PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2333 — Sistemas Operativos y Redes — 1/2019

Tarea 1

Viernes 22-Marzo-2019

Fecha de Entrega: Lunes 1-Abril-2019, 14:00
Ayudantía: Viernes 22-Marzo-2018, 10:00
Composición: Tarea individual

Descripción

El *scheduling* consiste en escoger el siguiente proceso a ejecutar en la CPU. Esto no es trivial para el sistema operativo pues no siempre se tiene suficiente información sobre los procesos para tomar una decisión justa. Sin embargo, la elección puede influir considerablemente en el rendimiento del sistema.

El objetivo de esta tarea es **implementar** el algoritmo de *priority scheduling*. Este algoritmo se encarga de que se le asigne la CPU al proceso de **mayor prioridad** que esté en espera. Este deberá incluir dos versiones: *non-preemptive* y *preemptive*. En la versión *non-preemptive* el proceso asignado a la CPU debe ejecutar en su totalidad, mientras que en el caso *preemptive* se asigna un intervalo de tiempo de ejecución fijo, que llamaremos *quantum*, para que el proceso asignado sea ejecutado y posteriormente interrumpido. Deberá implementar el algoritmo en sus dos versiones mediante un programa en C y **simular** su comportamiento de acuerdo a un conjunto de procesos descritos en un archivo.

Modelamiento de los procesos

Debe existir un `struct Process`, que modelará un proceso. Un proceso tiene un `PID`, un nombre de hasta 256 caracteres, una prioridad y un estado, que puede ser `RUNNING`, `READY`, `WAITING` o `FINISHED`. Note que el `PID` es un entero mayor o igual a 0 que se asigna de forma creciente según el orden de llegada de los procesos.

La simulación recibirá un archivo de entrada que describirá los procesos a ejecutar y el comportamiento de cada proceso en cuanto a uso de CPU. Una vez terminado el tiempo de ejecución de un proceso, éste terminará tal como si hubiese ejecutado `exit()`, y pasará a estado `FINISHED`. Durante su ciclo de vida el proceso alterna entre un tiempo A_i en que ejecuta código de usuario (*CPU burst*), y un tiempo B_i en que permanece bloqueado (*I/O burst*).

Modelamiento de la cola de procesos

Debe construir un `struct Queue` para modelar una cola de procesos. Esta estructura de datos deberá ser construida por usted y deberá ser eficiente en términos de tiempo[1]. La cola debe estar preparada para recibir **cualquier** cantidad de procesos que puedan estar en estado `READY` al mismo tiempo.

Scheduler

El *scheduler* decide qué proceso de la cola, en estado `READY`, entra en estado `RUNNING` de acuerdo a la prioridad mayor. En la versión *non-preemptive*, el *scheduler* se activa cada vez que el proceso `RUNNING` decide bloquearse (entra a un periodo B_i), mientras que en la versión *preemptive*, además, se activa cada vez que se consume el *quantum*

[1] Para efectos de esta evaluación, se considerarán eficientes simulaciones que tarden, a lo más, 30 segundos en ejecutar para cada archivo de entrada. Para ver el tiempo de ejecución de un programa en C puede usar `time`.

asignado. El *scheduler* debe sacar al proceso actual de la CPU, reemplazarlo por el siguiente y determinar la acción correspondiente para el proceso saliente, las que se pueden dividir de la siguiente forma:

- **Versiones *non-preemptive* y *preemptive***
  - Pasar a estado `FINISHED` al finalizar su secuencia.
  - Pasar a estado `WAITING` al bloquearse.
- **Versión *preemptive***
  - Pasar a estado `READY` en la cola al ser interrumpido.

El *scheduler* debe asegurar que el estado de cada proceso cambie de manera consistente.

Por otra parte, en la implementación de la versión *preemptive* del algoritmo, el *quantum* se establece en q unidades de tiempo. Esta constante debe ser entregada por línea de comandos y su valor por defecto es $q = 3$.

