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

IIC2333 — Sistemas Operativos y Redes — 2/2017

Tarea 3

Viernes 29-Septiembre-2017

Fecha de Entrega: Viernes 13-Octubre-2017 a las 23:59
Composición: grupos de n personas, donde $n \leq 2$

Parte I. Memoria Virtual - `mem_simulator`

Esta parte consiste en simular las etapas involucradas en paginación y manejo de memoria virtual con *swapping*, para operaciones de lectura. Se trabajará sobre una memoria virtual con direcciones de 16 bits, donde los primeros 8 representan el número de página, y los últimos representan el offset. Además, se trabajará con una tabla de páginas de un sólo nivel y con una TLB de 32 entradas. En relación a la memoria física, habrán 128 *frames* de 2^8 bytes cada uno. Inicialmente, la tabla de páginas y la TLB están vacías.

Funcionamiento

Para traducir una dirección lógica en una física, se intenta consultar el número de página en la TLB. Si hubo un TLB-*hit*, el número de *frame* es obtenido de la TLB. En caso de un TLB-*miss*, se debe consultar la tabla de páginas, de la cual es posible obtener el número de *frame* u obtener un *page-fault*.

En el caso de un *page-fault*, usted deberá implementar **paginación por demanda**. Para esto, el archivo `data.bin` actuará como "disco", por lo que ante un *page-fault* deberá leer el *frame* correspondiente en el archivo, y cargarlo en la memoria física. Una vez que haya cargado el *frame*, hay que actualizar la TLB y la tabla de páginas.

Tome en cuenta que el espacio de direcciones físicas es menor al de direcciones virtuales, por lo que habrá *swapping*. Es por ello que deberá implementar 2 políticas de reemplazo de páginas independientes. Implemente estas mismas políticas para actualizar la TLB.

- `LRU`: **Least Recently Used**
- `FIFO`: **First In - First Out**

Input

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

```
./mem_simulator <politica>
```

- `<politica>` corresponderá a `lru` o `fifo`

Su programa recibirá mediante `stdin` una secuencia de enteros que representan direcciones de memoria lógicas, desde el 0 hasta el 65535. Puede asumir que el archivo `data.bin` de 65536 bytes se encontrará en el mismo directorio del ejecutable. Por ejemplo, dos ejecuciones equivalentes podrían ser las siguientes:

```
./mem_simulator lru
    513
    65535
    0
    33
    Ctrl D
```

```
./mem_simulator lru < input.txt
```

Donde el archivo `input.txt` se vería de la siguiente forma:

```
513
65535
0
33
```

Output

- Para cada dirección lógica leída, deberá traducirla a su respectiva dirección física e imprimir en pantalla el valor del *byte* (sin signo) correspondiente.

Además, al finalizar el programa deberá imprimir en pantalla:

- El porcentaje de *page-faults*.
- El porcentaje de TLB-*hits*.
- Los contenidos finales de la tabla de páginas.
- Los contenidos finales de la TLB.

Al imprimir los contenidos finales de la tabla de páginas y la TLB, las entradas que no posean información se deben imprimir con dichos campos en blanco, como se puede observar en el formato de impresión en pantalla a continuación:

```
89
247
73
35
PORCENTAJE_PAGE_FAULTS = 75%
PORCENTAJE_TLB_HITS = 25%
TABLA DE PAGINAS
page_number        frame_number
0                  2
1
2                  0
...                ...
255                1
TLB
i        page_number        frame_number
0        2                  0
1        255                1
2        0                  2
3
...
31
```

En el ejemplo subido junto a la tarea se puede revisar en detalle el formato que debe seguir el output del programa.

Tome en cuenta

1. Por cada *page-fault* va a tener que leer el archivo `data.bin` y buscar una cierta posición. Se recomienda usar las funciones `fopen`, `fread`, `fseek` y `fclose`. **No debe mantener este archivo en memoria.**

2. No debe imprimir **nada más** que lo especificado anteriormente. Se penalizarán diferencias en el formato.

3. En esta simulación la TLB no ayuda a hacer más rápido encontrar una página (de hecho lo contrario).

Parte II. Scheduling de accesos - `disk_scheduler`

Esta parte consiste en simular el proceso por el cual el sistema operativo solicita datos del disco duro. Para este fin, se considerará la representación de un sector por medio de *CHS*, tal como fue visto en clases.

