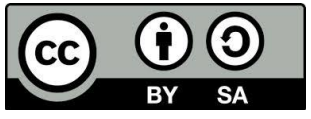
Práctica 2: Llamadas al Sistema

LIN - Curso 2014-2015



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

Contenido

Contenido

Práctica 2: Llamadas al sistema

Objetivos

- Familiarizarse con:
  - Implementación de llamadas al sistema en Linux y su procedimiento de invocación
  - Compilación del kernel Linux y creación de parches
  - Exportación de símbolos (funciones) para su uso desde módulos del kernel

Contenido

Ejercicios

- La entrada `/proc/cpuinfo` permite obtener información acerca de las CPUs del sistema

Terminal

```
kernel@debian:~$ cat /proc/cpuinfo
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 23
model name  : Intel(R) Xeon(R) CPU           E5450  @ 3.00GHz
stepping    : 10
cpu MHz     : 2003.000
cache size  : 6144 KB
physical id : 0
siblings    : 4
core id     : 0
cpu cores   : 4
apicid      : 0
initial apicid  : 0
fpu     : yes
fpu_exception   : yes
cpuid level : 13
wp      : yes
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse
bogomips    : 5984.67
clflush size    : 64
cache_alignment : 64
address sizes   : 38 bits physical, 48 bits virtual
power management:
...
```

Ejercicios

Ejercicio 1

- Estudiar la implementación del programa cpuinfo.c
  - Este programa imprime por pantalla el contenido de `/proc/cpuinfo` haciendo uso de las llamadas al sistema `open()` y `close()`, y las funciones `printf()` y `syscall()`.
  - ¿Qué llamada al sistema invoca el programa mediante `syscall()`?
  - Reescribir el programa anterior reemplazando las llamadas a `open()`, `close()` y `printf()` por invocaciones a `syscall()` que tengan el mismo comportamiento.

Contenido

Sistema de ficheros /proc

- El sistema de ficheros /proc de Linux resulta de gran utilidad para distintos subsistemas del kernel
  1. Interfaz para comunicación entre kernel/módulos y el modo usuario
  2. Interfaz extensible
     - Los módulos pueden crear nuevas entradas en cualquier directorio del árbol
- En otros SSOO tipo UNIX /proc no existe o no puede emplearse como mecanismo de interacción de "propósito general" entre modo usuario y modo kernel
  - Inexistente en Mac OS X
  - En Solaris, /proc se emplea únicamente para alojar información sobre los procesos
    - Los módulos cargables (también existentes en Solaris) no usan el /proc como interfaz *ad-hoc* con el usuario o las aplicaciones

Especificación de la práctica (I)

- En esta práctica construiremos un mecanismo alternativo a /proc llamado KIFS (*Kernel InterFace System*)

Entradas KIFS (`struct kifs_entry`)

- El SO mantiene una lista enlazada de entradas KIFS

```
struct kifs_entry {
    char entryname[MAX_SIZE_KIFS_ENTRY_NAME];
    struct kifs_operations* ops;
    void *private_data;
    struct list_head links;
};
```

  - `entryname`: Nombre de la entrada (ej.: *"mi_entrada"*)
    - Identificador único
  - `ops`: Interfaz de operaciones (*read/write callbacks*)
  - `private_data`: Datos privados de la entrada (uso opcional)
  - `links`: Enlaces de la lista

Especificación de la práctica (II)

Interfaz de operaciones entrada KIFS

```
struct kifs_operations {
    int (*read)(struct kifs_entry* entry, char *user_buffer,
     unsigned int maxchars);
    int (*write)(struct kifs_entry* entry, const char *user_buffer,
     unsigned int maxchars);
};
```

- Versión simplificada de `struct file_operations`
- Operaciones se comportan como *callbacks* de entradas /proc
  - `entry`: Descriptor de entrada cuya operación se invoca
    - Permite acceder a `entry->private_data` si es necesario
  - `user_buffer`: Puntero al espacio de usuario donde el usuario recibe/envía los datos
  - `maxchars`: Para `read()`, máximo número de bytes que podemos escribir en `user_buffer`; para `write()`, número de bytes que usuario ha escrito en `user_buffer`.

