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

IIC2333 — Sistemas Operativos y Redes — 1/2022

Tarea 2

Fecha de Entrega: Lunes 11-Abril-2022 a las 21:00
Composición: Tarea en parejas

Objetivos

- Modelar la ejecución de un algoritmo de *scheduling* de procesos.

Scheduler

Una de las partes más importantes de un sistema operativo es el Scheduler, un componente que se encarga de escoger el o los procesos que se ejecutarán en las CPU del computador. Las decisiones que toma este programa pueden influir considerablemente el rendimiento del sistema.

El objetivo de esta tarea en particular será **simular** el scheduler *Multi-level Feedback Queue*, el cual estará compuesto de tres colas y buscará optimizar tanto el tiempo de respuesta de un proceso, como el *turnaround time*.

Modelamiento de los procesos

Debe existir un `struct Process`, que modelará un proceso. Un proceso tiene un `PID`[1], un nombre de hasta 32 caracteres, una prioridad y un estado[2], que puede ser `RUNNING`, `READY`, `WAITING` o `FINISHED`.

La simulación recibirá un archivo de entrada que describirá los procesos[3] 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*, 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 ser eficiente. La colas deben estar preparadas para recibir **cualquier** cantidad de procesos que puedan estar en estado `READY` al mismo tiempo.

Las dos colas con más alta prioridad poseerán un número correspondiente a su *quantum*, el cual esta regido por la siguiente fórmula:

$$quantum = p_i \times q$$

Donde `q` es un input que recibe el programa y p_i es la prioridad de la cola, siendo $p_i = 2$ para la cola de mayor prioridad y $p_i = 1$ para la siguiente. En caso de que el quantum sea consumido por un proceso, este bajará de prioridad.

[1] Se recomienda usar un `int`
[2] Se recomienda usar un `enum`
[3] La cantidad de procesos es variable, sin embargo será siempre menor que `2048`

Además, estas dos colas siguen un esquema FIFO[4].

En cambio, la última cola sigue un esquema del tipo SFJ[5], donde aquellos procesos que presenten un tiempo restante de ráfaga de CPU menor serán los primeros en ejecutarse.

Scheduler

El *scheduler* decide qué procesos de la cola en estado `READY` entrarán en estado `RUNNING`. 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 tiempo total de ejecución.
- Pasar a estado `READY` en la cola al ser interrumpido sin haber terminado.

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[6].

Multi-level Feedback Queue

El scheduler que deberán implementar se basa en las siguientes reglas:

a) Aspectos generales:

- Cuando un nuevo proceso entra al sistema comienza en estado `READY` en la cola de mayor prioridad.
- Si la prioridad de un proceso A es mayor a la de un proceso B, es decir, se encuentran en colas distintas, se debe ejecutar A antes que B.
- Cada proceso cuenta con un valor de envejecimiento S independiente, el cual será entregado por input e indica cada cuanto tiempo el proceso vuelve a la cola de mayor prioridad, comenzando el conteo desde que el proceso ingreso al sistema.[7]
- Los procesos solo interrumpen su ejecución si acaban su quatum o ceden la CPU.
- Si un proceso se encuentra al inicio de la cola y en estado `WAITING` debe ser ignorado.

b) Primeras dos colas:

- Si existen dos procesos con igual prioridad dentro de las primeras dos colas y estos se encuentran en estado `READY`, serán ejecutado el proceso de la cola de mayor prioridad.
- Lo anterior implica que sus colas actuan bajo un esquema FIFO para entregar la CPU.
- Cuando un proceso usa todo su quantum en una determinada cola, se reduce su prioridad pasando a la cola siguiente.
- Cuando un proceso cede el control de la CPU por su cuenta, se aumenta su prioridad. Si el proceso estaba en la cola de prioridad máxima, su prioridad no cambia.
- El quatum se reinicia cada vez que se ingresa a la CPU.

c) Última cola:

[4] *First In First Out*
[5] *Shortest Job First*
[6] Para efectos de esta evaluación, se considerarán eficientes simulaciones que tarden, a lo más, un minuto en ejecutar para cada archivo de entrada. Para ver el tiempo de ejecución de un programa en C puede usar `time`.
[7] Visto de otra manera, un proceso volverá a la cola de mayor prioridad cuando $(t - t\_inicio)\ \%\ S = 0$.

- Si existen dos procesos en la última cola en estado `READY` serán ejecutados a partir de *Shortest Job First* basado en el tiempo restante de ráfaga de CPU, en caso de que existan dos o más procesos con el mismo tiempo restante, se ejecutarán bajo el esquema FIFO.
- Los procesos que entran a la CPU desde esta cola no son expropiados de la CPU, es decir, siempre lograrán completar su ráfaga, luego volverán a ubicarse al final de esta misma cola.