Ejecución de la Simulación

El simulador será ejecutado por línea de comandos con la siguiente instrucción:

```
./scheduler <file> <output> <version> [<quantum>]
```

- `<file>` corresponderá a un nombre de archivo que deberá leer como *input*.
- `<output>` corresponderá a la ruta de un archivo `CSV` con las estadísticas de la simulación, que debe ser escrito por su programa.
- `<version>` corresponderá a la versión del algoritmo: `np` (*non-preemptive*) o `p` (*preemptive*).
- `[<quantum>]` es un parámetro que corresponderá al valor de q. Si no es ingresado, debe usarse por defecto $q = 3$.

Por ejemplo, algunas ejecuciones podrían ser las siguientes:

```
./scheduler input.txt output.csv p 5
./scheduler input.txt output.csv np
./scheduler input.txt output.csv np 4
```

Archivo de entrada (*input*)

Los datos de la simulación se entregan como entrada en un archivo de texto con múltiples líneas, donde cada una tiene el siguiente formato:

```
NOMBRE_PROCESO P TIEMPO_INICIO N A_1 B_1 A_2 B_2 ... A_N-1 B_N-1 A_N
```

Donde:

- N es la cantidad de ráfagas de CPU, con $N \geq 1$.
- La secuencia $A_1\ B_1\ \ldots\ A_{N-1}\ B_{N-1}\ A_N$ describe el comportamiento de cada proceso. Cada A_i es la cantidad de unidades de tiempo que dura un ráfaga de CPU (*CPU burst*), es decir, la cantidad de tiempo que el proceso solicitará estar en estado `RUNNING`. Cada B_i es la cantidad de unidades de tiempo que dura una *I/O burst*, es decir, la cantidad de tiempo que el proceso se mantendrá en estado `WAITING`. En la versión *preemptive*, si la *ráfaga* A_i es mayor al *quantum* asignado, el proceso es interrumpido, liberando la CPU y volviendo a la cola con estado `READY`. Notar que en este caso, al haber alcanzado a ejecutar q unidades de tiempo, la próxima vez que pase a estado `RUNNING` al proceso le quedará un total de $A_i - q$ unidades de tiempo restantes de ejecución en esa ráfaga.

- `NOMBRE_PROCESO` debe ser respetado para la impresión de estadísticas.
- `P` es la prioridad del proceso. Esta es **constante** durante toda la simulación.
- `TIEMPO_INICIO` es el tiempo de llegada a la cola. Considere que `TIEMPO_INICIO` ≥ 0.
- Los tiempos son entregados sin unidades, por lo que pueden ser trabajados como enteros adimensionales.

Puede utilizar los siguientes supuestos:

- Cada número es un entero no negativo y que no sobrepasa el valor máximo de un `int` de 32 bits.
- El nombre del proceso no contendrá espacios, ni será más largo que 255 caracteres.
- No habrá dos o más procesos cuyo tiempo de inicio sea el mismo.
- Habrá al menos un proceso descrito en el archivo.
- Todas las secuencias comienzan y terminan con un *burst* A_i. Es decir, un proceso nunca estará en estado `WAITING` al ingresar a la cola por primera vez ni antes de terminar su ejecución.

El siguiente ejemplo ilustra cómo se vería un posible archivo de entrada, siendo `PROCESS1` el proceso de mayor prioridad:

```
PROCESS1 5 21 5 10 4 30 3 40 2 50 1 10
PROCESS2 1 13 3 14 3 6 3 12
```

Importante: Es posible que en algún momento de la simulación no haya procesos en estado `RUNNING` o `READY`. Los sistemas operativos manejan esto creando un proceso especial de nombre `idle` que no hace nada hasta que llega alguien a la cola `READY`. Pueden considerarlo en su simulación, pero no debe influir en los valores de la estadística.

