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

IIC2333 — Sistemas Operativos y Redes — 1/2025

Tarea 1 - *Scheduling*

Viernes 4 de abril 2025

Fecha de Entrega: Miércoles 16 de Abril a las 21:00 hrs.
Composición: Tarea en parejas

Objetivos

- Modelar un *scheduler* de procesos utilizando `structs`.
- Simular la ejecución de dicho *scheduler* sobre un grupo de procesos.

DCCScheduler

Una vez resuelto el desafío de gestionar los procesos, surge la necesidad de garantizar que todos sean atendidos de manera eficiente. Para abordar este problema, los profesores sugieren una solución: **`DCCScheduler`**, un programa que distribuye los procesos a medida que llegan, asegurando que todos puedan ser atendidos y finalicen sus actividades. Para ello, se propone implementar un **scheduler** de tipo MLFQ (*MultiLevel Feedback Queue*) que, de manera eficiente, atiende de forma óptima y efectiva a los procesos.

Descripción

El *scheduling* consiste en elegir el siguiente proceso a ejecutar en la CPU. Dado que no siempre se tiene información completa sobre los procesos, existen diferentes algoritmos para intentar tomar la mejor decisión. Uno de ellos es el MLFQ, que deberán **implementar**.

Un MLFQ es un *scheduler* en que se utilizan varias colas con diferentes prioridades, a modo de reducir la probabilidad de inanición de un proceso. Ustedes deberán modelar un *scheduler* MLFQ con tres colas, una de alta prioridad, otra de media prioridad y una de baja prioridad.

Modelamiento de procesos

Debe construir un `struct Process`, que modelará un proceso. Cada proceso tiene al menos las siguientes variables asociadas:

1. Nombre.
2. PID.
3. Estado $\in$ {`RUNNING, READY, WAITING, FINISHED`}.
4. Tiempo de ejecución por ráfaga (*burst*).
5. Número de ráfagas de ejecución en CPU.
6. Tiempo de espera para I/O entre ráfagas.
7. Prioridad (numero entre 1 y 30).

Modelamiento de las colas de procesos

También debe existir un `struct Queue`, el cual modelará una cola de procesos. Esta estructura debe estar diseñada por ustedes y debe ser capaz de recibir un número arbitrario de procesos en estado `READY` de manera simultánea.

La cola *High* y *Medium* tienen un *quantum* asociado. En el caso de la cola *High*, el *quantum* se calcula según la fórmula $quantum = 2 * q$, y para la cola *Medium* según $quantum = q$, donde q es un parámetro que recibe el programa.
La cola *Low* no tendrá un quantum asociado y seguirá el método FIFO para el manejo de procesos.

Scheduler

El *scheduler* puede contener procesos en las colas. Los procesos en una cola pueden estar en dos estados posibles:

- `READY`, indica que el proceso está listo para ejecutar.
- `WAITING`, indica que el proceso está esperando y no puede ejecutar.

En caso de que un proceso no se encuentre dentro de una cola, sus estados pueden ser:

- `RUNNING`, indica que el proceso está ejecutando dentro de la CPU. Sólo puede haber un proceso en este estado a la vez.
- `FINISHED`, indica que el proceso ya ha terminado su ejecución.

El scheduler debe asegurar que el estado de cada proceso cambie de manera consistente.

DCCScheduler

Su programa debe implementar el *scheduler* MFLQ de tres colas, *High*, *Medium* y *Low*, de acuerdo a las siguientes reglas:

- Los procesos de la cola *High* siempre tienen prioridad sobre la cola *Medium* y *Low*. Los procesos de la cola *Medium*, a su vez, siempre tendrán prioridad sobre los de la cola *Low*.
- Todo proceso que ingresa al *scheduler* por primera vez, ingresa a la cola según su *Prioridad*:
  - A la cola *High* si $1 \leq$ *Prioridad* ≤ 10
  - A la cola *Medium* si $11 \leq$ *Prioridad* ≤ 20
  - A la cola *Low* si $21 \leq$ *Prioridad* ≤ 30
- Dentro de una misma cola, los procesos se ordenan según el valor de *Prioridad*, en caso de empate, tiene mayor prioridad el proceso con mayor *PID*.
- en *High* y *Medium* tenga en cuenta:
  - Un proceso en estado `RUNNING` solo sale de la CPU si se acaba su $quantum$ o cede la CPU (ya sea por que se acabo el burst entrando a estado `WAITING` por I/O).
  - Si el proceso cede la CPU , este se mantiene en la misma cola en la que se encontraba, y el *quantum* no se reinicia.
  - Si un proceso consume todo su *quantum*, este pasa a la cola siguiente con menor prioridad.
- Los procesos en la cola *Low* también son ejecutados por orden de llegada, pero al no poseer *quantum* se ejecuta todo su *burst*.

