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

IIC2333 — Sistemas Operativos y Redes — 1/2021

**Tarea 2**

**Fecha de Entrega: Miércoles 28 de Abril, 23:59**
**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 busca 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`, un nombre de hasta 32 caracteres, una prioridad y un estado, que puede ser `RUNNING`, `READY`, `WAITING` o `FINISHED`.

La simulación recibirá un archivo de entrada que describirá los procesos[1] 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.

Cada una de estas colas poseerá un número correspondiente a su *quantum*, el cual aumenta a medida que disminuye la prioridad, regido por la fórmula:

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

Donde `Q` es la cantidad total de colas del sistema, `q` es un input del programa y $p_i$ es la prioridad de la cola. Terminado este quantum el proceso bajará de prioridad luego de terminada su ejecución.

[1] La cantidad de procesos es variable, sin embargo será siempre menor que `2048`

Toda cola tiene una prioridad asociada, la primera cola tendrá la prioridad mas alta, $Q-1$, cada cola sucesiva disminuye en 1 la prioridad hasta la última que tendrá prioridad 0. Para sacar una proceso de una cola, se debe seguir un esquema FIFO[2].

## *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[3].

## *Multi-level Feedback Queue*

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

- Si la prioridad de un proceso A es mayor a la de un proceso B (están en colas distintas), se ejecutará A y no B.
- Si dos procesos listos para ejecutar tienen la misma prioridad, serán ejecutados a partir de *Round Robin*.
  - Lo anterior implica que sus colas actuan bajo un esquema FIFO para entregar la CPU.
  - Para este esquema FIFO debe ignorar los procesos en espera.
- Cuando un nuevo proceso entra al sistema (pasa a estado READY por primera vez), comienza en la cola de mayor prioridad.
- 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 y se deja al final de la cola correspondiente. Si el proceso estaba en la cola de prioridad máxima, su prioridad no cambia.
- Luego de un período de tiempo $S$, todos los procesos del sistema vuelven a la cola de mayor prioridad. Esto debe ser en orden de colas de mayor a menor prioridad, respetando el esquema FIFO de las colas.
- El único motivo por el que se interrumpe la ejecución de un proceso, es el término de su quantum. Si llega un proceso con mayor prioridad, se esperará que el proceso actual termine para elegir uno nuevo.

---

[2] *First In First Out*

[3] 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`.

# Ejecución de la Simulación

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

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

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.[4]
- `<Q>` corresponderá a la cantidad de colas que tendrá su programa.
- `<q>` corresponderá a la variable usada para el cálculo de los quantums de las colas.
- `<S>` período en el cual los procesos pasan a la cola de mayor prioridad.

Por ejemplo, algunas ejecuciones podrían ser las siguientes:

```
./mlfq input.txt output.csv 5 10 100
./mlfq procesos.txt salida.csv 20 5 30
```

## 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 PID TIEMPO_INICIO CYCLES WAIT WAITING_DELAY
```

Donde:

- `CYCLES` es la cantidad tiempo que el proceso utilizará la CPU.
- `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` $\geq 0$.
- `WAIT` son la cantidad de ciclos que el proceso correrá antes de conceder la CPU para esperar el input del usuario. Si este número es 0, el proceso nunca concederá el control por su cuenta.
- `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`.
- 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.
- Habrá al menos un proceso descrito en el archivo.

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

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

```
3
PROCESS1_RAUL 21 2 120
PROCESS2_RICHI 13 1 150
PROCESS3_TRINI 5 3 200
```

**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 entrega la CPU por su cuenta 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 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**.

## 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:

- 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[5]. 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 Miércoles 28 de Abril, 23:59.
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.
- 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* [6].
- **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**[7].

## Preguntas

Cualquier duda preguntar a través del foro oficial.

[5] `iic2333.ing.puc.cl`
[6] Este puntaje será evaluado según la cantidad de *tests* a utilizar.
[7] Es decir, debe reportar 0 *leaks* y 0 errores para todo *test*.