Especificación de la práctica (III)

Llamada al sistema `kifs()`

- Los procesos de usuario pueden invocar la operación de lectura o escritura de una entrada mediante una nueva llamada al sistema

```
int kifs(const char* entryname, unsigned int read_write,
         char* user_buffer, unsigned int maxchars);
```

- Esta "función" se invocará desde un programa de usuario
- **Parámetros**
  - `entryname`: nombre de la entrada cuya *callback* queremos invocar
  - `read_write`: 0 → lectura, 1 → escritura
  - `user_buffer` y `maxchars`: parámetros que se pasan a *callback*
- **Valor de retorno**
  - Similar a `read()`/`write()` → Número de caracteres escritos/leídos en el/del buffer o negativo (error).

Especificación de la práctica (IV)

API KIFS para los módulos

- El sistema KIFS, que requiere modificaciones del kernel, exporta las funciones `create_kifs_entry()` y `remove_kifs_entry()`
  - Permiten añadir/eliminar entradas desde módulos del kernel
    - Funcionalidad análoga a `proc_create()` y `remove_proc_entry()`
  - Necesario exportar funciones con `EXPORT_SYMBOL()`

```
struct kifs_entry* create_kifs_entry(const char* entryname,
                 struct kifs_operations* ops);
int remove_kifs_entry(const char* entry_name);
```

- **Parámetros**
  - `entryname`: nombre de la entrada a crear/eliminar
  - `ops`: Instancia de interfaz de operaciones de la entrada

KIFS vs. /proc

Propiedad	`/proc`	KIFS
Callbacks de lectura y escritura	Sí	Sí
Interfaz Extensible	Sí	Sí
Organización de entradas	Jerárquica (Árbol)	Lineal (Lista enlazada)
Entradas tienen presencia en sistema de ficheros	Sí	No
Puntero de posición asociado a entrada	Sí	No (No hay fichero asociado)
Invocación *callbacks*	Llamadas al sistema `read()` y `write()`. Alternativamente, podemos usar `echo` y `cat` desde el shell.	Llamada al sistema `kifs()`. No podemos usar `echo` y `cat`.

KIFS vs. /proc

Propiedad	`/proc`	KIFS
Callbacks de lectura y escritura	Sí	Sí
Interfaz Extensible	Sí	Sí
Organización de entradas	Jerárquica (Árbol)	Lineal (Lista enlazada)
Entradas tienen presencia en sistema de ficheros	Sí	No
Puntero de posición asociado a entrada	Sí	No (No hay fichero asociado)
Invocación *callbacks*	Llamadas al sistema `read()` y `write()`. Alternativamente, podemos usar `echo` y `cat` desde el shell.	Llamada al sistema `kifs()`. No podemos usar `echo` y `cat`.

- Se proporciona programa de usuario (kifs_invoke.c) para invocar `kifs()`
  - Modo de uso:
    - (Lectura): `./kifs_invoke -r <nombre_entrada>`
    - (Escritura): `./kifs_invoke -w <nombre_entrada> <cadena>`

Partes de la práctica

(Parte A.) Crear llamada al sistema "Hola Mundo"

- Seguir instrucciones del tema "Llamadas al Sistema"

(Parte B.) Implementación de KIFS

- Por simplicidad, KIFS se implementará en un módulo del kernel
  - Realiza procesamiento asociado a llamada al sistema `kifs()`
  - Implementa el API de KIFS (creación/eliminación de entradas KIFS)
    - Gestión de lista enlazada de entradas
  - Creará dos entradas KIFS: `list` y `clipboard`.
- Necesario modificar el kernel para incluir llamada al sistema `kifs()`

(Parte C.) Crear un módulo que haga uso de KIFS

- Gestiona una nueva entrada KIFS llamada `counter`
  - Escritura → Incremento de un contador (inicialmente a 0)
  - Lectura → se escribirá en el buffer del usuario el valor del contador

Implementación Parte B (I)