- Los procesos en estado `WAITING` no son elegibles para ejecutar.
- Transcurridos n *ticks* **TODOS** los procesos suben de prioridad, es decir, pasan a la cola superior.
- Si un proceso está en estado `RUNNING` no se actualiza su prioridad.

Flujo del *scheduler*

Por cada *tick*, el *scheduler* debe realizar las siguientes tareas, en el orden indicado:

1. Actualizar los procesos que hayan terminado su tiempo de espera de I/O de `WAITING` a `READY`.
2. Si hay un proceso en estado `RUNNING`, actualizar su estado según corresponda. Esto podría incluir sacarlo de la CPU si su *quantum* ha expirado o ha terminado su ráfaga de ejecución (*burst*).
3. Ingresar los procesos a las colas según corresponda:
   3.1) Si un proceso salió de la CPU, ingresarlo a la cola que corresponda.
   3.2) Si el tiempo de inicio de un proceso se cumple, ingresarlo a la cola que corresponda.
   3.3) Si han pasado n *ticks*, subir la prioridad de todos los procesos, ingresandolos a la cola siguiente correspondiente.
4. Si no hay un proceso en estado `RUNNING`, ingresar el proceso de mayor prioridad en estado `READY` a la CPU, esto implica ignorar a todos los que se encuentren en estado `WAITING`, sin moverlos de su posición actual. Si la cola con mayor prioridad se encuentra sin procesos que puedan ser ejecutados, se ejecutan los de la siguiente cola.

Consideraciones especiales

- Si el proceso termina su ráfaga de CPU al mismo momento que se acaba su *quantum*, se considera que el proceso cedió la CPU.
- Si el proceso termina su ejecución al mismo momento que se acaba su *quantum*, este pasa a estado `FINISHED`.

Ejecución de la simulación

La simulación se debe ejecutar con el siguiente comando:

```
./DCCScheduler <input_file> <output_file> <q> <n>
```

Donde `input_file` es el nombre del archivo de *input*, `output_file` corresponde a la ruta a un archivo `CSV` con las estadísticas que debe guardar el programa, `q` es el ponderador para el *quantum* y `n` son la cantidad de *ticks* a esperar antes de un cambio de prioridad. Una posible ejecución sería `./DCCScheduler procesos.txt salida.csv 3 10`.

Archivo de entrada (*input*)

El archivo de entrada contiene los datos de la simulación, donde la primera línea contiene un entero K que indica la cantidad de procesos y las siguientes K líneas tienen el siguiente formato:

```
NOMBRE_PROCESO PID T_INICIO T_CPU_BURST N_BURSTS IO_WAIT PRIORITY
```

Donde:

- `PID` es el Process ID del proceso.
- `T_INICIO` es el tiempo en el que el proceso entra a la cola por primera vez.
- `T_CPU_BURST` es el tiempo que dura una ráfaga de ejecución en la CPU.
- `N_BURSTS` es la cantidad de ráfagas de ejecución en la CPU que tiene el proceso.
- `IO_WAIT` es el tiempo de espera entre ráfagas de ejecución.
- `PRIORITY` es la prioridad de ejeccucion del proceso.

Puede suponer que no habrá tiempos negativos, también pueden asumir que `IO_WAIT >= 0`. Todos los tiempos se entregan sin dimensión y caben en un entero sin signo de 32 bits. Tampoco se entregarán archivos vacíos.

Archivo de salida (*output*)

En su archivo de salida, debe incluir una fila con las siguientes estadísticas para cada proceso:

- Nombre del proceso
- PID
- El *turnaround time*
- El *response time*, considerando el tiempo que tardó en entrar por primera vez a la CPU.
- El *waiting time*, este es la suma de todos los intervalos en que estuvo en estado `WAITING` o `READY` para ejecutar.
- Numero de cambios de cola.

