Sistemas Operativos

Primer Cuatrimestre de 2010

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Taller N°1

Schedulers

Grupo

Integrante	LU	Correo electrónico
Mataloni Alejandro	706/07	amataloni@gmail.com
Mancuso Emiliano	597/07	emiliano.mancuso@gmail.com

1. Ejercicio 1

Partiendo de los 2 gráficos de la cátedra identificamos los distintos tiempos de cada proceso. Identificamos 3 procesos y un tiempo IDLE del procesador, lo que nos dio el pie para averiguar a partir de que momento los procesos estaban listos.
Hasta ahora son 5 ciclos que el procesador esta en IDLE, el P1 demora 10 ciclos y P2 y P3 demoran 5 ciclos cada uno. Con el primer gráfico (FCFS) sabemos el orden de llegada de los procesos, que es justamente P1,P2,P3 y con el segundo (SJF) nos damos cuenta de que los tres procesos están listos al mismo tiempo.

Configuración de procesos

	Demora	Listo
P1	10	5
P2	5	5
P3	5	5

2. Ejercicio 2

Para calcular el *waitingTime* de cada tarea sin tener que modificar cada algoritmo, implementamos una función *finalize* que además de dar por finalizada la tarea, calcula el tiempo de espera.

```
1: public void finalize(int current_time){
2:     this.ftime = current_time;
3:     this.wtime = (ftime - rtime) - ptime;
4: }
```

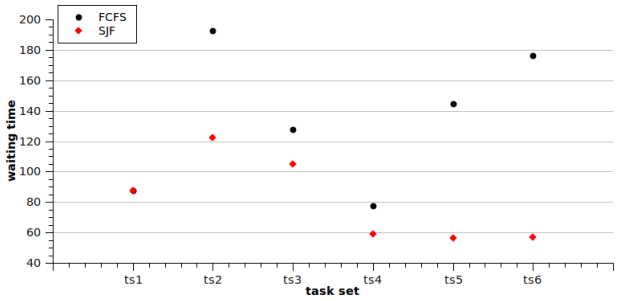
```
ftime: finished time
rtime: ready time
ptime: process time
```

3. Ejercicio 3

Gracias al diseño de la clase *Task* que almacena el el tiempo de espera de cada tarea y a los algoritmos de Scheduling que conservan las tareas finalizadas, calcular el tiempo de espera promedio se resuelve con una sencilla cuenta.

```
1: public double get_avg_wtime(){
2:     double avg = 0.0;
3:     for (Task t : this.task_set)
4:         avg += t.wtime;
5:
6:     return avg / this.task_set.size();
7: }
```

4. Ejercicio 4

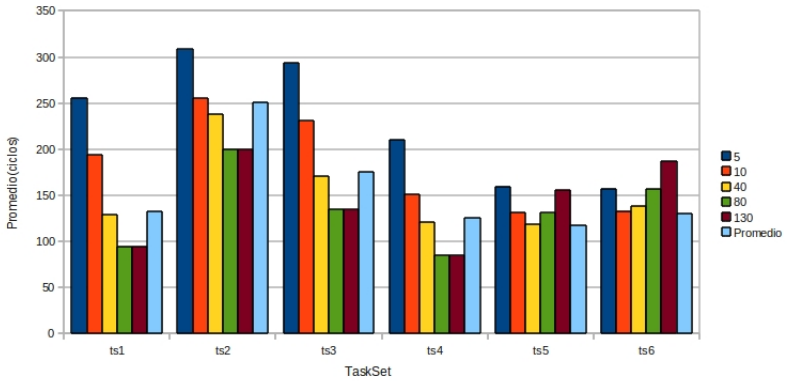


Realizando el estudio utilizando los *taskset* de la cátedra pudimos observar el comportamiento (en cuanto al waiting time de los procesos) de los schedulers del tipo $FCFS$ y SJF. El waiting time promedio es menor en SJF. Esto se debe a que justamente con ésta política garantiza que los que menos tardan se ejecutan primero y de esta manera se evita que un proceso que tarda mucho le sume waiting time a otros procesos mas cortos.

5. Ejercicio 5

Realizando varios estudios sobre el comportamiento de RR utilizando diferentes quantums observamos que a medida que se le asigna un mayor valor, el tiempo de espera promedio disminuye. Esto se debe a que los procesos en un quantum asignado pueden ejecutar más, por lo tanto van a terminar más rápido. Es decir, mientras más interrupciones tenga un proceso, crece el tiempo de espera promedio. Para obtener un mejor promedio se aumenta el valor del quantum pero verificando que no se perjudiquen los beneficios del algoritmo de scheduler.

