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

IIC2333 — Sistemas Operativos y Redes — 1/2020

Tarea 2

Lunes 13-Abril-2020

Fecha de Entrega: Lunes 27-Abril-2020, 14:00
Ayudantía: Viernes 17-Abril-2019, 08:30
Composición: Tarea individual

Objetivos

- Modelar la ejecución de un algoritmo de *scheduling* de procesos.
- Analizar las diferencias de rendimiento entre distintas variantes de un mismo algoritmo de *scheduling*.

OSRScheduler

Luego de crear tu excelente simulación de *CRVID*, el profesor llega bastante agitado a la clase *online*, argumentando que, debido a la gran cantidad de procesos creados por las simulaciones del virus, su CPU no da abasto y no le permite jugar **Oldschool Ruzscape**, lo cual implica una gran cantidad de horas de juego y experiencia en **Ruzcrafting**. Luego de haber creado el programa que causó esto, deberás crear un nuevo *Scheduler*, para que la CPU del profesor le dé más tiempo a su juego favorito.

Descripción

El *scheduling* consiste en escoger el siguiente proceso a ejecutar en la CPU. Esto no es trivial para el sistema operativo, puesto que no siempre se tiene suficiente información sobre los procesos para tomar una decisión óptima, lo que es importante dado que la elección puede influir significativamente en el rendimiento del sistema.

El objetivo de esta tarea es **implementar** el algoritmo de *Shortest Remaining Time First (SRTF)*. Este algoritmo se encarga de asignarle la CPU al proceso cuyo tiempo total de ejecución[1] restante sea el menor.

Para esta tarea, entonces, deberán implementar dos versiones de este algoritmo:

- ***preemptive***: La CPU interrumpirá el proceso que se encuentre ejecutando en la CPU luego de un tiempo específico llamado `quantum`.
- ***non-preemptive***: Los procesos correrán ininterrumpidos hasta que hagan entrega de la CPU voluntariamente.

Modelamiento de los procesos

Debe existir un `struct Process`, que modelará un proceso. Un proceso tiene un `PID`, un nombre de hasta 32 caracteres, una prioridad y un estado, que puede ser `RUNNING`, `READY`, `WAITING` o `FINISHED`. 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

[1] Sólo considerando tiempo en CPU

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*, ráfaga de ejecución), y un tiempo B_i en que permanece bloqueado (I/O *burst*, tiempo de espera).

Modelamiento de la cola de procesos

Debe construir un `struct Queue` para modelar una cola de procesos. Esta estructura deberá ser construida por usted y deberá contener, como mínimo, el **arreglo** de estructuras `Process` de la simulación. La cola debe estar preparada para recibir **cualquier** cantidad de procesos que puedan estar en estado `READY` al mismo tiempo.

Si bien se modela una cola de procesos, **no es necesario** que creen exactamente una estructura de tipo cola. Por ejemplo, no es necesario que la compongan a partir de una lista enlazada con estructuras de clase "nodo", ni que posea métodos como *enqueue* o *dequeue*, basta con que se defina la `struct Queue` y que contenga, como mínimo, el **arreglo** de todos los procesos a simular.

Scheduler

El *scheduler* decide qué proceso de la cola, en estado `READY`, entra en estado `RUNNING` de acuerdo a la cantidad de tiempo que le quede al proceso. 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 pueden ser:

- Pasar a estado `FINISHED` al finalizar su última ráfaga de ejecución.
- Pasar a estado `WAITING` al bloquearse, *i.e.* al terminar la ráfaga de ejecución actual y tener un tiempo de espera posterior.
- Pasar a estado `READY` en la cola al ser interrumpido, lo que sucede cuando llega a la cola un proceso con menor tiempo de ejecución.

El *scheduler* debe asegurar que el estado de cada proceso cambie de manera consistente. Por otra parte, su ejecución deberá ser eficiente en términos de tiempo[2].

