Diseño de Sistemas Operativos

Planificación de Procesos

Fernando García Sanz - 100346043 - 100346043@alumnos.uc3m.es

Pablo Cañas Castellanos - 100346190 - 100346190@alumnos.uc3m.es

18 de marzo de 2019

Índice

1. Introducción

En un sistema operativo se encuentran multitud de procesos que operan de forma concurrente, y el sistema ha de decidir cómo organizarlos de tal forma que pueda completar todas las tareas que realiza. Debe planificar de entre los procesos activos cuáles son más importantes, a cuáles debe atender antes, etc.

En esta práctica procederemos a implementar un planificador de hilos según diversos algoritmos. Se han establecido tres políticas de planificación de procesos: *Round-Robin*, *Round-Robin* sumándose el algoritmo *FIFO* para hilos de alta prioridad y, finalmente, esta última política añadiendo cambios de contexto voluntarios por medio de accesos a la memoria.

La memoria se estructura de la siguiente manera: primero, partimos de los archivos y estructuras expuestas en el enunciado para así comentar nuestra implementación de los planificadores; segundo, estudiaremos la completitud y correctitud de estos por medio de una batería de pruebas; y por último, expondremos nuestras conclusiones sobre la práctica.

2. Planificador *Round-Robin*

El primer planificador implementa el algoritmo *Round-Robin*, mediante el cual se asigna a cada proceso una cantidad máxima de tiempo (*slice*) en la que este puede ser ejecutado. Una vez haya transcurrido dicho periodo, el proceso en ejecución le pasa el turno al siguiente proceso listo para ejecutarse y se coloca en la última posición de la cola de procesos listos. Cuando un proceso finaliza, el siguiente proceso listo en la cola pasará a ejecutarse.

En primer lugar, hemos definido una serie de estructuras de datos para facilitar la implementación. Tendremos una variable del tipo TCB* que servirá para referirse al hilo que se encuentra en ejecución (*running*). Se define también una cola que servirá para almacenar los procesos listos para ser ejecutados (*q*) y el hilo *idle*, el cual se mantiene a la espera ejecutando un bucle infinito.

La primera función que se define es *init_mythreadlib*, que se encarga de inicializar la biblioteca de *threads*.

Lo primero que se hace en ella es crear el contexto para el *thread idle*, dando valor a todos sus parámetros. A continuación, se inicializa el estado de todos los elementos del *array* de control de hilos (*t_state*) a *FREE*, se establece el primero de todos los hilos como el hilo en ejecución, se inicializan tanto las interrupciones de disco como de reloj y se inicializa la cola de procesos listos.

La siguiente función a tener en cuenta es *mythread_create*, que se encarga de inicializar un nuevo *thread*. La primera acción que se realiza es comprobar si la biblioteca de *threads* se ha inicializado, y si no, la inicializa. Después se busca la primera posición de *t_state* cuyo estado sea *FREE*, y se inicializa el proceso con todos sus parámetros, cambiando su estado a *INIT*. Finalmente, este se inserta en la lista de listos.

Después se encuentra la función *mythread_exit*, que se ejecuta cuando un hilo termina su ejecución. Se establecerá el estado *FREE* en la posición de *t_state* para poder reutilizarla más adelante, se liberará el espacio de memoria asignado al hilo, y se pondrá en ejecución el siguiente hilo de la lista.

La función *scheduler* se encarga de calcular cuál es el siguiente hilo a ejecutarse. Para implementar *Round-Robin*, simplemente tendremos que seleccionar el siguiente proceso de la cola de procesos listos y devolverlo como resultado para su posterior ejecución. En el caso de que no queden procesos, se finaliza la ejecución y se libera el espacio de memoria asignado a la cola.

Más adelante se encuentra la función *timer_interrupt*, que se ejecuta de forma atómica, deshabilitando interrupciones. La función es llamada cada interrupción de reloj, cada tick, y reduce en uno el numero de ticks que el hilo en ejecución tiene asignados. En el caso de que dicho número llegue a cero, se restablecen los ticks, se inserta dicho proceso en la última posición de la cola y se llama a la función *scheduler* para obtener el siguiente hilo a ejecutar (planificador *Round-Robin*). En el caso de que quede más de un proceso listo, se realiza un cambio de contexto entre ambos procesos mediante la función *activator*. Esta función cambia el contexto entre dos procesos y actualiza el valor del proceso en ejecución (*running*).

