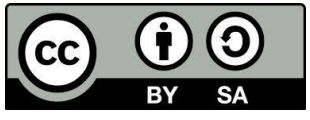
Práctica 3: Sincronización

Arquitectura Interna de Linux - 2016



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

Contenido

1 Introducción

2 Práctica

Práctica

Dos partes:

- **(Parte A)** Implementación *SMP-safe* de la Práctica 1B usando *spin locks*
  - Se ha de garantizar exclusión mutua entre las distintas regiones de código que acceden a la lista enlazada de enteros (estructura compartida)
- **(Parte B - Opcional)** Implementar el productor/consumidor con /proc presentado en el tema de sincronización
  - Sincronización gestionada mediante semáforos

Consideraciones adicionales Parte A

Algunas consideraciones importantes

- No es posible invocar funciones bloqueantes como `vmalloc()` dentro de `spin_lock()` y `spin_unlock()`
- Controlar situaciones de desbordamiento de buffer
  - ¿Qué ocurre si se intenta leer de la lista cuando ésta tiene muchos elementos?
- Para garantizar que el código funciona correctamente, se recomienda preparar scripts BASH (lanzados desde distintos terminales) para realizar operaciones simultáneas sobre la lista enlazada
  - Ejemplo:
    - *Terminal 1*: Se insertan números del 1 al *n* con un cierto retardo entre cada inserción
    - *Terminal 2*: Se consumen/eliminan números del 1 al *n* con un cierto retardo entre cada eliminación. Imprimir contenido de la lista tras eliminar un elemento.

Parte B: Especificación

Especificación

- Crear módulo del kernel que gestiona un buffer circular de enteros
  - Se proporciona implementación de buffer circular de enteros: tipo de datos `cbuffer_t` (cbuffer.c y cbuffer.h)
  - Crear un buffer con capacidad máxima de 5 enteros
- Se podrá alterar el estado del buffer circular realizando lecturas/escrituras en la entrada /proc/prodcons, gestionada por el módulo:
  - Para insertar al final del buffer: `$ echo <num> > /proc/prodcons`
  - Para extraer primer elemento del buffer: `$ cat /proc/prodcons`
- Las operaciones de inserción/eliminación del buffer tendrán semántica productor/consumidor
  1. Un proceso que intente insertar en buffer lleno se bloquea hasta que haya hueco
  2. Proceso que consume de buffer vacío se queda bloqueado hasta que se inserte algún elemento

Parte B: Ejemplo de ejecución

terminal

```
kernel@debian:~/ProdCons1$ echo 4 > /proc/prodcons
kernel@debian:~/ProdCons1$ echo 5 > /proc/prodcons
kernel@debian:~/ProdCons1$ echo 6 > /proc/prodcons
kernel@debian:~/ProdCons1$ cat /proc/prodcons
4
kernel@debian:~/ProdCons1$ cat /proc/prodcons
5
kernel@debian:~/ProdCons1$ cat /proc/prodcons
6
kernel@debian:~/ProdCons1$ cat /proc/prodcons
<<proceso se queda bloqueado>>
```

Parte B: Implementación

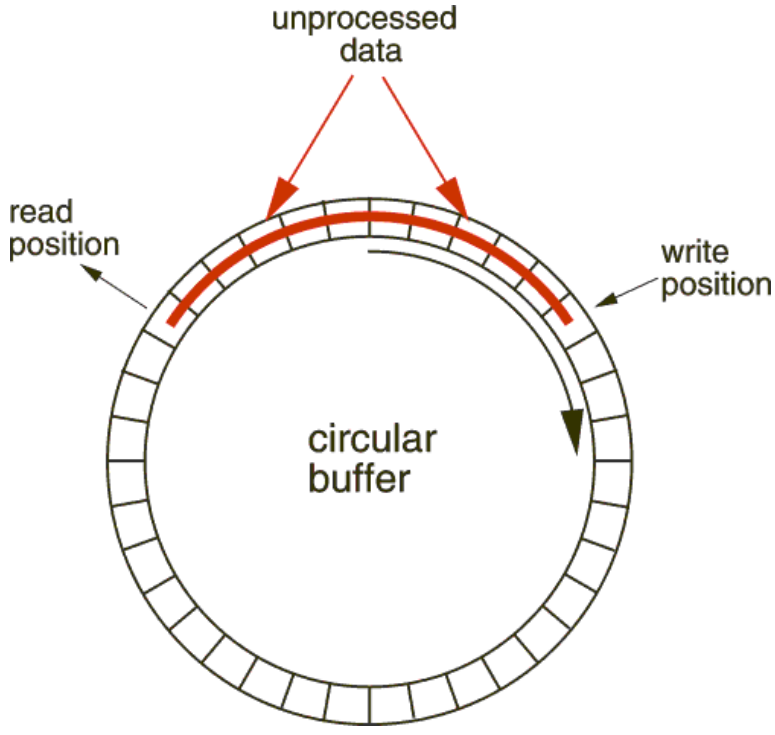
- Módulo constará de los siguientes ficheros fuente:
  1. prodcons.c: Fichero principal
  2. cbuffer.h: Declaración del tipo `cbuffer_t` y operaciones sobre el mismo
  3. cbuffer.c: Implementación de las operaciones del tipo `cbuffer_t`
- Necesario crear `Makefile` para compilar módulo que consta de varios ficheros .c

