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

IIC2333 — Sistemas Operativos y Redes — 2/2017

Tarea 1

Viernes 11-Agosto-2017

Fecha de Entrega: Viernes 25-Agosto-2017 a las 23:59
Composición: grupos de n personas, donde $n \leq 2$

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 **simular** algunos *scheduler* y determinar estadísticas de rendimiento en base a una secuencia de procesos que desean ejecutarse. La simulación debe estar construida en C de acuerdo a los requerimientos que se detallan a continuación.

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 entre 1 y 64 (inclusive), y un estado, que puede ser `RUNNING`, `READY`, o `WAITING`.

Como esto es una simulación, no podemos pedirle que ejecute cualquier programa. Un proceso tendrá un tiempo de ejecución total, después del cual deberá destruirse. Durante su ciclo de vida el proceso alterna entre un tiempo A_i en que ejecuta código de usuario, y un tiempo B_i en que permanece bloqueado. El comportamiento de cada proceso estará especificado en un archivo de entrada.

Modelamiento de la cola de procesos

Debe existir un `struct Queue`, que modelará una cola de procesos. Esta estructura de datos deberá ser construida por usted, y deberá ser eficiente. Se recomienda que utilice la misma cola para los tres *scheduler* que tendrá que implementar, y que sólo cambie el modo de elegir el proceso de acuerdo a cada uno. Considere que usted **no conoce** cuántos procesos pueden estar en estado `READY` a la vez.

Scheduler

El *scheduler* decide qué proceso de la cola `READY` entra en estado `RUNNING`.

Cada vez que el *scheduler* ejecute, debe determinar si es necesario que el proceso activo (`RUNNING`) entregue la CPU y, de ser así, sacar un proceso de la cola `READY` para pasarlo a estado `RUNNING`. El *scheduler* deberá permitir a cada proceso bloquearse y desbloquearse cuando éste lo requiera. El *scheduler* debe asegurarse que el estado de cada proceso cambie de acuerdo a lo que está ocurriendo en el sistema.

Para esta tarea deberá implementará tres algoritmos de *scheduling*:

1. **Non-Preemptive FCFS**. Este *scheduler* elige cada proceso en el orden en que entraron a la cola, sin importar la prioridad asociada a cada uno.

2. **Preemptive Round Robin con prioridades**. Este *scheduler* asigna un *quantum* Q_k a un proceso con identifi-

cador k en función de su prioridad P_k. La prioridad se calculada de acuerdo a la siguiente fórmula:

$$P_k(p_k, q) = p_k \cdot q + (-1)^{round(p_k/q)} \cdot p_k$$

Donde p_k corresponde a la prioridad obtenida del archivo para el proceso k; *round* es una función de redondeo sin decimales; y q es una constante que se entrega por línea de comandos, con valor por defecto $q = 3$.

A partir de P_k, se asignan Q_k unidades de tiempo al proceso k, donde $Q_k = \left\lceil \frac{P_k(p_k,q)}{64} \right\rceil$.

3. **Non-Preemptive Priority-based**. Este *scheduler* ordena los procesos por orden de prioridad. En caso de que dos o más procesos tengan igual prioridad, éstos se atienden según FCFS.

Ejecución de la Simulación

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

```
./simulator <scheduler> <file> [<quantum>]
```

- `<scheduler>` corresponderá a `fcfs`, `roundrobin` o `priority`
- `<file>` corresponderá a un nombre de archivo que deberá leer como *input*
- `<quantum>` es un parámetro opcional que corresponderá al valor de q. Este valor es usado por el *scheduler* `roundrobin`. Este parámetro podría no ser entregado incluso si el *scheduler* elegido es `roundrobin`, y usted debe manejar adecuadamente esa situación.

Por ejemplo, algunas ejecuciones podrían ser las siguientes:

```
./simulator fcfs input.txt
./simulator roundrobin input.txt 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 PRIORIDAD TIEMPO_INICIO N A_1 B_1 A_2 B_2 ... A_N-1 B_N-1 A_N
```

- N indica la cantidad de valores A_i que hay en la secuencia.
- La secuencia $A_1\ B_1\ \ldots\ A_{N-1}\ B_{N-1}\ A_N$ representa la cantidad de tiempo en que el proceso estará en estado `RUNNING` (A_i), seguido de una cantidad de tiempo en que el proceso se bloquea y está en estado `WAITING` (B_i)[1].

