Trabajo Práctico - *Scheduling* de tareas

Sistemas Operativos - 2do cuatrimestre de 2010

Fecha de entrega: Hasta el Domingo 12 de Agosto de 2010, 23:59

Parte 1: Entrenamiento en el Simulador

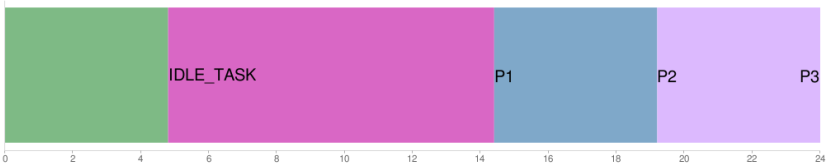
Una tarea (`Task`) se crea indicando los siguientes valores:

- Nombre: una etiqueta para identificar la tarea.
- Processing time: tiempo necesario para ejecutar la tarea en el procesador sin interrupción.
- Release time: tiempo en que la tarea pasa al estado *Ready*, lista para ser ejecutada.

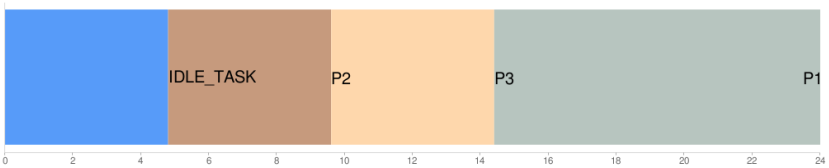
Un conjunto de tareas (`TaskSet`) se define como una lista de tareas.

Ejercicio 1 Completar el método `TaskSet::build_ej_1()` con una posible definición para un TaskSet tal que :

Su planificación FCFS es :



Su planificación SJF es :



Parte 2: Evaluación de algoritmos de scheduling

Nonpreemptive Scheduling

El Simulador cuenta con una implementación del algoritmo FCFS (*First-Come, First-Served*) y otra del algoritmo SJF (*Shortest-Job-First*).

Ejercicio 2 Modificar las clases `FCFS` y `SJF` para que calculen el *waiting time*[1].

Ejercicio 3 Completar la clase `SchedulerAnalyzer` para calcular el tiempo de espera promedio de un trabajo.

Ejercicio 4 Realizar un estudio del comportamiento de estas implementaciones utilizando los *tasksets* de la cátedra (aquellos con prefijo ts...). Calcular el *waiting time* promedio para cada caso.

Preemptive Scheduling

Ejercicio 5 Implementar el algoritmo de Round-Robin (en un único procesador) y realizar un estudio de su comportamiento utilizando los *tasksets* de la cátedra, utilizando varios valores de *quantum*.

- Para obtener un resultado más realista, utilice un retardo por cambio de contexto de **2 unidades de tiempo**. Durante ese período, ninguna tarea puede utilizar el procesador.

Ejercicio 6 Implementar el algoritmo de *Multilevel Feedback Queue*, utilizando tres colas con Round-Robin en cada una (similar a Windows NT) con los parámetros que se detallan a continuación, y realizar un estudio de su comportamiento utilizando los *tasksets* de la cátedra.

- La cola de alta prioridad tiene *quantum* 5, la de prioridad media tiene *quantum* 15 y la de prioridad baja tiene *quantum* 45.
- Utilice, tal como en el ejercicio anterior, un retraso de 2 unidades de tiempo en concepto de cambio de contexto entre tareas.
- Al ingresar un proceso en el sistema, el mismo se ubica al final de la cola de alta prioridad.
- Si hay algún proceso de alta prioridad se ejecuta el primero de ellos, si no hay ninguno se ejecuta el primer proceso de prioridad media. Si no hay procesos de prioridad alta ni media, se ejecuta el primer proceso de prioridad baja. Si no hay ningún proceso se espera la llegada de algún proceso.

[1] Suma de los períodos invertidos por el proceso en esperar en la cola de procesos preparados. (Silberschatz p.141

- Un proceso que agota su *quantum* es enviado a la cola inmediata inferior.
- Una vez que un proceso es pasado a una cola de prioridad inferior no puede volver.

Ejercicio 7 Modifique el algoritmo anterior de manera tal de que algunas veces los procesos realicen operaciones de entrada/salida. Esto se representará mediante la utilización de tan sólo una parte del *quantum* disponible.

- Utilizar números aleatorios para decidir si un proceso utilizará todo su tiempo asignado, o se irá voluntariamente antes. Utilice números aleatorios también para decir qué proporción de su tiempo usará, en caso de irse de forma anticipada.
- Un proceso que se retira voluntariamente sin utilizar todo su *quantum* es premiado, por lo tanto se lo coloca al final de la cola de prioridad inmediata superior a la que se encontraba.