Departamento de Computacin,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

Trabajo Práctico 1

Sistemas Operativos

22 de Abril de 2014

Apellido y Nombre	LU	E-mail
Silvio Vileriño	106/12	svilerino@gmail.com
Ezequiel Gambaccini	715/13	ezequiel.gambaccini@gmail.com
Martin Arjovsky	683/12	martinarjovsky@gmail.com

Contents

1 Introducción

El objetivo de este documento es recopilar las resoluciones de los ejercicios presentados en el enunciado del trabajo practico. En las secciones a continuación se detalla cada ejercicio.
Nota: Los ejercicios 1, 3, 5, 6 son implementaciónes y no serán tenidos en cuenta en la siguiente sección.

2 Desarrollo

2.1 Ejercicio 1

Ejercicio implementado en el código.

2.2 Ejercicio 2

A continuación se muestran los gráficos del lote 2 para 1, 2 y 3 núcleos.

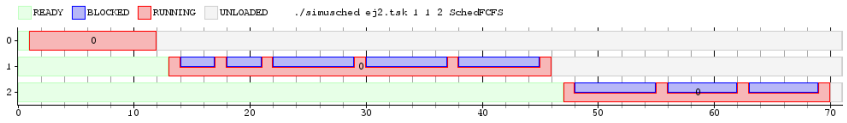

Figure 1: Diagrama de Gantt para el lote 2 en FCFS con un núcleo

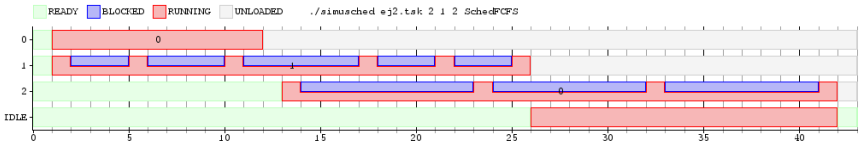

Figure 2: Diagrama de Gantt para el lote 2 en FCFS con dos núcleos

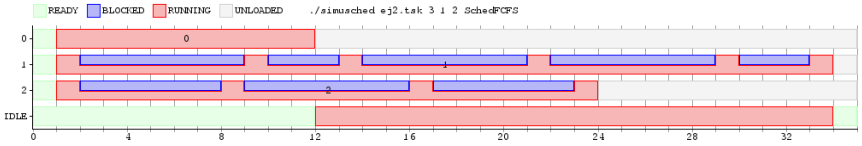

Figure 3: Diagrama de Gantt para el lote 2 en FCFS con tres núcleos

2.3 Ejercicio 3

Ejercicio implementado en el código.

2.4 Ejercicio 4

A continuación se muestran los gráficos del lote 4 ejecutado en el scheduler Round-Robin.

En ambos gráficos se observa el comportamiento esperado de un scheduler de tipo Round-Robin. En particular, en la figura 4 se ve como la tarea 0 corre

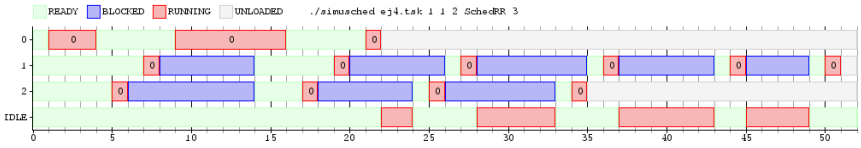

Figure 4: Diagrama de Gantt para el lote 4 en RR con un núcleo

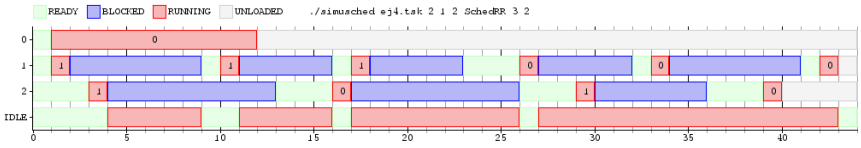

Figure 5: Diagrama de Gantt para el lote 4 en RR con dos núcleos

en el único núcleo hasta que en el tiempo 4 se le acaba el quantum disponible y el scheduler busca otro proceso disponible en la cola global. También se muestra repetidamente como cuando un proceso realiza una llamada bloqueante, el CPU deja esa tarea para ir a ejecutar otra. En la figura 5 se observa el mismo comportamiento, pero además podemos observar la migración de procesos entre núcleos, por ejemplo a tiempo 16 en la tarea 2.

