Sistemas Operativos
Pauta Interrogación 3

2 horas

Junio de 1999

Pregunta 1

Escriba el código de la función `PageFault` del sistema operativo, que se invoca automáticamente cuando el proceso referenció una página que no está en memoria. Suponga que usted es el Sistema Operativo, y que recibe los argumentos:

```
PageFault(PAGETABLE *pagetable, int page, int wss)
```

Que son la tabla de páginas, el número de la página que falta y el tamaño del working set del proceso. La tabla de páginas tiene los bits de estado asociados y el arreglo de las páginas:

```
typedef struct {
int size;      /* taman~o de la tabla */
struct {
    bool invalid;  /* true si la pagina esta en disco */
    bool dirty;    /* true si la pagina ha sido modificada */
    DISK disk_ptr; /* posicion en el disco */
    int  frame;    /* posicion en RAM */
} page_array[];
} PAGETABLE;
```

Por ejemplo, para poner el bit de inválido a la página `p`, hago:

```
pagetable->page_array[p].invalid = TRUE;
```

Use las siguientes funciones para su implementación:

```
int FindFreeFrame();
```

Retorna un frame libre en la memoria RAM. Suponga que siempre funciona y encuentra uno libre, pero sólo deben invocarla si el proceso se mantiene dentro de su Working Set Size (`wss`).

```
int FindLRU(PAGETABLE *pagetable);
```

Escoge una página víctima de la tabla para ser reemplazada, utilizando LRU.

```
swapout(DISK dptr, int frame);
```

Copia la página de RAM a disco.

```
swapin(DISK dptr, int frame);
```

Copia de disco a RAM una página.

Recuerde no copiar a disco una página que no ha sido modificada, y suponga que el puntero a disco siempre existe en la tabla de páginas y está al día, o sea que la RAM actúa como cache del espacio de swap.

```
PageFault(PAGETABLE *pagetable, int page, int wss)
{
    if(!pagetable->page_array[page].invalid)
        return;

    for(i=0; i < pagetable->size; i++)
       if( !pagetable->page_array[i].invalid )
           npages++;

    if(npages < wss )
        fr = FindFreeFrame();
    else {
        p = FindLRU(pagetable);
        if(pagetable->page_array[p].dirty) {
            pagetable->page_array[p].disk = swapout(pagetable[p].frame);
            pagetable->page_array[p].invalid = TRUE;
        }
        fr = pagetable->page_array[p].frame;
    }

    swapin(pagetable->page_array[page].disk, fr);
    pagetable->page_array[page].invalid = FALSE;
    pagetable->page_array[page].frame = fr;
    pagetable->page_array[page].dirty = FALSE;
    return;
}
```

Pregunta 2

Parte I

Suponga que tiene una arquitectura con sólo bits de invalidación de las páginas. Cuando el espacio en RAM pasa un umbral mínimo, diseñe un esquema que

permita detectar las páginas que no están siendo referenciadas en últimamente para sacarlas al swap.

Cuando se sobrepasa el umbral de RAM, se invalidan todas las páginas de los procesos. Cuando se accesan, se pasa el bit de invalidación a 0 y se re-ejecuta el proceso. Al buscar páginas no-referenciadas, se usan aquellas que todavía tienen el bit de invalidación en 1.

Parte II

Explique porqué no es razonable hacer memoria virtual y paginamiento sin que el hardware soporte la tabla de páginas.

Para la traducción de direcciones, se debe ejecutar una secuencia de código por cada acceso a memoria. Si esto tuviese que hacerse por software, tendría que hacerlo el S.O. (por protección). Para implementarlo, se requiere que haya una interrupción por cada acceso a memoria, lo que es demasiado ineficiente y en algunos casos imposible.

Parte III

¿Cómo se implementa la protección de las páginas de un proceso, para que ningún otro pueda accesarlas?

La protección se da automáticamente por la existencia de traducción de direcciones. Resulta imposible para un proceso accesar memoria de otro proceso puesto que no tiene como apuntarla.

Parte IV

En general las páginas son potencias de dos. Escriba el pseudo-código de la traducción de direcciones que se realiza si las páginas fuesen de tamaño 10. Explique porqué es una buena idea que sean potencias de dos.

```
page = address / 10;
phys_addr = table[page] + address % 10;
```

Se usan potencias de 2, porque permiten evitar la división entera simplemente extrayendo los bits de orden superior para las páginas y los de orden inferior para el desplazamiento.

Parte V

¿Cómo evitamos que el overhead de accesar memoria paginada sea de 100 %?

Cada acceso a memoria requiere un acceso a la tabla de páginas y luego el verdadero acceso a memoria. Para evitar un overhead de 100 %, usamos una tabla de páginas por hardware, con al menos un cache de los elementos más usados de la tabla. La búsqueda en esa tabla debe ser muy eficiente, evitando accesar la memoria.

Parte VI

Explique cómo hace el S.O. para entregarle CPU a un proceso, pasando a modo usuario, pero garantizando que después la recupera en modo supervisor. Recuerde que la instrucción que cambia el modo de la CPU es privilegiada, por lo que no se puede ejecutar en modo usuario.

Las interrupciones de hardware son atrapadas por el S.O. en modo supervisor. Por lo tanto, basta con garantizar que habrá una interrupción de hardware en el futuro. Para ello, se setea una interrupción de reloj en el tiempo máximo de ejecución de un proceso.