Diseño de Sistemas Operativos

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

Planificación de procesos

Florin Isaila
Carlos Gómez Carrasco

Febrero, 2015

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 práctivas de la asignatura Diseño de Sistemas Operativos del Grado en Ingeniería Informática que se imparte 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 alumno 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, crea un número determinado de threads y llama al planificador implementado en mythreadlib.c

- *mythread.c*
  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 alumno debe usar como base este archivo para realizar las distintas políticas de planificación. Se debe rehacer la función scheduler e invocarla desde la interrupción de reloj.

- *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 alumno debe realizar los siguientes planificadores:

3.1.1 Round-Robin

El alumno debe realizar un planificador cuya política de planificación sea Round-Robin. Cada hilo se ejecutará un número de ticks marcado en la variable ticks de la tabla TCB de cada thread. Esta variable debe ser inicializada al valor QUANTUM_TICKS al crear el hilo. Cuando un thread finalice su rodaja, será expulsado del procesador y restablecido el valor de su variable ticks a QUANTUM_TICKS.

- El fichero resultante de esta planificación será **RR.c**

3.1.2 Round-Robin/FIFO con prioridades

El alumno debe realizar un planificador cuya política de planificación sea Round-Robin para los procesos de prioridad baja, y una política de planificación FIFO para los procesos de prioridad alta.
Cuando un proceso de prioridad alta entra en el sistema y el proceso en ejecución es de prioridad baja éste debe ser expulsado, al ser expulsado se insertará al final de la lista de procesos listos de prioridad baja y se volverá a restablecer su valor de ticks. Si el proceso en ejecución es de prioridad alta, el nuevo proceso se almacenará en la lista de procesos de prioridad alta y esperará a que llegue su turno de ejecución. Cuando un proceso de prioridad alta entra en ejecución deberá ejecutarse desde el principio hasta el fin. Cuando no existan proceso de prioridad alta, se procederá a ejecutar el planificador Round-Robin como el descrito en el primer punto.

- El fichero resultante de esta planificación será **RRF.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 <idestino>
```

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

```
$ THREAD <expulsado> EJECTED: SETCONTEXT OF <destino>
```

- 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 (número 4) ya está incluida en el código base. El resto deben ser incorporadas por el alumno 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, y *expulsado* es el identificador del thread que se expulsa para dejar paso a otro de mayor prioridad.

3.3 Requisitos 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 procesos. **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 se realizarán con las funciones indicadas para ello:

   - Setcontext
   - Makecontext
   - Getcontext
   - Swapcontext

4. La función *scheduler()* sólo puede ser invocada desde el manejador de la señal *timer_interrupt()* y *mythread_exit*.

5. Cuando un thread termina debe reiniciar el valor de QUANTUM_TIME al valor inicial con la función reset_timer.

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

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

4 Procedimiento de evaluación

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

- **Código (8 puntos)**
  - Round-Robin (*4 puntos*)
  - Round-Robin/FIFO con prioridades (*4 puntos*)

- **Memoria (2 puntos)**

5 Aspectos Importantes

5.1 Normas Generales

1. Las prácticas que no compilen o no se ajusten a la funcionalidad y requisitos planteados, **obtendrán una calificación de 0**.
2. Las prácticas que tengan *warnings* o no estén comentadas **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 sin autorización previa**.
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 y 2.2.C06) 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).
- **Índice de contenidos**
- **Descripción del código** detallando las principales funciones implementadas. NO incluir códio fuente de la práctiva 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.
  - Evite pruebas duplicadas que evalúan los mismo 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 evalúan diferentes casos, a muchas que evalúan 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.

***NOTA*: NO DESCUIDE LA CALIDAD DE LA MEMORIA DE SU PRÁCTICA**

La calidad de la memoria es tan importante para aprobar la práctica como el correcto funcionamiento de la misma.

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**.

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...B, 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**

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).