Grado en Ingeniería Informática

Diseño de Sistemas Operativos

Curso 2018-2019

Grupo 83

Práctica – 1

PLANIFICACIÓN DE PROCESOS

David Maroto Gómez – 100363850 – 100363850@alumnos.uc3m.es

Sabrina Riesgo Reyes – 100363834 – 100363834@alumnos.uc3m.es

Rafael Rus Rus – 100363907 – 100363907@alumnos.uc3m.es

ÍNDICE

1. INTRODUCCIÓN

El contenido de este documento pretende analizar los resultados obtenidos durante la realización de la práctica 1 de la asignatura consistente en desarrollar tres planificadores de procesos diferentes.
Para la elaboración de esta práctica se ha utilizado el material proporcionado por los profesores donde encontrábamos la codificación de un planificador FIFO en el lenguaje de programación C.
En los siguientes apartados de este documento se comentarán las decisiones tomadas en el código para los distintos planificadores, así como la batería de pruebas realizadas para satisfacer el funcionamiento deseado. Por último, finalizaremos con las conclusiones sacadas de este trabajo así como las dificultades encontradas.

2. DESCRIPCIÓN DEL CÓDIGO

Descripción del código detallando las principales funciones implementadas; no se incluye ningún código fuente de la práctica en este apartado. Todos los planificadores, además de realizar sus correspondientes funciones, tienen en cuenta su control de errores, como por ejemplo: cuando la variable 'QUANTUM_TICKS' es menor que 0, o cuando no existe ningún hilo por ejecutar.

2.1. PLANIFICADOR ROUND-ROBIN

Para la codificación del planificador Round Robin se ha creado una nueva variable a los hilos llamada 'ticks', que será el número de ticks que necesita cada hilo para terminar su ejecución. A su vez, hemos creado una nueva constante llamada 'QUANTUM_TICKS' que será la rodaja de tiempo durante la cual se ejecutará un proceso.
La función 'TCB* scheduler()' se encargará de gestionar la cola de procesos. Para ello comprueba la cola de procesos, y en caso de que esté vacía el programa termina puesto que no quedan procesos por ejecutar; en caso contrario, el programa se encargará de devolver el proceso siguiente a ejecutar.
La función 'activator(TCB* next)' simplemente cambia el contexto entre el anterior hilo ejecutado y el ejecutado actual haciendo el control de errores pertinente.
La función 'void timer_interrupt(int sig)' es la encargada de gestionar la rodaja de tiempo de los procesos en ejecución. Cuando la rodaja de tiempo llega a 0, reinicia la variable 'ticks' a la constante 'QUANTUM_TICKS' y encola el proceso en la cola, después de esto se encargará de llamar a 'TCB* scheduler()' para recibir el siguiente hilo a ejecutar y de pasárselo a 'activator(TCB* next)'.

2.2. PLANIFICADOR ROUND-ROBIN/FIFO CON PRIORIDADES

Para la codificación del planificador Round-Robin/FIFO con prioridades se parte de la solución del apartado anterior. A este código hemos decidido implementarle dos colas en lugar de una, la primera cola controlará los procesos de baja prioridad que se rigen por una política Round-Robin, la otra controlará los procesos de alta prioridad que se rigen por una política FIFO.
Se ha modificado la función que crea los hilos añadiendo una variable 'priority' que controlará la prioridad de los hilos. Ahora al crear un nuevo proceso se comprobará su prioridad, en caso de ser un hilo de prioridad alta y se esté ejecutando un hilo de prioridad baja, pasará a ejecutarse inmediatamente el hilo de prioridad alta. El hilo de prioridad baja se encolará en la cola correspondiente a los hilos de prioridad baja previo reinicio del tiempo de rodaja. En el caso de que el nuevo hilo sea de prioridad alta y se