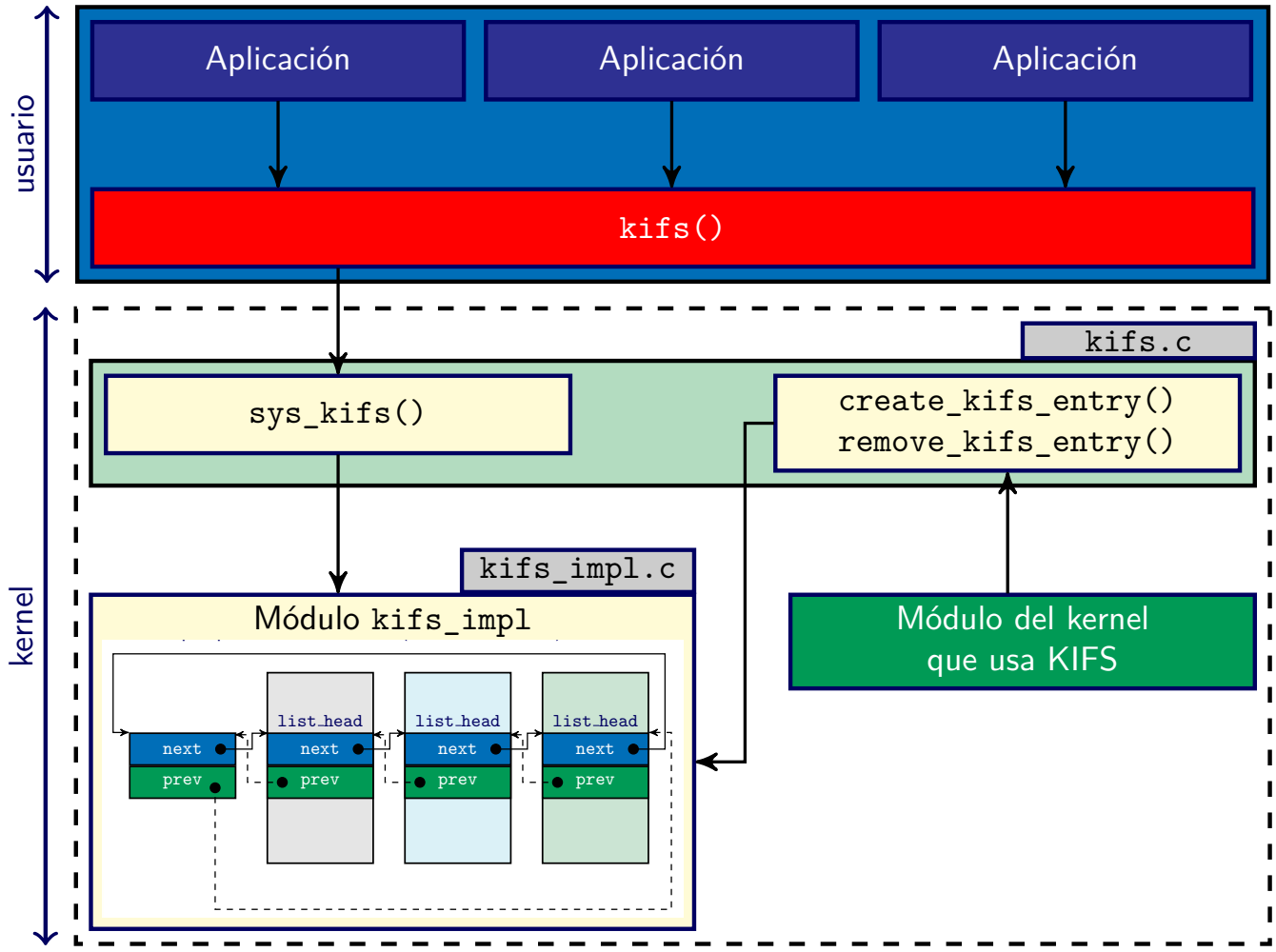
- Es posible implementar KIFS completamente en el kernel pero el proceso de desarrollo es tedioso
- Por cada fallo detectado:
  1. Modificar código del kernel
  2. Recompilar y reinstalar kernel
  3. Reiniciar la máquina

Implementación Parte B (II)

Estrategia de implementación con 2 componentes

1. Módulo del kernel `kifs_impl`
   - Implementa la mayor parte de la funcionalidad de KIFS
   - Gestiona lista enlazada de entradas `struct kifs_entry`
2. Código del kernel que actúa como intermediario entre `kifs_impl` y los programas de usuario y módulos del kernel que usen KIFS
   - Se proporciona el código (kifs.h y kifs.c)
     - Es preciso comprender su funcionamiento
   - Ofrece un API para registrar el módulo implementador `kifs_impl`
   - Invoca código de `kifs_impl` en las siguientes situaciones:
     - Programa de usuario invoca `kifs()`
     - Módulo del kernel desea crear/eliminar entrada KIFS

Visión global de KIFS



Implementación Parte B (III)

