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

IIC2333 — Sistemas Operativos y Redes — 2/2018

Tarea 1

Jueves 23-Agosto-2018

Fecha de Entrega: Viernes 31-Agosto-2018, 23:59
Ayudantía: Viernes 24-Agosto-2018, 08:30
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 *scheduling Round Robin* (RR). Este algoritmo asigna un intervalo de tamaño fijo de tiempo de ejecución, que llamaremos *quantum* para cada proceso dentro de una única cola. El *quantum* se asigna por turnos a cada proceso, permitiendo que todos los procesos puedan ser atendidos. Deberá implementar el algoritmo 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, y un estado, que puede ser `RUNNING`, `READY`, `WAITING` o `FINISHED`.

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 al orden FIFO de la cola. El *scheduler* se activa cada vez que se consume un *quantum* o bien cuando el proceso `RUNNING` decide bloquearse (entra a un periodo B_i). El *scheduler* debe sacar al proceso actual de la CPU, reemplazarlo por el siguiente, y determinar la acción correspondiente para el proceso saliente (regresar a la cola, pasar a `WAITING`, o terminar).

[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`.

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

En la implementación del algoritmo RR, el *quantum* se establece en q unidades de tiempo. Esta constante el *quantum* 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> [<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.
- `[<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 5
./scheduler input.txt output.csv
```

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 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 proceos. 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 ed tiempo que el proceso se mantendrá en estado `WAITING`. Si la *ráfaga* A_i es mayor al *quantum* asignado, el proceso es interrumpido, liberando la CPU y volviendo a la última posición de la cola, y 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.
- `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:

```
PROCESS1 21 5 10 4 30 3 40 2 50 1 10
PROCESS2 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`.

Salida (*Output*)

Usted deberá dar información de lo que está ocurriendo. La simulación deberá imprimir los siguientes eventos:

- Cuando un proceso es creado, cambia de estado (y a qué estado), o termina.
- Cuando el *scheduler* elige un proceso para ejecutar en la CPU (cuál). Aquí se puede incluir el proceso `idle`.

El formato de impresión de cada evento es libre, pero se espera que sea conciso y explicativo, idealmente de una línea. Por ejemplo:

```
...
[t = 5] El proceso GERMY ha pasado a estado RUNNING.
[t = 6] El proceso RICHI ha sido creado.
[t = 8] El proceso GERMY ha pasado a estado READY.
[t = 8] El proceso CRUZ ha pasado a estado RUNNING.
[t = 9] El proceso CRUZ ha pasado a estado FINISHED.
...
```

Una vez que el programa haya terminado, 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*.
- El *turnaround time*.
- El *response time*.
- El *waiting time*. Este equivale a la suma del tiempo que 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. En estas estadísticas **no se debe incluir al proceso** `idle`**.**

Casos especiales

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

- Si el *quantum* de un proceso se agota al mismo tiempo que su ráfaga o *burst*, se considerará como un **bloqueo**, iniciando inmediatamente el tiempo B_i. Es decir, su estado debe pasar directamente a `WAITING` y no de forma intermedia a `READY`. Si el tiempo B_i no existe (es decir, se ejecutó la última ráfaga), se sigue considerando como bloqueo, solo que su estado debe pasar directamente a `FINISHED`, en vez de `WAITING`.
- 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, al terminar su *quantum*, éste sale de la CPU, es regresado a la cola `READY` e inmediatamente vuelve a ser elegido para su ejecución. Esto se considera como una ocasión más en que el proceso fue interrumpido y elegido.
- Si un proceso nuevo llega a la cola al mismo tiempo que otro pasa de estado `WAITING` a `READY`, arbitrariamente ingresa primero el proceso nuevo y luego el que haya estado en espera.
- Si un proceso pasa de estado `RUNNING` a `READY` al mismo tiempo que otro pasa de estado `WAITING` a `READY`, arbitrariamente este último es el primero en ingresar a la cola.
- Si dos procesos pasan de estado `WAITING` a `READY` al mismo tiempo, vuelven a ingresar a la cola **en el mismo orden** que entraron a `WAITING`.

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. ¿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.
2. A continuación se muestra una lista de *schedulers*. De estos debe elegir uno de acuerdo al último dígito de su número de alumno.
   - 0, 3, 6, 9: *Lottery scheduling*
   - 1, 4, 7, J: *Completely Fair Scheduling* (CFS)
   - 2, 5, 8: $O(1)$ *scheduling*

   Investigue el funcionamiento del *scheduler* asignado. Describa brevemente su funcionamiento, para qué sistema operativo ha sido usado, qué características de diseño hace que el sistema sea interactivo y cómo se comportaría ante una carga intensiva en uso de CPU. Debe respetar el *scheduler* asignado, pues en otro caso esta pregunta no será corregida.

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 Viernes 31-Agosto-2018, 23:59.

- **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 esa misma carpeta. 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 ésta **no compila** o **no funciona** (*segmentation fault*), obtendrán la nota mínima, teniendo como base 1 punto menos en el caso 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:

- **5.0 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.
  - **3.5 pts.** *Scheduler* RR.
- **1.0 pts.** Debe entregar mensajes claros y precisos y que permitan entender lo que está pasando, idealmente que no ocupen más de una línea. Debe respetar el formato de las estadísticas.

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.

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

Política de atraso

Se puede hacer entrega de la tarea con un máximo de 2 días de atraso. La fórmula a seguir es la siguiente:

$$N_{T_1}^{\text{Atraso}} = N_{T_1} - 1{,}5 \cdot d$$

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.

Bonus (+0.5 pts): manejo de memoria perfecto

Se aplicará este *bonus* a la nota final si `valgrind` reporta en su código 0 *leaks* y 0 errores de memoria, considerando que el programa funcione correctamente. El *bonus* a su nota se aplica solo si la nota final correspondiente es $\geq 3{,}95$.

Preguntas

Cualquier duda preguntar a través del foro.