esté ejecutando otro de prioridad alta o el nuevo hilo creado sea de prioridad baja se encolará el nuevo hilo creado en la cola correspondiente.
La función 'TCB* scheduler()' se encarga de gestionar las colas de procesos. Para ello lo primero que hace es comprobar la cola de procesos de prioridad alta, en caso de que esté vacía comprobará la cola de procesos de prioridad baja, puesto que no quedan procesos de prioridad alta por ejecutar, en caso de que la cola de prioridad alta no esté vacía se encarga de devolver el siguiente proceso de prioridad alta que debe ejecutar. Una vez vaciada la cola de procesos de prioridad alta en caso de que la cola de procesos de prioridad baja contenga procesos se devolverá el primer proceso de la cola de prioridad baja. Si ambas colas se vacían el programa terminará su ejecución.
La función 'activator(TCB* next)' simplemente cambia el contexto entre el anterior hilo ejecutado y el ejecutado actual haciendo el control de errores pertinente.
La función 'void timer_interrupt(int sig)' es la encargada de gestionar la rodaja de tiempo de los procesos de prioridad baja. La función comprueba la prioridad del proceso en ejecución, en caso de que la prioridad sea alta la función no hace nada, si por el contrario el proceso es de prioridad baja se reduce el número de ticks. Cuando la rodaja de tiempo llega a 0 reinicia la variable 'ticks' a la constante 'QUANTUM_TICKS' y encola el proceso en la cola de prioridad baja, después de esto se encarga de llamar a 'TCB* scheduler()' para recibir el siguiente hilo a ejecutar y pasárselo a 'activator(TCB* next)'.

2.3. PLANIFICADOR ROUND-ROBIN/FIFO CON POSIBLES CAMBIOS DE CONTEXTO

Para la codificación del planificador Round-Robin/FIFO con posibles cambios de contexto se parte también de la solución del apartado anterior, aunque en este caso se editarán las funciones 'int read_disk()' y 'void disk_interrupt(int sig)'. A este nuevo código hemos decidido implementarle (junto a las dos colas de baja prioridad y alta prioridad realizadas para el anterior planificador) una cola de threads en espera llamada 'queue_wait'. Todas las colas se inicializarán en la función 'void init_mythreadlib()', al igual que en los otros planificadores.
Con respecto a la función que crea los hilos, no se ha realizado ninguna modificación del anterior planificador Round-Robin/FIFO con prioridades. No obstante, si se han cambiado las funciones 'int read_disk()' y 'void disk_interrupt(int sig)' con el objetivo de introducir la nueva funcionalidad que permite a un thread realizar un posible cambio de contexto voluntario. En relación a la primera, cuando un thread realice la llamada *read_disk*, esta liberará la CPU (haciendo que el proceso deje la CPU) e introducirá ese thread en la cola 'queue_wait'; no obstante, esto sólo se producirá si y sólo si los datos solicitados por el proceso ya no están en la caché de paginas (es decir, si la función 'data_in_page_cache' devuelve un valor distinto de 0). Posteriormente, se activará el proceso new_thread que será el primer proceso de mayor prioridad que esté listo.
También hemos modificado la función 'void disk_interrupt(int sig)' añadiendo la siguiente situación (similar a la función de interrupción de reloj): si la cola de threads no aparece vacía al llegar a una interrupción, el primer hilo en la cola de espera pasará a la cola de listos correspondiente a su prioridad; sin embargo, no se realizarán acciones adicionales en caso de que no haya ningún thread en espera.
La función 'TCB* scheduler()' también ha sido modificada, ya que se ha tenido en cuenta que cuando no existe ningún thread listo para ejecutar pero si en la cola 'queue_wait', se deberá ejecutar el thread 'idle'. Así pues, actualizamos el thread 'running' y su id, que ahora será el thread 'idle' y su id siempre será -1. Este thread se encargará de ejecutar un bucle infinito de tal forma que este planificador consultará cada tick de reloj para ver si existe algún hilo para ser ejecutado.
De la misma forma, la función 'void timer_interrupt(int sig)' también ha sido modificada,

ya que hay que tener en cuenta cuando el 'mythread_gettid()' sea -1, pues en este caso el thread que se está ejecutando será el 'idle'. A continuación, comprobaremos de que las colas de baja prioridad y de alta prioridad no estén vacías, que en cuyo caso vamos a proceder a activar el primer proceso de mayor prioridad que esté listo. En caso de que no sea -1 el 'mythread_gettid()', deberemos consultar si el thread que se está ejecutando es de baja prioridad, ya que se encargará de gestionar la rodaja de tiempo de todos los procesos de prioridad baja (tal y como hacía en el anterior planificador).
Por último, cabe destacar que se ha editado la función 'activator(TCB* next)' para tener en cuenta cuando realizamos un cambio de contexto desde el thread 'idle' a un thread listo para ejecutar.