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.
- La prioridad no será mayor ni menor a los límites indicados (entre 1 y 64, inclusive).
- El nombre del proceso no contendrá espacios, ni será más largo que 256 caracteres.
- $N \geq 1$. La secuencia de tiempos es de tamaño $2N - 1$, con N tiempos A_i y $N - 1$ tiempos B_i.

[1] El tiempo B_i simula un bloqueo por I/O, por ejemplo.

- La secuencia siempre iniciará y terminará con un intervalo de tiempo en que el proceso debe estar en estado `RUNNING`, después de lo cual el proceso terminará.
- No van a haber procesos cuyo tiempo de inicio sea el mismo.
- Habrá al menos un proceso descrito en el archivo.

El siguiente ejemplo ilustra cómo se vería un posible archivo de entrada:

```
PROCESS1 21 80 4 4 2 6 5 6 1 1
PROCESS2 13 91 9 2 4 1 8 3 7 5 5 4 4 7 7 6 6 7 2 8
```

Importante: es posible que en algún momento de la simulación no haya procesos en estado `RUNNNING` o `READY`. Los sistemas operativos manejan esto creando un proceso 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 es destruido.
- Cuando el *scheduler* elige un proceso para ejecutar en la CPU.
- En el caso del proceso activo, cuántos intervalos de tiempo ha ejecutado, y cuántos faltan para que termine.

Una vez que el programa haya terminado, su simulación deberá imprimir la siguientes estadísticas:

- Cuántos procesos han terminado su ejecución.
- El tiempo de duración de la simulación.

Además para cada proceso (terminado o no terminado), debe indicar:

- El número de veces que el proceso fue elegido para usar la CPU.
- El número de veces que se bloqueó [2].
- El *turnaround time*.
- El *response time*.
- El *waiting time* [3].

Importante: El simulador podría ser terminado de manera forzosa en mitad de la simulación usando Ctrl C. En este caso el simulador debe escribir las estadísticas que haya alcanzado a recolectar antes de terminar.

README

Deberá incluir un archivo `README` que indique quiénes son los autores de la tarea, cuáles fueron las principales decisiones de diseño para construir el programa, qué supuestos adicionales ocuparon, y cualquier información que considere necesarias para facilitar la corrección de su tarea. Se sugiere utilizar formato **markdown**.

[2] Equivale al número de veces que el proceso pasó de `RUNNING` a `WAITING`.
[3] Equivale a todo el tiempo que no está en `RUNNING`.

Formalidades

A cada alumno se le asignó un nombre de usuario y una contraseña para el servidor del curso (`iic2333.ing.puc.cl`). 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 código fuente** necesario para compilar su tarea, además del README y un `Makefile`. **NO debe incluir archivos binarios (será penalizado)**. Se revisará el contenido de dicha carpeta el día Viernes 25-Agosto-2017 a las 23:59.

- Puede ser realizada en forma individual, o en grupos de 2 personas. En cualquier caso, recuerde indicar en el `README` los autores de la tarea.
- Su tarea deberá compilar utilizando el comando `make` en la carpeta `T1`, y generar un ejecutable llamado `simulator` en esa misma carpeta.

Evaluación

- 0.5 pts. Formalidades. Esto incluye cumplir las normas de la sección formalidades.
- 4.5 pts. Simulación correcta.
  - 0.5 pts Estructura y representación de procesos.
  - 0.5 pts Estructura y manejo de colas.
  - 0.25 pts Activación del *scheduler*.
  - 0.25 pts Línea de comandos
  - 1.0 pts *Scheduler* FCFS.
  - 1.0 pts *Scheduler* RR.
  - 1.0 pts *Scheduler* Priority.
- 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 capturar el término forzado.

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.

Bonus (+0.5 pts): manejo de memoria perfecto

Se aplicará este bonus 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 correspondiente es $\geq 3{,}95$.

Preguntas

Cualquier duda preguntar a través del foro.