El resto de funciones que podemos encontrar sirven para llamar a las interrupciones, obtener y establecer prioridades y obtener el identificador de cada hilo.

Una vez explicado el código para implementar *Round-Robin*, el funcionamiento en conjunto es el siguiente:

El hilo principal del sistema procederá a iniciar el resto de hilos y éstos se asignan a distintas tareas. Cada vez que se produce una interrupción de reloj, se reduce en uno el número de ticks del hilo actual, y cuando este llega a cero, se sustituye el hilo en ejecución por el primero que se encuentra en la lista de listos, mediante un cambio de contexto. El hilo que previamente se estaba ejecutando pasa a la última posición de la lista de listos, restableciendo sus parámetros. Una vez todos los hilos han terminado con la ejecución de las tareas asignadas, se finaliza el programa.

3. Planificador *Round-Robin/FIFO* con prioridades

En este nuevo planificador se implementa la estructura del anterior, al que se incorporan varios añadidos.

En el sistema a desarrollar ha de tenerse en cuenta que ahora los distintos hilos tienen una prioridad, que bien puede ser alta o baja. Los hilos de prioridad alta tienen preferencia, siguiendo su ejecución un algoritmo *FIFO*, mientras que aquellos de baja prioridad seguirán la estructura anterior, *Round-Robin*.

Igual que en la descripción del sistema anterior, primero se procede a explicar los cambios en el algoritmo de planificación:

En vez de utilizar una única cola de hilos, se crean dos, para las dos prioridades. Dichas colas se inicializan en la función *init_mythreadlib*.

En la función *mythread_create* se añade lo siguiente: una vez se ha inicializado en nuevo hilo se deshabilitan las interrupciones para realizar una operación atómica que puede suponer un cambio de contexto. Se comparan las prioridades del hilo en ejecución y del nuevo hilo. Si el nuevo hilo es preferente, el hilo en ejecución se colocará en el último lugar de la cola de procesos de prioridad baja habiendo restablecido sus ticks. Se produce entonces un cambio de contexto entre ambos hilos. En cambio, si los hilos creados son de prioridad menor o igual al ejecutado, serán colocados en el último lugar de la cola correspondiente.

Se producen también cambios en la función *scheduler*. Se comprueba que la cola de procesos de alta prioridad no esté vacía, para obtener el primer elemento y devolverlo como el proceso a ejecutar. Si no es el caso, se comprueba en la cola de baja prioridad. Si ambas están vacías se procede a liberar el espacio de memoria asignado a ambas y a terminar la ejecución. Las operaciones de obtener elementos de las colas se realizan de forma atómica.

La última función en sufrir cambios es *timer_interrupt*. Se comprueba que el proceso que se está ejecutando es de prioridad baja para reducir el número de ticks, y si éste es igual a cero, se obtiene el siguiente proceso a ejecutar mediante la función *scheduler* y se produce un cambio de contexto. Si la prioridad es alta, no se hará nada durante la interrupción, ya que los procesos de prioridad alta han de ejecutarse en su totalidad.

Una vez descritos los cambios en las funciones del sistema se procede a detallar el funcionamiento del sistema en su conjunto:

El funcionamiento es igual que en el planificador anterior, con el añadido de hilos prioritarios.

Si el hilo en ejecución es de baja prioridad y se inicializa uno de prioridad alta, se produce un cambio de contexto y el de mayor prioridad se ejecutará completamente. El otro se asigna a la última posición de la cola correspondiente restableciendo sus ticks. Mientras no haya hilos de alta prioridad, el resto funcionan según el algoritmo del apartado anterior. Una vez todos los hilos han terminado con la ejecución de las tareas asignadas, se finaliza el programa.

4. Planificador *Round-Robin/FIFO* con posibles cambios de contexto voluntarios

En este último planificador se implementan posibles cambios de contexto voluntarios entre hilos en el sistema. Se parte de la estructura del planificador anterior, en el que aparte de las prioridades de los distintos hilos, se permite que estos realicen un cambio de contexto de forma voluntaria mediante la llamada al sistema *read_disk*.

