Departamento de Arquitectura

Examen de Sistemas Operativos

22 de julio de 2023

Lea detenidamente las siguientes instrucciones. No cumplir los requerimientos puede implicar la pérdida del examen.

Formato

- Indique su nombre completo y número de cédula en cada hoja (no se corregirán las hojas sin nombre). Numere todas las hojas e indique la cantidad total de hojas en la primera.
- Escriba las hojas de un solo lado y empiece cada problema en una hoja nueva.
- Si se entregan varias versiones de un problema solo se corregirá la primera de ellas.

Dudas

- Sólo se contestarán dudas de letra.
- No se aceptarán dudas en los últimos 30 minutos del examen.

Material

- El examen es SIN material (no puede utilizarse ningún apunte, libro ni calculadora). Sólo puede tenerse las hojas del examen, lápiz, goma y lapicera en su banco. Todas sus demás pertenencias debe colocarlas en el piso debajo de su asiento.

Aprobación

- Para aprobar el examen se debe tener un mínimo de 60 puntos.

Finalización

- El examen dura **3 horas**.
- Al momento de finalizar el examen no se podrá escribir absolutamente nada en las hojas, debiéndose parar e ir a la fila de entrega. Identificar cada una de las hojas con nombre, cédula y numeración forma parte de la duración del examen.

Problema 1 (31 pts)

(a) (5 pts) Describa brevemente el método de asignación en forma de lista (linked allocation) y el método de asignación indexada (indexed allocation) para los sistemas de archivos. Describa y justifique una ventaja de cada uno de ellos con respecto al otro.

(b) (5 pts) Describa las técnicas de agrupación (grouping) y conteo (counting) utilizadas para la administración de bloques libres en un sistema de archivos.

(c) (3 pts) Describa qué es un hardlink y mencione dos diferencias con un softlink.

(d) Sea un sistema de archivos de tipo FAT que cuenta con las siguientes definiciones:

```
type entrada_dir = record
    usado: boolean                \\ 1 bit
    tipo: (file, dir)             \\ 1 bit
    inicioFAT: integer            \\ 2 bytes
    nombre: array [0..26] of char  \\27 bytes
    tamaño: unsigned integer      \\ 2 bytes
    reservado: array [0..5] of bit \\ 6 bits
end;  \\32 bytes en total
type fat = array [0..(MAX_BLOQUES-1)] of -2..(MAX_BLOQUES-1);
type block = array [0..4095] of byte;
```

Tomando en cuenta que `MAX_BLOQUES` es 65536 (i.e. 2^{16}) y que el primer bloque se encuentra reservado para el directorio raíz, responda:

I. (4 pts) ¿Cuál es la máxima cantidad de archivos que puede contener el directorio raíz?

Solución: El caso con máxima cantidad de archivos ocurre cuando el directorio raíz ocupa la máxima cantidad de bloques y se encuentra lleno de archivos de tamaño 0 bytes. La cantidad máxima de bloques que puede usar el directorio raíz es `MAX_BLOQUES`$= 2^{16}$. Cada uno de estos bloques puede contener hasta $2^{12}/2^5 = 2^7$ entradas. Entonces la cantidad máxima de archivos es $2^{16} \times 2^7 = 2^{23}$.

II. (4 pts) ¿Cuál es el mayor tamaño que puede tener un archivo en este sistema?

Solución: El mayor tamaño que puede tener un archivo se encuentra limitado por el campo `tamaño` y por la cantidad máxima de bloques disponibles, `MAX_BLOQUES`−1 (porque un bloque está asignado al directorio raíz). Según el campo tamaño un archivo no puede tener más de 2^{16} bytes. Por otro lado, según la cantidad máxima de bloques un archivo podría tener $(2^{16} - 1) \times 2^{12}$. Como $2^{16} < (2^{16} - 1) \times 2^{12}$, entonces el tamaño máximo de un archivo es 2^{16}.

(e) (5 pts) Describa brevemente los cuatro mecanismos de hardware necesarios para garantizar la protección del sistema de forma eficiente.

(f) (5 pts) ¿En qué circunstancias ocurren los fallos de página en un sistema con memoria virtual? Describa las acciones llevadas a cabo por el sistema operativo en cada caso cuando ocurre un fallo de página.

Problema 2 (32 pts)

Sea un sistema que implementa memoria virtual con paginación bajo demanda. En este sistema, los procesos pueden direccionar hasta 2 GiB de memoria (i.e. 2^{31} bytes). Además, la traducción de una dirección se realiza a través de cuatro niveles de tabla de página. Cada entrada de la tabla de páginas es de 32 bits y se desea que la tabla de página de primer nivel ocupe sólo una página y completamente. Las tablas del resto de los niveles contienen la misma cantidad de entradas. Finalmente, los marcos en memoria principal tienen un tamaño de 4096 bytes.

