



## IIC2333 — Sistemas Operativos y Redes — 1/2018 Tarea 1

Jueves 22-Marzo-2018

**Fecha de Entrega: Lunes 9-Abril-2018, 23:59**

**Ayudantía: Viernes 23-Marzo-2018, 10:00**

**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** el *scheduler Multi-level Feedback Queue* (MLFQ). A diferencia de los *schedulers* tradicionales, MLFQ busca optimizar tanto el *turnaround time* como el *response time* a través de múltiples colas, donde cada una de ellas denota una prioridad mayor o menor de los procesos que se encuentran contenidos en ella. En particular, se basa en las siguientes reglas:

- **Regla 1:** Si la prioridad de un proceso  $A$  es mayor a la de un proceso  $B$  (es decir, están en colas distintas), entonces  $A$  es ejecutado y no  $B$ .
- **Regla 2:** Si la prioridad de un proceso  $A$  es igual a la de un proceso  $B$  (es decir, se encuentran en la misma cola), entonces ambos son ejecutados a partir de *Round Robin*.
- **Regla 3:** Cuando un proceso entra al sistema, comienza en la cola de mayor prioridad.
- **Regla 4:** Cuando un proceso ha usado todo su *quantum*, se reduce su prioridad (pasa a la siguiente cola).
- **Regla 5:** Después de un periodo  $S$ , todos los procesos de las colas inferiores pasan a la de mayor prioridad.

Además de las reglas antes mencionadas, la mayoría de las variantes de MLFQ aumentan la *quanta* de cada proceso a medida que cambian de cola (*i.e.* bajan su prioridad). De esta forma, los procesos de menor prioridad tienen mayor probabilidad de terminar su ejecución al ser seleccionados por el *scheduler* al tener un tiempo de trabajo mayor.

La simulación de esta tarea debe determinar estadísticas de rendimiento en base a una secuencia de procesos que desean ejecutarse. Esta, además, 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, y un estado, que puede ser `RUNNING`, `READY`, o `FINISHED`.

La simulación recibirá un archivo de entrada que describirá los procesos a ejecutar y su comportamiento en cuanto a uso de CPU, lo que servirá como entrada a la simulación. Una vez terminado su tiempo de ejecución, el proceso terminará como si hubiese ejecutado `exit()`, y pasa a estado `FINISHED`.

### 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. La cola debe estar preparada para recibir cualquier cantidad de procesos que puedan estar en estado `READY` al mismo tiempo.

Toda cola tiene una prioridad asociada. En MLFQ, la primera cola, a la cual entran a los nuevos procesos, tiene la prioridad más alta, que es igual  $Q - 1$  donde  $Q$  es un parámetro que indica la cantidad de colas en la simulación. Cada cola sucesiva disminuye en 1 la prioridad hasta la última cola, que tiene prioridad 0.

## Scheduler

El *scheduler* decide cuál proceso en estado READY debe pasar a estado RUNNING dependiendo de la cola en la que se encuentre. Ningún proceso de la cola de prioridad  $k$  puede ser ejecutado si hay otros procesos en estado READY dentro de una cola con prioridad mayor a  $k$ .

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

Dada la complejidad de este algoritmo de *scheduling*, esta tarea se construirá de forma gradual a partir de **tres** versiones del algoritmo. Cada versión debe ser ejecutable de acuerdo a los argumentos que se entreguen en la línea de comandos.

1. **Primera versión:** Corresponde a la versión más básica. Sigue las reglas 1, 2, 3 y 4, es decir, los procesos solo disminuyen su prioridad.
2. **Segunda versión:** Contiene todas las reglas, es decir, después de un periodo  $S$  todos los procesos pasan a la cola de mayor prioridad.
3. **Tercera versión:** Además de las reglas antes mencionadas, esta versión hará lo que se comentó anteriormente: El valor del *quantum* utilizado por cada proceso **aumentará** a medida que **disminuya** su prioridad. Esto se regirá por la siguiente fórmula:

$$q_i = (Q - p_i) \times q$$

Donde  $q_i$  es la *quanta* de los procesos en la cola  $i$ ,  $p_i$  es la prioridad de la cola  $i$ ,  $q$  es la *quanta* ingresada como *input*, y  $Q$  es la cantidad de colas en el sistema. Esta será la versión completa del *scheduler* MLFQ de esta tarea.