Primero se procede a explicar los cambios en el algoritmo de planificación:

Se añade una tercera cola para almacenar aquellos procesos que se encuentran en estado de espera. Dicha cola se inicializa junto a las otras dos.

La función *read_disk* ahora, en vez de únicamente devolver el valor "1", comprueba si los datos leídos en la interrupción de disco están o no en la memoria caché. Si no están, se restablecen los ticks del proceso en ejecución al valor por defecto y dicho proceso cambia a estado de espera (*WAITING*), colocándose en la cola correspondiente. La función *scheduler* devuelve el proceso que ha de ser puesto en ejecución. Se produce un cambio de contexto, voluntario en este caso, entre ambos procesos. Si la información estuviera en la caché, no es necesaria una interrupción de disco, por lo que únicamente se devuelve el valor "1".

En cuanto a la función *disk_interrupt*, se han producido grandes cambios. Las operaciones realizadas dentro de la función se realizan de forma atómica. Se comprueba que existen procesos en la cola de procesos en espera, para obtener el primer proceso de la cola, que cambiará de estado a *INIT*. Se comprueba también la prioridad de dicho proceso. Si es alta, el proceso actualmente en ejecución es de prioridad baja y su identificador es distinto de "-1"(*idle*), se restablecen los ticks del proceso en ejecución y se coloca en la cola de baja prioridad, produciéndose un cambio de contexto entre el nuevo proceso y el que se encontraba en ejecución. En el caso de que el proceso obtenido de la cola de espera sea de prioridad baja, simplemente se almacena en la última posición de la cola de baja prioridad. En el caso de que el identificador del proceso en ejecución sea "-1", se obtiene el siguiente proceso a ejecutar mediante la función *scheduler* y se produce un cambio de contexto.

Los cambios restantes se producen en las funciones *scheduler* y *timer_interrupt*.

En el caso de la primera, se comprueba también que la cola de procesos en espera no esté vacía, después de comprobarlo en la cola de procesos de alta y baja prioridad, que tienen preferencia. Si estas dos colas están vacías la función devolverá como el siguiente proceso a ejecutar el hilo *idle*, en ejcución hasta que alguna de las dos primeras listas deje de estar vacía. En el caso de que las tres colas estén vacías, todos los procesos habrán concluido, por lo que se libera la memoria utilizada y finaliza el programa.

En la segunda función, el único cambio que se realiza es el de comprobar cuando se produce una interrupción de reloj que el hilo en ejecución es distinto de *idle*, ya que en ese caso no deben ni restablecerse valores ni habrá cambios de contexto.

Una vez explicados los cambios en cada una de las funciones anteriores, se pasa a explicar el funcionamiento en conjunto del sistema:

El sistema funciona de igual forma que el planificador anterior, con el añadido de las interrupciones de disco. Si es necesaria una lectura de disco, ese proceso pasa a bloqueado y mediante la función *scheduler* se obtiene el siguiente en a ejecutar. Cuando el bloqueado se desbloquee pasará a la cola correspondiente. Si no hay procesos listos pero si bloqueados, el hilo *idle* se ejecutará hasta que alguno se desbloquee, cambiando el contexto. Por lo demás, el funcionamiento es el mismo que en el planificador anterior.

5. Batería de Pruebas

Una vez se han implementado los tres tipos de planificadores se han diseñado una serie de pruebas específicas para cada uno de ellos para comprobar su correcto funcionamiento. Para cada una de las pruebas, comentaremos los resultados obtenidos con cada planificador, para así poder apreciar las diferencias.

Para las primeras pruebas, se asume que tenemos la configuración aportada en el código inicial. Es decir, tendremos 8 hilos, uno principal y siete que se activan desde el principal; y tendremos las mismas funciones y lecturas de disco que se especifican en el enunciado. Más adelante, cambiaremos esa configuración, para realizar pruebas más exhaustivas.

