DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico I

Scheduling

Sistemas Operativos

Primer Cuatrimestre 2015

Integrante	LU	Correo electrónico
Aldasoro Agustina	86/13	agusaldasoro@gmail.com
More Ángel	XXX/XX	mail
Zimenspitz Ezequiel	155/13	ezeqzim@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Resumen

En el presente trabajo se describe la problemática de ...

Índice

1. Parte I

1.1. Ejercicio 1: TaskConsola

Programar un tipo de tarea TaskConsola, que simulará una tarea interactiva. La tarea debe realizar n llamadas bloqueantes, cada una de una duración al azar entre bmin y bmax (inclusive). La tarea debe recibir tres parámetros: n, bmin y bmax (en ese orden) que serán interpretados como los tres elementos del vector de enteros que recibe la función.

Al momento de ejecutar la tarea *TaskConsola*, lo que realiza nuestro algoritmo es *uso_IO* (sistema operativo???) n veces, eligiendo cada vez un número al azar -mediante la función rand()- entre bmin y bmax.

1.2. Ejercicio 2: Ejecución de tres tareas

Escribir un lote de 3 tareas distintas: una intensiva en CPU y las otras dos de tipo interactivo (TaskConsola). Ejecutar y graficar la simulación usando el algoritmo FCFS para 1, 2 y 3 núcleos.

Las tareas utilizadas en el lote de tareas fueron las siguientes:

```
TaskConsola 4 2 7
TaskCPU 3
@3:
TaskConsola 5 1 10
```

Para los tres casos se consideró un costo de una unidad para cambiar de contexto y para cambiar un proceso de núcleo de procesamiento.

Los diagramas de Gantt obtenidos fueron los siguientes:

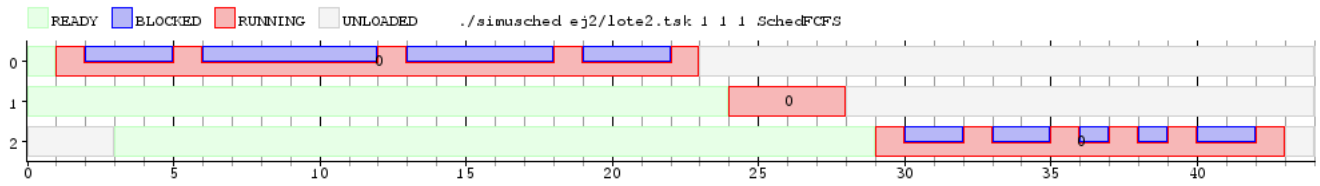

Figura 1: Diagrama de Gantt para la ejecución del lote de tareas bajo 1 núcleo.

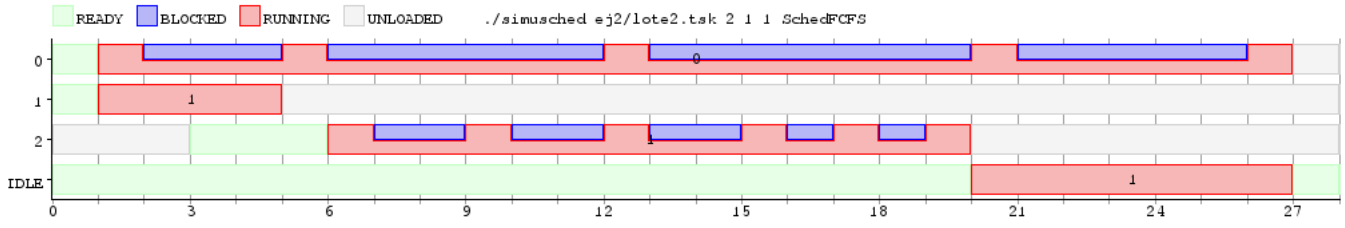

Figura 2: Diagrama de Gantt para la ejecución del lote de tareas bajo 2 núcleos.

