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

IIC2333 — Sistemas Operativos y Redes — 2/2021

Tarea 2

Martes 7-Septiembre-2021

Fecha de Entrega: Jueves 16-Septiembre-2021, 23:59
Ayudantía: Viernes 10-Septiembre-2021, 08:30
Composición: Tarea en parejas

Objetivos

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

Descripción

Luego de crear exitosamente el sistema de DCCuber, este escaló y hay 4 fábricas que envían paquetes por distintas rutas. Lamentablemente solo hay un servidor para manejar los procesos. Las fábricas pequeñas encuentran injusto que las más grandes usen toda la CPU ya que tienen muchos procesos. Por esto se les pide a ustedes que implementen un algoritmo de *scheduling* basado (vagamente) en *Fair Share*, que se encargue de repartir la CPU equitativamente entre las fábricas.

El objetivo de esta tarea es **implementar** un algoritmo de *scheduling Round Robin* (RR) que determine el quantum de ejecución dependiendo del grupo (fábrica) al cual pertenece el proceso. Este algoritmo asigna un tiempo de ejecución, que llamaremos *quantum* para cada proceso dentro de una única cola. El *quantum* se asigna por turnos a cada proceso, permitiendo que todos los procesos puedan ser atendidos. Deberá implementar el algoritmo mediante un programa en C y **simular de forma discreta**[1] su comportamiento de acuerdo a un conjunto de procesos descritos en un archivo.

Modelamiento de los procesos

Debe existir un `struct Process`, que modelará un proceso. Un proceso tiene un `PID`, un nombre de hasta 255 caracteres, un `NÚMERO DE FÁBRICA`, y un estado, que puede ser `RUNNING`, `READY`, `WAITING` o `FINISHED`.

La simulación recibirá un archivo de entrada que describirá los procesos 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*), y un tiempo B_i en que permanece bloqueado (*I/O burst*). Un proceso que está ejecutando deja de hacerlo si: cede la CPU porque se acabó su ráfaga de ejecución(*CPU burst*), o es **interrumpido** por el *scheduler* porque consumió su quantum, o porque terminó.

Modelamiento de la cola de procesos

Debe construir un `struct Queue` para modelar una cola de procesos (es decir, de `struct Process`), la forma de implementar esta cola queda a criterios de ustedes (array, lista ligada, etc). Esta estructura de datos deberá ser

[1] Una simulación discreta tiene una unidad de tiempo que va subiendo de a una unidad, a diferencia de una simulación por eventos que se guía por una cola de eventos a ejecutar.

construida por usted y deberá ser eficiente en términos de tiempo[2]. La cola debe estar preparada para recibir **hasta ocho** procesos al mismo tiempo.

Scheduler

El *scheduler* decide qué proceso de la cola que esté en estado `READY` pasa a ser ejecutado, pasando al estado `RUNNING`. Este elección sigue un orden FIFO. Se debe elegir el primer proceso de la cola que esté `READY`. Los procesos de la cola que están `WAITING` se deben ignorar. El *scheduler* se activa cada vez que se consume un *quantum* o bien cuando el proceso `RUNNING` decide bloquearse (entra a un periodo B_i). El *scheduler* debe sacar al proceso actual de la CPU y ponerlo al final de la cola solo si es que no ha terminado. Si el proceso sale porque cedió la CPU (terminó un periodo A_i) debe cambiar su estado a `WAITING`. Si el proceso fue interrumpido durante su *CPU burst*, entonces pasa a estado `READY`.

En la implementación del algoritmo RR, el *quantum* de los procesos que pertenecen a una misma fábrica i debe ser calculado de la siguiente manera:

$$q_i = \left\lfloor \frac{Q}{n_i * f} \right\rfloor$$

Donde Q es un input del programa, n_i es la cantidad de procesos de la fábrica i que se encuentran actualmente en la cola. Finalmente, f es la cantidad de fábricas con al menos un proceso dentro de la cola. La constante Q debe ser entregada por línea de comandos y su valor por defecto es $Q = 100$.

El quantum debe calcularse cada vez que a un proceso le toca ejecutar. Se deben tomar en cuenta los procesos que hay en la cola en ese momento (incluyendo los que llegaron en ese instante).

Durante cada momento (que corresponderá a un aumento de la unidad de tiempo de la simulación), se deberá revisar e informar los cambios que ocurre en cada proceso, en caso de que ocurra alguno.

