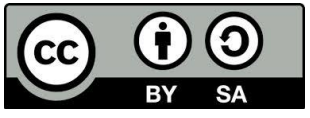
Práctica 5: Trabajos diferidos

LIN - Curso 2019-2020



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

Contenido

Contenido

Práctica 5: Trabajos diferidos

Objetivos

- Familiarizarse con:
  - Temporizadores del kernel
  - Mecanismos para diferir el trabajo en el kernel Linux
  - Uso avanzado de mecanismos de sincronización en el kernel

Contenido

Ejercicios (I)

Ejercicio 1

- Analizar el módulo `example_timer.c` que gestiona un temporizador que se activa cada segundo e imprime un mensaje con `printk()`

terminal 1

```
kernel@debian:~/Ejemplos$ sudo insmod example_timer.ko
```

terminal 2

```
kernel@debian:~$ sudo tail -f /var/log/kern.log
[sudo] password for kernel:
...
Jan  4 14:15:22 debian kernel: [233644.504010] Tic
Jan  4 14:15:23 debian kernel: [233645.524021] Tac
Jan  4 14:15:24 debian kernel: [233646.544028] Tic
Jan  4 14:15:25 debian kernel: [233647.564029] Tac
Jan  4 14:15:26 debian kernel: [233648.584021] Tic
Jan  4 14:15:27 debian kernel: [233649.604031] Tac
...
```

Ejercicios (II)

Ejercicio 2

- Estudiar la implementación de los módulos de ejemplo `workqueue1.c`, `workqueue2.c` y `workqueue3.c`, que ilustran el uso de las workqueues

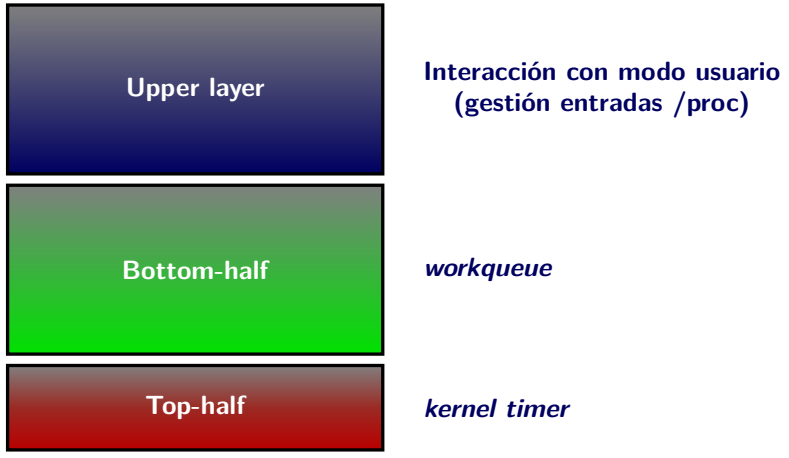
Contenido

Especificación de la práctica (I)

- Implementar un módulo del kernel (`modtimer`) que **genera una secuencia de números aleatorios**
  - Los números se generan periódicamente y se van insertando en una lista enlazada
- El módulo permitirá que un único programa de usuario **"consuma" los números de la lista leyendo de la entrada `/proc/modtimer`**
  - El programa se bloqueará si la lista está vacía al hacer una lectura
  - No se debe permitir que la entrada sea abierta por varios procesos simultáneamente
- El proceso de generación de números (gestionado mediante un temporizador) estará activo mientras un programa de usuario esté leyendo de la entrada /proc

Especificación de la práctica (II)

- El módulo constará de tres capas/componentes:
  1. **"Top Half"**: temporizador del kernel que al activarse genera un número aleatorio y lo inserta en un buffer circular acotado
     - No es un manejador de interrupción pero se ejecuta en contexto de interrupción → No es posible invocar funciones bloqueantes
  2. **Bottom Half**: Tarea diferida que transfiere los enteros del buffer circular a la lista enlazada (vacía el buffer)
  3. **Upper Layer**: Implementación de operaciones asociadas a las entradas /proc exportadas por el módulo



