Tecnológico de Monterrey

Sistemas Operativos
TC2008-Verano 2020
Proyecto final

Objetivo: Ejercicio de programación para mostrar su comprensión de algunos de los algoritmos de administración de memoria vistos en clase

Procedimiento: Desarrollar un programa en cualquier lenguaje con las siguientes especificaciones:

1. El programa deberá simular un manejador de memoria virtual utilizando paginación ***parecido*** al visto en clase, y medir su rendimiento. Se utilizarán las estrategias de reemplazo FIFO y LRU, para que comparen el rendimiento de ambas estrategias. Asuman que la computadora tiene un solo procesador; una memoria real de 2048 bytes, y que las páginas son de 16 bytes.

2. Para la simulación, su programa contará con las siguientes estructuras de datos:
   a. Un área de memoria que simulará la parte de la memoria real de una computadora reservada a marcos de página. *Por ejemplo*, puede ser un vector llamado **M** de 2048 elementos, cada uno de 1 byte. La primera posición del arreglo, y por tanto del primer marco de página, representa la dirección 0 de la memoria real. La última dirección sería la 2047. Inicialmente la memoria está vacía. Otro tipo de estructura de datos para simular los marcos de página es posible.

   b. Una segunda área de memoria contigua, que simulará el área de disco reservada para el swapping de páginas. Podría ser por ejemplo un vector llamado S de 4096 elementos, cada uno de 1 byte.

   c. Otras estructuras de datos que encuentren Uds. convenientes o indispensables para lograr la simulación y que se encuentran en áreas de memoria distintas de las áreas M y S.

3. INPUT: Simulará las peticiones que hace el sistema operativo a su propio algoritmo manejador de memoria virtual, tales como asignar o liberar espacios de memoria o acceder direcciones virtuales para obtener la dirección en memoria real.

Los datos de entrada al programa consistirán de un archivo en disco con varias líneas que representan estas solicitudes Cada línea del archivo puede tener uno de los siguientes formatos. Para cada formato, se indica cuál sería *parte* del PROCESS y el OUTPUT requeridos. Infieran Uds. qué estructuras de datos van a necesitar para cumplir con estas especificaciones.

a. P n p (cargar un proceso)
   Se trata de una solicitud de "n" bytes para cargar un proceso a la memoria.

   "p" es un número entero arbitrario que indica el identificador de proceso.
   Ejemplo:

   P 534 5834 (Se solicita asignar 534 bytes al proceso 5834)

   NOTA En esta versión de memoria virtual con paginación, un proceso se carga completito a memoria real en una sola solicitud al manejador de memoria, incluyendo páginas que contienen el código del proceso y otras para su área de datos. La longitud máxima de un proceso es de 2048 bytes. *Observen que esto es diferente a la forma como se revisó en clase, donde aprendimos que los procesos se pueden cargar a memoria por partes en base a un esquema por demanda.*

   PROCESS: El programa deberá asignar los marcos de página necesarios, *no necesariamente contiguos, pues podría no encontrarse en la memoria real un área contigua de la longitud requerida.* En el ejemplo anterior, 534 bytes requieren 34 marcos [(534/16) = 33.375)]. Si no hay marcos de página vacíos suficientes en **M**, se tendrán que swappear-out las páginas contenidas en algunos marcos ya ocupados, de acuerdo con el algoritmo de reemplazo correspondiente. *Las páginas a reemplazar podrían pertenecer a cualquier proceso; es decir, se utiliza un enfoque de "reemplazo global"*. Si hay swapping, su programa deberá registrar dónde queda cada página en el área **S** de swapping, pues posteriormente puede ser necesario volver a cargar esas páginas. Además puede ser necesario registrar el "*timestamp"* u otra información como podría ser el *"bit de referencia"* y *"bit de modificación"* al momento que se asigna o se utiliza en memoria cada página del proceso, en concordancia con lo que requiera la estrategia de reemplazo que se esté utilizando.

Para propósitos de la simulación, se supone que cargar una página a memoria inicialmente toma 1 segundo. Ese mismo tiempo toman las operaciones de swap-in y swap-out por cada página que se mueve de o al área de swapping. Accesar o modificar una dirección de una página ya cargada en memoria, toma sólo 0.1 segundos. La simulación comienza en tiempo 0. Para el ejemplo indicado arriba, *si **P 534 5834** fuera el primer comando de la simulación*, el timestamp una vez terminada la operación de carga sería igual a 34.

