Sistemas Operativos

TP Scheduling

Grupo 20

Daniel Grosso	694/08	dgrosso@gmail.com
Mariano De Sousa Bispo	389/08	marian_sabianaa@hotmail.com

Septiembre 2010

1. Ejercicio 4

1.1. Análisis de los resultados de `ts1`

El valor de *Waiting Time Promedio (WTP)* obtenido tanto para `FCFS` como para `SJF` fue de 87 unidades de tiempo. Esto se debe a que los procesos tienen duración creciente a medida de que llegan a la cola, por lo tanto no difiere el orden según duración al orden de llegada. Esto implica que el tiempo de espera en ambos casos es el óptimo, ya que cada proceso nuevo que llega debe a lo sumo esperar a que terminen todos los procesos incluídos anteriormente, que son los de menor duración.

1.2. Análisis de los resultados de `ts2`

En este caso, el *WTP* obtenido con `FCFS` fue de 192 unidades de tiempo, mientras que con `SJF` fue de 122 unidades de tiempo. Esta diferencia se debe a que los procesos tienen duración decreciente a medida de que están listos, por lo tanto el orden según su duración es inverso al orden de llegada. Como la primera tarea del *taskset* tiene una duración de 80 y el resto de las tareas tiene un tiempo de liberación menor a 80, todas las tareas restantes están listas para ser ejecutadas antes de que la primera tarea termine. Esto implica que el tiempo de espera con `SJF` para las tareas es el óptimo, porque ejecuta primero las de menor duración. En el caso de `FCFS`, esto sucede al revés, es decir, el tiempo de espera es el peor posible.

1.3. Análisis de los resultados de `ts3`

Para `ts3`, el *WTP* obtenido con `FCFS` fue de 127 unidades de tiempo, mientras que para `SJF` fue de 105 unidades de tiempo. Todas las tareas del *taskset* llegan en el mismo instante, por lo que `FCFS` podría establecer cualquier orden. Esto hace que, en este caso, el *WTP* no dependa de la política de `FCFS` y que pierda sentido el valor obtenido.

1.4. Análisis de los resultados de `ts4`

Con `FCFS`, el *WTP* es de 77 unidades de tiempo, mientras que para `SJF` fue de 58 unidades de tiempo. La mejora de tiempo por parte de `SJF` viene dada porque se ordena tanto en el instante 0 como en el instante 100 los procesos

que pasan a estar en estado *ready* de menor a mayor (sobre la variable de la cantidad de tiempo que necesita el `CPU` para terminar de procesar). Al igual que en `ts3`, el orden en el que se agregan las tareas que ingresan en un mismo instante en `FCFS`, hace que el análisis de la diferencia de *WTP* obtenida pierda sentido.

1.5. Análisis de los resultados de `ts5`

El task set cinco, modela los procesos entre un servidor, y varios clientes. Los valores de *WTP* son para `FCFS` de 144 unidades de tiempo y para `SJF` de 56. Si bien a primera vista consideramos que conviene usar el scheduling implementado con el algoritmo de `SJF`, al analizar el output otorgado por la corrida del programa notamos que, el orden en el que ejecutan los procesos no es un orden que modele la relación entre un servidor y clientes. La base de datos como el servidor son apagados previo al encendido, y finalmente se corren los clientes. Podemos decir entonces que no es correcto, ya que el servidor, una vez que todos los clientes son procesados debería apagarse y esto no ocurre. En `FCFS` no ocurre este problema, ya que debido a su implementación inicialmente los procesos son ingresados a la cola en el orden correcto. Si no supiéramos en qué momento ha de ingresar un nuevo proceso, podría llegar a pasar una situación similar al caso que tenemos en `SJF` con este task set. Es importante saber cuál es el entorno en el que los procesos van a correr para poder decidir entonces qué tipo de *scheduling* se usa, intentando maximizar el rendimiento sin obtener resultados inconsistentes.

1.6. Análisis de los resultados de `ts6`

El sexto *taskset* modela también un servidor con varios clientes, en particular con dos clientes más que el *taskset* anterior. El *WTP* de `FCFS` es de 176 unidades de tiempo, contra 57 unidades de `SJF`. Si bien uno tendería a decir que en materia de performance el *taskset* con el *scheduling* `SJF` es mejor, pero incurriríamos en el mismo error que con el `ts5`. En este, se ejecutan clientes antes de que se inicialice la base de datos y el servidor. Como conclusiones, cabe destacar las mismas consideraciones que en el `ts5`.