Si el quantum toma el máximo tiempo de proceso, de algún proceso en el taskset, se pierde el preemption que ofrece el algoritmo. Por eso, nosotros encontramos como buen valor el promedio de los tiempos de proceso dentro de un mismo TaskSet, para disminuir el tiempo de espera promedio.



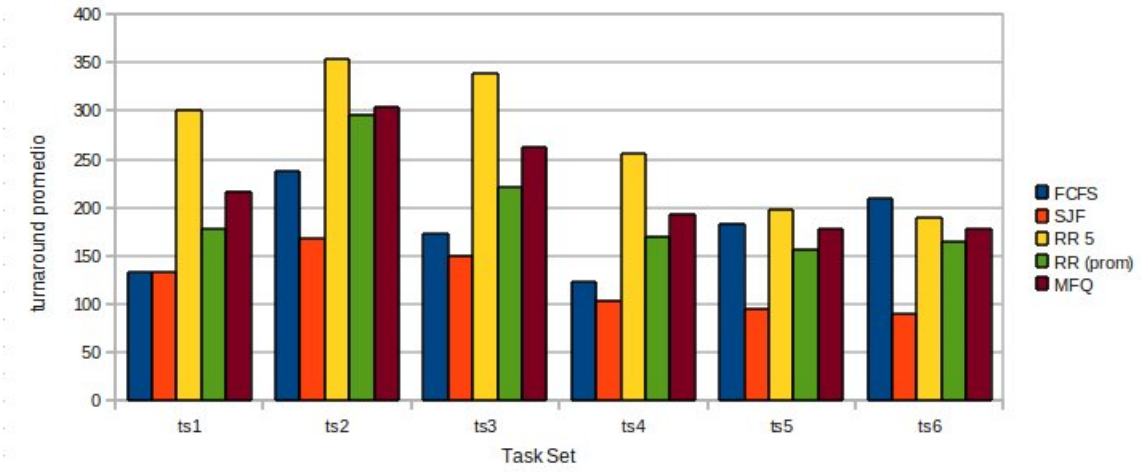
	5	10	40	80	130	Promedio
ts1	255.75	193.5	128.5	94.5	94.5	132.25
ts2	308.25	255	237.5	199.5	199.5	251.25
ts3	293.375	230.5	170.25	134.5	134.5	175.25
ts4	210.125	150.5	120.25	84.5	84.5	125.25
ts5	159.625	131.125	118.125	130.75	155.75	117.325
ts6	157.3	132.8	138.7	156.9	186.9	130.1

Como se puede ver el en gráfico, para los primeros TaskSet el quantum de tamaño promedio es de los que mantienen el menor Waiting Time.

6. Ejercicio 6

En el caso de Multilevel Feedback Queue, es un algoritmo más complejo que los anteriores y se podría ubicar entre los algoritmos de SJF y RR.

Para comparar los algoritmos de scheduling, los comparamos analizando el turnaround promedio.



En el gráfico podemos observar que el algoritmo MFQ se comporta muy parecido al RR, esto se debe al preemption que define la especificación de la cátedra.

Lo que queremos decir es que, para optimizar el turnaround los procesos más cortos están en las colas con más prioridad, mientras que los más largos van disminuyendo su prioridad a través de las colas.
A su vez, para no comportarse como un SJF se introducen las colas y el preemption y ahi es donde se acerca a la idea de un RR. Con ésta optimiza el tiempo de respuesta en el sistema, beneficiando los procesos interactivos y reduciendo la probabilidad de que un proceso caiga en inanición en comparación con SJF, sin embargo, no elimina por completo la inanición.

Supongamos el caso en que existe un proceso que requiere un largo tiempo de ejecución y junto con el, muchos procesos con un tiempo de ejecución suficiente para ser atendidos en la cola de alta prioridad. El proceso más largo se ejecuta al principio, cuando permanece en la cola de alta prioridad y va descendiendo por las diferentes colas. Al mismo tiempo nuevos procesos ingresan en la cola de alta prioridad, haciendo que el proceso largo no sea atendido y caiga en inanición mientras los procesos en la cola de alta prioridad entran y se ejecutan.

La mayoría de los procesos interactivos, son procesos largos pero requieren ser atendidos de inmediato, por lo tanto para no caer en inanición estos procesos son 'premiados' cuando hacen E/S (en el algoritmo de MFQ con IO).