



IIC2333 — Sistemas Operativos y Redes — 1/2018 Tarea 2

Jueves 12-Abril-2018

Fecha de Entrega: Jueves 26-Abril-2018 a las 23:59

Composición: grupos de n personas, donde $n \leq 2$

Esta tarea se compone de dos partes, independientes entre sí, donde deberán utilizar las *syscalls* de manejo de procesos e implementar una simulación basada en sus conocimientos de memoria virtual, paginación y TLB.

Parte I. Doer - doer

El objetivo de esta parte es construir un programa que sea capaz de ejecutar comandos recibidos como *input*, respetando ciertas restricciones del sistema. Este programa se llamará *doer* (*/ˈduː.ə/*), y deberá recibir como entrada un archivo de texto con una cantidad arbitraria m de tareas (una tarea por línea, $m \geq 1$) esto es, comandos a ejecutar con sus respectivos argumentos. También recibirá como parámetro la cantidad máxima de procesos n **concurrentes**¹ con los que puede contar para ejecutar dichas tareas.

doer debe ser capaz de asignar a cada proceso una tarea y. Si quedan tareas pendientes, debe esperar que un proceso termine antes de ejecutar la siguiente. En caso de un término fallido, debe reintentar la tarea, pero solo una vez.

Al finalizar la ejecución de las tareas asignadas a *doer*, se debe entregar una estadística de procesos ejecutados y, para cada proceso, reportar su *output* y *exit code*. La estadística global debe reportar la configuración de m y n para la ejecución, el tiempo total real de ejecución (tiempo paralelo), y el tiempo total de sistema de ejecución (tiempo secuencial)².

Input



El archivo de texto que será entregado como *input* tendrá en cada línea una tarea a realizar con sus respectivos argumentos. Un ejemplo de archivo de *input* es el siguiente:

```
ls -al
echo something
ps T
date "+DATE: %Y-%m-%d%nTIME: %H:%M:%S"
```

Ejecución

doer será ejecutado por línea de comandos con la siguiente instrucción:

```
./doer <file> <n>
```

Al presionar   se debe terminar la ejecución de *doer*, esto es, que el programa deje de asignar tareas a sus procesos. Sin embargo, debe imprimir las estadísticas hasta ese momento. Los procesos que estén corriendo deben dejar de hacerlo también.

¹No puede haber más de n procesos en la cola *ready* ejecutando estas instrucciones

²El tiempo total real o paralelo corresponde al delta de tiempo entre que comienza a ejecutarse el programa y termina. El tiempo total de sistema o secuencial corresponde al tiempo que hubiera tomado el programa si no existiese paralelización de ningún tipo proveída por el sistema, por lo tanto la suma de los tiempos de cada proceso. Se puede hacer cierto símil con la diferencia entre *real time* y *usr + sys time* de los procesos.

Tenga en cuenta

1. Deberá usar `fork`, `exec` y `wait` (no `threads`).
2. Deberá construir adecuadamente `argc` y `argv` para pasarlos a los comandos que deberá ejecutar.

Parte II. Memoria Virtual - `simulator`

Esta parte consiste en simular las etapas involucradas en paginación y manejo de memoria virtual con *swapping*, para operaciones de lectura.

Direccionamiento eficiente

Se trabajará sobre una memoria virtual con direcciones de 30 bits, con una tabla de páginas **de nivel variable**, con una TLB de 64 entradas y con 256 *frames* de 2^8 bytes cada uno. Además de lo anterior, por cada PTE (*Page Table Entry*) que apunte a un *frame*, se incluirán 3 bits adicionales de uso personal³. En la dirección virtual, los 8 bits menos significativos corresponderán al *offset* (lo que a su vez determina el tamaño de las páginas y los *frames*), mientras que los 22 restantes serán utilizados para el direccionamiento. La cantidad de niveles a utilizar en el esquema de direccionamiento será un **parámetro** a ingresar para la ejecución, cuyo valor puede ser desde 1 hasta 5 inclusive. El objetivo de esta parte será asignar a cada nivel del esquema de paginación una cantidad de bits que permita obtener **un menor uso de memoria en las tablas necesarias para traducir una dirección**. Dicho de otra forma, es la suma de los tamaños de una tabla en cada nivel.

Para entender mejor lo que se busca en esta parte, se verán dos ejemplos. Suponga que se está trabajando solo con dos niveles. Si se utilizan 18 bits para direccionar en el primer nivel y los 4 restantes para el segundo, se tiene el siguiente uso de espacio⁴:

- **Tabla de páginas primer nivel:** $2^{18} \times 4b = 2^{17} \times 2^3b = 2^{17}B$ (2^{18} entradas, 4 bits por entrada para acceder a la tabla de segundo nivel).
- **Tabla de páginas segundo nivel:** $2^4 \times 11b = 2^4 \times 1,375B$ (2^4 entradas, $8 + 3 = 11$ bit por entrada para encontrar el *frame*, incluyendo los bits de uso personal).
- **Total para traducir una dirección:** $2^{17}B + (2^4 \times 1,375B) = 128KB + 22B$ (1 tabla de páginas del primer nivel más 1 tabla de páginas del segundo nivel).