API para el módulo implementador (kifs.h)

```
/* El módulo kifs_impl debe instanciar una interfaz de este tipo */
struct kifs_impl_operations {
    struct kifs_entry* (*create_kifs_entry)(const char* entryname,
                                            struct kifs_operations* ops);
    int (*remove_kifs_entry)(const char* entry_name);
    long (*sys_kifs)(const char* entry_name, unsigned int op_mode,
                     char* user_buffer,unsigned int maxchars);
};

/* Función que permite registrar la implementación de KIFS
    (Invocar función al cargar el módulo)
 */
int register_kifs_implementation(struct kifs_impl_operations* ops);

/* Función que permite desregistrar la implementación de KIFS
    (Invocar función al descargar el módulo)
*/
int unregister_kifs_implementation(struct kifs_impl_operations* ops);
```

Implementación Parte B (IV)

Pasos a seguir

1. Realizar modificaciones en el código del kernel
   - Copiar kifs.h en ${KERNEL_SOURCE}/include/linux
   - Copiar kifs.c en ${KERNEL_SOURCE}/kernel
   - Modificar ${KERNEL_SOURCE}/kernel/Makefile para que compile kifs.c
   - Añadir llamada al sistema `kifs()` a la tabla de *syscalls*
2. Compilar el kernel modificado
3. Instalar paquetes (*image* y *headers*) en la máquina virtual y reiniciar
4. Implementar módulo `kifs_impl`
5. Compilar y cargar el módulo `kifs_impl`
6. Probar código usando programa `kifs_invoke`

Implementación del módulo `kifs_impl` (I)

Inicialización

1. Inicializar la lista enlazada de entradas KIFS
2. Registro de la implementación de KIFS
   - `register_kifs_implementation()`
3. Creación de entradas KIFS "por defecto"
   - `list`: (Sólo lectura) Imprime un listado de las entradas KIFS registradas en el sistema
     - Recorrido de la lista de entradas
     - Emula `ls` para KIFS
   - `clipboard`: (Lectura/Escritura) Comportamiento similar al ejemplo `clipboard.c` de la práctica 1
   - El módulo debe incluir *callbacks* de lectura y escritura asociadas a ambas entradas

Implementación del módulo `kifs_impl` (II)

Instanciación de interfaz en módulo (`kifs_impl.c`)

```
    ...
struct kifs_impl_operations ops = {
    .create_kifs_entry=my_create_kifs_entry,
    .remove_kifs_entry=my_remove_kifs_entry,
    .sys_kifs=my_sys_kifs,
};

static int __init init_module(void) {
    ...
    register_kifs_implementation(&ops);
    ...
}

static void __exit cleanup_module(void){
    ...
    unregister_kifs_implementation(&ops);
    ...
}
```

Implementación del módulo `kifs_impl` (III)

Implementación API KIFS (`kifs_impl.c`)

```
struct kifs_entry* my_create_kifs_entry(const char* entryname,
                  struct kifs_operations* ops) {...}
```

- Se invoca cuando otro módulo del kernel ejecuta `create_kifs_entry()`
- Crea entrada kifs con nombre `entryname` y la inserta en la lista enlazada

```
int my_remove_kifs_entry(const char* entry_name) {...}
```

- Se invoca cuando otro módulo del kernel ejecuta `remove_kifs_entry()`
- Elimina de la lista enlazada de entradas la entrada con nombre `entryname`

```
long my_sys_kifs(const char* entry_name, unsigned int op_mode,
       char* user_buffer, unsigned int maxchars) {...}
```

- Se invoca cuando programa de usuario invoca llamada al sistema `kifs()`

Implementación del módulo `kifs_impl` (IV)

Pseudocódigo de `my_sys_kifs()`

```
long my_sys_kifs(const char* entry_name, unsigned int op_mode, char*
     user_buffer, unsigned int maxchars) {
    copiar entryname al espacio de kernel:
       strncpy_from_user(e_name,entry_name,???);

    kifs_entry=Buscar entrada e_name en la lista enlazada de entradas;

    Si la entrada no existe
       -> return -EINVAL;

    Si la operación solicitada no está implementada para la entrada
       (kifs_entry->read == NULL o kifs_entry->write == NULL)
        -> return -EINVAL;

    Ejecutar la operación solicitada mediante el puntero a función y
     los parámetros correspondientes y devolver el resultado
     como valor de retorno de my_sys_kifs();
}
```

Implementación del módulo `kifs_impl` (V)