Orden de prioridad llegada

En cada unidad de tiempo pueden llegar múltiples procesos al final de la cola, estos pueden venir de la CPU ya que acaba de terminar su *CPU burst* o su *quantum*, o pueden ser procesos nuevos. Si es que llega más de un proceso en la misma unidad de tiempo debemos hacer un "desempate" para determinar quien entra primero a la cola. Se seguirán las siguientes prioridades en orden para romper dicho el empate:

1. Proceso que pasó a `WAITING`.
2. Proceso consumió su quantum y está en `READY`.
3. Procesos que llegan por primera vez a la cola.
   3.1) Con menor número de fábrica f.
   3.2) Con menor `NOMBRE_PROCESO`. Para esta parte deberás usar **strcmp**[3].

[2]Para efectos de esta evaluación, se considerarán eficientes simulaciones que tarden, a lo más, 60 segundos en ejecutar para cada archivo de entrada. Para ver el tiempo de ejecución de un programa en C puede usar `time`.
[3]Recomendamos leer la manpage de **strcmp**

Orden de ejecución de eventos en una unidad de tiempo

La siguiente lista muestra el orden en el que deben ser procesados todos los posibles eventos en cada unidad de tiempo. Los procesos **NUNCA** podrán contabilizar dos estados distintos en la misma unidad de tiempo:

1. Si hay un proceso en la CPU se debe hacer lo siguiente:
   - `IF` Proceso cede la CPU, pasa a `WAITING`, y se va al final de la cola.
   - `ELSE IF` Proceso termina su ejecución, pasa a `FINISHED` y sale del sistema.
   - `ELSE IF` Proceso consume todo su quantum, pasa a `READY` y se va al final de la cola.
   - `ELSE`, el proceso continua en `RUNNING`.
2. Procesos creados entran a la cola, incluyendo el proceso que salga de la CPU.
3. Si no hay un proceso en la CPU se elige uno para que pase a `RUNNING`.
4. Se actualizan las estadísticas de los procesos. Si un proceso salió de la CPU, se considera como su hubiera estado `RUNNING`.
5. Los procesos `WAITING` que terminaron su *I/O Burst* (B_i) pasan a `READY`.

Ejecución de la Simulación

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

```
./scheduler <file> <output> [<Q>]
```

- `<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.
- `[<Q>]` es un parámetro que corresponderá al valor de Q. Si no es ingresado, debe usarse por defecto $Q = 100$.

Por ejemplo, algunas ejecuciones podrían ser las siguientes:

```
./scheduler input.txt output.csv 120
./scheduler input.txt output.csv
```

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:

La primera línea contendrá la cantidad de líneas que contiene el archivo (sin considerarse a si misma):

```
N
```

Cada una de las siguientes `N` líneas representará a un proceso:

```
NOMBRE_PROCESO TIEMPO_INICIO F N A_1 B_1 A_2 B_2 ... A_N-1 B_N-1 A_N
```

Donde:

- `NOMBRE_PROCESO` debe ser respetado para la impresión de estadísticas.
- `TIEMPO_INICIO` es el tiempo de llegada a la cola. Considere que `TIEMPO_INICIO` ≥ 0.
- F es el número de la fábrica a la que pertenece (identificadas del 1 al 4).
- N es la cantidad de ráfagas de CPU, con $N \geq 1$.
- La secuencia $A_1\ B_1\ \ldots\ A_{N-1}\ B_{N-1}\ A_N$ describe el comportamiento del proceso. Donde cada número A_i representa la duración de una *CPU burst* (tiempo que el proceso deberá estar `RUNNING`) y cada número B_i representa la duración de una *I/O burst* (tiempo que el proceso deberá estar `WAITING`). Se debe considerar lo siguiente:
  - Un proceso siempre empezará y terminará con una *CPU burst*.
  - Cada *CPU burst* será seguida por una *I/O burst* a excepción de la última.
  - Cada A_i como B_i serán enteros mayores o iguales a 1.
  - Si un A_i es mayor al *quantum* calculado para el proceso este será interrumpido en algún momento, pasando de `RUNNING` a `READY`. Si un proceso es interrumpido, deberá considerar cuanto tiempo de *CPU burst* le quedaba.[4].

