Sistema Operativos Practica 4

Ejercicio 1

Explicar la diferencia entre los conceptos de fragmentación interna y externa.

Fragmentacion externa:

Cuando tenemos espacio libre disperso en la memoria que se desperdicia. Por ejemplo puedo tener 3 MB libres pero en particiones no contiguas de 1MB cada una. Si quiero darle 3MB a un proceso no puedo.

(Se resuelve separando la memoria en bloques fijos y a cada proceso se le asignan bloques, asi puede usar espacio fisico no contiguo)

Fragmentacion Interna:

Cuando en los bloques fijos que describi arriba se empieza a desperdiciar memoria tenemos fragmentacion interna. Por ejemplo un proceso podria necesitar 1 MB de memoria, pero si los bloques son de 8MB, al asignarle un bloque se van a estar desperdiciando 7MB

Ejercicio 2

Se tiene un sistema con 16 MB de RAM que utiliza particiones fijas para ubicar a los programas en memoria. Cuenta con particiones de 8 MB, 1 MB, 4 MB, 512 KB, 512 KB y 2 MB, en ese orden. Se desea ejecutar 5 programas de los siguientes tamaños: 500 KB, 6 MB, 3 MB, 20 KB, 4 MB, en ese orden

a)

formato que voy a usar:

Programa memoria → particion (desperdicio)

Best fit:

P1 500KB → particion 4 (desperdicia 12kb)

P2 6MB → particion 1 (desperdicia 2MB)

P3 3MB → particion 3 (desperdicia 1MB)

P4 20KB → particion 5 (desperdicia 492 kb)

P5 4MB → no me entra!

Desperdicio:

2MB + 1MB + 1MB + 12KB + 492KB + 2MB = 6,5MB(aprox)

b)

worst fit

P1 500KB → particion 1 (desperdicia 7,5 MB APROX)

P2 6MB → no me entra!

P3 3MB → particion 3 (desperdicia 1MB)

P4 20KB → particion 6 (desperdicia aprox 2 MB)

P5 4MB → no me entra!

desperdicio: aprox 12,5 MB

first fit

P1 500KB → particion 1 (desperdicia 7,5 MB APROX)

P2 6MB → no me entra!

P3 3MB → particion 3 (desperdicia 1MB)

P4 20KB → particion 2 (desperdicia aprox 1 MB)

P5 4MB → no me entra!

desperdicio: aprox 12,5 MB

c)

El mejor fue best fit (aprox 6,5 MB de desperdicio)

Ejercicio 3

Considerar un sistema con paginación por demanda donde los procesos están haciendo acceso secuencial a los datos de acuerdo a los siguientes patrones de uso:

- Uso de CPU: 20 %.
- El sistema hace thrashing.
- Uso del resto de los dispositivos de E/S: 10 %.

Como se ve, la CPU está siendo ampliamente desaprovechada. Para cada uno de los siguientes cambios en el sistema indicar si es probable o no que mejore la utilización de la CPU.

- Instalar una CPU más rápida.
- Instalar un disco de paginado más grande.
- Incrementar el grado de multiprogramación.
- Disminuir el grado de multiprogramación.
- Instalar más memoria principal.
- Instalar un disco más rápido.
- Incrementar el tamaño de página.
- Incrementar la velocidad del bus de E/S.

Instalar una CPU mas rapida:

No implicaria una mejora (al menos significativa). El gran problema es que se se esta utilizando tanto como deseamos la CPU, una mas rapida sera igual de desutilizada.

Instalar un disco de paginado más grande:

Agrandar el disco no deberia resolver nada, si esta haciend thrashing se esta quedando sin espacio en la memoria.

Incrementar el grado de multiprogramación.:

Si el sistema esta haciendo thrashing aumentar el switcheo entre tareas no es la mejor idea. Si lo hacemos vamos a profundizar aun mas la penalizacion del thrashing de estar swapeando paginas entre el disco y la memoria principal

Disminuir el grado de multiprogramación:

Al disminuir el switcheo entre programas, vamos a reducir la cantidad de veces que swapeamos paginas entre el disco y la memoria. De esta manera aumenta el uso de la CPU.