Hint. Para facilitar su debugging, podrían registrar en las áreas **M** y **S** o en alguna otra estructura de datos el número del proceso y número de página de ese proceso que está cargado en cada marco. Esto no es indispensable.

OUTPUT: Imprimir el comando de INPUT y la lista de marcos de página asignados. Si fue necesario hacer swapping, una lista de los marcos de memoria real que fue necesario swappear para hacer hueco; para cada uno, indicar, acerca de la página que contenía el marco, a qué proceso y número de página del proceso pertenecía, y donde quedó esa página en el área de swapping. Se muestran ejemplos del output solicitado más adelante.

b. A d p m

Es una solicitud para accesar la dirección virtual “d” del proceso “p”. Si “m” es 0, la dirección correspondiente sólo se lee; si es 1, también se modifica.“d” puede tener un valor desde cero hasta la dirección virtual máxima del proceso.
Ejemplo:

A 17 5 0 (accesar para lectura la dirección virtual 17 del proceso 5)

PROCESS: Localizar dónde se encuentra la página que contiene esa dirección. En el ejemplo, se trata de la página 1 del proceso, pues la longitud de página es 16 y las páginas de un proceso se numeran desde cero. Esa página 1 puede estar en algún lugar de la memoria real, y no necesariamente junto a la página 0; o bien puede estar en el área de swapping pues ya la habíamos “corrido” de su marco en memoria real

en alguna operación anterior. Si ese es el caso, ha ocurrido un "*page fault"*, y es necesario cargarla de nuevo (swapping-in) en alguna parte de la memoria real, para lo cual podría ser necesario swappear-out algún otro marco de página si es que todos están ocupados (si no hay marcos libres). Nuevamente, podría ser necesario registrar en alguna parte el *"timestamp"* en que ocurre el acceso o alguna otra información pertinente.

OUTPUT: El comando de INPUT; la dirección en memoria real correspondiente a la dirección virtual "d" del proceso "p". Si la página necesaria se encontraba en el área de swapping, indicar donde estaba en el arreglo S, y donde quedó en memoria real; si fué necesario para cargarla swappear-out alguna otra página de algún proceso de acuerdo al algoritmo de reemplazo, indicar ésto de manera similar a como se describe en el inciso a).

c. L p
Liberar las páginas del proceso "p".

PROCESS: Se liberan todas las páginas del proceso "p", tanto las que estaban en memoria real como aquellas que se encontraban en el área de swapping, quedando varios marcos de página o pedazos del área de swapping vacíos y disponibles para otras operaciones. *Toma 0.1 segundos liberar cada página de cualquiera de las dos áreas.*

OUTPUT: El comando de INPUT; lista de marcos de página que se liberaron.

d. C comentario. Es una línea de comentarios. Desplegarla

d. F
Fin. Es siempre la última línea de un conjunto de solicitudes; **pero pueden seguir otras líneas más para otro conjunto de solicitudes.**

PROCESS: lo necesario para el OUTPUT.

OUTPUT: El comando de INPUT; reporte de estadísticas, que incluye:

- *turnaround time* de cada proceso que se consideró, desde que se

comienza a cargar un proceso (P) hasta que se terminan de liberar todas sus páginas (L). Puedes obtenerlo por medio de una diferencia de timestamps.

- *turnaround promedio*.

- *número de page faults por proceso*. Recuerda que un page fault ocurre únicamente cuando un marco de página necesario no se encuentra en memoria real. → *En esta versión del manejador de memoria virtual, el comando P (cargar un proceso completo a memoria real) no produce page faults. Los habría si es necesario reemplazar páginas de memoria real para hacerle espacio al proecso.* ←

- *número total de operaciones de swap-out y de swap-in* que fueron necesarias por cualquier motivo

e. E
Exit. Última línea del archivo.

PROCESS: se termina el programa.

OUTPUT: El comando de INPUT y mensaje de despedida...

4. Los algoritmos de reemplazo que vamos a utilizar para el proyecto para cada uno de los equipos serán FIFO (First In – Fisrt Out) y LRU (Least Recently Used) En caso de empate entre varias páginas que según su algoritmo pueden reemplazarse, el desempate es por FIFO.

   El cargar una página a memoria como resultado del comando P, no genera un uso o modificación de la página. Una página solo se usa o modifica como resultado del comando A.

