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

IIC2333 — Sistemas Operativos y Redes — 1/2017

Tarea 1

Viernes 17-Marzo-2017

Fecha de Entrega: Viernes 31-Marzo de 2017, 23:59
Composición: grupos de n personas, donde $n \leq 2$

Descripción

El *scheduling* es el arte de escoger procesos. Esto no es trivial, pues no se conoce mucho de ellos. La elección es importante para el sistema, ya que se busca que todos los procesos puedan avanzar de manera "justa", pero a la vez se intenta respetar la importancia de los procesos.

El objetivo de esta tarea es **simular** algunos *scheduler* como los que encontraría en cualquier sistema operativo. Tendrá que construir un programa en C con varios requerimientos, que serán detallados en este documento.

Modelamiento de los procesos

Debe existir un `struct Process`, que se encargará de modelar un proceso. Un proceso tiene un `PID`, una prioridad, un nombre de hasta 256 caracteres, una prioridad entre 1 y 64 (inclusive), y un estado. El estado 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. El proceso tendrá asociado una lista de tiempos t_i en los que se bloqueará por k_i unidades de tiempo. Más adelante se detalla cómo conseguir esta información.

Modelamiento de la cola de procesos

Debe existir un `struct Queue`, que se encargará de modelar una cola de procesos. Esta estructura de datos deberá ser modelada por usted, y deberá ser razonablemente eficiente. Se recomienda que utilice la misma cola para los tres schedulers que se le pedirán, y que sólo cambie el modo de elegir el proceso de acuerdo a cada uno. 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 se ejecute el *scheduler*, debe determinar si es necesario que el proceso activo (`RUNNING`) entregue la CPU y luego volver a sacar un proceso de la cola `READY`. Además, 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 acorde a lo que está ocurriendo. Puede aprovechar también de ingresar o desbloquear los procesos que lo requieran.

Usted implementará tres *schedulers* distintos:

1. FCFS (non-preemptive). Tomará cada proceso en el orden en que entraron a la cola, sin importar la prioridad asociada a éstos.

2. Round Robin con prioridades (*preemptive*). A diferencia del visto en clases, este *scheduler* dará un *quantum* proporcional a la prioridad del proceso, de forma que si a un proceso con prioridad por defecto (32) se le asignan q unidades de tiempo, a un proceso con prioridad máxima se le asigne $2q$ unidades, y a uno con prioridad mínima

se le asigne $\lfloor \frac{q}{2} \rfloor$ unidades de tiempo. El parámetro q deberá ser entregable a través de la línea de comandos, y por defecto será $q = 3$.

3. Random (*non-preemptive*). Luego de que un proceso termine de ejecutarse, tomará otro al azar.

Otros detalles

Línea de comandos

Su simulador será llamado con la siguiente instrucción:

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

- `<scheduler>` corresponderá a `fcfs`, `roundrobin` o `random`
- `<file>` corresponderá a un nombre de archivo que deberá leer como input (en la misma carpeta del ejecutable)
- `<quantum>` corresponderá al valor de q en el caso del *scheduler* Round-Robin. Este parámetro podría no ser entregado (incluso si el *scheduler* elegido es Round-Robin), y usted debe manejar adecuadamente esta situación.

Input

Usted deberá leer un archivo de texto al inicio de la ejecución, con 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
```

donde N indica el tamaño de la secuencia que viene, y 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 estará en estado `WAITING` (B_i). La secuencia siempre iniciará y terminará con un intervalo de tiempo en que el proceso deba estar en estado `RUNNING`, después de lo cual el proceso terminará. Podrá suponer que:

1. Cada número es un entero no negativo y que no sobrepasa el valor máximo de un `int` de 32 bits.
2. La prioridad no será mayor ni menor a los límites indicados anteriormente (entre 1 y 64, inclusive).
3. El nombre del proceso no contendrá espacios, ni será más largo que 256 caracteres.
4. $N \geq 1$, y que la secuencia de tiempos es del tamaño declarado en N.

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`.

Output

Usted deberá dar información de lo que está ocurriendo en `stdout`. Usted deberá indicar:

1. Cuando un proceso es creado, cambiado de estado (y a qué estado), o destruido.
2. Cuando el *scheduler* elija un proceso para continuar.
3. En el caso del proceso activo, cuántos intervalos de tiempo se ha ejecutado y cuántos faltan para que termine.

Cuando el programa haya terminado, deberá imprimir cuántos procesos han terminado su ejecución y el tiempo de duración de la simulación. Además para cada proceso (terminado o no terminado), debe indicar: (1) el número de veces que fue elegido para usar la CPU, (2) el número de veces que se bloqueó, (3) el *turnaround time*, el (4) *response time*, y el (5) *waiting time* (tiempo que permaneció esperando).

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, a grandes rasgos cuáles fueron las decisiones de diseño para hacer el programa, qué supuestos adicionales ocuparon, entre otras cosas que considere necesarias para una mejor corrección de su tarea. Se sugiere utilizar formato **markdown**.

Formalidades

La tarea será entregada mediante `git` en un repositorio **privado** que ustedes deberán crear. Se revisará el contenido de la rama `master` el día martes 28 de marzo de 2017 23:59.

- Puede ser realizada en forma individual, o en grupos de 2 personas.
- Su tarea deberá compilar utilizando el comando `make` en la raíz de su repositorio, y generar un ejecutable llamado `simulator` en esa misma carpeta.
- Su repositorio **DEBE ser privado**, de lo contrario calificará como copia (alguien les podría haber copiado). Esta restricción es fundamental. Si no la cumple **no** se revisará su tarea.
- La entrega será automatizada, basta que **registren** su grupo y repositorio mediante un formulario que habilitaremos para ello (no por email).
- Como el repositorio debe ser privado, tendrá que permitir acceso especial al curso, autorizando al servidor de tareas acceder al contenido. Para esto basta registrar la llave pública del curso en las **Deployment Keys** de su repositorio.
- Puede utilizar Bitbucket, Github u otro host, mientras sea privado y permita el acceso a los ayudantes mediante la **deployment key**.

Evaluación

- 10 %. Formalidades. Esto incluye cumplir las normas de la sección formalidades.
- 75 %. Simulación correcta.
  - 10 % Estructura y representación de procesos.
  - 10 % Estructura y manejo de colas.
  - 5 % Activación del *scheduler*.
  - 5 % Línea de comandos
  - 15 % *Scheduler* FCFS.
  - 15 % *Scheduler* RR.
  - 15 % *Scheduler* Random.
- 15 %. Debe entregar mensajes claros y precisos y que permitan entender lo que está pasando, idealmente que no ocupen más de una línea. Además debe capturar el término forzado.

El no respeto de las formalidades o un código extremadamente desordenado podría originar descuentos adicionales.

Preguntas

A través del foro de EdX.