Instalar más memoria principal:

Con mas memoria va a haber espacio para tener mas programas. De esta manera se reduce el thrashing y se van a tener que swappear menos paginas entre memoria y disco. Lo que conlleva en un mayor uso de CPU

Instalar un disco más rápido:

Cuando hay thrashing el sistema esta constantemente swappeando paginas entre memoria y disco. Con un disco mas rapido, si bien el thrashing sigue estando, se mitiga el tiempo que tarda el swapping. Por esta razon aumenta el CPU

(Si el disco fuera infinitamente rapido, el swapeo seria instanteneo, asi que el thrashing no implicaria un problema).

Incrementar el tamaño de página.:

No se, consultar.

Incrementar la velocidad del bus de E/S.:

Creo que no, el bus de E/S no esta saturado. El cuello de botella en este caso particular es que el disco no escribe suficientemente rapido las paginas, sino saturaria el bus.

Ejercicio 4

¿Bajo qué circunstancias se produce un page fault? ¿Cuáles son las acciones que realiza el sistema operativo para resolver la situación?

Se produce un page fault cuando tratamos de leer una pagina que no tenemos en la memoria.

El sistema operativo llama a la rutina de atencion del page fault. Busca la pagina que tiene que traer a la memoria (y se fija que el proceso tenga permiso). Esta pagina se copia en algun frame que este libre y en caso de que no haya ninguno lo libera con el algoritmo que use.

Ademas si la pagina que desalojamos tiene el bit dirty prendido, antes hay que bajarla a disco porque fue modificada. Despues de hacer eso se retoma la ejecucion.

Ejercicio 5

Considerar la siguiente secuencia de referencias a páginas:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

¿Cuántos fallos de página se producirán con los siguientes algoritmos de reemplazo, suponiendo que se tienen 1, 2, 3, 4, 5, 6 o 7 frames? Al comenzar todos los frames se encuentran vacíos, por lo que la primer referencia a una página siempre genera fallo de página.

a) Con reemplazo LRU.

completar

b) Con reemplazo FIFO.

completar

c) Con reemplazo Second chance

completar

Ejercicio 8

Une estudiante dijo: "Meh, los algoritmos básicos de reemplazo de página son idénticos, salvo por el atributo utilizado para seleccionar la página a ser reemplazada".

a) ¿Qué atributo utiliza el algoritmo FIFO? ¿Y el algoritmo LRU? ¿Y Second Chance?

FIFO: Cuando fue cargada (se va la de fecha mas vieja) **LRU**: Cuando fue referenciada por ultima vez (se va la de fecha mas vieja) **Second chance**: Igual que FIFO pero ademas mira el bit R para decidir si le da otra oportunidad

b) Pensar el algoritmo genérico para estos algoritmos de reemplazo de páginas.

```
page_t desalojo(){
    while(1){
        page_t pagina_a_desalojoar = buscar_pagina_mas_vieja(atributo);
        //asumo que el bit R siempre esta implementado pero que por ejemplo en FIFO no se us
        if(pagina_a_desalojoar.R == 1){
            pagina_a_desalojar.R = 0;
            actualizar(pagina_a_desalojar)//si tenia S-C la mandamos al final de la cola/act
        }else{
            mmu.desalojar(pagina_a_desalojar);
            return pagina_a_desalojar;
        }
    }
}
```

Ejercicio 9

Una computadora tiene cuatro marcos de página. El tiempo de carga, tiempo de último acceso, y el bit R (referenciado) para cada página están como se muestra a continuación:

a) ¿Qué página reemplazará el algoritmo FIFO?

Reemplaza la pagina 3, es la de tiempo de carga mas viejo.

b) ¿Qué página reemplazará el algoritmo LRU?

Reemplaza la pagina 1, es la de tiempo de acceso mas viejo.

c) ¿Qué página reemplazará el algoritmo Second Chance?

Reemplaza la pagina 2, es la de tiempo de carga mas viejo sin SC.