Especificación de la práctica (III)

"Top Half" (temporizador)

- 3 parámetros configurables (var. globales modificables vía /proc):
  1. `timer_period_ms`
  2. `max_random`
  3. `emergency_threshold`
- Temporizador se activa cada `timer_period_ms` milisegundos
- Cada vez que se invoque la función del *timer* se generará un número aleatorio *n* y se insertará en un buffer circular
  - $n \in \{0..max\_random - 1\}$
    - Usar función `get_random_int()` que devuelve número aleatorio
    - Declarada en `<linux/random.h>`
  - Usar la implementación buffer circular de bytes del kernel (`struct kfifo`)
    - 1 entero sin signo $\rightarrow$ 4 bytes
    - Capacidad máxima del buffer: 8 enteros (32 bytes)

Especificación de la práctica (III)

"Top Half" (cont.)

- Cuando el buffer de enteros haya alcanzado un cierto grado de ocupación → activar *tarea de vaciado* del buffer (*bottom half*)
  - Parámetro `emergency_threshold` indica el porcentaje de ocupación que provoca la activación de dicha tarea
  - La tarea diferida se debe activar con `queue_work()`, y usando una workqueue privada del módulo del kernel (`create_workqueue()`)
  - No se debe planificar la tarea diferida de nuevo hasta que no se haya completado la ejecución de la última tarea planificada y el umbral de emergencia se vuelva a alcanzar

Especificación de la práctica (IV)

Bottom Half

- Tarea (`struct work_struct`) que se encolará en una workqueue privada del módulo del kernel
- La función asociada a la tarea volcará los datos del buffer a la lista enlazada de enteros
  1. Se extraerán todos los elementos del buffer circular (vacía buffer)
  2. Se ha de reservar memoria dinámica para los nodos de la lista vía `vmalloc()`
  3. Si el programa de usuario está bloqueado esperando a que haya elementos en la lista, la función le despertará
- Esta función se ejecuta en contexto de proceso en modo kernel
  - Un *kernel thread* se encarga de ejecutarla
  - Es posible invocar funciones bloqueantes siempre y cuando no se haya adquirido un *spin lock*

Especificación de la práctica (V)