3. BATERÍA DE PRUEBAS

Baterías de pruebas utilizadas y de resultados obtenidos (todos los archivos mencionados en el pdf se encuentran en las dos carpetas adjuntas).

3.1. PLANIFICADOR ROUND-ROBIN

Explicación	Prueba realizada	Resultado esperado	Resultado obtenido
La variable QUANTUM_TICKS es menor que 0	- En Makefile, se escribe RR.o en OBJS - En mythread.h se ha modificado lo siguiente: #define QUANTUM_TICKS -2 - Por terminal, se escribe el comando *make* y luego *./main*	ERROR: the variable QUANTUM TICKS must be greater than zero	ERROR: the variable QUANTUM TICKS must be greater than zero
Error al inicializar el planificador debido al valor de la variable N (no es un número natural mayor que 0), no se inicializa el primer hilo	- En Makefile, se escribe RR.o en OBJS - En mythread.h se ha modificado lo siguiente: #define N 0 - Por terminal, se escribe el comando *make* y luego *./main*	thread failed to initialize	thread failed to initialize
Son creados los siguientes hilos en el *main*: int i, j, k, l, m, n, a, b = 0; Da lo mismo las prioridades de los hilos anteriormente creados	- En Makefile, se escribe RR.o en OBJS - En mythread.h se ha modificado lo siguiente: #define N 10 #define QUANTUM_TICKS 40 - Por terminal, se escribe el comando *make* y luego *./main*	resultado_esperado1.txt	resultado_obtenido1.txt

Son creados los siguientes hilos en el *main*: int i, j, k, l, m, n, a, b = 0; Da lo mismo las prioridades de los hilos anteriormente creados	- En Makefile, se escribe RR.o en OBJS - En mythread.h se ha modificado lo siguiente: #define N 10 (no se realiza una prueba cambiando la variable N porque se obtiene lo mismo y se deben evitar las pruebas duplicadas) #define QUANTUM_TICKS 500 - Por terminal, se escribe el comando *make* y luego *./main*	resultado_esperado2.txt	resultado_obtenido2.txt
Son creados los siguientes hilos en el *main*: int i, j, k, l, m, n, o, a, b = 0; Da lo mismo las prioridades de los hilos anteriormente creados	- En Makefile, se escribe RR.o en OBJS - En mythread.h se ha modificado lo siguiente: #define N 10 #define QUANTUM_TICKS 40 - Por terminal, se escribe el comando *make* y luego *./main*	resultado_esperado3.txt	resultado_obtenido3.txt

3.2. PLANIFICADOR ROUND-ROBIN/FIFO CON PRIORIDADES

Explicación	Prueba realizada	Resultado esperado	Resultado obtenido
La variable QUANTUM_TICKS es menor que 0	- En Makefile, se escribe RRF.o en OBJS - En mythread.h se ha modificado lo siguiente: #define QUANTUM_TICKS -2 - Por terminal, se escribe el comando *make* y luego *./main*	ERROR: the variable QUANTUM TICKS must be greater than zero	ERROR: the variable QUANTUM TICKS must be greater than zero
Error al inicializar el planificador debido al valor de la variable N (no es un número natural mayor que 0), no se inicializa el primer hilo	- En Makefile, se escribe RRF.o en OBJS - En mythread.h se ha modificado lo siguiente: #define N 0 - Por terminal, se escribe el comando *make* y luego *./main*	thread failed to initialize	thread failed to initialize

Son creados los siguientes hilos en el *main*: int i, j, k, l, m, n, a, b = 0;
mythread_setpriority(HIGH_PRIORITY);
read_disk();
Hilo i – prioridad baja en fun1 read_disk()
Hilo j – prioridad baja en fun2
Hilo k – prioridad baja en fun3
Hilo l – prioridad alta en fun1
Hilo m – prioridad alta en fun2
Hilo n – prioridad alta en fun3
for(a = 0; a < 10; ++a) {
for(b = 0; b < 30000000; ++b);}
Hilo a – prioridad alta en fun1
Hilo b – prioridad alta en fun1
	- En Makefile, se escribe RRF.o en OBJS
	- En mythread.h se ha modificado lo siguiente: #define N 10
	#define QUANTUM_TICKS 40
	- Por terminal, se escribe el comando *make* y luego *./main*
		resultado_esperado4.txt	resultado_obtenido4.txt