Orden de ejecución

Podrá notar que el orden exacto en el que realice los cambios no es fundamental, no obstante, **debe tener cuidado** con que su programa realice más de una modificación dentro de una misma iteración. Por ejemplo, un proceso que termina la ejecución de una de sus ráfagas A_i y pasa a estado `WAITING`, **no puede aumentar en una unidad su tiempo de espera dentro de la misma iteración**. Es importante que tenga esto en consideración para todos los eventos que puedan ocurrir.

Salida (*Output*)

Una vez que el programa haya terminado la simulación, su programa deberá escribir un archivo `CSV` con los siguientes datos por proceso:

- El nombre del proceso.
- El número de veces que el proceso fue elegido para usar la CPU.
- El número de veces que fue interrumpido. Este equivale al número de veces que el *scheduler* sacó al proceso por el uso completo de su *quantum* (en la versión *non-preemptive*, siempre será igual a 0).
- El *turnaround time*.
- El *response time*.
- El *waiting time*. Este equivale a la suma del tiempo en el que el proceso está en estado `READY` y `WAITING`.

Es importante que siga **rigurosamente** el siguiente formato:

```
nombre_proceso_i,turnos_CPU_i,interrupciones_i,turnaround_time_i,response_time_i,waiting_time_i
nombre_proceso_j,turnos_CPU_j,interrupciones_j,turnaround_time_j,response_time_j,waiting_time_j
...
```

Podrá notar que, básicamente, se solicita un `CSV` donde las columnas son los datos pedidos por proceso. Las líneas **deben** estar en el mismo orden del archivo de entrada.

Casos especiales

Dada la naturaleza de este problema, existirán algunos casos límite (algunos solo en la versión *preemptive*) que podrían generar resultados distintos según la implementación. Para evitar este problema, **deberán** considerar que:

- Si al momento de escoger un proceso existen dos con la misma prioridad (independiente de su estado previo), entonces será seleccionado el que posea un **mayor** `PID`.
- Si el *quantum* de un proceso se agota al mismo tiempo que su ráfaga o *burst*, se considerará como un **bloqueo** (es decir, se suma a su cantidad de bloqueos), iniciando inmediatamente la espera B_i en estado `WAITING`. Si el tiempo B_i no existe (es decir, se ejecutó la última ráfaga), se sigue considerando como bloqueo, solo que posteriormente el proceso pasa a estado `FINISHED`, terminando su ejecución.
- Un proceso **puede** llegar en tiempo $t = 0$. Su programa no se puede caer si llega un archivo *input* con ese valor.
- En el caso de un único proceso en el sistema y ejecutando con *scheduling preemptive*, al terminar su *quantum* sigue en ejecución, pero aumenta en uno su cantidad de bloqueos.

Reporte

Además de su programa, deberá incluir un reporte en formato `PDF` con su nombre y número de alumno. En este, debe responder las siguientes preguntas:

1. ¿Puede identificar una desventaja de la versión *preemptive* del algoritmo de *scheduling* propuesto? Si es así, ¿cómo la podría mejorar?
2. ¿Cómo se podría extender su implementación para funcionar en un sistema *multicore* (> 1 núcleos)? ¿Podría disminuir alguno de los tiempos de los procesos (*turnaround*, *response*, *waiting*)? Describa qué decisiones de diseño debería tomar. No es necesario que lo implemente.
3. ¿Cuál versión del algoritmo se comporta mejor para un sistema interactivo? Proponga un caso de prueba con un conjunto de procesos *I/O bound* y compárelo en ambas versiones del algoritmo. Justifique el resultado.

No importa en qué plataforma escriba su reporte (LaTeX, Word, Bloc de notas, Markdown, etc.) siempre y cuando se respete el formato de entrega solicitado (`PDF`).

Formalidades

A cada alumno se le asignó un nombre de usuario y una contraseña para el servidor del curso[2]. Para entregar su tarea usted deberá crear una carpeta llamada `T1` en el directorio principal de su carpeta personal y subir su tarea a esa carpeta. En su carpeta `T1` **solo debe incluir el código fuente** necesario para compilar su tarea, además del reporte y un `Makefile`. Se revisará el contenido de dicha carpeta el día Lunes 1-Abril-2019, 14:00.