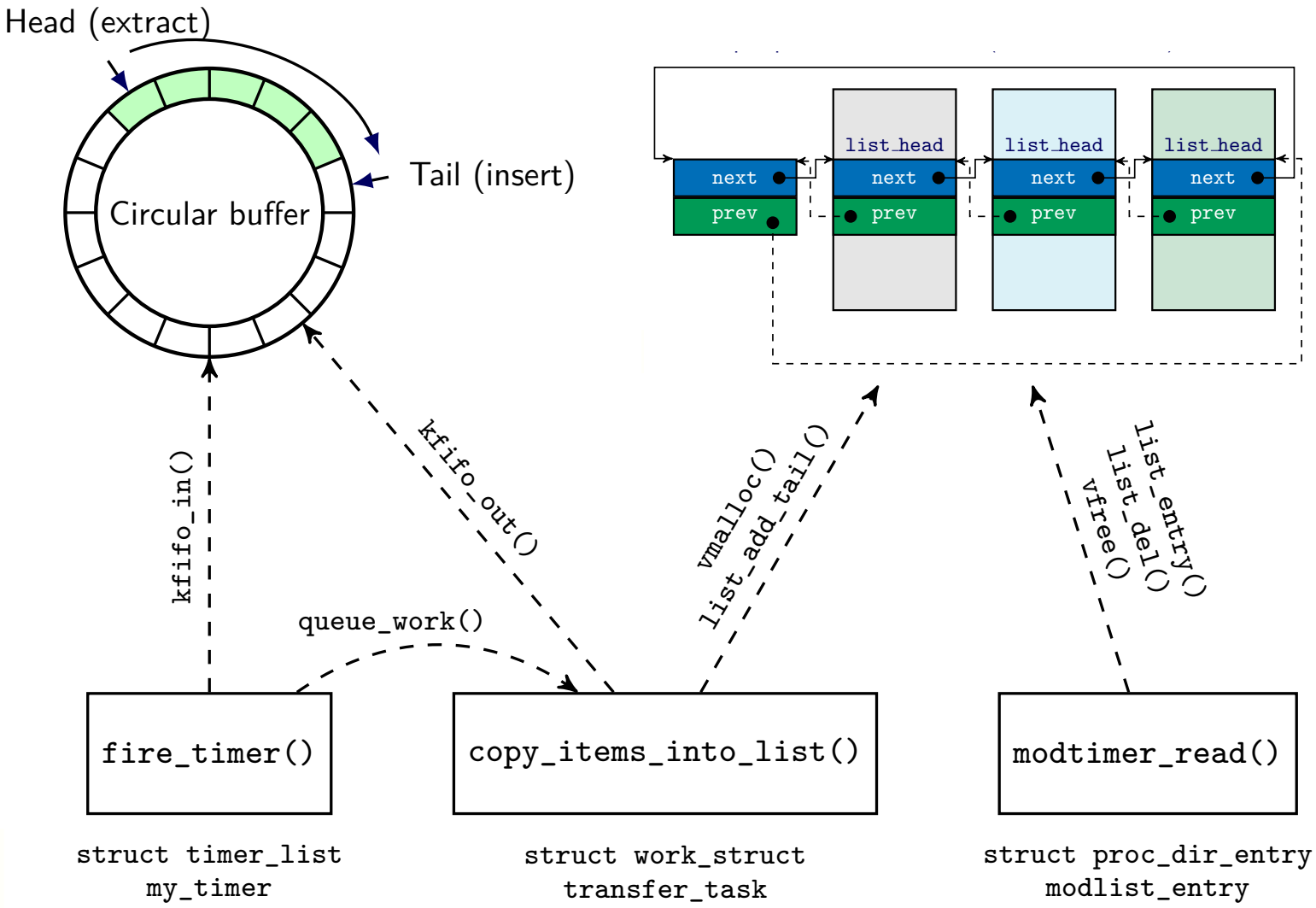
Upper Layer

- Código del módulo que implementa las operaciones sobre entradas /proc del módulo
- Dos entradas /proc:
  1. `/proc/modconfig`: permite cambiar/consultar valor de parámetros de configuración `timer_period_ms`, `emergency_threshold` y `max_random`

```
kernel@debian:~$ echo timer_period_ms 500 > /proc/modconfig
kernel@debian:~$ cat /proc/modconfig
timer_period_ms=500
emergency_threshold=75
max_random=300
```

  2. `/proc/modtimer`: Al leer de esta entrada (`cat`) se consumen elementos de la lista enlazada de enteros
     - Implementar operaciones `open()`, `release()` y `read()`

Implementación



Implementación

Necesarios 3 recursos de sincronización

1. El buffer debe protegerse mediante un *spin lock*
   - Accedido desde *timer* (cont. interrupción) y otras funciones que se ejecutan en contexto de proceso
   - Necesario deshabilitar interrupciones antes de adquirir el *spin lock*
     - Emplear `spin_lock_irqsave()` y `spin_unlock_irqrestore()`
2. La lista enlazada puede protegerse usando *spin lock* o semáforo
   - Siempre se accede a la lista desde funciones que se ejecutan en contexto de proceso
   - Por simplicidad/flexibilidad, se recomienda usar un semáforo
3. Se hará uso de un semáforo (cola de espera) para bloquear al programa de usuario mientras la lista esté vacía
   - Actúa como cola de una "variable condición" que tiene como "mutex" asociado el spin lock o semáforo de la lista enlazada