Orden de ingreso

Cuando dos o más procesos ingresan al mismo tiempo a una cola, estos deben seguir el siguiente orden de llegada:

1. Proceso que sale de la CPU.
2. Proceso que ingresa por primera vez al sistema. En caso de existir dos o más procesos que cumplan esto, se debe seguir el orden del archivo de input.
3. Proceso que aumenta su prioridad luego de su tiempo *S*, proveniente de la segunda cola. En caso de existir dos o más procesos que cumplan esto, se debe seguir el orden de ingreso en sus cola actual.
4. Proceso que aumenta su prioridad luego de su tiempo *S*, proveniente de la tercera cola. En caso de existir dos o más procesos que cumplan esto, se debe seguir el orden de ingreso en sus cola actual.

Flujo

Por cada unidad de tiempo el *scheduler* debe realizar:

1. Actualizar los procesos que cumplan su I/O burst de `WAITING` a `READY`.
2. En caso de existir un proceso en estado `RUNING`, actualizar su estado según corresponda.
3. Ingresar los procesos a sus colas correspondientes siguiendo la orden de ingreso.
   3.1) Si un proceso salió de la CPU, ingresarlo a la cola correspodiente.
   3.2) Por cada proceso p comprobar si $t = t\_inicio_p$ e ingresarlo a la primera cola.
   3.3) Por cada proceso p en la segunda cola verificar si se cumple $(t - t\_inicio_p) \,\%\, S_p = 0$ e ingresarlo a la primera cola.
   3.4) Por cada proceso p en la tercera cola verificar si se cumple $(t - t\_inicio_p) \,\%\, S_p = 0$ e ingresarlo a la primera cola.
4. Ingresar un proceso a la CPU si corresponde, esto implica cambiar su estado de `READY` a `RUNNING`.

Casos borde

- Si el proceso termina su ráfaga de CPU al mismo momento que se acaba su quatum, se asume que el proceso cedió la CPU.
- Si el proceso termina su ejecución al mismo momento que se acaba su quatum, este pasa a estado `FINISHED`
- Si al momento de transcurrir el tiempo *S* de un proceso este se encuentra en estado `RUNNING`, este volverá a la cola de mayor prioridad cuando salga de la CPU en cualquier caso. Para que vuelva a transcurrir *S* unidades de tiempo nuevamente se comienza a contar desde que se cumplió el tiempo anterior y no desde que se ingreso a la cola.

Ejecución de la Simulación

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

```
./mlfq <file> <output> <q>
```

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.[8]
- `<q>` corresponderá a la variable usada para el cálculo de los quantums de las colas.

Por ejemplo, algunas ejecuciones podrían ser las siguientes:

```
./mlfq input.txt output.csv 5
./mlfq procesos.txt salida.csv 20
```

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:
Donde:

```
NOMBRE_PROCESO PID TIEMPO_INICIO CYCLES WAIT WAITING_DELAY S
```

Donde:

- `NOMBRE_PROCESO` debe ser respetado para la impresión de estadísticas.
- `PID` es el Process ID del proceso.
- `TIEMPO_INICIO` es el tiempo de llegada a la cola. Considere que `TIEMPO_INICIO` ≥ 0.
- `CYCLES` es la cantidad tiempo que el proceso utilizará la CPU.
- `WAIT` son la cantidad de ciclos que el proceso correrá antes de conceder la CPU para esperar el input del usuario. Considere `WAIT` ≥ 1.
- `WAITING_DELAY` son la cantidad de ciclos que el proceso esperará el input del usuario, es decir, la cantidad de tiempo que el proceso quedará en `WAITING`.
- `S` es el tiempo de envejecimiento del proceso, es decir, la canrtidad de tiempo que debe pasar para que el proceso vuelva a la cola de mayor prioridad.

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.
- Habrá al menos un proceso descrito en el archivo.

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

[8] El output de su programa **debe** ser el archivo ingresado, de lo contrario no se contará como correcto.

```
3
PROCESS1 1 0 100 30 20 50
PROCESS2 13 5 200 40 10 60
PROCESS3 5 3 150 20 60 100
```

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.

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 alguna 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, **sin espacios de por medio**.

Formalidades

A cada alumno se le asignó un nombre de usuario y una contraseña para el servidor del curso[9]. 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 11-Abril-2022 a las 21:00.
Solo uno de los integrantes de cada grupo debe entregar en su carpeta. Los grupos los elegirán ustedes y en caso de que alguien no tenga compañero, puede crear una issue en el foro del curso buscando uno.
Antes de acabado el plazo de entrega de la tarea, se enviará un form donde podrán registrar los grupos junto con el nombre de usuario de la persona que realizará la entrega vía el servidor.

- **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 `T2`, y generar un ejecutable llamado `edf` en esta misma. Si su programa **no tiene** un `Makefile`, tendrá un descuento de 0.5 puntos en su nota final.

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

- 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:

- **5.0 pts.** Simulación correcta.
  - **0.5 pts.** Estructura y representación de procesos.
  - **1.5 pts.** Estructura y manejo de colas.
  - **3.0 pts.** *Multi-level Feedback Queue* [10].
- **1.0 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**[11].

Preguntas

Cualquier duda preguntar a través del foro oficial.

[10] Este puntaje será evaluado según la cantidad de *tests* a utilizar.
[11] Es decir, debe reportar 0 *leaks* y 0 errores para todo *test*.