DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico I

22 / 04 / 2015 Sistemas Operativos

Integrante	LU	Correo electrónico
Abdala, Leila	950/12	abdalaleila@gmail.com
Enrique, Natalia	459/12	natu_2714@hotmail.com
Salinas, Pablo	456/10	salinas.pablom@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

Índice

1. Introducción **3**

2. Ejercicios **3**
2.1. Ejercicio 1: TaskConsola . . . 3
2.2. Ejercicio 2: Experimentando FCFS . . . 3
2.3. Ejercicio 3: Implementando Round-Robin . . . 4
2.4. Ejercicio 4: Experimentando Round-Robin . . . 4
2.5. Ejercicio 5 . . . 5
2.6. Ejercicio 6: TaskBatch . . . 5
2.7. Ejercicio 7 . . . 5
2.8. Ejercicio 8: Round Robin sin migracion . . . 5
2.9. Ejercicio 9 . . . 5
2.10. Ejercicio 10 . . . 5

3. Concluciones **5**

1. Introducción

En este informe presentaremos la implementacion de diversos schedulers y una breve experimentacion comparativa entre estos. La idea de este TP es conocer las distintas formas de administrar el scheduler, entendiendo asi que ventajas presenta cada modo y en que contexto. El desarrollo del informe se basa en el enunciado, por lo que cada item del mismo tiene en este informe una seccion que responde al mismo.

2. Ejercicios

2.1. Ejercicio 1: TaskConsola

En éste ejercicio debemos programar una tarea *TaskConsola* la cual debe realizar n llamadas bloqueantes, cada una con una duración al azar entre bmin y bmax, ambas pasadas por parámetro.

Para ello, decidimos utilizar un contador de 0 hasta n y generar un número pseudo-alaeatorio por medio de la función *rand*. Es decir, cada vez que se aumenta el contador, realizamos una llamada bloqueante (uso_IO) que durará la cantidad de ciclos que se haya generado en la llamada a rand.

2.2. Ejercicio 2: Experimentando FCFS

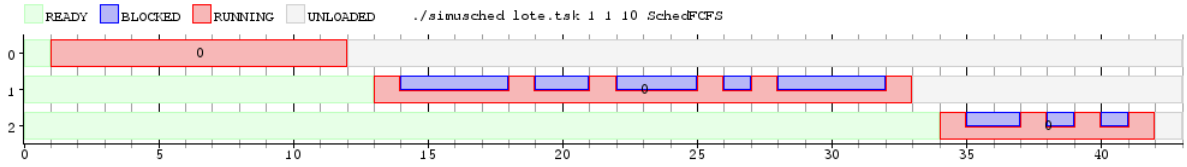
Para probar el Scheduler First Came First Served usaremos el siguiente lote de tareas:

TaskCPU 10
TaskConsola 5 1 5
TaskConsola 3 1 2

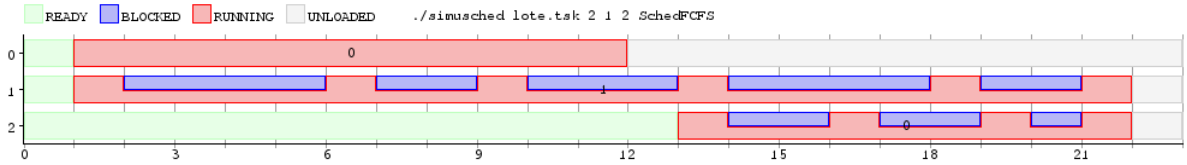
En la primera, utlizaremos TaskCPU y para darle uso intensivo correrá durante 10 ciclos de reloj. Las siguientes, son de tipo TaskConsola implementado anteriormente, pasándole como parámetro el número de llamadas bloqueantes y el rango en el cual debe seleccionar el número aleatorio.

En el algoritmo FCFS, la CPU se asigna a los procesos en el orden en que la solicitan.
Por lo tanto, esperamos observar que, con un solo núcleo, un proceso no pueda correr hasta que no terminaron los anteriores a el.

