Práctica – 13

Simulación de un pequeño Scheduler y manejo de Memoria Central

de un Sistema de Cómputo (co).

Scheduling FCFS y Asignación en memoria de un proceso.

Usted agregará al programa concurrente desarrollado en la práctica 12 la funcionalidad de asignar los procesos a una memoria central (main memory).

Así ahora un ejemplo del archivo batch *cmdbatchSM.txt*, sería el siguiente:

|  |
| --- |
| *run* prog9.exe 3 4 |
| *run* nevada.exe 2 3 |
| *run* tres.exe 1 2 |
| - - - - - - - - - |
| *run* puertas.exe 3 1 |
| *run* chequera.exe 2 4 |

Donde en cada renglón se guarda el comando de mandar a ejecutar (*run*) un programa, un primer número que indica la cantidad de *CPU Bursts* que tendrá como proceso, y un segundo número que indica la cantidad de páginas de memoria a ocupar. Obviamente estos números son parte de la simulación.

El algoritmo de planificación empleado seguirá siendo el non-preemptive FCFS, ya programado en la práctica 12.

Dado que ahora tendremos que manejar los procesos en memoria, necesitamos de algunas estructuras como serían la clase *MemCen* y la clase *Swap*, de las cuales instanciaríamos un objeto de cada clase.

El objeto de la clase *MemCen* es el que representaría a la memoria central, que constaría de 30 páginas. Inicialmente estas 30 páginas están disponibles.

Un objeto de la clase *Swap*, que representa un área de un disco, sería capaz de almacenar hasta 50 procesos, en otras palabras puede almacenar hasta 50 referencias a objetos *ProcesoG*. Dentro de un objeto *Swap* no estamos manejando páginas.

Así, cada vez que un comando *run* con su programa ejecutable es leído del archivo, el estado NEW seguiría formando el nuevo proceso a partir del programa ejecutable, pero ahora tiene que marcar en *MemCen*, las páginas que ocuparía dicho procesos. También se requiere que se tenga una *PageTable*, tal vez dentro de cada proceso, que indique las páginas ocupadas. Suponga que un proceso jamás ocupara más de diez páginas. Como se ha hecho la vez pasada el proceso se pasaría a READY.

Pero qué pasaría si no hubiera suficientes páginas para acomodar el proceso (*ProcesoG*), entonces hay que ubicarlo en el área de *Swap*, y vamos a suponer (“invento”) que ahora el proceso está en estado READY-SWAP.

En general, toda la funcionalidad de la práctica 12 se sigue conservando, con las adecuaciones por la asignación de memoria a un proceso.

La gran pregunta es ¿en cuales puntos, estados o transiciones hay que producir un *Roll-in* o *Roll-out* de *Swap*, ya que si hay algún proceso pendiente de asignación de memoria y hay páginas suficientes en *MemCen*, dicho proceso deberá asignarse a memoria? Puede ser en READY, al terminar RUNNING (para ir a WAITING), en WAITING. En definitiva no manejaremos el de cuando un proceso está en WAITING.

Obviamente, el estado TERMINATED deberá encargarse de liberar memoria, dada su labor de dar por terminada la existencia de un proceso en el sistema de cómputo.

Dado que el sistema operativo es una aplicación concurrente, significa que muchas actividades llevándose a cabo “al mismo tiempo” o sea concurrentemente, debe evitar el tener grandes actividades de manera seriada. Además de lo solicitado en la práctica pasada debe observar:

* Que el sistema no debe terminar hasta que el último programa haya sido ejecutado.
* Cada vez que haya un cambio de estado deberá imprimirse tanto en la terminal como en el archivo de log, dicho cambio.
* Cada vez que un proceso sea asignado a memoria, o desasignado, deberán indicarse las páginas asignadas o las páginas desasignadas, imprimiendo el *PageTable* del proceso.

Bien, seguimos diciendo que parece que los cinco estados pudieran ser THREADS, que se arrancan y se mantienen según FCFS. También tenemos una serie de objetos comunes como:

* Las Colas de Estados
* La referencia al archivo de *bitácora*.
* Las referencias a los procesos.
* Algún otro objeto común que tenga el estado del sistema que permita al *main* sólo terminar cuando corresponda.

A lo mejor estos objetos comunes hay que aplicarles en algún momento *Exclusión Mutua* donde las secciones críticas deben estar en los métodos del objeto común. También se pudieran ayudar de los métodos para *Deadlocks*, pero esto déjenlo como última solución.