Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2333 - Sistemas Operativos y Redes

Reporte Tarea 1

1 de abril de 2019

1º semestre 2019 - Profesor Cristián Ruz

Paul Heinsohn Manetti - 1562305J

Pregunta 1

La primera desventaja identificable sucede cuando un proceso que aun no termina su *burst* es interrumpido y vuelve a usar la CPU inmediatamente, ya que este procedimiento conlleva a un costo extra de tiempo para suspender el proceso actual, cambiar el contexto y despachar el nuevo proceso, por lo tanto, si lo anterior ocurre ya sea porque no hay otro en estado *Ready*, o bien, porque es aquel con mayor prioridad en la cola, entonces se pierde tiempo de ejecución que podría ser aprovechado de otra manera.

También, al realizar estos cambios de contexto, la caché se va llenando con datos de los procesos y puede que reemplace aquellos datos que serán ocupados en posterioridad, por lo que se generan más *cache miss* causando que el acceso a los datos sea más lento, por lo tanto, los procesos demoran más y se pierde eficiencia.

Para el primer caso, basta que cada proceso tenga un *booleano* o *entero* que marque verdadero si el proceso fue head anteriormente o no. De esta manera, es posible realizar control de flujo sobre el tamaño de la cola y los elementos que la forman, en donde, si no hay elementos en la cola o la prioridad es menor a la del proceso en *running*, se reinicia el contador del *quantum* para que no cambiar de contexto. Por otra parte, para el segundo caso expuesto, supongo que bastaría tener una caché más grande.

Pregunta 2

En el caso de la presencia de un sistema *multicore*, bastaría con permitir la misma cantidad de elementos en *running* que de CPUs, pero siempre con el cuidado de fijar una prioridad distinta a cada una, es decir, debe haber una jerarquía entre las CPUs para saber a cual asignarle el siguiente proceso.

Para lograr lo anterior, supondré como código base el de mi tarea. En primer lugar, se podría tener un puntero que almacene la misma cantidad de *booleanos* que de CPUs, cuya prioridad sea definida acorde a su posición en dicho 'arreglo', así cada *bool* (CPU) va a representar un estado; T: disponible, F: ocupado. Además, en caso de que importe cuál CPU tiene qué proceso, va a ser necesario agregarle una variable del tipo *entero* a la estructura de 'Proceso' que indique la posición del *booleano* asignado para liberarlo cuando el proceso termine o sea interrumpido. Finalmente, se debe ir revisando en cada iteración cuáles son las CPUs que estan disponibles, ocupadas, deben ser liberadas y deben ser ocupadas por algún proceso.

En cuanto a las estadísticas, es muy probable que todos los tiempos disminuyan, ya que las probabilidades de que un proceso ocupe la CPU aumentan producto a la disponibilidad de estas, por lo tanto, un proceso tenderá a ser atendido por primera vez antes (*response time* disminuye), estará menos tiempo esperando (*waiting time* disminuye), y justo por esperar menos, tendremos que se completará antes (*turnaround time* disminuye).

Pregunta 3

La versión del algoritmo que mejor se comporta frente a un sistema interactivo es la *preemptive*, ya que estos sistemas son aquellos que dependen de las acciones de los usuarios, y como la velocidad de procesamiento de una CPU es muy alta, un usuario realmente la termina ocupado por un periodo de tiempo reducido, por lo que, al usar la versión *non-preemptive* la CPU estaría bastante tiempo 'ociosa' cuando pudiese ser aprovechada por otros procesos, que es exactamente lo que ofrece la versión escogida.

Bibliografía

- `http://retis.sssup.it/~giorgio/slides/rts/w06-lim-pre.pdf`