Funcionamiento

Para ordenar los accesos a disco, en primer lugar se obtienen tanto los accesos encolados (*queue*) como la posición donde el brazo inicia la búsqueda (*head*) y luego se determina el orden de los accesos, según el algoritmo de *scheduling* correspondiente.

Se pide calcular el orden de desplazamiento, la cantidad de desplazamientos entre cilindros, y el tiempo de desplazamiento, considerando que el tiempo de movimiento hacia cada cilindro buscado (*seek time*) toma τ msec y que en cada cambio de sentido del brazo hay un retardo de δ msec.

Input

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

```
./disk_scheduler <politica> <input_file>
```

- `<politica>` corresponderá a `fcfs`, `sstf`, `scan` o `c-look`.
- `<input>` corresponderá a la ruta a un archivo de texto que contendrá la ubicación del *head* en la primera línea, y un número arbitrario de accesos encolados en las siguientes líneas.

Una ejecución del programa podría ser la siguiente:

```
./disk_scheduler sstf input.txt
```

Donde un ejemplo de archivo `input.txt` es de la siguiente forma:

```
20
10
22
20
2
6
38
```

Siendo la cabeza del brazo (*head*) `20`, y la cola de accesos (*queue*) `10,22,20,2,6,38`.

Output

Al finalizar el programa deberá imprimir en pantalla:

- Orden de desplazamiento como una secuencia de accesos separados por coma.
- Cantidad de desplazamientos entre cilindros como un entero.
- Tiempo de desplazamiento como $X\tau + Y\delta$ msec. Para la impresión puede sustituir τ y δ por T y D, respectivamente.

Un ejemplo de *output*, correspondiente al *input* de ejemplo, es el siguiente:

```
20,22,10,6,2,38
58
58T+2D msec
```

Tome en cuenta

1. Puede asumir cuando sea necesario, que el brazo se está moviendo a los cilindros con mayor número.
2. Puede asumir también, cuando sea necesario, que los cilindros van desde el 0 al 255.
3. No debe imprimir **nada más** que lo especificado anteriormente. Se penalizarán diferencias en el formato.

README y Formalidades

Deberá incluir un archivo `README` que indique quiénes son los autores de la tarea (**con sus respectivos números de alumno**), 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. Se sugiere utilizar formato **markdown**.

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 `T3` en el directorio `Entregas` de su carpeta personal y subir su tarea a esa carpeta. En su carpeta `T3` **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 Viernes 13-Octubre-2017 a las 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 con sus respectivos números de alumno.
- La parte I de esta tarea debe estar en el directorio `Entregas/T3/memv`, y la parte II de esta tarea en el directorio `Entregas/T3/disk`. Cada una de las partes debe compilar utilizando el comando `make` en los directorios señalados anteriormente.

Evaluación

- 0.5 pts. Formalidades. Esto incluye cumplir las normas de la sección formalidades.
- 0.5 pts. Manejo de memoria. *valgrind* reporta en su código 0 *leaks* y 0 errores de memoria, considerando que los programas funcionen correctamente. **No es bonus.**
- 2.5 pts. Memoria virtual.
  - 1.5 pts. Modelación de TLB y tabla de páginas.
    - 0.75 pts. LRU.
    - 0.75 pts. FIFO.
  - 1.0 pts. Correctitud y estadísticas.
- 2.5 pts. Scheduling de accesos.
  - 0.5 pts. Modelación del scheduler.
  - 2.0 pts. Schedulers
    - 0.5 pts. FCFS.
    - 0.5 pts. SSTF.

- 0.5 pts. SCAN.
- 0.5 pts. C-LOOK.

El no respeto de las formalidades o un código extremadamente desordenado podría originar descuentos adicionales, lo mismo para errores de memoria significativos. 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.

Preguntas

Cualquier duda preguntar a través del foro.