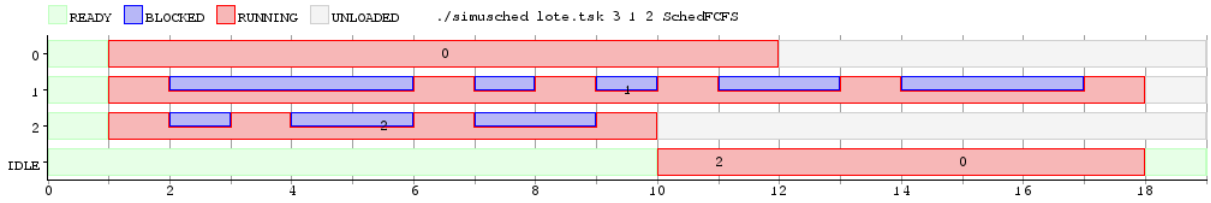
Con dos nucleos, correrán dos procesos simltáneamente y el último empezará cuando alguno de los otros dos finalicen, y por último, si el procesador tiene 3 núcleos, los 3 correrán al mismo tiempo.



Simulacro FCFS con un núcleo



Simulacro FCFS con dos núcleos



Simulacro FCFS con tres núcleos

Como podemos observar, los experimentos resultaron satisfactoriamente con nuestra hipótesis.

Además, notemos que en los casos en los que utilizamos la tarea TaskConsola, se observan claramente las llamadas bloqueantes de manera random, dado que el tiempo tanto de la ejecución como el de las llamadas bloqueantes varía. En el caso en que el procesador tiene tres núcleos, se observa que los núcleos 2 y 0 ejecutan Idle ya que están desocupadas, esperando la solicitud del próximo proceso.

2.3. Ejercicio 3: Implementando Round-Robin

La idea del scheduler *Round-Robin* es darle un quantum a cada procesador, cuando un proceso lo solicita, si algún núcleo esta desocupado, corre la tarea el tiempo que sea el quantum del procesador seleccionado.

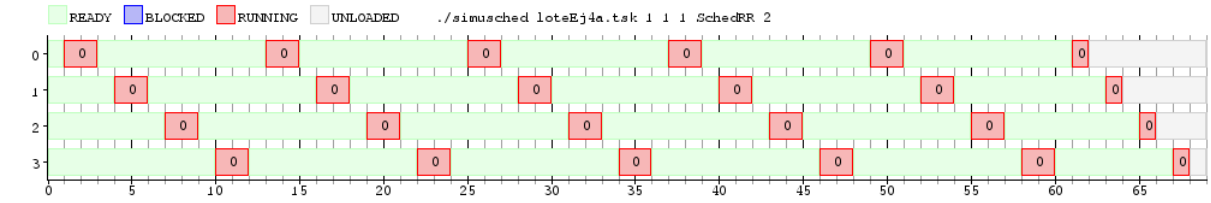
Con esta idea desarrollamos nuestro Scheduler Round-Robin. Utilizamos como estructura una Cola (q), para las llamadas a los procesos, un vector (*quantum*) de tamaño cantidad de cores del procesador que asignará el quantum del iésimo núcleo, y otro vector (*contador*) que va a llevar cuenta del tiempo corrido por el proceso en el iésimo núcleo hasta llegar al quantum del mismo.

Para el correcto funcionamiento en la función *tick* se ven reflejados los casos en el cual el proceso debe dejar de correr ya sea porque terminó su tiempo o el quantum del procesador en el que corría. En éste último caso, la posición correspondiente al núcleo en *contador* volverá a cero y el proceso se encolará para terminar con su tiempo.
Una vez realizada dicha acción, debe dar lugar a la siguiente en la cola, o en caso de no haber una la Idle deberá hacerlo hasta el llamado de una nueva.

2.4. Ejercicio 4: Experimentando Round-Robin

En este ejercicio nos proponemos experimentar con el scheduler del punto anterior para verificar que el comportamiento es el esperado. Haremos esto de manera incremental, es decir, empezaremos probando las cosas mas basicas e iremos subiendo la complejidad.

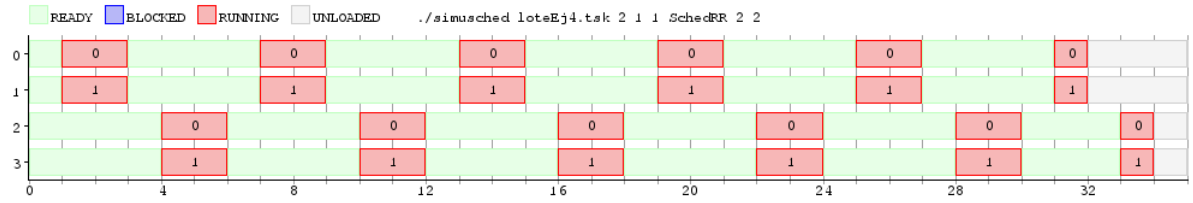
Para empezar, probaremos RR con un solo nucleo. Notese que si le asignamos una sola tarea, su comportamiento no diferiria de algun otro scheduler, por lo que empezamos probando con cuatro tareas simultaneas. Estas tareas solo usan al cpu, por lo tanto no se bloquean. Esperamos verificar que el scheduler RR le asigna el tiempo del quantum a cada tarea antes de comenzar de nuevo la lista de tareas pendientes.