Implementación

Comentarios adicionales

- El módulo no debe poder descargarse mientras programa de usuario esté usando sus funciones
  - Incrementar el contador de referencias (CR) del módulo cuando programa haga `open()` y decrementarlo al hacer `close()`
    - Incrementar CR ⇒ `try_module_get(THIS_MODULE);`
    - Decrementar CR ⇒ `module_put(THIS_MODULE);`
- Realizar las siguientes acciones cuando el programa de usuario cierre `/proc/modtimer`
  1. Desactivar el temporizador con `del_timer_sync()`
  2. Esperar a que termine todo el trabajo planificado hasta el momento en la workqueue creada
  3. Vaciar el buffer circular
  4. Vaciar la lista enlazada (liberar memoria)
  5. Decrementar contador de referencias del módulo

Ejemplo de ejecución

terminal1

```
kernel@debian:~$ sudo insmod modtimer.ko
kernel@debian:~$ cat /proc/modconfig
timer_period_ms=500
emergency_threshold=75
max_random=300
kernel@debian:~$ cat /proc/modtimer
61
176
74
298
87
221
100
235
114
249
128
7
141
20
^C
```

Ejemplo de ejecución (cont.)

terminal2

```
kernel@debian:~$ sudo tail -f /var/log/kern.log
....
Jan  3 18:13:30 kernel kernel: [161532.116030] Generated number: 61
Jan  3 18:13:30 kernel kernel: [161532.652020] Generated number: 176
Jan  3 18:13:31 kernel kernel: [161533.192020] Generated number: 74
Jan  3 18:13:31 kernel kernel: [161533.728018] Generated number: 298
Jan  3 18:13:32 kernel kernel: [161534.268018] Generated number: 87
Jan  3 18:13:32 kernel kernel: [161534.804023] Generated number: 221
Jan  3 18:13:33 kernel kernel: [161535.924681] 6 items moved from the buffer to the
Jan  3 18:13:34 kernel kernel: [161536.424019] Generated number: 114
Jan  3 18:13:34 kernel kernel: [161536.964019] Generated number: 249
Jan  3 18:13:35 kernel kernel: [161537.504026] Generated number: 128
Jan  3 18:13:36 kernel kernel: [161538.044024] Generated number: 7
...
```

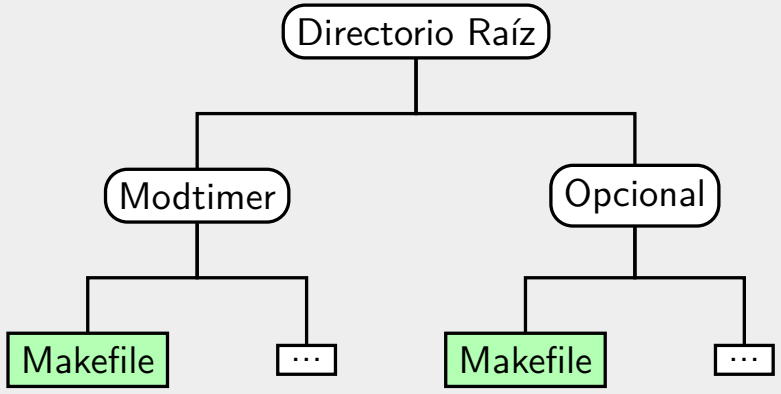
Parte opcional

- Implementar una versión alternativa de la práctica en la cual los números pares generados se inserten en una lista enlazada y los impares en otra
  - El módulo permitirá que dos programas de usuario abran la entrada `/proc/modtimer`
  - El primer proceso en abrir la entrada procesará los números pares y el segundo los impares
    - Pista: Para poder distinguir entre ambos procesos puede modificarse el campo `private_data` de `struct file` al abrir el fichero
  - La secuencia de números comenzará a generarse cuando ambos procesos hayan abierto la entrada /proc
- En la parte opcional, el trabajo diferido (tarea de vaciado del buffer) se encolará en una worqueue privada del módulo

Entrega de la práctica

- A través del Campus Virtual
  - Hasta el 19 de diciembre (Test: 18 de diciembre)
  - No se permitirán entregas tardías
- Es aconsejable mostrar el funcionamiento antes de hacer la entrega

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



Licencia

LIN - Práctica 5: Trabajos diferidos
Versión 1.2

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