Prueba 1: Todos los hilos de baja prioridad

Round-Robin: el hilo principal, durante su primera rodaja, inicializa cinco hilos. Como puede verse, los dos restantes son incializados cuando el contexto vuelve al hilo principal. Anteriormente se han

producido los cambios de contexto esperados. Podemos observar como se producen unos cambios de contexto coherentes con el programa escrito, por lo que el comportamiento es el esperado.

Round-Robin/FIFO: al tener todos los hilos prioridad baja, este planificador se comporta exactamente igual que el *Round-Robin*, como es de esperar.

Round-Robin/FIFO + CCV: el comportamiento es igual que el anterior planificador, con la diferencia de que ahora se pueden producir cambios de contexto voluntarios (lectura de disco) además de los involuntarios (fin de rodaja). Al ser todos los hilos de baja prioridad, ningún hilo podrá ser expulsado debido a la aparicón de otro de mayor prioridad. El comportamiento es el esperado en la ejecución.

Prueba 2: Todos los hilos de alta prioridad

Round-Robin: estos resultados no varían con respecto a las pruebas anteriores, ya que las prioridades de los hilos no se tienen en cuenta con este planificador.

Round-Robin/FIFO: en este planificador, un hilo de prioridad alta que se empiece a ejecutarse ha de terminar su ejecución. Como el hilo principal inicializa al resto de hilos, éstos son ejecutados en orden, empezando la ejecución del siguiente cuando termina el anterior. El comportamiento es el de un planificador FIFO, como esperábamos.

Round-Robin/FIFO + CCV: en este caso el comportamiento es similar al del *Round-Robin/FIFO*, salvo que si se produce una lectura de disco, ese hilo pasa a bloqueado, y una vez se desbloquee se inserta en el último lugar de la lista de listos. Si no hay interrupciones de disco, los hilos se ejecutan hasta terminar la tarea asignada. Si todos los hilos se encuentran bloqueados, se ejecutara el hilo *idle*.

Prueba 3: Hilo principal en baja prioridad, y el resto de hilos en alta prioridad

Round-Robin: estos resultados no varían con respecto a las pruebas anteriores, ya que las prioridades de los hilos no se tienen en cuenta con este planificador.

Round-Robin/FIFO: como el hilo principal es el que invoca al resto, este es el primero en ejecutarse. Sin embargo, en el instante en que el primer hilo pasa a estar listo, se realiza un cambio de contexto puesto que este tiene mayor prioridad. Este se ejecuta hasta su finalización (por la planificación FIFO) y se vuelve al hilo principal, el cual activará el segundo hilo y realizará un cambio de contexto. Este proceso se repite para todos los hilos, por lo que el resultado es el esperado.

Round-Robin/FIFO + CCV: el comportamiento es parecido al anterior, con la diferencia de que cuando un hilo de alta prioridad realiza una lectura de disco, se vuelve al hilo principal, por lo

que este puede continuar su ejecución y continúa activando los siguientes hilos. En la prueba que hemos realizado, los tres últimos hilos se quedan bloqueados por una lectura de disco, por lo que cambiamos el contexto a *idle* hasta que finaliza su bloqueo. Por lo tanto, el comportamiento es el esperado.

Prueba 4: Hilo principal en alta prioridad, y el resto de hilos en baja prioridad

Round-Robin: estos resultados no varían con respecto a las pruebas anteriores, ya que las prioridades de los hilos no se tienen en cuenta con este planificador.

Round-Robin/FIFO: como el hilo principal tiene alta prioridad, este se va a ejecutar hasta el final, activando así el resto de hilos. Una vez este finaliza, el resto de hilos se empiezan a ejecutar en orden con la política *Round-Robin*, realizando los cambios de contexto cuando el *slice* asignado finaliza.

Round-Robin/FIFO + CCV: el hilo principal comienza, pero tiene que realizar una lectura de disco antes de activar alguno de los otros hilos, por lo que se cambia al contexto de *idle*. Una vez la lectura finaliza, el hilo se ejecuta hasta el final, y el resto de hilos de baja prioridad comienzan con la planificación Round-Robin. Por lo tanto, realizan cambios de contexto, involuntarios cuando su *slice* finaliza, o voluntarios cuando realizan una lectura de disco.

Podemos concluir que los resultados son satisfactorios y los programas se comportan según los requisitos.