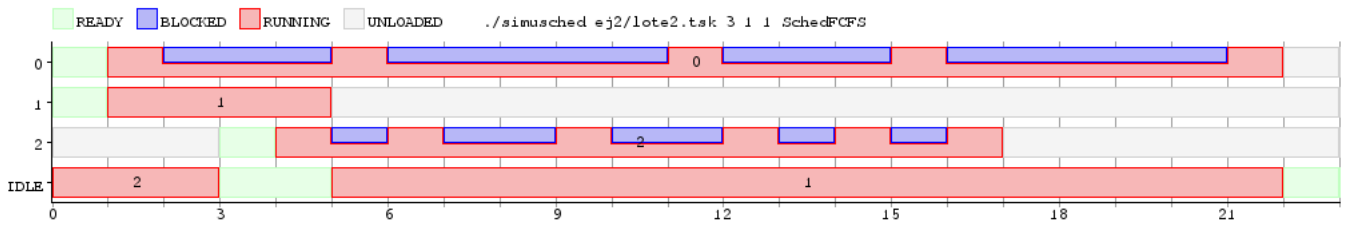

Figura 3: Diagrama de Gantt para la ejecución del lote de tareas bajo 3 núcleos.

Aca habria que hablar un poco :D

2. Parte II

2.1. Ejercicio 3: Scheduler Round-Robin

Completar la implementación del scheduler Round-Robin implementando los métodos de la clase SchedRR en los archivos sched_rr.cpp y sched_rr.h. La implementación recibe, como primer parámetro, la cantidad de núcleos y a continuación los valores de sus respectivos quantums. Debe utilizar una única cola global, permitiendo así la migración de procesos entre núcleos.

Las estructuras de datos con las que vamos a trabajar, en la clase *SchedRR*, son las siguientes:

```
int cores;
vector<int> quantums;
vector<int> quantums_timer;
vector<int> actuales;
int siguiente;
queue< int, deque<int> > cola;
```

- **cores** es la cantidad de núcleos.
- **quantums** es un vector de *cores* posiciones que guarda en quantums[i] el valor del quantum del núcleo i.
- **quantus_timer** es un vector, tal que quantus_timer[i] representa el valor de lo que resta del quantum de la tarea corriendo en el núcleo i. es asi esto que puse?
- **actuales** son las *cores* tareas que están corriendo ahora. Para la posición actuales[i] será la tarea que esté corriendo en el núcleo i (si no hay nada, sera la Tarea Iddle).
- **siguiente** indica el core que corresponde a utilizar próximo.
- **cola** es la cola de tareas que restan ser ejecutadas.

Modificamos a la funcion *next* para que reciba un parametro más (enum Motivo { TICK, BLOCK, EXIT }).

```
    int next(int cpu, const enum Motivo m);
```

Constructor Scheduler Round-Robin

Al construir un Scheduler Round-Robin, se instancian las estructuras de datos de modo que: se le asigna la cantidad de cores correspondientes (con sus respectivos quantums), todas las tareas actuales se definen como Iddle, la cola está vacía y el siguiente núcleo que le corresponde ejecutar es el primero ingresado como parámetro.

Función Load

Recibo un *pid* como parámetro, si tengo algún nucleo libre se la asigno a él sino encolo el *pid* en la cola global *cola*. No se porque hace siguiente = cores

Función Tick

El tick pasa para un solo núcleo, cada uno tiene su clock??.Recibe *cpu* como parámetro. Sólo voy a actualizar el quantum (restarle 1) de la tarea en *cpu* si no es la Iddle, porque esta corre indefinidamente hasta que la desplace otra tarea.

Si se acabó el quantum de la tarea actual (ya sea porque terminó o no) o se bloqueó, voy a querer actualizar la tarea actual. Para esto, invoco a la función *next()*.

Devuelve el *pid* de la tarea actual ejecutándose, al pasar el tick del clock.

Función Next

La función Next también va a ser invocada para un sólo núcleo *cpu* pasada por parámetro.

De este modo, si la tarea que se estaba ejecutando previo a la invocación de *next()* terminó de ejecutarse y la cola se encuentra vacía, la tarea que pondrá a ejecutar va a ser la Tarea Iddle.

Si la tarea que se estaba ejecutando no terminó su ejecución o se bloqueo, se deberá encolar en la cola global de pendientes.

Si todavía no asigné una tarea actual para el núcleo cpu, le asigno la primera de la cola dándole todo el quantum disponible para el núcleo *cpu*. (Si la cola se encontraba vacía al llamar la función next() y la tarea ejecutándose no había concluído se la encolará para luego volver a asignársela al núcleo).

Devuelve el *pid* de la tarea actual ejecutándose.