   Un page fault ocurre al tener que enviar una página de M hacia S, como resultado de comandos P o A

   Cualquier equipo puede decidir utilizar un algoritmo de reemplazo diferente a los anteriores (nuevo), previa declaración por parte de ustedes y aprobación de sus instructores del curso.

5. El archivo de entrada que utilizarán todos los equipos será dado a conocer dos días antes de la fecha de entrega. Podrá ser el mismo para todas las estrategias de reemplazo, o diferente para alguna(s). Ya para entonces deberá estar ampliamente probado su programa, de tal forma que no tenga ningún problema al ejecutar este archivo de entrada "sorpresa". Este archivo podrá incluir datos erróneos, como procesos demasiado grandes, comandos inválidos ETC. Es responsabilidad de cualquier programa que hagan en su vida el verificar que sus datos de entrada sean correctos.

6. Lenguaje. El que gusten. Si conocen Python o Ruby o algún otro lenguaje igualmente poderoso, se les puede simplificar la vida. Si no los conocen puede quedarles un tiempo muy corto para aprenderlos y además hacer, probar y documentar el programa, así pueden utilizar algún lenguaje que ya conozcan.

7. Entregables: 3 de Agosto de 2020 a las 11:59 horas:

    a. Lenguaje y versión del lenguaje que utilizaron. Número total de líneas y número de líneas sin los comentarios. Esto es sólo para propósitos informativos y no afecta para nada su calificación.

    b. El output especificado.

    c. El listado del programa, que deberá utilizar programación modular. Ampliamente documentado mediante comentarios en el mismo código, como sigue. Adapten lo siguiente a la nomenclatura usada por el lenguaje que decidan utilizar.

    d. Los autores del programa

    e. Breve descripción sobre qué hace el programa.

    f. Una tabla comparativa resumen que muestre claramente el comportamiento de la corrida de simulación con cada estrategia de reemplazo, comparando el rendimiento obtenido con cada una de ellas, para el archivo de datos proporcionado por sus instructores del curso.

g. El significado de cada variable global o local. Nombres de variable muy claros; sin embargo, el comentario sobre el significado preciso de la variable puede ser necesario para no tener que deducirlo a partir de la forma como se utiliza la variable.

h. La función de cada función, rutina, gem, método o como se llame, así como su estado de entrada y estado de salida. Nombres igualmente muy claros. El estado de entrada incluye nombres y valores de variables globales y argumentos de entrada a la función. El estado de salida, lo mismo al terminar la función. Siéntete *función* y pregúntate: ¿qué es lo que me llega, tanto de argumentos como de la situación global de lo que se está haciendo, incluyendo tal vez parte de su historia? ¿cómo lo dejo?

i. Comentarios de bloques de código que indican qué hace el bloque, si es necesario. Lo mismo para ciertas líneas individuales, si es necesario.

   Prueba de fuego acerca de la estructura y claridad de su programa: *la prueba de la bicicleta*. No es necesario pensar mucho para darse cuenta de cómo funciona una bici. Basta mirarla. Lo mismo debe ocurrir al echarle una mirada a su programa --cualquier programa que hagan en su vida. Un programa es una obra literaria; debe estar hecho para leerse con facilidad y aprender, no es solo un conjunto de instrucciones para una máquina. Incluso hubo hace unos años un influyente movimiento llamado [literate programming].

   En una escala de 1 a 100, un programa que funciona bien vale sólo 30. Uno que funciona ante cualquier conjunto posible de datos y genera mensajes entendibles si algo anda mal, 50. Uno que fácilmente se puede comprender cómo funciona s*in recurrir para nada a los programadores originales*, vale 80. Uno que además está tan bien documentado y modularizado que es posible hacerle rápidamente los cambios más probables a futuro, vale 100. Los mejores programas “open source” son de este último tipo, pues precisamente la idea es que cualquier persona pueda entenderlos rápidamente y contribuir a mejorarlos.

   Su programa debe ser robusto y defenderse ante cualquier error, por ejemplo de input incorrecto, procesos más grandes que lo permitido, direcciones fuera del tamaño del proceso, área de swapping agotada ETC. No son todos… Les sugerimos probar su programa con input aleatorio…

Para este proyecto en particular, *claridad y robustez mata eficiencia*. Sin exagerar...

