Diseño de Sistemas Operativos

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

Planificación de procesos

David del Río Astorga

Febrero, 2018

Índice

1. Introducción

El presente documento representa el cuaderno de prácticas para el desarrollo de **un planificador de procesos en C**.

Forma parte de las prácticas de la asignatura Diseño de Sistemas Operativos que se imparten en el Grado en Ingeniería Informática y en el Doble grado en Ingeniería Informática y ADE, que se imparten en la Universidad Carlos III de Madrid.

Para comprobar el aprendizaje y asimilación, al final del documento se indica el material que se ha de entregar y la forma de entrega.

La práctica va a consistir en la creación de varias políticas de planificación de procesos, para ello el estudiante debe implementar dos algoritmos de planificación a nivel de usuario para un grupo de hilos.

2. Código base

Al descomprimir el archivo *p1_planificador.zip*, adjunto con el material en Aula Global, aparecerán los siguientes archivos:

- *main.c*
  Programa principal de ejemplo que crea un número determinado de threads y llama al planificador implementado en mythreadlib.c

- *mythread.h*
  Fichero de cabeceras que contiene las definiciones de las funciones necesarias para manejar los threads creados y la estructura TCB

- *mythreadlib.c*
  Implementación de la librería de hilos definida y un planificador FIFO.
  El estudiante debe usar como base este archivo para realizar las distintas políticas de planificación. Se deben rehacer las funciones scheduler y activator. Así mismo deberán realizarse las llamadas a ambas funciones donde corresponda como, por ejemplo, en la interrupción de reloj.

  ```
  TCB* scheduler(){
    ...
  }
  void activator(TCB *next){
    ...
  }
  ```

  La función scheduler deberá devolver el siguiente thread a ejecutar mientras que la función activator deberá realizar el cambio de contexto correspondiente.

- *interrupt.c* e *interrupt.h*
  Fichero de cabeceras e implementación de funciones para el manejo de las interrupciones.

```
void init_interrupt();
void enable_interrupt();
void disable_interrupt();
```

- *queue.c* y *queue.h*
  Fichero de cabeceras e implementación de funciones para la creación y manejo de procesos.

```
struct queue* queue_new();
struct queue* enqueue (struct queue*, void *);
void* dequeue (struct queue*);
int queue_empty (struct queue* s);
```

A continuación se muestra un ejemplo de creación, inserción y obtención de un elemento en una cola:

```
//Creates an empty queue
struct queue *q = queue_new();
//Take the first element from the TCB table
TCB *t = &t_state[0];
//insert a TCB into the queue
enqueue(q, t);
// remove a TCB from the queue
TCB s* = dequeue(q);
```

3. Trabajo a realizar

3.1. Políticas de planificación

El estudiante debe realizar los siguientes planificadores:

3.1.1. Round-Robin

El estudiante debe realizar un planificador cuya política de planificación sea Round-Robin en el cual cada hilo se ejecutará un número de ticks definido dando paso al siguiente hilo listo para ejecutar al finalizar su rodaja. Los requisitos de este planificador serán los siguientes:

1. El número de ticks restantes vendrá definido en la variable *ticks* de la tabla TCB de cada thread cuyo valor inicial será el valor indicado en la macro QUANTUM_TICKS definida en el fichero mythread.h.
2. Al finalizar su rodaja el thread actual deberá ser expulsado del procesador restableciendo su valor de ticks a QUANTUM_TICKS dando paso al siguiente thread a ejecutar.
3. El fichero resultante de esta planificación será **RR.c**

3.1.2. Round-Robin/FIFO con prioridades

El estudiante debe realizar un planificador cuya política de planificación sea Round-Robin para los hilos de prioridad baja y una política de planificación FIFO para los hilos de prioridad alta. Los requisitos de este planificador serán los siguientes:

1. Cuando un proceso de prioridad alta entra en el sistema:
   - Si el proceso en ejecución es de baja prioridad, este deberá ser expulsado, en favor del nuevo hilo introducido en el sistema, y se insertará al final de la lista de hilos listos de prioridad baja restableciendo su valor de ticks a QUANTUM_TICKS.
   - Si el proceso en ejecución es de prioridad alta, el nuevo proceso se almacenará en la lista de hilos de prioridad alta y esperará a que llegue su turno de ejecución.
2. Cuando un proceso de prioridad alta entra en ejecución deberá ejecutarse desde el principio hasta el fin.
3. Cuando no existan hilos de prioridad alta, se procederá a ejecutar el planificador Round-Robin como el descrito en el primer punto.
4. El fichero resultante de esta planificación será **RRF.c**

3.1.3. Round-Robin/FIFO con cambios de contexto voluntarios

El último planificador a desarrollar en esta práctica es una extensión del planificador round-robin/FIFO con prioridades desarrollado anteriormente. Este planificador deberá introducir una funcionalidad extra que permite a un thread realizar un cambio de contexto voluntario mediante la llamada al sistema *read_network*. Los requisitos de este planificador serán los siguientes:

1. El thread que realiza la llamada *read_network* deberá liberar la CPU e introducirse en una nueva cola de threads en espera.