[2] `iic2333.ing.puc.cl`

- **NO debe incluir archivos binarios**. En caso contrario, tendrá un descuento de 0.5 puntos en su nota final.
- Su tarea deberá compilar utilizando el comando `make` en la carpeta `T1`, y generar un ejecutable llamado `scheduler` en esta misma. Si su programa **no tiene** un `Makefile`, tendrá un descuento de 1 punto en su nota final.
- Es muy importante que su tarea corra dentro del servidor del curso. Si esta **no compila** o **no funciona** (*segmentation fault*), obtendrán la nota mínima, teniendo como base 1 punto menos en el caso de que soliciten recorrección.

El no respeto de las formalidades o un código extremadamente desordenado podría originar descuentos adicionales. Se recomienda modularizar, utilizar funciones y ocupar nombres de variables explicativos. En el caso de no entregar en la carpeta especificada, la tarea **no** se corregirá.

Evaluación

Simulación: La parte programada de esta tarea sigue la siguiente distribución de puntaje:

- **5.5 pts.** Simulación correcta.
  - **0.5 pts.** Estructura y representación de procesos.
  - **0.5 pts.** Estructura y manejo de colas.
  - **0.5 pts.** Línea de comandos.
  - **2.0 pts.** *Priority scheduler*, versión *non-preemptive*[3].
  - **2.0 pts.** *Priority scheduler*, versión *preemptive*[4].
- **0.5 pts.** Manejo de memoria. Se obtiene este puntaje si `valgrind` reporta en su código 0 *leaks* y 0 errores de memoria en **todo caso de uso**[5].

Reporte: Cada pregunta del reporte sigue la siguiente distribución de puntaje:

- **2 pts.** La respuesta es correcta.
- **1 pts.** La respuesta se acerca a lo solicitado, pero posee aspectos incorrectos o no bien detallados.
- **0 pts.** La respuesta está incorrecta o se deja en blanco.

La nota final de la evaluación es, entonces:

$$N_{T_1} = 0{,}7 \cdot N_S + 0{,}3 \cdot N_R$$

Donde N_S es la nota obtenida en la simulación y N_R la nota obtenida en el reporte.

Política de atraso

Se puede hacer entrega de la tarea con un máximo de 4 días de atraso. La fórmula a seguir es la siguiente:

$$N_{T_1}^{\text{Atraso}} = N_{T_1} - 0{,}75 \cdot d$$

Siendo d la cantidad de días de atraso.

Cabe destacar que se considerarán **ambas** partes de la tarea para definir el atraso. Es decir, si se entrega el reporte a tiempo, pero no el código, sigue aplicando el descuento según la fórmula antes descrita.

[3] Este puntaje será evaluado según la cantidad de *tests* a utilizar.
[4] Ver nota al pie de página 3.
[5] Es decir, debe reportar 0 *leaks* y 0 errores para todo *test*.

Bonus (+0.5 pts): tercera versión

Se aplicará este *bonus* a la nota final si propone una tercera versión que sea superior a las anteriores, es decir, se espera que disminuya, al menos, uno de los siguientes valores promedio: *turnaround time*, *response time* y *waiting time*. Se espera que, naturalmente, se utilicen como mínimo **los elementos de las versiones anteriores** (prioridad, *quantum*). Para corroborar sus resultados, debe hacer uso de los mismos *tests* de prueba que serán compartidos y mostrar la mejora a partir de la comparación de los resultados de los tres algoritmos de *scheduling*. Note que su propuesta podría añadir **nuevos casos borde**, los que se espera que sean resueltos según estime conveniente. El *bonus* a su nota se aplica si, y solo si la nota final correspondiente es $\geq 3{,}95$.

Preguntas

Cualquier duda preguntar a través del foro oficial.