2.5 Ejercicio 5

Ejercicio implementado en el código.

2.6 Ejercicio 6

Ejercicio implementado en el código.

2.7 Ejercicio 7

Para estudiar la performance del Round-Robin variando los cuantos creamos el lote7 y realizamos varias simulaciones. Variamos los quantums entre 1 y 10 inclusive y evaluamos el throughput y INSERTAR SEGUNDA MÉTRICA con 2 y 4 núcleos. Como las tareas TaskConsola son pseudoaleatorias, para cada seleccioón de parámetros corrimos 10 simulaciones y graficamos el promedio con la variación estandard.

En la figura 6 se muestran los resultados del throughput. En la misma se observa que (considerando las desviaciones) el throughput aumenta al aumentar el quantum, tanto para 2 como 4 núcleos. Esto tiene mucho sentido, ya que se gasta menos tiempo en cambios de contexto y es menos probable que un proceso cambie de CPU, bajando el costo total de migración de procesos entre CPUs. Obviamente, este crecimiento converge, ya que el tiempo que tardan en

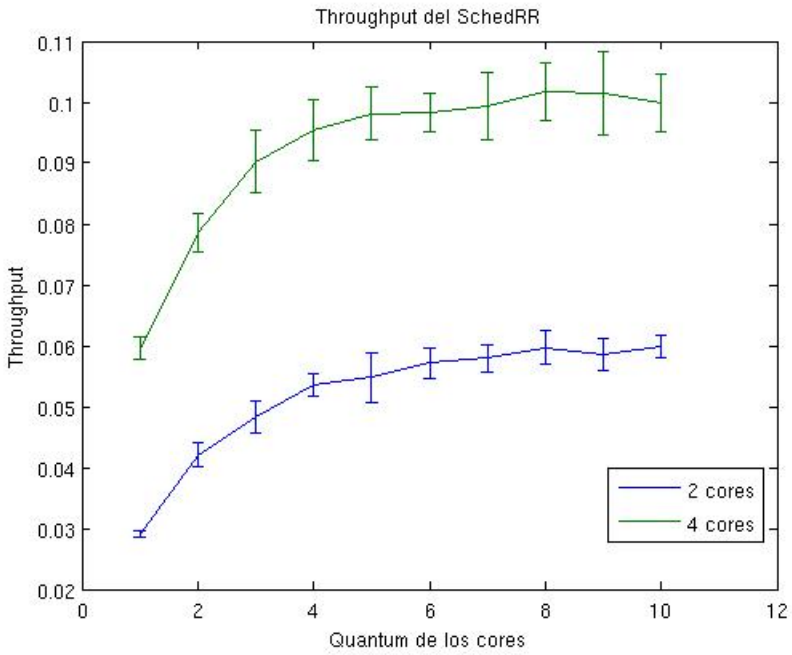

Figure 6: Throughput para SchedRR con dos y cuatro núcleos

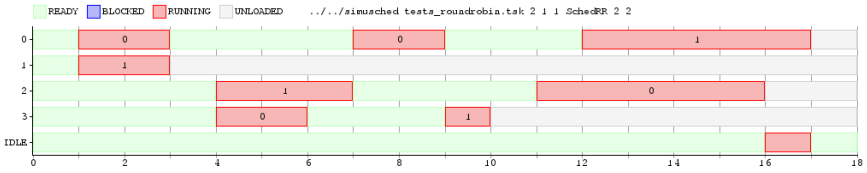

Figure 7: Diagrama de Gantt para el lote 8 en RR con dos núcleos

Figure 8: Diagrama de Gantt para el lote 8 en RR2 con dos núcleos

terminar los procesos está acotado por el uso total del CPU dividido la cantidad de núcleos, por lo que el throughput también converge.