Contador de referencias del módulo

- No debemos permitir que módulo `kifs_impl` pueda descargarse si hay algún otro módulo cargado que haya creado una entrada KIFS
- Solución: Incrementar/decrementar contador de referencias (CR) del módulo cuando se crea/destruye una entrada
  - Incrementar CR: `try_module_get(THIS_MODULE);`
  - Decrementar CR: `module_put(THIS_MODULE);`

Descarga del módulo

1. Liberar memoria
2. Desregistro de la implementación de KIFS
   - `unregister_kifs_implementation()`

Parte B: Ejemplo de ejecución

terminal

```
kernel@debian:p2$ gcc -g kifs_invoke.c -o kifs_invoke
kernel@debian:p2$ ./kifs_invoke
Usage: ./kifs_invoke [-r|-w] <entry_name> [value]
kernel@debian:p2$ ./kifs_invoke -r list
Error when reading entry: 'list' (return value: -1)
Error message: Function not implemented
kernel@debian:p2$ sudo insmod KifsImpl/kifs_impl.ko
[sudo] password for kernel:
kernel@debian:p2$ ./kifs_invoke -r list
*** READING list ENTRY **
clipboard
list

*****
kernel@debian:p2$ ./kifs_invoke -w list prueba
Error when writing to entry: 'list' (return value: -1)
Error message: Invalid argument
```

Parte B: Ejemplo de ejecución (cont.)

terminal

```
kernel@debian:p2$ ./kifs_invoke -w clipboard Hoooola
kernel@debian:p2$ ./kifs_invoke -r clipboard
*** READING clipboard ENTRY **
Hoooola
*****
kernel@debian:p2$ ./kifs_invoke -r clipboard
*** READING clipboard ENTRY **
Hoooola
*****
kernel@debian:p2$ ./kifs_invoke -w clipboard test
kernel@debian:p2$ ./kifs_invoke -r clipboard
*** READING clipboard ENTRY **
test
*****
kernel@debian:p2$
```

Parte C: Ejemplo de ejecución

terminal

```
kernel@debian:p2$ sudo insmod ./KifsClient/kifs_client.ko
kernel@debian:p2$ ./kifs_invoke -r list
*** READING list ENTRY **
clipboard
list
counter

*****
kernel@debian:p2$ ./kifs_invoke -r counter
*** READING counter ENTRY **
0
*****
kernel@debian:p2$ ./kifs_invoke -w counter foo;  ./kifs_invoke -w counter foo
kernel@debian:p2$ ./kifs_invoke -r counter
*** READING counter ENTRY **
2
*****
kernel@debian:p2$ sudo rmmod kifs_client
kernel@debian:p2$ ./kifs_invoke -r list
*** READING list ENTRY **
clipboard
list

*****
```

Partes opcionales (I)

- **(Opcional 1)** Crear implementación de KIFS sin memoria dinámica
  - Por motivos de robustez de la implementación nos interesa evitar `vmalloc()` y `vfree()` para gestionar la memoria de las entradas KIFS de la lista
    - `vmalloc()` es bloqueante
    - No es posible invocar funciones bloqueantes desde algunos puntos del código del núcleo
  - En esta implementación, KIFS soportará hasta un máximo de 10 entradas cuya memoria se reservará de forma estática
    - *Pool* de entradas (*array* definido estáticamente)

```
#define MAX_KIFS_ENTRIES 10

struct kifs_entry pool[MAX_KIFS_ENTRIES];
```

Gestión del espacio libre (Opcional 1)

- Dos alternativas para la gestión de las entradas libres:
  1. Mapa de bits
     - Un bit por cada elemento del array de entradas (*pool*)
     - *unsigned int* $\rightarrow$ Cada bit valdrá 0 (libre) ó 1 (ocupada)
  2. Lista enlazada de entradas libres
     - Inicialmente lista de "libres" (10 elementos) y lista de "ocupadas" vacía

```
struct list_head free_list; /* Lista de entradas libres */
```

Partes opcionales (II)

- **(Opcional 2)** Extender KIFS del siguiente modo:
  - Añadir un campo `description` de tipo cadena de caracteres a `struct kifs_entry`
    - Este campo almacenará una descripción de la entrada kifs
    - `description` se inicializará adecuadamente tras la invocación de `create_kifs_entry()`
  - Modificar KIFS de tal forma que el campo `description` de una entrada específica pueda leerse desde el programa 'kifs_invoke'
    - Ej: `kifs_invoke -d <nombre_entrada>`
- **(Opcional 3)** Ampliar el módulo de la parte C para que cree una entrada KIFS llamada 'modlist' que tenga el mismo comportamiento que la entrada /proc de la Práctica 1

