Sistemas Operativos 2019-2
Tarea 5. Administración de memoria (Simulación de intercambio)

Profesor : Salvador López Mendoza
Ayudante : Jorge Erick Rivera López

Facultad de Ciencias, UNAM

Fecha de entrega: 16 de Abril de 2019

El *sistema operativo* como ya se ha visto puede administrar los procesos, sin embargo, también puede *administrar la memoria*, un técnica utilizada en la administración de memoria es el **intercambio**, la cual consiste en que se otorga cierto tiempo de ejecución a un proceso, luego de ese tiempo su información en la RAM se muda al disco duro, permitiendo que otro proceso se le asigne un espacio de direcciones en la memoria principal para que en un cierto momento utilice el procesador, después de cierto tiempo, se trae del disco duro a algún proceso cuya información se haya guardado en él a la RAM, para que pueda continuar su ejecución donde se había quedado.

1. Problema

Resulta que para poder realizar el intercambio se necesita conocer que lugares de la RAM están o no ocupados, algunas técnicas conocidas son **mapa de bits** y **listas ligadas**.

El **mapa de bits** puede verse un arreglo de 1's y 0's, donde 1 significa un bloque de memoria ocupado y 0 un bloque de memoria no ocupado, el tamaño de bloque que se quiera representar con cada bit dependerá de la

elección del programador.

La **lista ligada** funciona así, una celda de la lista contiene un *estado*, *dirección base*, *longitud de memoria*, el *estado* puede ser **O**cupado o **H**ueco, la *dirección base* es donde inicia el **H**ueco o el bloque **O**cupado, si una celda de la lista tiene estado Ocupado, significa que en la RAM, hay un proceso ocupando espacio desde la *dirección base* hasta *la dirección base + la longitud de memoria - 1*, es importante notar que no puede haber dos celdas seguidas de Hueco.

2. Instrucciones

Para esta tarea tendrás que hacer tu propio mapa de bits y lista ligada (la técnica, no la estructura de datos).

Tu programa recibirá a través de línea de comandos, un nombre de archivo, dicho archivo contendrá lo siguiente:

1. Una secuencia de letras del alfabeto que indicará que está ocupado un bloque en la memoria, el símbolo *, indicará un hueco, para este caso se supondrá que cada letra o símbolo representará un bloque de la RAM de 32 bits (i.e. 4 bytes), habra un total de 256 caracteres (i.e. una RAM de capacidad de 1024 bytes) que representarán la memoria principal.

2. Un secuencia de caracteres que representará el espacio de direcciones de un proceso a colocar en RAM.

3. Una letra que indicará que proceso deberá salir de la RAM hacia el disco duro.

El programa leerá la secuencia de caracteres, luego se utilizarán las técnicas para determinar en dónde colocar el nuevo proceso, primero deberá de salir el proceso indicado y posteriormente entrar el nuevo proceso. Deberá de imprimirse en un archivo de texto la RAM cuando se ha ya sacado el proceso y cuando se haya introducido el nuevo proceso.

En caso de no haber espacio, no se modificará la RAM. Como notarás más adelante, cada espacio de direcciones en la RAM esta representado por una

secuencua de i letras del mismo símbolo, el kernel del sistema operativo estará representado cómo una cadena de k's, la cuál es inamovible.

2.1. Mapa de bits

El *mapa de bits* estará representado como un arreglo de 1's y 0's, deberás de buscar n *bits* necesarios para el nuevo proceso. Deberás buscar el primer lugar que tenga los bits necesarios, el tamaño del bloque representado por cada bit será de 64 bits (i. e. 8 bytes).

2.2. Listas ligadas

Para este punto podrás utilizar alguna implementación de lista genérica que encuentres en la red, dicha lista guardará un estructura que contendrá los campos antes descritos, recuerda que cada vez que se haga un cambio la lista ligada deberá modificarse, asigna el primer espacio que encuentres disponible.

2.3. Compactación de memoria

Deberás hacer tu función para compactar la memoria de tal forma que esta actue antes de la inserción de nuevos procesos, solo en caso de que no haya un bloque libre del tamaño requerido (deberá imprimirse la RAM resultante de la compactación).

3. Ejemplo de entrada y salida

Archivo de entrada (ejemplo pequeño): *El primer caracter es la dirección 0x0, el siguiente bloque de texto es el nuevo proceso y el último caracter es el proceso a sacar.*

```
kkkkyyyy
zzz***aa
***bbbbb
*****ccc

tttttttt
```

a

Un ejemplo de salida: *La primer tercia de bloques de letras son la RAM resultante de utilizar el Mapa de bits y el segundo par de bloques utilizando listas ligadas.*

```
usuario@maquina ~] ./intercambio entrada.txt
usuario@maquina ~] cat salida.txt
-- Mapa de bits --
kkkkyyyy
zzz*****
***bbbbb
*****ccc

kkkkyyyy
zzzbbbbb
ccc*****
********

kkkkyyyy
zzzbbbbb
ccc*tttt
tttt****

-- Listas Ligadas --
kkkkyyyy
zzz*****
***bbbbb
*****ccc

kkkkyyyy
zzztttttt
tttbbbbb
*****ccc
```

4. Puntos extras

1. (1 pt) Haz que en caso de que no haya espacio suficiente para insertar un elemento, se seleccione un proceso para quitarlo y así meter el nuevo proceso (deberá imprimirse en el archivo de salida la RAM resultante de esta acción y deberá omitirse la compactación, aun así la función de compactación deberá existir en el código).

2. (2 pts) Haz que al utilizar las listas ligadas en vez de utilizar *el primer ajuste*, se utilice *el siguiente ajuste* ó *el peor ajuste* ó *el mejor ajuste*.

Especifica en el README los puntos extras que realizaste.

5. Observación

El código podrás escribirlo en **Java**, **Python** ó **C** según te acomodes, deberás especificar en el readme la versión de *Java* o *Python* que utilizaste. Para el caso del uso del mapa de bits si en un bloque hay un espacio ocupado y otro no, se considerará por lo general ocupado (Ej. ttt***** $\rightarrow$ 1100). Recuerda también que esto es simplemente un ejemplo ilustrativo, la implementación real es más complicada.

6. Política de retraso

Por cada semana de retraso en la entrega se restará un punto.

7. Reglas importantes

El trabajo se realizará y entregará **a lo más en parejas**, deberás entregar tu código fuente en una carpeta nombrada **src** con el nombre de uno de los integrantes del equipo con su respectivo **README** que contendrá los nombres, además los números de cuenta, y comprimido en un archivo **ZIP** que deberá llevar el nombre de un integrante del equipo (el código fuente también deberá llevarlo). La entrega será a **más tardar a las 23:59**, cualquier copia detectada entre compañeros será considerada con calificación 0 en la tarea para los implicados.