Makefile

```
obj-m =  prodconsmod.o  #prodconsmod.c no ha de existir
prodconsmod-objs =  prodcons.o cbuffer.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
```

Parte B: Buffer circular

- El tipo de datos `cbuffer_t` implementa **buffer circular de enteros**
  - Las inserciones en el buffer circular se realizan siempre al final
  - La cabeza del buffer (*head*) es el extremo de lectura, por donde se extraen los elementos



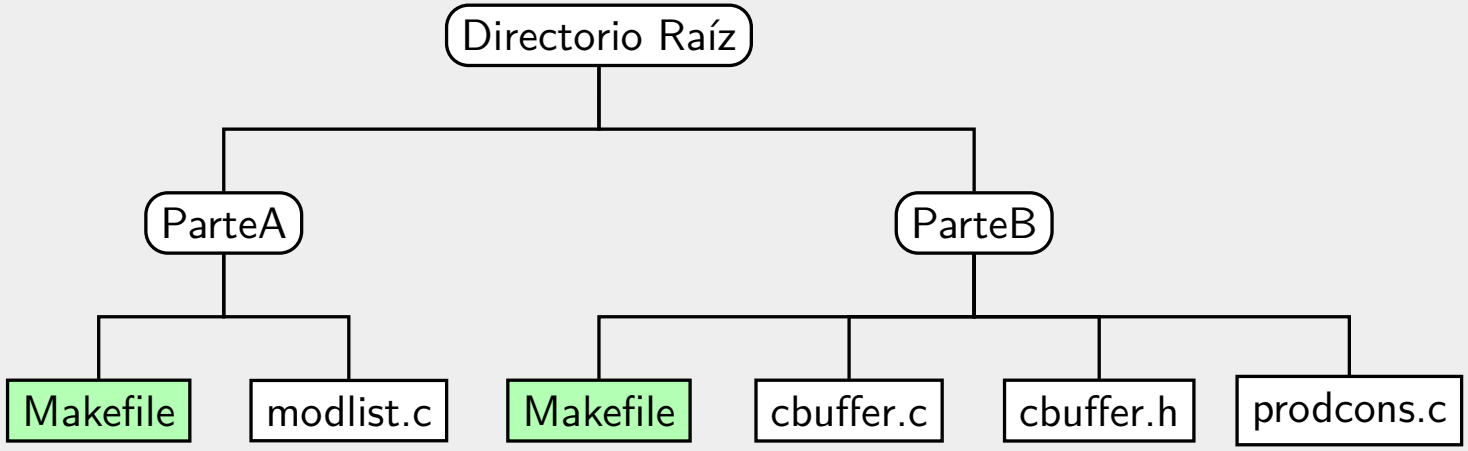
Parte B: *cbuffer_t*

```
typedef struct {
    int* data;        /* int vector */
    unsigned int head;   /* Index of the first element in [0 .. max_size-1] */
    unsigned int size;   /* Current Buffer size // size in [0 .. max_size] */
    unsigned int max_size;  /* Buffer max capacity */
}cbuffer_t;

/* Creates a new cbuffer (takes care of allocating memory) */
cbuffer_t* create_cbuffer_t (unsigned int max_size);

/* Release memory from circular buffer */
void destroy_cbuffer_t ( cbuffer_t* cbuffer );

/* Returns a non-zero value when buffer is full */
int is_full_cbuffer_t ( cbuffer_t* cbuffer );

/* Returns a non-zero value when buffer is empty */
int is_empty_cbuffer_t ( cbuffer_t* cbuffer );

/* Inserts an item at the end of the buffer */
void insert_cbuffer_t ( cbuffer_t* cbuffer, int new_item );

/* Removes the first element in the buffer and returns a copy of it */
int remove_cbuffer_t ( cbuffer_t* cbuffer);
```

Entrega de la práctica

- A través del Campus Virtual

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



Licencia

Arquitectura Interna de Linux - Práctica 3: Sincronización
Versión 0.3

Este documento (o uno muy similar) está disponible en
`https://cv4.ucm.es/moodle/course/view.php?id=70009`