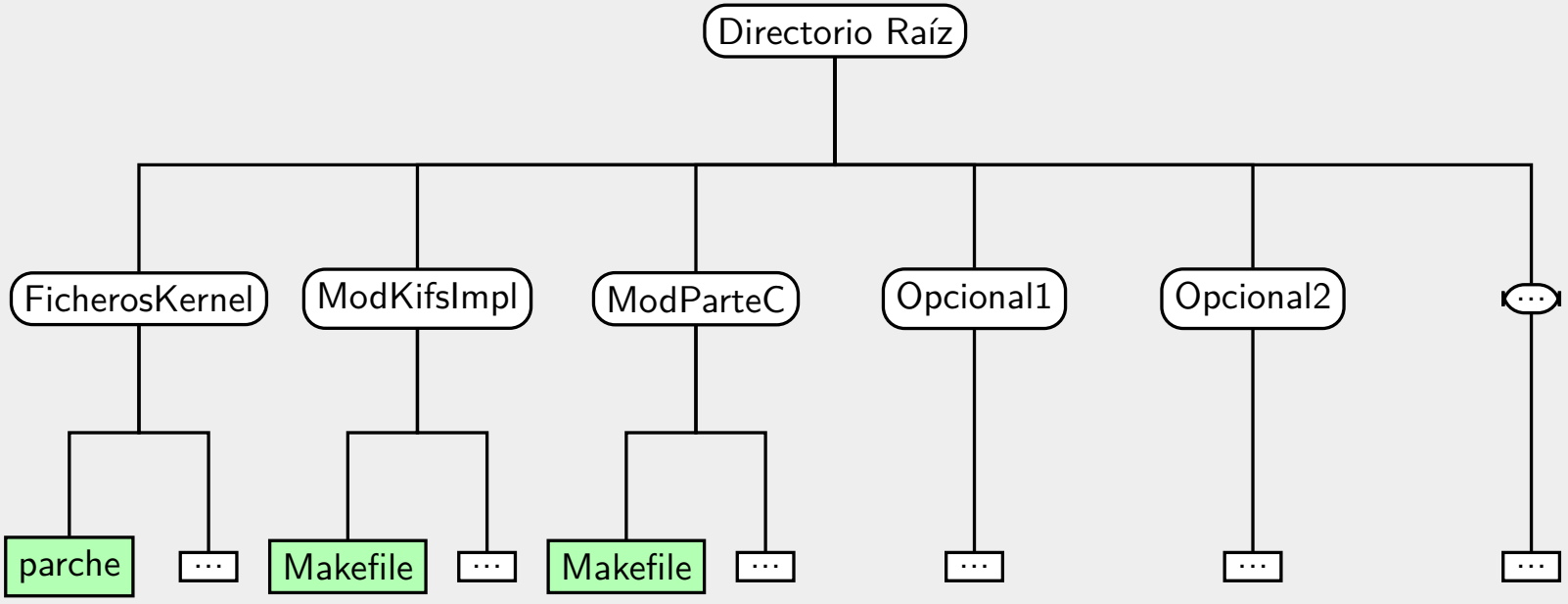
Partes opcionales (III)

- **(Opcional 4)** Implementar un módulo que cree *k* entradas KIFS con nombre <prefijo>0, <prefijo>1, ... ,<prefijo>k-1
  - Tanto el prefijo (cadena de caracteres) como el número de entradas a crear *k* se pasarán como parámetros al módulo
  - Cada entrada exhibirá el mismo comportamiento que la entrada 'counter' de la Parte C de la práctica y tendrá asociado un contador privado
  - Si no es posible crear *k* entradas por falta de espacio, el módulo eliminará las entradas creadas previamente antes de devolver un error en tiempo de carga

Entrega de la práctica

- A través del Campus Virtual
  - Hasta el 21 de noviembre
- Obligatorio mostrar el funcionamiento después de hacer la entrega

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



Licencia

LIN - Práctica 2: Llamadas al Sistema
Versión 0.1

Este documento (o uno muy similar) está disponible en
`https://cv4.ucm.es/moodle/course/view.php?id=49276`