Supuestos Input

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 254 caracteres.
- Como máximo existirán 2 procesos por fábrica que tengan el mismo tiempo de inicio.
- Habrá al menos un proceso descrito en el archivo.

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

```
2
PROCESS1 21 3 5 10 4 30 3 40 2 50 1 10
PROCESS2 13 1 3 14 3 6 3 12
```

Salida (*Output*)

Output de la consola

Se deberá dar información de lo que está ocurriendo. Deberá **imprimir en consola** 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 (indicar cuál proceso).
- Si es que la CPU no está ejecutando ningún proceso.

[4]Por ejemplo, para una ráfaga A_i y *quantum* q, luego de ser interrumpido todavía quedarían $A_i - q$ unidades de tiempo de ejecución de la ráfaga

El formato de impresión de cada evento es libre, pero se espera que sea conciso y explicativo, idealmente de una línea. Por ejemplo:

```
...
[t = 3] No hay ningun proceso ejecutando en la CPU.
[t = 4] El proceso CARS ha pasado a estado RUNNING.
[t = 5] El proceso REPARTIDOR ha sido creado.
[t = 7] El proceso CARS ha pasado a estado READY.
[t = 8] El proceso CRUZ ha pasado a estado RUNNING.
[t = 9] El proceso CRUZ ha pasado a estado FINISHED.
...
```

Output del archivo

Una vez que el programa haya terminado, su programa deberá escribir un archivo `CSV`[5] **que debe tener la extensión .csv**, con los siguientes datos por proceso:

- El nombre del proceso.
- El número de veces que el proceso fue elegido para usar la CPU.
- El número de veces que fue interrumpido. Este equivale al número de veces que el *scheduler* sacó al proceso por el uso completo de su *quantum*.
- El *turnaround time*. Definido como el tiempo que el proceso permaneció en el sistema, desde que el proceso llega hasta que termina de ser atendido (termina su última ráfaga).
- El *response time*. Definido como el tiempo desde que el proceso llega al sistema hasta que es atendido (pasa a `RUNNING`) por primera vez.
- El *waiting time*. Este equivale a la suma del tiempo que 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. Las líneas **deben** estar en el mismo orden del archivo de entrada.

Ejemplo

Sea un proceso con un $q = 4$, y cuyo archivo de input es

```
1
PROCESS1 0 1 3 5 6 4 2 4
```

Su output en consola debe corresponder a:

[5] *Comma Separated Values*

```
[t = 0] El proceso PROCESS1 ha sido creado.
[t = 0] El proceso PROCESS1 ha pasado a estado RUNNING.
[t = 4] El proceso PROCESS1 ha pasado a estado READY.
[t = 4] El proceso PROCESS1 ha pasado a estado RUNNING.
[t = 5] El proceso PROCESS1 ha pasado a estado WAITING.
[t = 11] El proceso PROCESS1 ha pasado a estado READY.
[t = 11] El proceso PROCESS1 ha pasado a estado RUNNING.
[t = 15] El proceso PROCESS1 ha pasado a estado WAITING.
[t = 17] El proceso PROCESS1 ha pasado a estado READY.
[t = 17] El proceso PROCESS1 ha pasado a estado RUNNING.
[t = 21] El proceso PROCESS1 ha pasado a estado FINISHED.
```

No hay ningún tiempo en que el proceso esté en estado `READY`.

Los tiempos que el proceso está en estado `RUNNING` son:

$$0, 1, 2, 3, 4, 11, 12, 13, 14, 17, 18, 19, 20$$

Los tiempos que el proceso esta en estado `WAITING` son:

$$5, 6, 7, 8, 9, 10, 15, 16$$

Los tiempos donde la CPU no es usada son:

Ninguno

La explicación es la siguiente:

- En $t = 0$, el proceso llega al *scheduler* en estado `READY`, y pasa a `RUNNING`, posteriormente, se guarda el estado del proceso para informar en el archivo de output.
- En $t = 0, 1, 2, 3$, el proceso ejecuta hasta que se cumple su *quantum*, en cada uno de esos tiempos anota su estado.
- Para $t = 4$, el proceso luego de haber guardado su estado, se cumple su *quantum*, por lo que pasa a `READY`
- En $t = 4$, el proceso pasa a `RUNNING` y luego anota su estado.
- En $t = 5$, luego de guardar su estado, el proceso pasa a `WAITING`.
- EN $t = 6, 7, 8, 9, 10$, el proceso hace `WAITING`.
- En $t = 11$, luego de haber guardado su estado, el proceso pasa a `READY`.
- En $t = 11$, el proceso pasa a `RUNNING`, y luego guarda su estado.
- En $t = 11, 12, 13, 14$, el proceso hace `RUNNING`.
- En $t = 15$, luego de guardar su estado, el proceso pasa a `WAITING`, ya que pese a que se le cumple su *quantum*, también se acaba su segunda ráfaga de CPU.
- En $t = 15, 16$, el proceso hace `WAITING`, guardando ese estado.
- En $t = 17$, después de guardar su estado, el proceso pasa a `READY`.
- En $t = 17$, el proceso pasa a `RUNNING`.
- En $t = 17, 18, 19, 20$, el proceso que está en `RUNNING` anota ese estado.

- En $t = 21$, el proceso pasa a `FINISHED` ya que complió su última ráfaga de CPU, pese a que en ese mismo instante cumplió su *quantum*.

Su output en el archivo es:

```
PROCESS1,4,3,21,0,8
```

El proceso fue elegido en:

$$t = 0, 4, 11, 17$$

El proceso fue interrumpido en:

$$t = 4, 15, 21$$

El *turnaround time* fue:

$$21 - 0 = 21$$

El *response time* fue:

$$0 - 0 = 0$$

El *waiting time* fue:

$$0 + 8 = 8$$

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:

- Si el *quantum* de un proceso se agota al mismo tiempo que su ráfaga o *burst*, se considerará como un **interrupción**, iniciando inmediatamente el tiempo B_i. Es decir, su estado debe pasar directamente a `WAITING` y no de forma intermedia a `READY`. Si el tiempo B_i no existe (es decir, se ejecutó la última ráfaga), se sigue considerando como interrupción, solo que su estado debe pasar directamente a `FINISHED`, en vez de `WAITING`.
- Un proceso **puede** llegar en tiempo $t = 0$. Su programa no se puede caer si llega un archivo *input* con ese valor.
- Al momento de calcular el *quantum* del proceso a ejecutar, se deben contabilizar los procesos que llegan en esa unidad de tiempo.
- Al terminar un proceso se debe reflejar inmediatamente una resta en el n_i de su respectiva fábrica.
- Puede que un proceso pase de RUNNING a READY y luego a RUNNING. En este caso no se contabiliza que estuvo un momento READY, ya que no alcanzó a estar una unidad de tiempo completa en READY.[6]

Formalidades

A cada alumno se le asignó un nombre de usuario y una contraseña para el servidor del curso[7]. 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 Jueves 16-Septiembre-2021, 23:59.

- **NO debe incluir archivos binarios**. En caso contrario, tendrá un descuento de 0.5 puntos en su nota final.

[6] Esto ocurre cuando solo hay un proceso ejecutando, o cuando todos los otros procesos estan WAITING.
[7] `iic2333.ing.puc.cl`

- Su tarea deberá compilar utilizando el comando `make` en la carpeta `T2`, y generar un ejecutable llamado `scheduler` en esa misma carpeta. Si su programa **no tiene** un `Makefile`, tendrá un descuento de 1 punto en su nota final.
- Es muy importante que su tarea corra dentro del servidor del curso. Si ésta **no compila** o **no funciona** (*segmentation fault*), obtendrán la nota mínima, teniendo como base 1 punto menos en el caso 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 (ojo que T es mayúscula).

Evaluación

Simulación: La parte programada de esta tarea sigue la siguiente distribución de puntaje:

- **5.0 pts.** Simulación correcta.
  - **0.5 pts.** Estructura y representación de procesos.
  - **1.0 pts.** Estructura y manejo de colas.
  - **3.5 pts.** *Scheduler* RR (archivo de output correcto).
- **0.2 pts.** Lectura de `stdin`. Paso de argumentos. Construcción de `argc` y `argv`.
- **0.3 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.
- **0.5 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[8].

Preguntas

Cualquier duda preguntar a través del foro.

[8] Es decir, debe reportar 0 leaks y 0 errores para todo test.

Anexo

Explicación gráfica del siguiente ejemplo con *quantum* 4:

```
1
PROCESS1 0 1 3 5 6 4 2 4
```