Además, debe estar impreso por orden de término, es decir, el proceso que termine primero debe estar impreso al principio del archivo. Por lo tanto, el archivo de salida se vería como sigue:

```
nombre_proceso_a,pid_a,turnaround_a,response_a,waiting_a,n_cambios_de_cola_a
nombre_proceso_b,pid_b,turnaround_b,response_b,waiting_b,n_cambios_de_cola_b
nombre_proceso_c,pid_c,turnaround_c,response_c,waiting_c,n_cambios_de_cola_c
```

Es importante que **respeten el formato del output.**

Ejemplo Input y Output

A continuación, se les presenta un input y output válidos.
Input:

```
2
PROCESS1 0 3 4 2 1 1
PROCESS2 1 6 2 3 1 3
```

Output:

```
PROCESS1,0,10,0,2,0,1
PROCESS2,1,2,12,1,6,0,2
```

Formalidades

A cada alumno se le asignó un nombre de usuario y una contraseña para el servidor del curso[1]. Para entregar su tarea usted deberá crear una carpeta llamada **exactamente** `T1`[2] en el directorio principal de su carpeta personal y subir su tarea a esa carpeta. En su carpeta `T1` **solo debe incluir el código fuente** es necesario para compilar su tarea y un `Makefile`. Se revisará el contenido de dicha carpeta el día Miércoles 16 de Abril a las 21:00 hrs..

- La tarea debe ser realizada solamante en parejas.
- La tarea deberá ser realizada en el lenguaje de programación **C**. Cualquier tarea escrita en otro lenguaje de programación no será revisada.
- **NO debe incluir archivos binarios**[3]. En caso contrario, tendrá un descuento de 0.2 décimas en su nota final.
- **NO debe incluir un repositorio de git**[4]. En caso contrario, tendrá un descuento de 0.2 décimas en su nota final.
- **NO debe usar VSCode para entrar al servidor**[5]. En caso contrario, tendrá un descuento de 0,2 puntos en su nota final.
- Si inscribe de forma incorrecta su grupo o no lo inscribe, tendrá un descuento de 0.3 décimas
- Su tarea debe compilarse utilizando el comando `make`, y generar un ejecutable llamado `DCCScheduler` en esa misma carpeta. Si su programa **no tiene** un `Makefile`, tendrá un descuento de 0.5 décimas en su nota final y corre riesgo que su tarea no sea corregida.
- En caso de no cumplir con el formato de entrega, tendrá un descuento de 0.3 décimas en la nota final.
- Si ésta **no compila** o **no funciona** (*segmentation fault*), obtendrán la nota mínima, pudiendo recorregir modificando líneas de código con un descuento de una décima por cada cuatro líneas modificada, con un máximo de 20 lineas a modificar.

El no respeto de las formalidades o un código extremadamente desordenado podría originar descuentos adicionales, los cuales quedarán a discreción del ayudante corrector. 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

- **0.4 pts.** Correcta modelación de los procesos.
- **0.7 pts.** Correcta modelación de la cola *High*.
- **0.7 pts.** Correcta modelación de la cola *Medium*.
- **0.7 pts.** Correcta modelación de la cola *Low*.
- **2.0 pts.** Correcta simulación del *scheduler*. Esto será evaluado con una serie de *inputs* de prueba.
- **1.0 pts.** *Output* correcto.
- **0.5 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**[6].
- **Bonus**: Se otorgará 0.2 puntos adicionales por entregar un archivo README.md explicativo que funcione como *guía para la corrección*, siempre y cuando se entregue dentro del plazo estipulado. El bonus no se aplicará si se entrega fuera de plazo o si se incurre en la política de atrasos.

---

1 `iic2333.ing.puc.cl`
2 Se debe respetar el uso de mayúsculas, sino, la tarea no sera recolectada.
3 Los archivos que resulten del proceso de compilar, como lo son los ejecutables y los *object files*
4 Si es que lo hace, puede eliminar la carpeta oculta `.git` antes de la fecha de entrega.
5 Si es que lo hace, puede eliminar la carpeta oculta `.vscode-server` antes de la fecha de entrega.
6 Es decir, debe reportar 0 *leaks* y 0 errores para todo *test*

Política de atraso

Se puede hacer entrega de la tarea con un máximo de 2 días hábiles de atraso. La fórmula a seguir es la siguiente:

$$N_{T_1}^{\text{Atraso}} = \text{mín}(N_{T_1}, 7{,}0 - 0{,}75 \cdot d)$$

Siendo d la cantidad de días de atraso. Notar que esto equivale a un descuento *soft*, es decir, cambia la nota máxima alcanzable y no se realiza un descuento directo sobre la nota obtenida. El uso de días de atraso no implica días extras para alguna tarea futura, por lo que deben usarse bajo su propio riesgo.

Preguntas

Cualquier duda, preguntar a través del foro oficial.