8. Ejemplo de input y output con el algoritmo de reemplazo: “Reemplaza una página contenida en un marco aleatorio”. No muy efectivo pero muy simple. Input en bold. Los //comentarios no son parte de su output. Todo lo demás sí.

9. **C comenzamos…**
   C comenzamos…

   P 2048 1 ignora cualquier comentario en la misma línea del comando, como éste.
   P 2048 1
   Asignar 2048 bytes al proceso 1
   Se asignaron los marcos de página 0-127 al proceso 1
   //¡se llenó la memoria!
   // como es el primer proceso, se cargó en memoria contigua

   A 1 1 0
   A 1 1 0
   Obtener la dirección real correspondiente a la dirección virtual 1 del proceso 1
   Dirección virtual: 1. Dirección real: 1
   // el proceso 1 comienza a ejecutar...

   A 33 1 1
   A 33 1 1
   Obtener la dirección real correspondiente a la dirección virtual 33 del proceso 1
   y modificar dicha dirección
   Página 2 del proceso 1 modificada.
   Dirección virtual: 33. Dirección real: 33

   P 32 2
   P 32 2
   Asignar 32 bytes al proceso 2
   página 5 del proceso 1 swappeada al marco 0 del área de swapping
   página 78 del proceso 1 swappeada al marco 1 del área de swapping
   // para hacer hueco y poder cargar el proceso 2 pues no había ningún marco libre.
   // 5 y 78 son dos números aleatorios de acuerdo a algoritmo de reemplazo
   // como el proceso 1 estaba en memoria contigua
   // empezando desde 0, los marcos de página obtenidos aleatoriamente coinciden
   // con los números de página del proceso
   Se asignaron los marcos de página [5, 78] al proceso 2

A 15 2 0
A 15 2 0
Obtener la dirección real correspondiente a la dirección virtual 15 del proceso 2
Dirección virtual:15. Dirección real: 95
// es la página 0 del proceso 2, que se encuentra en el marco 5
// Este empieza en la dirección 16*5 = 80.

A 82 1 0
A 82 1 0
Obtener la dirección real correspondiente a la dirección virtual 82 del proceso 1
Página 80 del proceso 1 swappeada al marco 3 del área de swapping.
Se localizó la página 5 del proceso 1 que estaba en la posición 0 de swapping
y se cargó al marco 80.
Dirección virtual: 82**.** Dirección real: 1282
// Como no había espacio en M,
// para cargar esa página 5 (82/16=5 y sobran 2) fué necesario antes sacar otra, y resultó
// aleatoriamente que estaba en el marco 80. Ese marco 80 ya
// estaba ocupado por otra página del mismo proceso 1 que fue necesario
// swappear al marco 3 de **S**
// Posición 0 del swapping queda libre y debe reutilizarse después si es necesario.

L 2
L 2
Liberar los marcos de página ocupados por el proceso 2
Se liberan los marcos de página de memoria real: [5, 78]

P 32 3522
P 32 3522
Asignar 32 bytes al proceso 35225
Se asignan los marcos [5, 78]
// pues justo los acabamos de librerar

L 1
L 1
Liberar los marcos de página ocupados por el proceso 1
Se liberan los marcos de memoria real: 0-4, 6-77 y 79-127
Se liberan los marcos [1,2] del área de swapping

F
F
Fin. Reporte de salida:
(etc… ver la instrucción F...)

(**OJO!!** → puede continuar una nueva serie de solicitudes. Se re-inicializa todo antes de proseguir. Algunas de las solicitudes que tendrá que manejar su programa incluyen errores intencionales para probar la robustez y resiliencia de su programa y la claridad de sus mensajes de error).

10. Rúbrica de evaluación:

    La simulación contempla contrastar 2 políticas de reemplazo.

    a. Verificaremos el documento y el código
    b. Evaluaremos, además de los criterios de programación ya explicados en este documento; distintos escenarios utilizando nuestro archivo de datos:
    c. Si no funciona su simulación.
    d. Si funciona sólo para 1 de las 2 políticas de reemplazo.
    e. Si funciona para ambas.
    f. Para los programas que funcionen correctamente al 100%, podrá haber un bono extra de 1 punto sobre los 5 puntos que vale el trabajo final; inapelable y totalmente a juicio de los instructores, según el grado de cumplimiento de los otros criterios previamente establecidos en este documento.