2.2. Ejercicio 4: Ejecución de lotes de tareas

Diseñar uno o más lotes de tareas para ejecutar con el algoritmo del ejercicio anterior. Graficar las simulaciones y comentarlas, justificando brevemente por qué el comportamiento observado es efectivamente el esperable de un algoritmo Round-Robin.

Primero ejecutamos el mismo lote de Tareas utilizado anteriormente:

```
TaskConsola 4 2 7
TaskCPU 3
@3:
TaskConsola 5 1 10
```

También lo corrimos para 1, 2 y 3 núcleos, cada uno con un quantum de 1, 2 y 3 respectivamente. Para los tres casos se consideró un costo de una unidad para cambiar de contexto y para cambiar un proceso de núcleo de procesamiento.

Los diagramas de Gantt obtenidos fueron los siguientes:

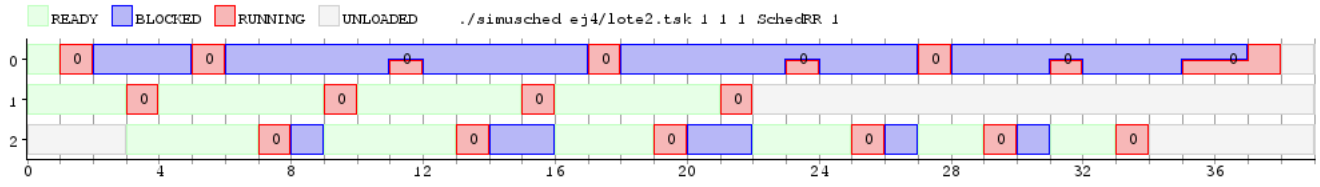

Figura 4: Diagrama de Gantt para la ejecución del lote de tareas bajo 1 núcleo.

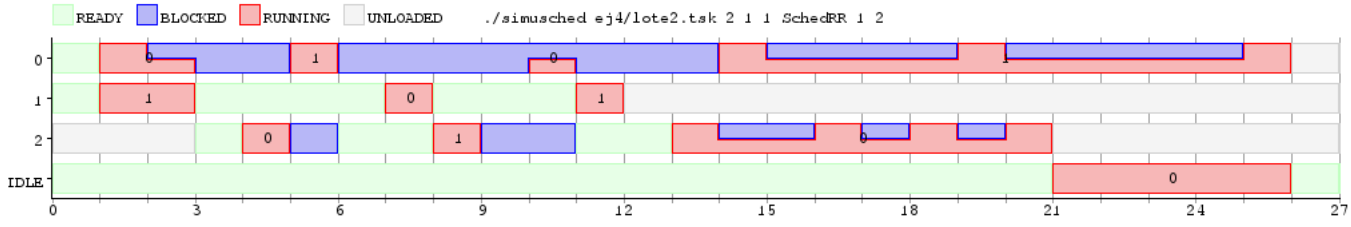

Figura 5: Diagrama de Gantt para la ejecución del lote de tareas bajo 2 núcleos.

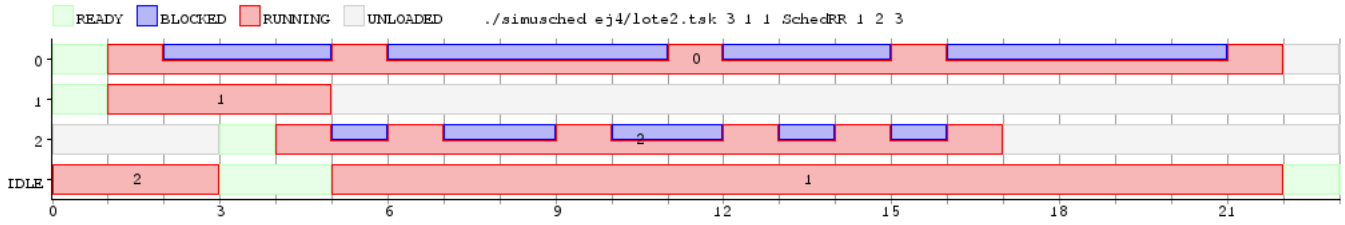

Figura 6: Diagrama de Gantt para la ejecución del lote de tareas bajo 3 núcleos.