Ejercicio 10

Se tiene la siguiente matriz:

int A[][] = new int[100][100];

donde A[0][0] está cargado en la posición 200, en un sistema de memoria paginada con páginas de tamaño 200. Un proceso de manipulación de matrices se encuentra cargado en la primer página, de la posición 0 a 199, por lo que todo fetch de instrucciones es a la misma página. Si se tienen sólo 3 frames de páginas, ¿Cuántos fallos de página serán generados por los siguientes ciclos, utilizando el algoritmo de reemplazo LRU? Suponer que el programa se encuentra en el primer frame, y los otros dos están vacíos.

a)

```
for (int j = 0; j < 100; j++)
    for (int i = 0; i < 100; i++)
        A[i][j] = 0;
```

Primero unas observaciones:

1. El proceso ocupa solo una pagina.
2. A[0][0] ocuapa 1 en la memoria, es decir que toda la matriz ocupa 10000 $\Rightarrow \frac{10000}{200} = 50$ paginas.
3. como es LRU no vamos a desalojar nunca a la pagina que contiene al programa, va a estar todo el tiempo accediendo al codigo.

El programa recorre toda la matriz en orden y pone 0. Si este ya esta cargado fetchear sus instrucciones no produce page faults. Cuando escribimosi el primer A[i][j] (j=0,i=0) tenemos un page fault y cargamos su pagina que tiene a ese y las siguientes 199 posiciones:

A[0][0],A[0][1],...,A[0][99], A[1][0],... A[1][99]. Es decir dos filas.

En la siguiente iteracion nos topamos con A[1][0], que lo tenemos cargado.

Despues llega A[2][0], de vuelta tenemos un page fault pero vamos a cargar al A[3][0].

Se ve que el comportamiento es que despues de un page fault se trae el siguiente elemento que necesita y se salva del siguiente. Hace un page fault cada dos elementos.

Page fault totales: $\frac{10000}{2} = 5000$

b)

```
for (int i = 0; i < 100; i++)
    for (int j = 0; j < 100; j++)
        A[i][j] = 0;
```

Ahora itera al reves.

1. A[0][0] → page-fault, carga A[0][0],A[0][1],. . . ,A[0][99], A[1][0],. . . A[1][99].
2. A[0][1] → hit.

. . .

100. A[0][99] → hit

101. A[1][0] → hit

. . .

200. A[1][99] → hit

201. A[2][0] → page-fault, carga las dos siguientes filas.

Este ciclo se repite. Se ve que por cada 2 filas o 200 entradas hace un unico page fault.

Page fault totales: $\frac{10000}{200} = 50$

Ejercicio 11

Dado un sistema que no realiza copy on write, ¿cómo le agregaría esa funcionalidad? Considerar:

- Llamadas al sistema a modificar.
- Cambios de hardware.
- Cambios en el manejo de segmentos y páginas.

Cambios de hardware.

Las paginas deben saber si mas de un proceso las leen, para eso se puede implementar un bit "S" que indica si mas de un proceso la esta leyendo. Ademas un contador para ver cuantos procesos referencian a la pagina.

Cambios de manejo de segmento y paginas

Las paginas que tengan el bit S prendido deben marcarse como read-only para que nadie pueda escribir, solo se van a poder escribir las que tengan una unica referencia.

Llamadas al sistema a modificar

Habria que modificar la rutina de atencion del page fault, para que cuando un proceso intente de escribir una pagina con el bit S prendido suceda lo siguiente:

1. Restarle 1 al contador de procesos que referencian la pagina
2. Se cree una copia de la pagina que el proceso este intenando de escribrir. Ahora con el bit S apagado, permiso de escritura y con el contador en 1.
3. Si el contador original quedo en 1, sacarle el bit S a la pagina original y otorgarle permisos de escritura.
4. Se devuelva el control al proceso que se interrumpio para que escriba en la pagina nueva que se le asigno

Logicamente cada vez que se crea un proceso nuevo, no va a copiar las paginas del padre sino que va a referenciar a las mismas. Esas paginas van a tener S prendido y seran read-only (toda la idea de copy on write es reusar esas paginas siempre que se pueda)

Ejercicio 12

Se tienen dos sistemas embebidos:

A: Hace procesamiento secuencial de archivos. Los bloques se leen, se procesan y se escriben.

