Trabajo Práctico 1

Martes 16 de Septiembre de 2014 — Sistemas Operativos

Scheduling

Integrante	LU	Correo electrónico
Pedro Rodriguez	197/12	pedro3110.jim@gmail.com
Gonzalo Benegas	???	gsbenegas@gmail.com
Ezequiel Barrios	???	ezequiel.barrios@gmail.com

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

Contents

Abstract

En el presente trabajo práctico estudiaremos algunos de los métodos más comúnmente utilizados en la actualidad por distintos sistemas operativos para manejar correcta y eficientemente los diversos procesos que se ejecutan concurrentemente en máquinas con uno o más procesadores.

Intentaremos detectar las ventajas y desventajas de cada método, así como los escenarios en los cuales uno puede ser más eficiente que otro. Para esto, dividimos el TP en tres partes: en la primera, presentamos el simulador simusched y lo corremos para algunas tareas específicas. En la segunda parte, extendemos el simulador con nuevos schedulers implementados por nosotros y finalmente, en la tercera parte evaluamos y analizamos los distintos algoritmos de scheduling ya presentados.

1 Parte I - Entendiendo el simulador simusched

1.1 Ejercicio 1

En este ejercicio, programamos la tarea TaskConsola, que simularía una tarea interactiva con el usuario.
Para la implementación de esta tarea, primero tuvimos que registrarla para que el simulador la reconozca en el archivo tasks.cpp, con la función tasksinit(void). En ese mismo archivo implementamos la tarea, que para el proceso número pid recibe los tres parámetros: n, bmin y bmax. Simplemente hacemos un ciclo que cicle n veces y que cada vez tome un entero al azar r entre $bmin$ y $bmax$, y cada vez llamamos a la función usoIO(pid, r), que simula hacer uso de dispositivos de entrada/salida.
Nota(1): se denomina llamada bloqueante a aquellas llamadas en las que, si lo que esta solicita no está disponible, entonces el proceso llamador se queda bloqueado a la espera de un resultado. Es decir, no permite que otros procesos dependientes de este sigan ejecutando. Por el contrario, en una llamada no bloqueante, si el proceso llamador no encuentra lo que estaba buscando, el proceso devuelve de todas formas algún resultado, permitiendo que otros procesos sigan ejecutando sin tener que esperar a que este proceso llamador reciba el resultado que está esperando.
Nota(2): para elegir el número pseudo-aleatorio hacemos uso de la función rand() provista por la librerá standard de C.

1.2 Ejercicio 2

Para seguir entendiendo el funcionamiento del simulador simusched, ahora pasamos a escribir un lote de 3 tareas distintas: una intensiva en CPU y las otras dos de tipo interactivo. Vamos a ejecutar y graficar la simulación usando el algoritmo FCFS para 1, 2 y 3 núcleos.

2 Parte II - Extendiendo el simulador con nuevos schedulers

2.1 Ejercicio 3

2.2 Ejercicio 4

2.3 Ejercicio 5

En este ejercicio, basados en el paper de Waldspurger, C.A. y Weihl en el cuál presentan un novedoso sistema de Scheduling llamado *LotteryScheduling*, implementamos una clase que llamamos SchedLottery que recibe como parámetros el quantum que se va a estar simulando y una semilla pseudo-aleatoria, que el scheduler va a utilizar para el manejo de los procesos. Como dice en el enunciado, básicamente nos interesó implementar la idea básica del algoritmo y la optimización de tickets compensatorios.

3 Parte III - Evaluando los algoritmos de scheduling

3.1 Ejercicio 6

3.2 Ejercicio 7

3.3 Ejercicio 8

3.4 Ejercicio 9

3.5 Ejercicio 10