2. Ejercicio 5

El tiempo de espera promedio para `ts1..4` con un *quantum* de 5 unidades de tiempo está entre 255 y 309 unidades de tiempo. Vemos que el tiempo de espera es alto. A pesar de que se desconoce la funcionalidad específica de las tareas que comprenden estos *tasksets*, podemos notar algunas consideraciones. Mantener este valor bajo de espera promedio nos serviría si se necesita tener algún tipo de interacción con el usuario, notar que este valor de *quantum* provoca que continuamente se esté haciendo cambios de contextos, con un costo importante debido a la gran cantidad de *task switch* que puede provocar. El valor del *quantum* debe analizarse con gran detalle sabiendo la carga y cantidad de trabajos que tendría el `CPU` corriendo con `Round Robin`. Si este *quantum* disminuye aumenta el porcentaje de tiempo que gasta realizando cambio de contexto en pos de una sensación de fluidez si se trabaja en entorno interactivos, pero si se aumenta considerablemente nos acercamos más a tener un modelo de scheduler `FCFS` (el cual aumentaría el rendimiento si tuviéramos procesos *batch*). Notar que en los tasks analizados si el *quantum* aumenta, el tiempo de espera media disminuye. Al igual que con el análisis del ejercicio cuatro, hemos notado la importancia de saber qué procesos son los que se están utilizando, para que el *scheduler* sea el más adecuando posible.

En los *tasksets* 4 y 5, tenemos waiting times de 154 unidades y 153 respectivamente pero no se está modelando el problema original que consistía en iniciar una base de datos, iniciar el servidor, permitir que clientes se conecten; y una vez desconectados todos cerrar el servidor y la base de datos. *Round Robin* otorga un *quantum* predefinido a cada proceso, y como en este caso, inicializar el servidor y base de datos cuesta más unidades de las otorgadas, se completan después que cliente comienza a ejecutarse. Una buena solución a este problema, hubiera sido ejecutar de forma `FCFS` todos los procesos que inicializan el server, pudiendo así poder rotar entre los distintos clientes con política `Round Robin`, con un *quantum* suficientemente bueno para dar la sensación de continuidad y además no derrochar excesiva cantidad de tiempo realizando cambios de contexto. Notar que una vez que todos los usuarios hayan terminado de realizar sus tareas, debería aplicarse nuevamente un scheduler `FCFS` para una correcta finalización del server y la base de datos.

3. Ejercicio 6

3.1. Análisis de los resultados de `ts1..6`

Los valores de *WTP* obtenidos con `MFQ` fueron de 171, 259, 217, 148, 138 y 144 unidades de tiempo respectivamente. Estos altos valores de *waiting time* se deben a que los procesos son bajados de prioridad cuando se les agota el *quantum* y deben esperar a que todos los procesos que tienen prioridad superior terminen para poder continuar su ejecución. En este caso, la cola de prioridad alta posee un *quantum* bajo de 5 unidades de tiempo. Como la mayoría de los procesos tienen una duración mayor a ese *quantum*, cuando este se les acaba son desalojados hacia la cola de prioridad media con un nuevo quantum ahora de 15 unidades. Esto implica que el tiempo de espera de cada tarea en la cola de prioridad media es alto. La gran desventaja de la cola de alta prioridad es que debido a su bajo *quantum*, se produce una gran cantidad de *task switch*, por lo que hay muchos tiempos muertos (2 por cada *switch*). Cuando las tareas son ejecutadas en la cola de prioridad media, se ejecutan por más tiempo para compensar el tiempo de espera y con un *quantum* mayor (de 15 unidades de tiempo), logrando en algunos casos terminar su ejecución. Por último, los procesos que no terminaron en prioridad media, son enviados a la cola de baja prioridad en donde se les otorga un *quantum* aún mayor al de la cola de prioridad media (de 45 unidades de tiempo), pero el tiempo de espera de los procesos que se encuentran en esta cola es muy alto. Estas tareas permanecen las próximas rondas allí hasta terminar su ejecución.