2. Deberá implementarse la función *network_interrupt()*, similar a la función de interrupción de reloj, que simula una tarjeta de red la cual recibe un paquete cada segundo. Al llegar un paquete nuevo de red, el primer hilo en la cola de espera deberá pasar a la cola de listos correspondiente a su prioridad. Si no hay ningún hilo en espera deberá descartarse el paquete.

3. Cuando no exista ningún hilo listo para ejecutar pero si en espera, se deberá poner en ejecución el thread *idle*. Este thread ejecutará un bucle infinito de tal forma que el planificador consulte cada tick de reloj si existe algún thread listo para ejecutar.

4. El id de este thread *idle* siempre sera -1 y no deberá ser introducido en ninguna cola.

5. El fichero resultante de esta planificación será **RRFN.c**

3.2. Formato de salida

En cada ejecución se deberán utilizar **únicamente** las siguientes impresiones por consola:

- Al cambiar de contexto entre dos threads que aún no han finalizado

```
***  SWAPCONTEXT FROM <origen> TO <destino>
```

- Cambio de contexto por finalización de un thread

```
***  THREAD <finalizado> TERMINATED: SETCONTEXT OF <destino>
```

- Expulsión de un proceso de prioridad baja para ejecutar un proceso de prioridad alta

```
***  THREAD <expulsado> PREEMTED: SETCONTEXT OF <destino>
```

- Al realizar un cambio de contexto voluntario haciendo uso de la funcion *read_network*

```
*** THREAD <current> READ FROM NETWORK
```

- Al mover un thread de la cola de espera a la cola correspondiente

```
*** THREAD <ready> READY
```

- Al realizar un cambio de contexto desde el thread idle a un thread listo para ejecutar

```
*** THREAD READY: SET CONTEXT TO <ready>
```

- Finalización de un thread

```
***  THREAD <finalizado> FINISHED
```

- Finalización del planificador, ya no existen más threads pendientes.

```
*** FINISH
```

La impresión por consola de finalización de un thread ya está incluida en el código base. El resto deben ser incorporadas por el estudiante tal y como se describen, donde:

- *origen* es el identificador del thread que ha finalizado su rodaja de tiempo.
- *destino* es el identificador del thread que inicia su rodaja de tiempo.
- *finalizado* es el identificador del thread que deja de ejecutarse por la finalización del mismo.
- *expulsado* es el identificador del thread que se expulsa para dejar paso a otro de mayor prioridad.
- *current* es el identificador del thread actual.
- *ready* es el identificador del thread listo para ejecutar.

NOTA: La salida por patalla de cualquier ejecución no puede contener más trazas que las descritas en este punto.

3.3. Requisitos extras de implementación

A continuación se detallan una serie de aspectos **importantes e imprescindibles** a la hora de implementar los planificadores descritos.

1. Para el manejo de los threads se deben crear colas de hilos. **No está permitido trabajar directamente con el array t_state**. Para ello se proporcionan las funciones implementadas en queue.c y queue.h.

2. Deben protegerse los accesos a las colas. Para ello se **debe utilizar las funciones enable_interrupt y disable_interrupt**.

3. Los cambios de contexto sólo se realizarán con las funciones indicadas para ello:

   - Setcontext
   - Makecontext
   - Getcontext
   - Swapcontext

4. La función *scheduler()* sólo puede ser invocada desde los manejadores de señales (*timer_interrupt()* y *network_interrupt()*), en la finalización de un thread (*mythread_exit*), para la expulsión de threads de menor prioridad por threads de alta prioridad (*mythread_create*) y para los cambios de contexto voluntarios (*read_network*).

5. **La salida por pantalla de cualquier ejecución no puede contener más trazas que las descritas en el punto 3.2**

6. No se realizarán cambios de contexto cuando el proceso inicial y final sea el mismo.

7. Se debe incluir código de control de errores.

4. Puntuación de la práctica

La evaluación de la práctica se va a dividir en dos partes.

