DISEÑO DE SISTEMAS OPERATIVOS

2017/18

PLANIFICACIÓN DE PROCESOS

Iván García Micó	100330132
Pablo Antonio Ferrer Calderón	100330656
Yang Chen	100330277

Contenido

Introducción

Esta práctica consiste en el desarrollo de planificadores de procesos en el lenguaje de programación C. Se implementarán los siguientes planificadores:

Round-Robin

Planificador que simplemente incorpora el algoritmo Round-Robin a través de una rodaja definida en uno de los ficheros.

Round-Robin/FIFO con prioridades

Planificador que ejecutará Round-Robin para aquellos hilos con prioridad baja y una política FIFO para aquellos procesos con prioridad alta.

Round-Robin/FIFO con cambios de contexto voluntarios

Planificador similar al anterior pero que permite a un hilo realizar un cambio de contexto voluntario mediante una llamada al sistema.

Descripción del código

Round-Robin (RR.c)

Las principales funciones implementadas son *TCB* scheduler ()*, *void timer_interrupt (int sig)* y *void activator (TCB* next)*.

También se utilizará una cola (llamada *thread_q*) que contiene todos los hilos que aun no han terminado.

Cada vez que se cree un hilo se encolara también en *thread_q*.

Timer_interrupt(int sig):

Va restando ticks del reloj al hilo que se está ejecutando actualmente, cuando se le acabe la rodaja (ticks del hilo iguales a 0), se llama al scheduler para determinar el siguiente hilo a ejecutar y pasárselo al activador (*void activator (TCB* next)*) para que realice los cambios de variables y contexto necesarios.

Scheduler ():

Es el planificador del algoritmo, si un proceso aún no ha terminado, se le vuelve a asignar la rodaja y se le pone al final de la cola.

Sacamos de la cola el siguiente hilo y lo devolvemos para que el activador haga sus funciones, si no hay siguiente elemento terminamos el programa y salimos del mismo.

Activator (TCB* next):

Parte del hilo que ha pasado el planificador, cambiamos la variable running que representa el hilo actual por el nuevo y la variable current que representa el índice del hilo actual por el nuevo índice.

Si el hilo antiguo ha terminado, simplemente establecemos el contexto del nuevo mediante setcontext. En cambio, si el hilo no ha terminado, cambiamos el contexto de uno a otro mediante swapcontext.

Además, se han editado as siguientes funciones:

Init_mythreadlib():

Para inicializar la cola de hilos.

Mythread_create(void (*fun_addr)(), int priority):

Para encolar los hilos que se van creando

Round-Robin/FIFO con prioridades (RRF.c)

Las principales funciones implementadas son *TCB* scheduler ()*, *void timer_interrupt (int sig)* y *void activator (TCB* next)*.

También se utilizarán un par de colas (llamas low_q y high_q) para representar las listas de hilos de baja prioridad y de alta prioridad respectivamente

Timer_interrupt(int sig):

Sirve para controlar los ticks del round robin, si actualmente se está corriendo un hilo de prioridad alta, se sale de esta función sin más.

En caso contrario, se utiliza como en RR.c, se baja 1 al contador de ticks y cuando se acabe la rodaja vamos a coger el siguiente hilo con scheduler y manejar el contexto con activator

Scheduler ():

Planificador del algoritmo, nos permite sacar un hilo de la cola correspondiente, tenemos dos casos:

- Si el hilo actual ha terminado, sacaremos un hilo de la cola de prioridad alta (si hay) y si no hay sacaremos un hilo de la cola de prioridad baja
- Si el hilo no ha terminado, nos encontramos en un caso de un hilo de prioridad baja que se le ha terminado su rodaja, y sacamos el siguiente hilo de la cola.

Activator (TCB* next):

Partimos de hilo que nos ha devuelto scheduler, si el hilo actual ha terminado, usamos setcontext para establecer el nuevo, cambiando variables running y current.

Si el hilo anterior simplemente ha terminado su rodaja y no ha terminado, cambiamos variables running y current y cambiamos el contexto con swapcontext

Además, se han editado las siguientes funciones:

Init_mythreadlib():

Para inicializar las colas de hilos.

Mythread_create(void (*fun_addr)(), int priority):

Para encolar los hilos que se van creando dependiendo de la prioridad que tengan

Mythread_exit():

Para añadir una finalización del programa cuando no queden hilos.

Round-Robin/FIFO con prioridades con cambios de contexto voluntarios (RRFN.c)

Las principales funciones implementadas son, TCB* scheduler(), void timer_interrupt(int sig), void activator(TCB* next), read_network() y network_interrupt(int sig).

Se utilizarán tres colas: low_q, high_q y wait_q. La cola low_q almacenará los hilos de baja prioridad. La cola high_q hace lo mismo para los hilos de alta prioridad. La cola wait_q es la utilizada cuando un hilo ejecuta la función read_network(), que hace que el hilo pase a esperar a un network_interrupt().