## Ejecución de la Simulación

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

```
./mlfq <version> <file> <quantum> <queues> [<S>]
```

- **<version>** corresponderá a v1, v2 o v3 (dependiendo de la versión que se desee ejecutar).
- **<file>** corresponderá a un nombre de archivo que deberá leer como *input*.
- **<quantum>** es un parámetro que corresponderá al valor de  $q$ .
- **<queues>** corresponderá al número de colas que usará el algoritmo ( $Q$ ). Debe controlar que el número ingresado sea mayor a 0. Notar que si es 1, entonces estará implementando *Round Robin*.
- **[<S>]** es un parámetro que corresponde al periodo  $S$  que determinará el momento en el que todos los procesos pasan a la cola de mayor prioridad. Este solo es válido para v2 y v3, si no es entregado en estos casos, usted debe manejar adecuadamente la situación<sup>1</sup>.

Por ejemplo, algunas ejecuciones podrían ser las siguientes:

```
./mlfq v1 input.txt 4 1
./mlfq v3 input.txt 3 3 50
```

---

<sup>1</sup>Con un manejo adecuado se espera que se impida la ejecución, explicándole al usuario el error

## 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 A_2 ... A_{N-1} A_N
```

---

Tenga en cuenta:

- $N$  indica la cantidad de valores  $A_i$  que hay en la secuencia.
- La secuencia  $A_1 A_2 \dots A_{N-1} A_N$  representa la cantidad de tiempo que el proceso requerirá usar la CPU, es decir, el tiempo que solicitará estar en estado `RUNNING`. Si esta *ráfaga* o *burst* es menor que el *quantum* el proceso libera la CPU y vuelve a la misma cola, en caso contrario, desciende de cola.
- `NOMBRE_PROCESO` debe ser respetado para la impresión de estadísticas.
- 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 256 caracteres.
- 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 5 10 30 40 50 10
PROCESS2 13 3 14 6 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 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 (cual).
- 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 fue interrumpido<sup>2</sup>.
- El *turnaround time*.
- El *response time*.
- El *waiting time*<sup>3</sup>.

El formato esperado tiene dos secciones, una libre y una estructurada. La primera corresponde a las impresiones durante la simulación, que deben ser concisas pero explicativas, idealmente de una línea. La segunda parte corresponde a las estadísticas y es estructurada, de modo que debe seguir **rigurosamente** el siguiente formato:



---

```
Procesos terminados: N
Tiempo total: T

NOMBRE_PROCESO_i:
Turnos de CPU: V
Bloqueos: W
Turnaround time: X
Response time: Y
Waiting time: Z
```

---

Donde habrán tantos procesos como corresponda (separados por una línea en blanco), y cada letra es reemplazada por el entero correspondiente. Debe cuidar que los saltos de línea, espacios y textos sean idénticos ya que esta parte de la tarea será revisada automáticamente. Se proporcionará un archivo de ejemplo.

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

## 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. Por otra parte, se **debe** realizar un breve análisis comparativo entre las versiones de MLFQ implementadas. En particular, se espera que bajo una misma parametrización se compare el *turnaround time*, *response time* y el *waiting time* de las distintas versiones de MLFQ y concluir si las funcionalidades añadidas conllevan a una ejecución de mejor rendimiento. Se sugiere utilizar formato **markdown**.

## 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 Lunes 9-Abril-2018, 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.

---

<sup>2</sup>Equivale al número de veces que el *scheduler* sacó al proceso por el uso completo de su *quantum*.

<sup>3</sup>Equivale a todo el tiempo que está en `READY`.

- Su tarea deberá compilar utilizando el comando `make` en la carpeta `T1`, y generar un ejecutable llamado `mlfq` en esa misma carpeta.

**Importante:** Si bien esta sección tiene su propio puntaje, especificado más adelante, su incumplimiento puede repercutir en que **NO** se corrija la tarea y que se califique con la nota mínima.

## Evaluación

- 0.25 pts. Formalidades. Esto incluye cumplir las normas de la sección formalidades.
- 4.25 pts. Simulación correcta.
  - 0.25 pts. Estructura y representación de procesos.
  - 0.25 pts. Estructura y manejo de colas.
  - 0.25 pts. Activación del *scheduler*.
  - 0.25 pts. Línea de comandos
  - 0.75 pts. Primera versión *Scheduler* MLFQ.
  - 1.25 pts. Segunda versión *Scheduler* MLFQ.
  - 1.25 pts. Tercera versión *Scheduler* MLFQ.
- 0.25 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.
- 0.25 pts. Debe capturar el término forzado.
- 1.0 pts. Se entrega un README con toda la información y análisis solicitado.

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](#).

## Material complementario

En el [siguiente enlace](#) se encuentra un documento que profundiza la construcción del algoritmo MLFQ. Se recomienda su revisión en caso de tener dudas con respecto de la implementación solicitada.