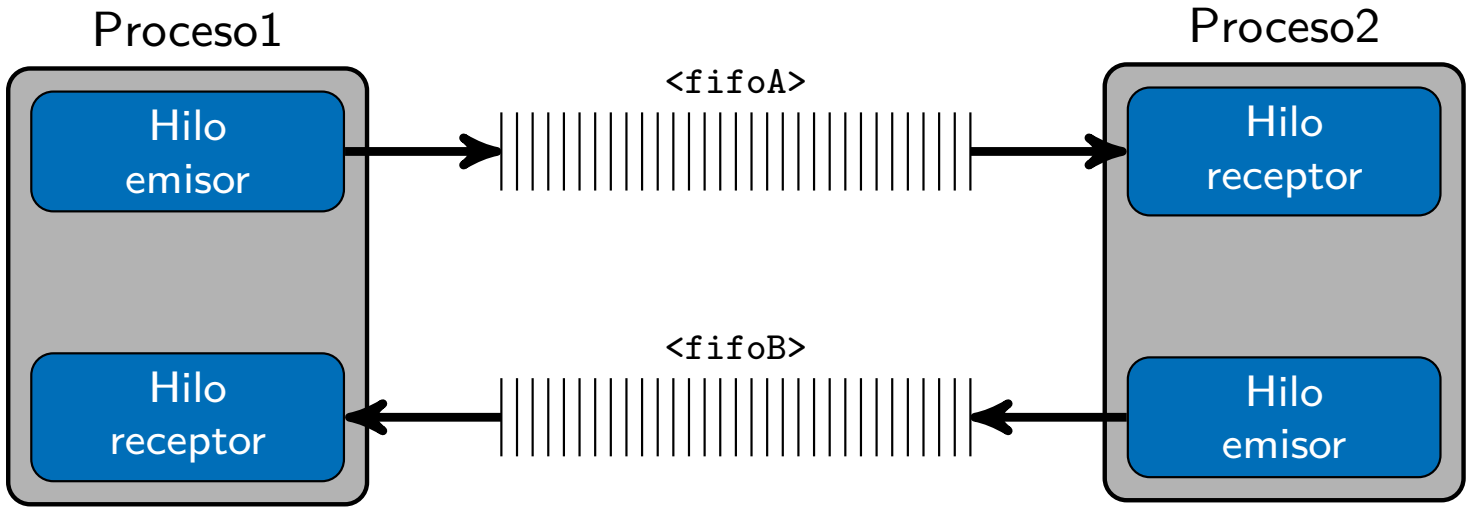
Variante 2 - Parte B: Implementación

Comportamiento hilo receptor de mensajes

1. Abre FIFO de recepción en modo lectura
2. Procesa mensaje de nombre de usuario (`MENSAJE_NOMBRE`) 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 `MENSAJE_FIN`
   - Al recibir un mensaje normal, imprime el campo `contenido` por pantalla con el siguiente formato:

     `<nombre_usuario_emisor> dice: <contenido>`
4. Cierra FIFO de recepción y sale

Parte B: Ejemplo de ejecución




```
$ ./chat Fulanito <fifoA> <fifoB>
Conexión de recepción establecida!!
Conexión de envío establecida!!
>Hola Menganito
>¿Estás 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 <fifoB> <fifoA>
Conexión de envío establecida!!
Conexión de recepción establecida!!
>
Fulanito dice: Hola Menganito

Fulanito dice: ¿Estás por ahí?

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

Práctica Final: Parte B

Consideraciones adicionales

- La máquina virtual de Android-x86 lleva un kernel de 64 bits
  - El código del kernel (fork de Linux v4.9.109) puede consultarse a través de `https://srcxref.dacya.ucm.es`
- Recomendable depurar el programa de usuario en Linux y luego llevarlo a Android
  - El chat se puede depurar también usando FIFOs reales creados con `mknod`

Licencia

LIN - Práctica Final
Versión 1.2

Este documento (o uno muy similar) está disponible en
`https://cv4.ucm.es/moodle/course/view.php?id=121225`