Prueba 5: Intercalamos prioridades entre los distintos hilos

5.1. Ponemos alta prioridad a los hilos: principal, tercero, sexto y séptimo; y baja prioridad a los hilos: primero, segundo, cuarto y quinto.

Round-Robin: estos resultados no varían con respecto a las pruebas anteriores, ya que las prioridades de los hilos no se tienen en cuenta con este planificador.

Round-Robin/FIFO: como el hilo principal tiene alta prioridad, se ejecutará hasta su finalización. Después, como el resto de hilos ya estarán activados, se ejecutarán con planificación *FIFO* los hilos de alta prioridad, estos son, el tercero, sexto y séptimo. Finalmente, se ejecutarán el resto de hilos con planificación *Round-Robin*.

Round-Robin/FIFO + CCV: el hilo principal realiza una lectura de disco y se pasa al thread *idle*. Después de que el primer hilo se active, el hilo principal vuelve a realizar otra lectura de disco, por lo que el primer hilo se ejecuta aun teniendo baja prioridad, puesto que es el único hilo listo. Después de que finalice el hilo principal y se activen el resto de hilos, comienzan a ejecutarse los de

alta prioridad. Sin embargo, tanto el sexto como séptimo hilos tienen que hacer lectura de disco, por lo que la ejecución pasa a hilos de menor prioridad. Tan pronto como alguno de los hilos de alta prioridad finaliza la lectura, el contexto vuelve a cambiarse a ellos, por lo que el requisito de este planificador funciona a la perfección.

5.2. Ponemos alta prioridad a los hilos: segundo, quinto y sexto; y baja prioridad a los hilos: principal, primero, tercero, cuarto y séptimo.

Round-Robin: estos resultados no varían con respecto a las pruebas anteriores, ya que las prioridades de los hilos no se tienen en cuenta con este planificador.

Round-Robin/FIFO: en este caso, en cuanto se inicie un hilo de alta prioridad, se producirá un cambio de contexto y el de alta prioridad se ejecutará hasta finalizar su tarea. El primer hilo de alta prioridad que se inicialice expulsará al hilo principal. Cada vez que el contexto vuelva al principal, y éste vuelva a iniciar un hilo de prioridad mayor ocurrirá el mismo proceso. Los resultados de la ejecución son los esperados.

Round-Robin/FIFO + CCV: ocurre lo mismo que en *Round-Robin/FIFO*, pero además cuando se produce una lectura de disco ese hilo se manda a la lista de bloqueados y se produce un cambio de contexto voluntario. Si el hilo bloqueado es de alta prioridad, cuando se desbloquee se producirá un cambio de contexto si el que se ejecuta es de baja prioridad. Si no, se mandará a la última posición de la lista de listos de alta prioridad.

Prueba 6: Prueba 4 + bucles de larga duración

En este caso se han añadido más bucles en las funciones de ejecución de los hilos, para así variar su duración y verificar el correcto funcionamiento en este caso.

Round-Robin: el hecho de añadir bucles entre la generación de hilos hace que el hilo principal agote antes los ticks asignados, por lo que se inicializarán menos hilos por rodaja. Por lo tanto, el tiempo que tardará en completar su tarea (inicializar hilos) el hilo principal será mayor. Luego al variar los bucles de las funciones de ejecución de cada hilo, algunos hilos tardarán varias rodajas más en finalizar, produciendo así más cambios de contexto.

Round-Robin/FIFO: el funcionamiento será igual que en el caso anterior, salvo que se tienen en cuenta las prioridades de los hilos.

Round-Robin/FIFO + CCV: este planificador funciona igual que el anterior, pero introduciéndose las interrupciones de disco. Por lo tanto, también podemos confirmar la validez de este caso

Prueba 7: Ejecución de un único hilo

Ejecutamos el hilo principal con baja prioridad con una duración larga, de manera que sus ticks se puedan agotar. También añadimos lecturas de disco.

Round-Robin: al ser un único hilo, no se produce ningún cambio de contexto. Con esto, podemos verificar el requisito de que no se realice un *swapcontext* cuando el hilo al que cambiamos es el mismo.

Round-Robin/FIFO: mismo comportamiento que *Round-Robin*.

