Pontificia Universidad Católica de Chile
Departamento de Computación
IIC-2333. Sección 1
Sistemas Operativos y Redes

Informe Tarea 1

Profesor: Cristian Ruz

Felipe Domínguez Claro.
1562188J

abril de 2019

Pregunta 1

Una posible desventaja es que los procesos con baja prioridad van a sufrir de inanición. En otras palabras, siempre se ejecutarán los procesos con mayor prioridad, dejando a los con menos sin posibilidad de acceder a la CPU.

En el output de la tarea podemos ver claramente esta consecuencia. El proceso RICHI tuvo un *waiting_time* de 21, lo que representa un 50% más que el promedio de los otros dos procesos.

Una medida para solucionar este problema, sería usar un "contra_quantum (CQ)". La idea sería que luego de CQ unidades de tiempo, el proceso se ejecute sin importar la prioridad de este. De esta manera no quedará siempre en el último lugar de la cola de procesos.

Otra medida para mitigar este problema, es la conocida como "Aging", que consiste en aumentar dinámicamente la prioridad de los procesos según el tiempo que llevan ejecutándose.

Otra posible desventaja de usar la versión preemptive, es que se perderán ciclos de la CPU en realizar los cambios de contexto luego de cada interrupción. Ya que luego de cada una se debe suspender el proceso actual, y traer otro de la cola para ejecutar. Esta desventaja no fue medida por la tarea, por lo tanto no tengo información empírica que respalde mi teoría.

Pregunta 2

Para poder usar mi algoritmo con un procesador *multicore*, es necesario tener referencias por separado de cada uno de los núcleos que tengamos (usaré un 4-core para la explicación). De esta manera tendremos la misma cola de procesos, pero los nodos pueden salir a 4 CPUs distintas. Para manejar esto, necesitamos tener para cada núcleo un estado, y en cada iteración verificar si alguno de los 4 esta libre para procesar. El resto de la implementación sería básicamente la misma, ya que el flujo de trabajo es exactamente el mismo.

Bajo este contexto, nuestra cola avanzará más rápido cuando existen muchos procesos en READY, ya que ahora hay más de una unidad procesadora. Esto, en general, ayudará a aumentar el *turnaround, response* y *waiting time.* Esta mejora de las 3 estadísticas es clara, ya que como ahora la cola de procesos en READY avanza mas rápido, entonces disminuye el *response* y *waiting time,* y por lo tanto también disminuirá el *turnaround time.* Mencioné que esto ocurrirá generalmente, ya que depende de la distribución de los procesos en el tiempo y de las proporciones de I/O que estos tengan.

Pregunta 3

Si queremos implementar una versión *interactive*, entonces el algoritmo de *scheduling* tendrá menos relevancia en el rendimiento final, ya que como los procesos son I/O bound, sus ráfagas de CPU suelen ser cortas, mientras que las de I/O son más largas. Para sacar conclusiones sobre cual es el mejor algoritmo realicé dos test que usan procesos de este tipo.

Test 1: se usaron los siguientes procesos:

```
GERMY 10 6 4 1 6 1 6 1 6 1
RICHI 1 0 3 1 6 1 6 1
CRISTIAN 5 5 3 1 6 1 6 1
```

Tipo	Turnos CPU	Interrupciones	Turnaround	Response	Waiting time
Preemptive 3	3.33	0	17.33	0	14
Preemptive 5	3.33	0	17.33	0	14
No preemptive	3.33	0	17.33	0	14

Tabla 1: Resumen de estadísticas promedio con procesos I/O Bound

En este caso podemos ver que como las ráfagas de CPU sólo tienen una duración de una unidad de tiempo, entonces las 3 implementaciones tienen el mismo despeño.

Test 2: se usaron los siguientes procesos:

```
GERMY 10 6 4 4 15 4 15 4 15 4
RICHI 1 0 3 5 16 5 16 5
CRISTIAN 5 5 3 4 14 4 14 4
```

Tipo	Turnos CPU	Interrupciones	Turnaround	Response	Waiting time
Preemptive 3	6.66	3.33	53.66	0.66	39.33
Preemptive 5	3.33	1	53.66	1	39.33
No preemptive	3.33	0	53.66	1	39.33

Tabla 2: Resumen de estadísticas promedio con procesos I/O Bound

En el test 2, podemos ver que los rendimientos también son casi idénticos entre sí. Existiendo sólo pequeñas diferencias con respecto a usar *quantum* más pequeño que la duración de las ráfagas. Aún así, el cambio es despreciable, por lo que no se tomará como influyente para tomar conclusiones.

Después de ver las distintas pruebas, podemos ver que cuando son sistemas interactivos con procesos marcados fuertemente por I/O bound, el algoritmo que usemos no tendrá mucha importancia a la hora de analizar el rendimiento. Esto se debe ya que los procesos están muy poco rato ejecutándose en la CPU, por lo tanto existen menos cuellos de botella a la hora de requerir su uso.

Bonus

Si queremos otorgar rapidez al iniciar programas, podríamos implementar un sistema en que en las primeras ejecuciones, los procesos tengan una alta prioridad, de esta manera poder comenzar su ejecución. Esto ayudará a reducir el *waiting time.* En la tabla 3 podemos ver que lo planteado anteriormente efectivamente se cumple. El tipo: "boost" reduce el *waiting time* a 12 unidades de tiempo. Junto con esto, el *turnarround time* también disminuyó a 24.

Para implementar este tipo de *scheduling*, lo que se hizo fue setear la prioridad del proceso en un valor muy alto (sea 1000), y luego de su primera ejecución, la prioridad vuelve a su valor inicial. Para poder probar esta implementación, hay que correr el programa en modo "boost".

Tipo	Turnos CPU	Interrupciones	Turnaround time	Response time	Waiting time
Preemptive 3	5	3	26.66	1.33	14.66
Preemptive 5	3.33	1.33	25.66	0.66	13.66
No preemptive	3.33	0	28.33	4	29
Boost	5	3.66	24	3.66	12

Tabla 3: Resumen de estadísticas promedio con procesos de test