Simulacro RR con un núcleo

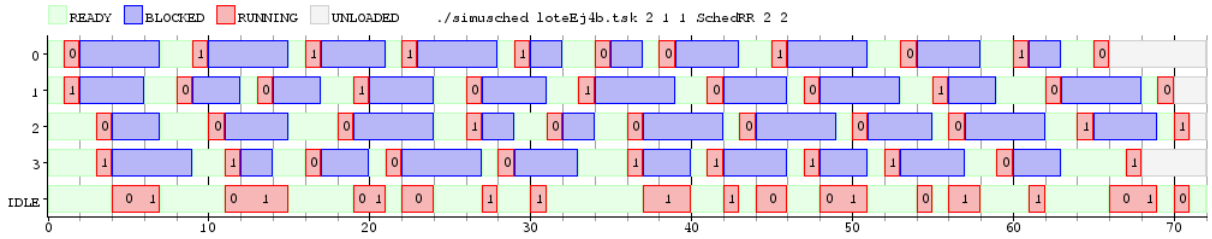
Efectivamente, cuando recibe k tareas simultaneas, el scheduler le asigna tiempo de ejecucion a las tareas de modo que todas ejecuten antes de regresar a ejecutar la primera.

A continuacion usaremos el mismo lote de tareas, pero agregaremos otro procesador. Asi veremos si el scheduler maneja correctamente los procesadores para asegurarse que todas las tareas ejecuten un tiempo *quantum* antes de volver a empezar.



Simulacro RR con dos núcleo

El comportamiento es identico al anterior, solo que agregando otro nucleo, es decir, podemos deducir las mismas concluciones. Para probar de modo mas realista este scheduler, vamos a simular un lote de cuatro tareas que realicen llamadas bloqueantes, con dos procesadores para su ejecucion. Lo que queremos mostrar, es que el scheduler trabajara de manera optima en este caso.



Simulacro RR con dos núcleo

Podemos ver que el scheduler trabaja de la manera esperada. Es decir, si hay un tarea disponible para ejecutar y procesador que no esta ejecutando nada, ese procesador carga la tarea y la ejecuta, o bien hasta que se le acabe el quantum o hasta que se bloquee. Si todas las tareas estan bloqueadas o ejecutando cuando el procesador termina la ejecucion de una tarea porque se bloqueo, este se poner a ejecutar la tarea IDLE hasta que una tarea se desbloque. Y por lo tanto, si todas las tareas estan bloqueadas, ambos procesadores ejecutan la tarea IDLE hasta que alguna tarea se desbloquee.

2.5. Ejercicio 5

2.6. Ejercicio 6: TaskBatch

En este ejercicio implementamos la tarea TaskBatch, que recibe como parametros totalcpu y cantbloqueos. Esta tarea dura totalcpu tiempo de cpu, y realiza cantbloqueos llamadas bloqueantes en momentos psudoaleatorios. Realizamos de dos maneras. La primera implementacion toma iteraba totalcpu veces pidiendo un numero aleatorio rand. Si rand >0.5 entonces llama a una tarea bloqueante, sino, a una tarea que use el cpu. Si la cantidad de iteraciones se acaba sin haber realizado todas las llamadas bloqueantes, entonces se realizaban las restantes seguidas al final. Sino, cuando se realizaba la ultima llamada bloqueante, se utilizaba el tiempo restante del cpu todo junto. Lo que observamos en esta implementacion fue que las llamadas bloqueantes se hacian para tiempos grandes siempre al principio del intervalo de uso del cpu, y para tiempos de medianos a cortos, siempre al final.**********

Por eso realizamos otra implementacion. En esta se seleccionan previamente en que momentos se van a realizar las llamadas bloqueantes, y luego se itera sobre el tiempo del cpu.

2.7. Ejercicio 7

2.8. Ejercicio 8: Round Robin sin migracion

2.9. Ejercicio 9

2.10. Ejercicio 10

3. Concluciones

Se vale concluir que odio el gdb?