Se pide:

(a) (5 pts) Determine el formato de las direcciones virtuales. Justificando adecuadamente cada una de sus partes.

Solución: Los procesos pueden direccionar hasta 2 GiB de memoria → direcciones virtuales de 31 bits con el siguiente formato:

| primer nivel | segundo nivel | tercer nivel | cuarto nivel | desplazamiento |

Los frames son de 4096 → páginas de 4096 → 12 bits para el desplazamiento. El primer nivel ocupa una página (i.e. 4096 bytes) y tiene entradas de 4 bytes → 10 bits para direccionar sus entradas. Para el resto de los niveles quedan 12 bits a repartir en partes iguales, por lo tanto la dirección virtual se divide de la siguiente manera: | 10 | 3 | 3 | 3 | 12 |

(b) (5 pts) Asumiendo que el sistema cuenta con una cache TLB (Translation Look-aside Buffer) muestre el esquema de traducción de una dirección virtual.

Solución: Ver teórico.

(c) (11 pts) Asumiendo que el sistema cuenta con una cache TLB que no contiene 'cacheado' ningún valor y que no cuenta con otras memorias cache entre el procesador y la memoria principal, determine la cantidad de accesos a memoria necesarios para leer un arreglo de 40.000 bytes:

```
var arreglo : array[0..39999] of byte;
for (i = 0; i < 40000; i++)
...arreglo[i]...
```

Asuma que la dirección de comienzo de la variable `arreglo` es al principio de una página y que la variable `i` está guardada en un registro de CPU (no tome en cuenta sus accesos).

Solución: Las páginas son de 4096 bytes. Un array se almacena de forma contigua en memoria virtual, por lo tanto se requieren 10 páginas contiguas para almacenarlo. Como las tablas de 4to nivel tienen 16 entradas, necesito acceder a 1 tabla de cuarto nivel $\rightarrow$ 1 tabla de de 3er nivel (solo requiere dos entradas) $\rightarrow$ 1 tabla de 2do nivel (solo requiere una entrada) $\rightarrow$ 1 entrada en la única tabla de primer nivel.

De esta forma se tiene:

1. un acceso a la página que contiene la tabla de primer nivel para leer la entrada que corresponde.
2. un acceso a la página que contiene la tabla de segundo nivel para leer el valor de la entrada correspondiente.
3. un acceso a la página que contiene la tabla de tercer nivel para leer el valor de la entrada correspondiente.
4. un acceso a la página que contiene la tabla de cuarto nivel para leer el valor de la entrada correspondiente.
5. un acceso al la página correspondiente para leer el primer byte.
6. se realizan 4095 accesos para leer cada uno de los elementos cargados en la primer página del array.
7. para leer la segunda página del array es necesario primero acceder a la página que contiene la tabla de todos los niveles para traer la página y almacenar la correspondencia en la TLB.
8. se realizan 4096 accesos para leer cada uno de los elementos cargados en la segunda página del array.
9. se repite el proceso hasta leer la décima página.
10. finalmente en la última página se leen 3136 bytes (con 10 páginas completas tenemos 40.960 bytes, por lo tanto de la última accedemos a 4096-960 bytes).

En total se tienen $(4 + 4096) \times 9 + (4 + 4096 - 960)$ accesos.

(d) (11 pts) El sistema tiene una estrategia de asignación de memoria local y utiliza un algoritmo de reemplazo LRU (Least Recently Used). Se tiene un proceso P que no tiene ninguna página en memoria principal y al que se le asignan 4 marcos para utilizar. Muestre, mediante un esquema en el tiempo, los fallos de páginas y el estado de la memoria si dicho proceso realiza los siguientes accesos:

Tiempo	Primer nivel	Segundo nivel	Tercer nivel	Cuarto nivel	Desplazamiento
t1	10	1	1	1	4
t2	10	1	2	1	1
t3	10	2	1	2	4
t4	12	2	2	2	2
t5	12	2	2	1	2
t6	10	1	2	1	2
t7	12	2	2	1	4
t8	10	1	1	1	2

Solución: La página será identificada en el esquema por:
(entrada_primer_nivel, entrada_segundo_nivel, entrada_tercer_nivel, entrada_cuarto_nivel)