Read_network()

Cuando un hilo llama a esta función, se introduce el hilo en la cola wait_q. A continuación, si no hay hilos que ejecutar en las colas high_q o low_q, se ejecuta el idle para esperar a un network_interrupt que recupere un hilo de la cola wait_q. El idle es una función que contiene un bucle infinito para esperar a las interrupciones de red. Si hay hilos para ejecutar en high_q o low_q se pasará a ejecutar el siguiente correspondiente.

Network_interrupt()

Cuando se recibe una interrupción simulada de la tarjeta de red, se saca al primer hilo disponible en la cola wait_q y se añade a la cola que le corresponda según su prioridad.

Timer_interrupt()

Se ha añadido la consideración del proceso idle. Si se está ejecutando dicho proceso y en ese momento hay algún elemento en alguna de las colas se sale del proceso idle.

Además, se han editado las siguientes funciones:

Mythread_exit()

Si las tres colas están vacías, se da por terminada la ejecución. Si se termina la ejecución de un hilo y no hay hilos en las colas high_q ni low_q, se ejecuta el proceso idle.

Mythread_init()

En esta organización se inicializan tres colas.

Las funciones scheduler() y activator() se mantienen práctica como en el apartado anterior.

Batería de pruebas

Las pruebas se realizarán cambiando el contenido del fichero main.c

Round-Robin (RR.c)

Lista de pruebas	
Descripción	**Resultado**
Prueba de ejecución estándar con los hilos que están en main por defecto	Ejecución normal
Ejecución sin creación de hilos con mythread_create	Ejecución solo del hilo principal
Ejecución con un solo hilo creado	Ejecución de forma seguida del hilo principal y el creado
Ejecución de tres hilos que ejecutan la misma función	Ejecución correcta de Round Robin
Ejecución de tres hilos cuya duración es la misma y además es menor que la rodaja	Se ejecutan de forma normal como si se tratara de un FIFO
Ejecución de tres hilos cuya duración es muy superior a la de rodaja y necesiten de múltiples rodajas para poder ejecutarse. Tiene como objetivo que el round robin funcione con más de un cambio de contexto para el mismo hilo	Ejecución correcta de Round Robin

Round-Robin/ FIFO con prioridades (RRF.c)

Las pruebas se realizarán cambiando el contenido del fichero main.c

Lista de pruebas	
Descripción	**Resultado**
Prueba de ejecución normal	Ejecución normal, los hilos de alta prioridad se ejecutan como FIFO, los hilos de baja se ejecutan como Round Robin
Ejecución con hilos solo de baja prioridad	Ejecución de Round Robin como en RR.c
Ejecución con hilos solo de alta prioridad	Ejecución del planificador como FIFO
Ejecución de tres hilos que ejecutan la misma función	Ejecución correcta de Round Robin
Ejecución de expulsión. Creamos un hilo de alta prioridad dentro de un hilo de baja prioridad	Ejecución con la traza de expulsión por la salida estándar
Ejecución con un solo hilo creado	Ejecución de forma seguida del hilo principal y el creado
Ejecución con dos hilos ejecutando la misma función, cambiando la prioridad	Se ejecuta primero el de mayor prioridad
Ejecución sin creación de hilos con mythread_create	Ejecución solo del hilo principal

Round-Robin/FIFO con prioridades con cambios de contexto voluntarios (RRFN.c)

Las pruebas se realizarán cambiando el contenido del fichero main.c e interrupt.c3

Lista de pruebas	
Descripción	Resultado
Prueba de ejecución normal	Ejecución normal, se responde al read_network() de forma correcta. Los hilos se sacan de la cola wait_q en el orden correcto.
Ejecución sin read_network()	La ejecución sin interrupciones de red, se lleva a cabo de forma similar a la organización RRF del apartado 2.
Ejecución únicamente del hilo principal, con un read_network()	Se inicia el hilo, cuando se llama a read_network se bloquea, y en la siguiente llamada a network_interrupt se desbloquea y termina.
Ejecución modificando la función 1, de manera que haya dos read_network() seguidos.	Los resultados de las pruebas son correctos. Los hilos que ejecutan la función 1 pasan por la cola wait_q dos veces.
Ejecución modificando el tiempo de espera entre interrupciones network_interrupt a 5.	Se notan con claridad los bloqueos de los hilos. La ejecución es correcta.

Conclusiones

El nivel de exigencia de esta práctica nos ha parecido adecuado. Hemos tenido tiempo de llevar a cabo todas las tareas requeridas en el enunciado antes de la fecha de entrega.

A lo largo de la práctica, nos han surgido algunas dudas respecto a su realización, sin embargo, todas han sido rápidamente resueltas investigando el código y leyendo la información dada en el enunciado.

Finalmente, se ha llevado a cabo la memoria explicando las distintas modificaciones realizadas sobre la librería base.

Como punto crítico, proponemos tener un enunciado que detalle más el código ofrecido para la realización de la práctica, o que aclare términos como “liberar CPU” o más aclaraciones de las funciones enable_interrupt() y disable_interrupt().