Y aca comparamos un poco con como fue la ejecucion antes blablabla

Y aca van otros dos lotes comparando un ejemplo bueno de round robin y uno malo

2.3. Ejercicio 5: Scheduling algorithms for multiprogramming in a hard-real-time environment

A partir del artículo Liu, Chung Laung, and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM) 20.1 (1973): 46-61.

1. *Responda:*

   a) *¿Qué problema están intentando resolver los autores?*

   The problem of multiprogram scheduling on a single processor is studied from the viewpoint of the characteristics peculiar to the program function that need guaranteed service. It is shown that an optimum fixed priority scheduler possesses an upper bound to processor utilization which may be as low as 70 percent for large task sets. It is also shown that full processor utilization can be achieved by dynamically assigning priorities on the basis of their current deadlines. A combination of these two scheduling techniques is also discussed.

   b) *¿Por qué introducen el algoritmo de la sección 7? ¿Qué problema buscan resolver con esto?*

   We turn now to study a dynamic scheduling algorithm which we call the deadline driven scheduling algorithm. Using this algorithm, priorities are assigned to tasks according to the deadlines of their current requests. A task will be assigned the highest priority if the deadline of its current request is the nearest, and will be assigned the lowest if the deadline of its current request is the furthest. At any instant, the task with the highest priority and yet fulfilled request will be executed. Such a method of assigning priorities to the tasks is a dynamic one, in contrast to a static assignment in which priorities of tasks do not change with time. We want now to establish a necessary and sufficient condition for the feasibility of the deadline driven scheduling algorithm.

   c) *Explicar coloquialmente el significado del teorema 7.*

   For a given set of m tasks, the deadline driven scheduling algorithm is feasible if and only if

   (C1/T1) + (C2/T2) + ... + (Cm/Tm) <=1:

2. *Diseñar e implementar un scheduler basado en prioridades fijas y otro en prioridades dinámicas. Para eso complete las clases SchedFixed y SchedDynamic que se encuentran en los archivos sched fixed.[h—cpp] y sched dynamic.[h—cpp] respectivamente.*

3. Parte III

3.1. Ejercicio 6: TaskBatch

Programar un tipo de tarea TaskBatch que reciba dos parámetros: total cpu y cant bloqueos. Una tarea de este tipo deberá realizar cant bloqueos llamadas bloqueantes, en momentos elegidos pseudoaleatoriamente. En cada tal ocasión, la tarea deberá permanecer bloqueada durante exactamente un (1) ciclo de reloj. El tiempo de CPU total que utilice una tarea TaskBatch deberá ser de total cpu ciclos de reloj (incluyendo el tiempo utilizado para lanzar las llamadas bloqueantes; no así el tiempo en que la tarea permanezca bloqueada).

3.2. Ejercicio 7: Ejecución lote de tareas

Elegir al menos dos métricas diferentes, definirlas y explicar la semántica de su definición. Diseñar un lote de tareas TaskBatch, todas ellas con igual uso de CPU, pero con diversas cantidades de bloqueos. Simular este lote utilizando el algoritmo SchedRR y una variedad apropiada de valores de quantum. Mantener fijo en un (1) ciclo de reloj el costo de cambio de contexto y dos (2) ciclos el de migración. Deben variar la cantidad de núcleos de procesamiento. Para cada una de las métricas elegidas, concluir cuál es el valor óptimo de quantum a los efectos de dicha métrica.

3.3. Ejercicio 8: Scheduler Round-Robin modificado

Implemente un scheduler Round-Robin que no permita la migración de procesos entre núcleos (SchedRR2). La asignación de CPU se debe realizar en el momento en que se produce la carga de un proceso (load). El núcleo correspondiente a un nuevo proceso será aquel con menor cantidad de procesos activos totales (RUNNING + BLOCKED + READY). Diseñe y realice un conjunto de experimentos que permita evaluar comparativamente las dos implementaciones de Round-Robin.

3.4. Ejercicio 9: Ejecución lote de tareas

Diseñar un lote de tareas cuyo scheduling no sea factible para el algoritmo de prioridades fijas pero sí para el algoritmo de prioridades dinámicas.

3.5. Ejercicio 10: Ejecución lote de tareas

Diseñar un lote de tareas, cuyo scheduling sí sea factible con el algoritmo de prioridades fijas, donde se observe un mejor uso del CPU por parte del algoritmo de prioridades dinámicas.