Tiempo	T1	T2	T3	T4	T5	T6	T7	T8
marco1	(10,1,1,1)	(10,1,1,1)	(10,1,1,1)	(10,1,1,1)	(12,2,2,1)	(12,2,2,1)	(12,2,2,1)	(12,2,2,1)
marco2		(10,1,2,1)	(10,1,2,1)	(10,1,2,1)	(10,1,2,1)	(10,1,2,1)	(10,1,2,1)	(10,1,2,1)
marco3			(10,2,1,2)	(10,2,1,2)	(10,2,1,2)	(10,2,1,2)	(10,2,1,2)	(10,1,1,1)
marco4				(12,2,2,2)	(12,2,2,2)	(12,2,2,2)	(12,2,2,2)	(12,2,2,2)
LRU	FALLO	FALLO	FALLO	FALLO	FALLO			FALLO
Mas RU	(10,1,1,1)	(10,1,2,1)	(10,2,1,2)	(12,2,2,2)	(12,2,2,1)	(10,1,2,1)	(12,2,2,1)	(10,1,1,1)
		(10,1,1,1)	(10,1,2,1)	(10,2,1,2)	(12,2,2,2)	(12,2,2,1)	(10,1,2,1)	(12,2,2,1)
			(10,1,1,1)	(10,1,2,1)	(10,2,1,2)	(12,2,2,2)	(12,2,2,2)	(10,1,2,1)
Menos RU				(10,1,1,1)	(10,1,2,1)	(10,2,1,2)	(10,2,1,2)	(12,2,2,2)

Luego de ejecutados los accesos a memoria se generan 6 fallos de página y el estado de la memoria es: marco 1 página (12,2,2,1), marco 2 página (10,1,2,1), marco 3 página (10,1,1,1) y marco 4 página(12,2,2,2).

Problema 3 (37 pts)

OSE ha instalado un tanque de agua potable para abastecer a varios hospitales de la ciudad. Los hospitales envían camionetas para llevar el agua a cada hospital. El tanque cuenta con cinco canillas para cargar las camionetas y se cuenta con cuatro empleados que atienden a las camionetas que llegan. Cada camioneta estaciona frente a una de las canillas hasta ser atendida y debe esperar en caso de que estén todas ocupadas. Cuando se agota el agua se debe avisar a un inspector de OSE. Este inspector se encargará de rellenar el tanque. Para esto el inspector requiere acceso exclusivo al tanque y tiene prioridad sobre las nuevas camionetas que llegan.

Se pide: Modelar usando mailboxes a los empleados, las camionetas y el inspector. No se permite usar procesos auxiliares.

Se dispone de los siguientes procedimientos auxiliares:

`hay_agua(): boolean`
Ejecutada por los empleados para saber si hay agua en el tanque para satisfacer el siguiente pedido.

`rellenar()`
Ejecutada por el inspector para llamar a OSE para rellenar el tanque. Se bloquea hasta que el tanque esté lleno nuevamente.

`cargar()`
Ejecutada por los empleados para cargar las camionetas.

`otras_tareas()`
Ejecutada por el inspector luego de recargar el tanque.

Solución:

```
var canillas_libres: mailbox of integer
    canillas_ocupadas: mailbox of integer
    espera_canilla: array [1..5] of mailbox of nil
    inspector: mailbox of nil
    avisar_inspector: mailbox of boolean
    cargando: mailbox of integer;
    espera_fin_inspector: mailbox of nil
    mutex_agua: mailbox of nil

procedure camioneta
var canilla: integer
begin
    canilla := receive(canillas_libres);
    send(canillas_ocupadas, canilla);
    receive(espera_canilla[canilla]);
    send(canillas_libres, canilla);
end camioneta

procedure empleado
var canilla: integer
    avisar: boolean
    cant_cargando: integer
    esperando: integer
begin
    while true do
        canilla := receive(canillas_ocupadas);
        receive(mutex_agua);
        if not hay_agua() then
            cant_cargando := receive(cargando);
            if cant_cargando = 0 then
                send(inspector, nil);
            else
                receive(avisar_inspector);
                send(avisar_inspector, true);
            end if
            send(cargando, cant_cargando);
            receive(espera_fin_inspector);
        end if
        cant_cargando := receive(cargando);
        send(cargando, cant_cargando + 1);
        send(mutex_agua, nil);

        cargar();

        cant_cargando := receive(cargando);
        avisar := receive(avisar_inspector);
        if cant_cargando = 1 and avisar then
            send(inspector, nil);
        end if
        send(avisar_inspector, avisar);
        send(cargando, cant_cargando - 1);

        send(espera_canilla[canilla]);
    end while
```

```
end empleado

procedure inspector
var esperando: integer
begin
    while true do
        receive(inspector);
        rellenar();
        receive(avisar_inspector);
        send(espera_fin_inspector, nil);
        send(avisar_inspector, False);
        otras_tareas();
    end while
end inspector

begin
    for i := 1 to 5 do
        send(canillas_libres, i);
    end for
    send(avisar_inspector, False);
    send(mutex_agua, nil);
    send(cargando, 0);

    cobegin
        empleado();
        empleado();
        empleado();
        empleado();
        inspector();
        camioneta();
        ...
        camioneta();
    coend
end
```