B: Medidor de clima. Hay un proceso principal que detecta fenómenos meteorológicos (lluvia, vientos, granizo, sol intenso) y lanza programas específicos para hacer mediciones apropiadas. El clima puede cambiar abruptamente y cuando aparece el fenómeno nuevo se lo debe medir de inmediato.

Indicar cuál de las siguientes políticas de reemplazo de páginas es más apropiada para cada uno. Justificar.

- Bajar la página más recientemente usada.
- LRU.
- Segunda oportunidad + páginas estáticas.

Sistema A

Va a mirar en orden los archivos, puedo pensar que primero va a usar las paginas de A_1 despues no las va a mirar mas (al menos en su mayoria) y va a pasar las de A_2 y asi sucesivamente.

Idealmente querria mantener en los pageframe a las paginas del archivo que este manipulando en ese momento, para eso lo mas adecuado pareceria ser LRU.

Cuando este manipulando un A_i y tenga que desalojar una pagina, cualquier pagina de A_{i-k} (las paginas de archivos anteriores), va a tener un tiempo de acceso mas viejo que las que se hayan cargado de A_i. De esta manera logro ir desalojando las de archivos viejos antes de reemplazar a las del actual.

Sistema B

Para este sistema puede ser util, Segunda oportunidad + páginas estáticas.

Como hay un proceso principal que siempre esta corriendo seria interesante tenerlo en una pagina/paginas estatica para que nunca se desaloje.

El resto de los programas pueden ser necesitados en cualquier momento pero seguramente hay algunos que se ejecutan mucho mas que otros. Por eso es util la segunda oportunidad, los procesos mas comunes tendrian segunda oportunidad mientras los mas “raros” probablemente no, asi que posiblemente si tenemos que hacer un desalojo no se elija a uno frecuente si es que hay uno mas “raro” para desalojar.

Mas aun si se pudiera estudiar que fenomenos son super comunes (capaz en la meteorologia se mide de manera muy frecuente la intensidad del sol o viento), a estos se les podria asignar paginas estaticas para mejorar aun mas la performance.

Ojo que este ultimo detalle va a depender de cuanta memoria tengamos, si nos vamos de mambo agregando paginas estaticas nos puede quedar poco espacio para el resto de los fenomenos generando comportamientos de desalojo indeseables.

Ejercicio 13

Suponer que se tiene un sistema con 2 MB de RAM y se desea ejecutar un programa de 4 MB ubicado en un disco de 200 GB.

a)

Explicar cómo funciona el mecanismo de paginación que permite ejecutar un programa más grande que la memoria física disponible.

Si bien todo el programa no entra en memoria lo podemos correr. Como?

Se carga en la memoria lo que se pueda del programa (2MB). A medida que avanza la ejecucion, cada vez que hay que ejecutar algo que no este en memoria ocurre un PF.

Se desaloja alguna página y se reemplaza por la que contiene lo que se necesite para ejecutar. (las pagina que se desalojan se bajan a disco. Se van swapeando a medida de lo que se necesita.)

b)

Si el tamaño de frame es de 4 KB y suponiendo que el programa tarde o temprano ejecuta todo su código. ¿Cuántos fallos de página se producirán como mínimo?

El caso con menos PF seria un codigo sin salto, se ejecuta de la linea 0 a la n secuencialmente y se van trayendo las paginas necesarias. En 2 MB tenemos 2048 KB, es decir 512 páginas. El código ocupa 1024 páginas.

Si inicialmente cargo las primeras 512 páginas de la memoria con las 512 páginas del programa, me ahorro 512 PF.

(notar que este caso es recontra artificial solo se usa la memoria para **leer** las lineas de codigo).

c)

¿Bajo qué contexto tiene sentido que varios procesos compartan páginas? Indique por lo menos 2 situaciones y justifique.

- i:

Si por ejemplo dos procesos corren el mismo código podrían compartir esa página de **lectura** para no tener que duplicar la info.

- ii:

Un proceso padre y un hijo idéntico. Capaz el hijo usa variables del padre, mientras nadie escriba esas páginas la data en ellas es válida. (Recién cuando uno la escriba es necesario realizar una copia)