Son creados los siguientes hilos en el *main*: int i, j, k, l, m, n, a, b = 0;
mythread_setpriority(HIGH_PRIORITY);
read_disk();
Hilo i – prioridad baja en fun1 read_disk()
Hilo j – prioridad baja en fun2
Hilo k – prioridad baja en fun3
Hilo l – prioridad alta en fun1
Hilo m – prioridad alta en fun2
Hilo n – prioridad alta en fun3
for(a = 0; a < 10; ++a) {
for(b = 0; b < 30000000; ++b);}
Hilo a – prioridad alta en fun1
Hilo b – prioridad alta en fun1
	- En Makefile, se escribe RRF.o en OBJS
	- En mythread.h se ha modificado lo siguiente: #define N 10 (no se realiza una prueba cambiando la variable N porque se obtiene lo mismo y se deben evitar las pruebas duplicadas)
	#define QUANTUM_TICKS 500
	- Por terminal, se escribe el comando *make* y luego *./main*
		resultado_esperado5.txt	resultado_obtenido5.txt

Son creados los siguientes hilos en el *main*: int i, j, k, l, m, n, o, a, b = 0;
mythread_setpriority(HIGH_PRIORITY);
read_disk();
Hilo i – prioridad baja en fun1 read_disk()
Hilo j – prioridad baja en fun2
Hilo k – prioridad baja en fun3
Hilo l – prioridad alta en fun1
Hilo m – prioridad alta en fun2
Hilo n – prioridad alta en fun3
Hilo o – prioridad baja en fun1
for(a = 0; a < 10; ++a) {
for(b = 0; b < 30000000; ++b);}
	- En Makefile, se escribe RR.o en OBJS
	- En mythread.h se ha modificado lo siguiente: #define N 10
	#define QUANTUM_TICKS 40
	- Por terminal, se escribe el comando *make* y luego *./main*
		resultado_esperado6.txt	resultado_obtenido6.txt

Hilo a – prioridad alta en fun1 Hilo b – prioridad alta en fun1			

Igual que el anterior, pero modificando la prioridad del primer hilo.
Son creados los siguientes hilos en el *main*: int i, j, k, l, m, n, o, a, b = 0;
mythread_setpriority(HIGH_PRIORITY);
read_disk();
Hilo i – prioridad alta en fun1
read_disk()
Hilo j – prioridad baja en fun2
Hilo k – prioridad baja en fun3
Hilo l – prioridad alta en fun1
Hilo m – prioridad alta en fun2
Hilo n – prioridad alta en fun3
Hilo o – prioridad baja en fun1
for(a = 0; a < 10; ++a) {
for(b = 0; b < 30000000; ++b);}
Hilo a – prioridad alta en fun1
Hilo b – prioridad alta en fun1
	- En Makefile, se escribe RR.o en OBJS
	- En mythread.h se ha modificado lo siguiente: #define N 10
	#define QUANTUM_TICKS 40
	- Por terminal, se escribe el comando *make* y luego *./main*
		resultado_esperado7.txt	resultado_obtenido7.txt

3.3. PLANIFICADOR ROUND-ROBIN/FIFO CON POSIBLES CAMBIOS DE CONTEXTO

Explicación	Prueba realizada	Resultado esperado	Resultado obtenido
La variable QUANTUM_TICKS es menor que 0	- En Makefile, se escribe RRFD.o en OBJS - En mythread.h se ha modificado lo siguiente: #define QUANTUM_TICKS -2 - Por terminal, se escribe el comando *make* y luego *./main*	ERROR: the variable QUANTUM TICKS must be greater than zero	ERROR: the variable QUANTUM TICKS must be greater than zero
Error al inicializar el planificador debido al valor de la variable N (no es un número natural mayor que 0), no se inicializa el primer hilo	- En Makefile, se escribe RRFD.o en OBJS - En mythread.h se ha modificado lo siguiente: #define N 0 - Por terminal, se escribe el comando *make* y luego *./main*	thread failed to initialize	thread failed to initialize