Ejecución de la Simulación

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

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

[2]Para efectos de esta evaluación, se considerarán eficientes simulaciones que tarden, a lo más, 10 segundos en ejecutar para cada archivo de entrada. Para ver el tiempo de ejecución de un programa en C puede usar `time`.

Donde:

- `<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>` argumento opcional, el cual denota al `quantum`. En caso de no ser ingresado, será por defecto 5.

Por ejemplo, algunas ejecuciones podrían ser las siguientes:

```
./osrs input.txt output.csv np
./osrs input.txt output.csv p
./osrs input.txt output.csv p 4
```

Archivo de entrada (*input*)

Los datos de la simulación se entregan como entrada en un archivo de texto, donde la primera línea indica la cantidad K de procesos a simular y las siguientes K líneas siguen 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 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`.
- `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 31 caracteres.
- 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:

```
3
PROCESS1_RAUL 21 5 10 4 30 3 40 2 50 1 10
PROCESS2_RICHI 13 3 14 3 6 3 12
PROCESS3_ZEZIMA 5 4 14 3 6 3 12 3 18
```

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, es decir, no se debe incluir en el archivo de salida.

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 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 de la CPU.
- 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.

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 al momento de escoger un proceso existen dos con el mismo tiempo restante, entonces será seleccionado el que posea un **menor** `turno inmediato`[3]. No obstante lo anterior, esto **no es válido para interrumpir**. Pueden asumir que no habrá empate en este caso.
- Un proceso **puede** llegar en tiempo $t = 0$. Su programa no se puede caer si llega un archivo *input* con ese valor.

Formalidades

A cada alumno se le asignó un nombre de usuario y una contraseña para el servidor del curso[4]. Para entregar su tarea usted deberá crear una carpeta llamada `T2` en el directorio principal de su carpeta personal y subir su tarea a esa carpeta. En su carpeta `T2` **solo debe incluir el código fuente** necesario para compilar su tarea y un `Makefile`. Se revisará el contenido de dicha carpeta el día Lunes 27-Abril-2020, 14:00.

- **NO debe incluir archivos binarios**. En caso contrario, tendrá un descuento de 0.5 puntos en su nota final.

[3] Notar que esto significa que estamos priorizando a los procesos que se demorarán menos en terminar su CPU burst actual.
[4] `iic2333.ing.puc.cl`

- Su tarea deberá compilar utilizando el comando `make` en la carpeta `T2`, y generar un ejecutable llamado `scheduler` en esta misma. Si su programa **no tiene** un `Makefile`, tendrá un descuento de 0.5 puntos 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 0.5 puntos 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

El puntaje de su programa seguirá la siguiente distribución de puntaje:

- **5.5 pts.** Simulación correcta.
  - **0.5 pts.** Estructura y representación de procesos.
  - **1.0 pts.** Estructura y manejo de colas.
  - **2.0 pts.** *Shortest Remaining Time First*, versión *non-preemptive*[5].
  - **2.0 pts.** *Shortest Remaining Time First*, versión *preemptive*[6].
- **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**[7].

La nota final de la evaluación es, entonces:

$$N_{T_2} = N_S + b$$

Donde N_S es la nota obtenida en la simulación y b el bonus.

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:

$$N_{T_2}^{\text{Atraso}} = \text{mín}(N_{T_2}, 7{,}0 - 0{,}75 \cdot d + b - D)$$

Siendo d la cantidad de días de atraso y D los descuentos obtenidos. Notar que esto equivale a un descuento *soft*, es decir, cambia la nota máxima alcanzable y no se realiza un descuento directo sobre la nota obtenida.

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*. Es libre de incluir los mecanismos que le parezcan pertinentes (por ejemplo, *aging*), siempre que **no modifique el formato de los archivos de entrada**. Además, debe ser ejecutable a partir de la versión `b`, por ejemplo:

```
./osrs input.txt output.csv bonus
```

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

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*, haciendo un análisis conciso en un archivo de nombre `bonus.md` dentro de la carpeta de su tarea. Note que su propuesta podría añadir **nuevos casos borde**, los que se espera que sean resueltos según estime conveniente. Finalmente, 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.