2.8 Ejercicio 8

A continuación se muestran los gráficos del lote 8 ejecutado sobre los schedulers Round-Robin con y sin migración entre procesos.

Se puede observar en el caso donde hay tareas con considerable diferencia de duracion, una mejor performance del scheduler RR1, dado que realiza una correccion permanente del balanceo de carga usando los recursos que tiene a su alcance en cada tick. Mientras que en el RR2, al quedar fija la afinidad al momento del dispatch, si ocurre un caso donde ambas tareas cortas quedan en un core y las tareas largas en otro core, al finalizar las tareas cortas, se desperdiciara un core durante toda la ejecucion de las tareas largas. Esta diferencia se ve claramente en los gráficos 7 y 8 si notamos que cuando se usa RR1 se termina de ejecutar a tiempo 18 y con RR2 a tiempo 27, lo que es una diferencia abismal. Cabe destacar que para que esto pase se necesita que haya una gran diferencia entre el uso de CPU de las tareas y que las largas se encuentren concentradas en un núcleo mientras las cortas en otro, lo que no suele ser un caso promedio.

2.9 Ejercicio 9

Como se puede ver en los graficos de lote2 (lote2.txt, lote2rr.txt, lote2rr2.txt, lote2fcfs.txt), el unico scheduler que puede hacer que todas las tareas con deadline se realicen dentro de su limite es el EDF. El scheduler EDF posee una propiedad que es condicion suficiente para asegurar esto, ademas de asegurar optimalidad. Esta propiedad consiste en que si la sumatoria de la cantidad de ciclos de cada tarea divido su deadline es menor a 1, entonces el EDF va a

encontrar una forma de ejecutar todas las tareas, y que adems es ptima. Como se ve en los casos de las 2 implementaciones de round robin, y con el fcfs, todos fallan en lograr que todas las tareas se cumplan dentro de su deadline. El FCFS falla al ir asignando las tareas como vienen, sin tener en cuenta nada ms, por lo que al no priorizar las tareas con deadline prximos aunque hayan llegado ultimos, estos fallan. El round robin no logra cumplir con los deadline debido a que su objetivo es tratar de balancear el uso de cpu por parte de tareas que tienen uso intesivo de cpu y tareas bloqueantes, de manera tal que no haya starvation de cpu para las tareas bloqueantes, y mientras estas estan bloqueadas, maximizar el uso de cpu, usando un quantum para mantener un balance entre las tareas. Este quantum evita que una tarea prioritaria con deadline proximo y un requirimiento de tiempo de cpu mayor a quantum se complete en el tiempo requerido.

2.10 Ejercicio 10

La propiedad del edf en estos casos (lote_edf.tsk, lote_edf3.tsk) no se mantiene, porque aunque la suma de las tareas y sus deadlines es menor a 2, el scheduler es incapaz de hacer que se cumpla la tarea 3, aun cuando la propiedad del edf debiera ser valida. Para mantener la propiedad de que se cumplan los deadline en un scheduler multicore, se debera usar otro algoritmo que tenga en cuenta como maximizar el uso de los procesadores, ya que para el caso de lote_edf3.tsk, si el primer core hubiera corrido las 2 tareas de 5 y el segundo hubiese corrido la de 8, todas las tareas se habran podido lograr en el tiempo necesario.

2.11 Apéndice: Lotes de prueba

Listing 1: "Lote de tests ejercicio 2"

```
TaskCPU 10
TaskConsola 5 3 7
TaskConsola 3 5 9
```

Listing 2: "Lote de tests ejercicio 4"

```
TaskCPU 10
TaskConsola 5 3 7
TaskConsola 3 5 9
```

Listing 3: "Lote de tests ejercicio 7"

```
TaskBatch 20 5
TaskBatch 20 2
TaskBatch 20 8
TaskBatch 20 3
```

Listing 4: "Lote de tests ejercicio 8"

```
@0
TaskCPU  8
TaskCPU  1
TaskCPU  7
TaskCPU  2
```