Son creados los siguientes hilos en el *main*: int i, j, k, l, m, n, a, b = 0;
mythread_setpriority(HIGH_PRIORITY);
read_disk();
Hilo i – prioridad baja en fun1 read_disk()
Hilo j – prioridad baja en fun2
Hilo k – prioridad baja en fun3
Hilo l – prioridad alta en fun1
Hilo m – prioridad alta en fun2
Hilo n – prioridad alta en fun3
for(a = 0; a < 10; ++a) {
for(b = 0; b < 30000000; ++b);}
Hilo a – prioridad alta en fun1
Hilo b – prioridad alta en fun1
	- En Makefile, se escribe RRF.o en OBJS
	- En mythread.h se ha modificado lo siguiente: #define N 10
	#define QUANTUM_TICKS 40
	- Por terminal, se escribe el comando *make* y luego *./main*
		resultado_esperado8.txt	resultado_obtenido8.txt

Son creados los siguientes hilos en el *main*: int i, j, k, l, m, n, a, b = 0;
mythread_setpriority(HIGH_PRIORITY);
read_disk();
Hilo i – prioridad baja en fun1 read_disk()
Hilo j – prioridad baja en fun2
Hilo k – prioridad baja en fun3
Hilo l – prioridad alta en fun1
Hilo m – prioridad alta en fun2
Hilo n – prioridad alta en fun3
for(a = 0; a < 10; ++a) {
for(b = 0; b < 30000000; ++b);}
Hilo a – prioridad alta en fun1
Hilo b – prioridad alta en fun1
	- En Makefile, se escribe RRF.o en OBJS
	- En mythread.h se ha modificado lo siguiente: #define N 10 (no se realiza una prueba cambiando la variable N porque se obtiene lo mismo y se deben evitar las pruebas duplicadas)
	#define QUANTUM_TICKS 500
	- Por terminal, se escribe el comando *make* y luego *./main*
		resultado_esperado9.txt	resultado_obtenido9.txt

Son creados los siguientes hilos en el *main*: int i, j, k, l, m, n, a, b = 0;
mythread_setpriority(LOW_PRIORITY);
read_disk();
Hilo i – prioridad alta en fun1 read_disk()
Hilo j – prioridad alta en fun2
Hilo k – prioridad alta en fun3
Hilo l – prioridad baja en fun1
Hilo m – prioridad baja en fun2
Hilo n – prioridad baja en fun3
for(a = 0; a < 10; ++a) {
for(b = 0; b < 30000000; ++b);}
Hilo a – prioridad baja en fun1
	- En Makefile, se escribe RRF.o en OBJS
	- En mythread.h se ha modificado lo siguiente: #define N 10
	#define QUANTUM_TICKS 40
	- Por terminal, se escribe el comando *make* y luego *./main*
		resultado_esperado10.txt	resultado_obtenido10.txt

Hilo b – prioridad baja en fun1			

Son creados los siguientes hilos en el *main*: int i, j, k, l, m, n, a, b = 0;
mythread_setpriority(LOW_PRIORITY);
read_disk();
Hilo i – prioridad alta en fun1
read_disk()
Hilo j – prioridad baja en fun2
Hilo k – prioridad alta en fun3
Hilo l – prioridad baja en fun1
Hilo m – prioridad alta en fun2
Hilo n – prioridad baja en fun3
for(a = 0; a < 10; ++a) {
for(b = 0; b < 30000000; ++b);}
Hilo a – prioridad alta en fun1
Hilo b – prioridad baja en fun1
	- En Makefile, se escribe RRF.o en OBJS
	- En mythread.h se ha modificado lo siguiente: #define N 10
	#define QUANTUM_TICKS 40
	- Por terminal, se escribe el comando *make* y luego *./main*
		resultado_esperado11.txt	resultado_obtenido11.txt

4. CONCLUSIONES Y PROBLEMAS ENCONTRADOS

Con el desarrollo de esta primera práctica hemos comprendido con mayor claridad el funcionamiento de los planificadores implementados (RR y FIFO con prioridades y bajo posibles cambios de contexto) y aumentado nuestros conocimientos sobre el tratamiento de hilos.

En cuanto a los problemas encontrados, nos ha resultado un poco complicado realizar el último apartado del planificador RRFD, apenas sabíamos por dónde empezar a realizarlo y cómo tratar los posibles cambios de contexto; por ello tuvimos que pedir una tutoría y así supimos dónde se encontraban nuestros fallos con todas las interrupciones y poder arreglarlos.