- **Código (7 puntos)**
  - Round-Robin (*2 puntos*)
  - Round-Robin/FIFO con prioridades (*2 puntos*)
  - Round-Robin/FIFO con cambios de contexto voluntarios (*3 puntos*)
- **Memoria (3 puntos)**
  - Diseño usado en el código (*1.2 puntos*)
  - Batería de pruebas (*1.5 puntos*)
  - Presentación (*0.3 puntos*)

5. Aspectos Importantes

5.1. Normas Generales

1. Las prácticas que no compilen o no se ajusten a la funcionalidad planteada, **obtendrán una calificación de 0**.
2. Las prácticas que tengan *warnings* **serán penalizadas**.
3. Un programa no comentado, **obtendrán una calificación de 0**.
4. La entrega de la práctica se realizará a través de los entregadores habilitados. **No se permite la entrega a través de correo electrónico**.
5. Se prestará especial atención a detectar funcionalidades copiadas entre dos prácticas. En caso de detectar copia, ambos grupos **perderán la evaluación continua**.
6. **La práctica debe funcionar en los ordenadores de las aulas Linux (4.0.F16, 4.0.F18, 2.2.C05, 2.2.C06, 1.1.A16 y 1.1.A14) o en guernika.**

5.2. Memoria de la práctica

La memoría tendrá que contener al menos los siguientes apartados:

- **Portada** donde figuren los autores incluyendo nombre completo, NIA y dirección de correo electrónico de cada uno de ellos.
- **Índice de contenidos**
- **Descripción del código** detallando las principales funciones implementadas. NO incluir código fuente de la práctica en este apartado.
- **Batería de pruebas** utilizadas y resultados obtenidos. Se dará mayor puntuación a pruebas avanzadas, casos extremos y en general a aquellas pruebas que garanticen el correcto funcionamiento de la práctica en todos los casos. Hay que tener en cuenta:
  - Que el programa compile correctamente y sin *warnings* no es garantía de que funcione correctamente.
  - Deben evitarse pruebas duplicadas o que evalúan los mismos flujos de programa. La puntuación de este apartado no se mide en función del número de pruebas, sino del grado de cobertura de las mismas. Es mejor pocas pruebas que evalúen diferentes casos a muchas que evalúen siempre el mismo caso.
- **Conclusiones**, problemas encontrados, cómo se han solucionado, y opiniones personales.

Se puntuarán también los siguientes aspectos relativos a la **presentación**:

- La memoria debe tener números de página en todas las páginas (menos en la portada).
- El texto de la memoria debe estar justificado.

IMPORTANTE: La longitud de la memoria no deberá superar 10 páginas (portada e índice incluidos)

6. Entrega

6.1. Plazo de entrega

La fecha límite de entrega de la práctica se podrá consultar en Aula Global.

6.2. Procedimiento de entrega

La entrega de la práctica ha de realizarse de forma electrónica. En Aula Global se habilitarán unos enlaces a través de los cuales podrá realizar la entrega de la práctica. En concreto, **se habilitará un entregador para el código de la práctica y otro de tipo TURNITIN para la memoria de la práctica**.

IMPORTANTE: Un miembro del grupo deberá enviar un correo electrónico antes del día 3 de Marzo indicando: miembros que componen el grupo indicando nombre, NIA y grupo de prácticas al que pertenecen. Si los miembros que conforman el grupo pertenecen a grupos reducidos diferentes se les asignará el grupo en el que deberán asistir a las clases de prácticas con el fin de garantizar los recursos suficientes para todos los estudiantes.

6.3. Documentación a entregar

Código. Se debe entregar un archivo comprimido en formato zip con el nombre
dssoo_p1_AAAAAAAAA_BBBBBBBBB_CCCCCCCCC.zip
donde A...A, B...B y C...C son los NIA de los integrantes del grupo. El archivo zip se entregará en el entregador correspondiente al código de la práctica y debe contener:

- **Makefile**
- **main.c**
- **mythread.h**
- **mythreadlib.c**
- **interrupt.c, interrupt.h**
- **queue.c, queue.h**
- **RR.c**
- **RRF.c**
- **RRFN.c**

Memoria. La memoria se entregará en formato PDF en un fichero llamado **dssoo_p1_AAAAAAAAA_BBBBBBBBB_CCCCCCCCC.pdf**.
Solo se corregirán y calificarán memorias en formato PDF.
El archivo PDF se entregará en el entregador correspondiente (TURNITIN).