Round-Robin/FIFO + CCV: el hilo principal se ejecuta indefinidamente, solo viéndose interrumpido cuando tiene que realizar un acceso al disco. En este caso, se cambia de contexto al hilo *idle*, hasta que el hilo principal vuelva a estar listo.

Prueba 8: Variable $QUANTUM\_TICKS = 0$

Round-Robin: en este caso, se produce un número mucho mayor de cambios de contexto entre hilos que en los casos anteriores. Cada uno de los hilos tiene una cantidad minúscula de tiempo asignada para cada *slice* y es por esto ese aumento de cambios de contexto. Sin embargo, el funcionamiento es el apropiado.

Round-Robin/FIFO: en esta situación, todos los hilos de alta prioridad tienen preferencia, ejecutándose hasta su finalización. Una vez éstos han terminado, se ejecutan los de baja prioridad, afectados por el planificador *Round-Robin*, por lo que se producen múltiples cambios de contexto entre ellos hasta su finalización, a una frecuencia mucho mayor que la producida con el valor de $QUANTUM\_TICKS$ por defecto.

Round-Robin/FIFO + CCV: este caso es igual al anterior, pero añadiéndose cambios de contexto cuando se produce una lectura de disco. Por lo tanto, los hilos de alta prioridad se ejecutan de forma ininterrumpida salvo que haya una interrupción de disco. En cuanto a los hilos de baja prioridad, el funcionamiento es el mismo que en los dos casos anteriores.

Prueba 9: Variable $QUANTUM\_TICKS = 10000$

Round-Robin: con un valor de ticks muy alto, esencialmente estamos provocando una planificación FIFO en los hilos. Todos los hilos son capaces de terminar su tarea, por lo que no se producen cambios de contexto debido a agotar los ticks asignados.

Round-Robin/FIFO: mismo comportamiento que con *Round-Robin*, con la diferencia de que aquí los hilos de prioridad alta se ejecutarán antes y, una vez todos se hayan ejecutado, lo harán los de

prioridad baja.

Round-Robin/FIFO + CCV: el comportamiento es igual que en el caso anterior, añadiéndose las interrupciones por lectura de disco, que provocan que el hilo pase a bloqueado. Si no hay más hilos listos, entrará en acción el hilo *idle* hasta que alguno se desbloquee.

Se han realizado muchas pruebas distintas para la verificación de las especificaciones de nuestro código. En cada una de ellas, hemos probado un número diferente de hilos, longitudes de función, y número de lecturas de disco. Sin embargo, las pruebas expuestas son las más representativas y engloban al conjunto de pruebas realizadas. Creemos que estas son suficientes para demostrar la correctitud y completitud de nuestra implementación para los requisitos de la práctica.

6. Conclusión

Con la realización de esta práctica hemos obtenido un entendimiento en profundidad de la forma en la que un sistema operativo gestiona los procesos que se están ejecutando. Los planificadores a implementar nos han servido para aprender cómo se organizan los distintos hilos, qué elementos se tienen en cuenta durante su planificación, etc.

La evolución gradual de los requisistos para el planificador a implementar nos ha permitido comprender cómo funciona un planificador *Round-Robin*, cómo se organizan los distintos hilos en el sistema en función de su prioridad, y cómo gestionar la ejecución cuando se producen peticiones al *hardware* por parte de los procesos.

El punto de mayor dificultad en la práctica ha podido deberse a esta misma evolución gradual del planificador. El cómo implementar los nuevos cambios y requisitos en el sistema previamente evaluado, de tal forma que se cumplimentase todo lo pedido sin provocar una malfunción en lo que ya se encontraba previamente implementado. El testeo de los resultados de la implementación de los distintos planificadores es una labor que se ha tenido que llevar de forma metódica y con sumo cuidado, teniendo en cuenta en todo momento en qué contexto se encontraba el hilo, los ticks restantes, su prioridad y si nuevos procesos que pasaban a estar listos para su ejecución podían provocar un cambio de contexto.

En resumen, la práctica ha sido de gran utilidad para mejorar nuestro entendimiento acerca de la planificación de procesos, y su dificultad gradual sirve para entender de forma ordenada las distintas variables que influyen el dicha planificación.