Ahora, suponga que utilizan 14 bits para direccionar en el primer nivel y los 8 restantes para el segundo:

- **Tabla de páginas primer nivel:** $2^{14} \times 8b = 2^{14}B$ (2^{14} entradas, 8 bits por entrada para acceder a la tabla de segundo nivel).
- **Tabla de páginas segundo nivel:** $2^8 \times 11b = 2^8 \times 1,375B$ (2^8 entradas, 11 bits por entrada para encontrar el *frame*, que incluyen los bits de uso personal).
- **Total para traducir una dirección:** $2^{14}B + (2^8 \times 1,375B) = 16KB + 352B$ (1 tabla de páginas de primer nivel más 1 tabla del segundo nivel).

La segunda elección de bits por nivel (14 y 8) lleva a un menor uso de memoria por parte de las tablas. El objetivo, entonces, es que su programa determine la cantidad de bits por nivel que debe utilizar para minimizar el espacio de memoria utilizado, además de indicar dicho valor obtenido.

³Por ejemplo, uno de esos bits puede corresponder al bit de validez. Si no los usa los puede dejar en cero.

⁴ $B = \text{byte}$, $b = \text{bit}$, $1b = 0,125B$.

Simulación de paginación

Después de haber obtenido la cantidad de bits por nivel para direccionar, se hará uso de dichos parámetros para llevar a cabo una simulación de la paginación de un proceso. En particular, se recibirá un archivo con una secuencia de direcciones virtuales a las que se busca acceder en ese orden y, para cada una de ellas, se seguirá el siguiente flujo para la obtención de la dirección física:

1. Se busca la **dirección completa**⁵ en la TLB.
2. Si hubo un TLB-*hit*, el número de *frame* es obtenido de la TLB.
3. En caso de un TLB-*miss*, se debe llegar a la tabla de páginas asociada a la dirección ingresada (dependiendo del número de niveles), de la cual es posible obtener el número de *frame* u obtener un *page-fault*.
4. 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 correspondiente.

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 una política de reemplazo de páginas LRU. Implemente esta misma política para actualizar la TLB.

Input y Output

En primer lugar, su programa será ejecutado con dos parámetros: *n* que indicará el número de niveles con el que trabajará en la simulación e *input.txt*, que corresponderá a las direcciones virtuales que busca acceder el proceso a simular. Antes de comenzar la simulación, imprimirá en consola el número de bits utilizado por nivel, además del espacio utilizado por sus tablas en bytes. Por ejemplo:

```
> ./simulator 2 input.txt
BITS NIVEL 1: 14
BITS NIVEL 2: 8
ESPACIO UTILIZADO: 5.515625 MB
```

Con respecto a *input.txt*, este corresponderá a una secuencia de enteros que representan direcciones de memoria lógicas, desde el 0 hasta el 1073741823. Puede asumir que el archivo `data.bin` de 1073741824 bytes se encontrará en el mismo directorio del ejecutable. El archivo `input.txt` se podría ver de la siguiente forma:

```
513
1073741820
6789
10105
```

⁵Todos los bits de dirección virtual que no incluyan al *offset*.

Por cada dirección recibida, deberá hacer lo especificado en la sección 'Simulación de paginación' e imprimir en pantalla tanto la traducción a dirección física como el valor del *byte* (sin signo) que haya sido obtenido del disco. Por ejemplo:

```
-513-
DIRECCIÓN FÍSICA: 2561
CONTENIDO: 1
-1073741820-
DIRECCIÓN FÍSICA: 22524
CONTENIDO: 125
...
```

Finalizada la simulación, usted imprimirá los siguientes resultados:

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

El formato de impresión en pantalla debe ser el siguiente:

```
PORCENTAJE_PAGE_FAULTS = 75%
PORCENTAJE_TLB_HITS = 25%
TLB
i          n1_number      n2_number      frame_number
0           2             123             0
1          255            1001            1
2           0             375             2
3
...
63
```

Notar que la impresión de la TLB variará con respecto al número de niveles que haya sido ingresado.

Tome en cuenta

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, sino que debe leerlo a medida que lo necesita. Si lo almacena en memoria, será penalizado.**

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 T2 en su carpeta personal y subir su tarea a esa carpeta. En su carpeta T2 **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 Jueves 26-Abril-2018 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/T2/door`, y la parte II de esta tarea en el directorio `Entregas/T2/mem`. Cada una de las partes debe compilar utilizando el comando `make` en los directorios señalados anteriormente.

Evaluación

- 0.25 pts. Formalidades. Esto incluye cumplir las normas de la sección formalidades.
- 2.75 pts. `door`
 - 0.25 pts. Lectura de `stdin`. Paso de argumentos. Construcción de `argc` y `argv`.
 - 1.5 pts. Ejecución de instrucciones de acuerdo a las restricciones
 - 1.0 pts. Estadísticas de ejecución
- 3 pts. `memory simulator`
 - 0.5 pts. Cálculo valores óptimos por nivel.
 - 0.5 pts. Construcción de los niveles.
 - 0.5 pts. Funcionamiento del TLB
 - 0.5 pts. Ejecución de la simulación. Traducción de direcciones.
 - 0.5 pts. Algoritmo de reemplazo
 - 0.5 pts. Estadísticas y *output*

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 los programas funcionen 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](#).