Departamento de Arquitectura

Examen de Sistemas Operativos

23 de diciembre de 2021

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 versión.

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). En su banco solo puede tener las hojas del examen, lápiz, goma y lapicera. Todas sus demás pertenencias debe colocarlas en el piso debajo de su asiento.

Aprobación

- Para aprobar el examen es necesario un mínimo de **60 puntos**.

Finalización

- La duración del examen es de **3 horas**.
- Al momento de finalizar el examen **no se podrá escribir absolutamente nada en las hojas**, el estudiante debe dirigirse a la fila de entrega. Identificar cada una de las hojas con nombre, cédula y numeración debe realizarse dentro de la duración del examen.

Problema 1 (20 pts)

Conteste justificando cada una de sus respuestas.

(a) (4 pts) Compare RAID 5 y RAID 6 e indique cuanto espacio utilizable resultaría de crear un RAID de cada uno de estos tipos sobre 4 discos de 2 TB.

(b) (4 pts) Describa brevemente que es una multilevel feedback queue y enumere las características mínimas necesarias para definir una.

(c) (4 pts) En el contexto de una operación de E/S, describa las diferencias entre una operación bloqueante y una no bloqueante.

(d) (4 pts) Describa brevemente el mecanismo de virtualización denominado paravirtualización.

(e) (4 pts) Indique que significa que un archivo tenga permisos 760 en UNIX.

Problema 2 (25 pts)

Se desea implementar un sistema operativo que utiliza una estrategia indexada multinivel de dos niveles y un mapa de bits para administrar los bloques libres. A continuación se presentan las estructuras de datos utilizadas por este sistema de archivos:

```
const MAX_BLOQUES = 65536;
const MAX_INODOS = 8192;
type bloque = array [0..4095] of byte; // 4096 bytes
type mapa_bits = array [0..MAX_BLOQUES-1] of bit;

type entrada_dir = record
    usado : bool; // 1 bit
    nombre : array [0..123] of char; // 124 bytes
    es_dir : bool; // 1 bit
    inodo_num : int16; // 2 bytes
    permisos : array [0..13] of bit; // 14 bits
end; // 128 bytes

type inodo = record
    usado: bool; // 1 bit
    inodo_num: int16; // 2 bytes
    es_dir: bool; // 1 bit
    tamaño: int32; // 4 bytes
    directo: array [0..7] of int16; // 16 bytes
    directo_tope: int16; // 2 bytes
    indirecto: int16; // 2 bytes
    indirecto_tope: int16; // 2 bytes
    reservado: array [0..30] of bit; // 30 bits
end; // 32 bytes

type inodos_tabla = array [0..MAX_INODOS-1] of inodo;
type disco = array [0..MAX_BLOQUES-1] of bloque;

var IT : inodos_tabla;
    MB : mapa_bits;
     D : disco;
```

Se sabe que:

- las variables IT, MB, y D son globales
- el inodo número 0 (cero) es el directorio raíz
- en MB se indica con 1 (uno) si un bloque está ocupado y con 0 (cero) en caso contrario
- las entradas de los directorios son almacenadas utilizando un array de entradas

Se dispone de los siguientes procedimientos:

- *procedure leerBloque(disco d, int bloque_num, byte[] &buff, bool &ok)*
  Lee desde el disco d el bloque con índice bloque_num en la variable buff.
  En ok se retorna verdadero si se ejecutó correctamente.
- *procedure obtenerDirPadre(char[] camino, char[] &dir)* Retorna en el parámetro dir el nombre del directorio padre que contiene el archivo o directorio referenciado en el parámetro camino.
  Ej: obtenerDirPadre('/home/sistoper/a.txt') retorna '/home/sistoper'
- *procedure obtenerNombre(char[] camino, char[] &base)* Retorna en el parámetro base el nombre del archivo o directorio referenciado en el parámetro camino.
  Ej: obtenerNombre('/home/sistoper/a.txt') retorna 'a.txt'

Se pide:

(a) (2 pts) Indique cuántas entradas de directorio hay por bloque.

Solución: Cada entrada de directorio ocupa 128 bytes y cada bloque tiene un de tamaño 4096 bytes, por lo tanto hay 32 entradas de directorio por bloque.

(b) (5 pts) Indique el tamaño máximo que puede tener un archivo utilizando el sistema de archivos planteado.

Solución: Cada bloque es de 4096 bytes = 2^{12} bytes.
Por inodo se tiene:

- 8 bloques de primer nivel (directo)
- Para referenciar un bloque se requiere de 2 bytes
- el sistema de archivos utiliza una estrategia de indexación multinivel de dos niveles.

Por lo que se cuenta con:

- 1 bloque que referencia a bloques ->$2^{12}/2$ = 2^{11} = 2048 bloques

Entonces: por archivo se puede tener hasta 2048 + 8 = 2056 bloques ->$2056 * 4096$ bytes

(c) (18 pts) Implemente la siguiente función que dada una ruta, encuentre el inodo referenciado:
procedure obtenerInodo(char[] ruta, int &inodo, bool &ok)
En ok se retorna verdadero si se ejecutó correctamente. En inodo se retorna el número de inodo correspondiente a la ruta.

Solución:

```
procedure buscarEntradaDir(entrada_dir[] bloque, char[] nombre,
    int16 &entrada) {
    entrada = -1;
    for(int i = 0; i < 32; i++) {
        if(bloque[i].usado && bloque[i].nombre == nombre) {
            entrada = i;
        }
    }
}

procedure buscarInodo(int16 inodo, char[] nombre, int &nroInodo,
    bool &ok) {
    entrada_dir[32] bloque;
    int16[2048] bloqueInd;
    inodo in = IT[inodo]
    ok=true;
    if(!in.es_dir) {
        ok = false;
    } else {
        int entry = -1;
        //busco en directos
```

```
            for(int i=0; i < in.directo_tope && entry<0 && ok; i++) {
                leerBloque(D, in.directo[i], bloque, ok);
                if(ok) {
                    buscarEntradaDir(bloque, nombre, entry)
                }
            }
            if(ok && entry < 0 && in.indirecto_tope > 0) {
                leerBloque(D, in.indirecto, bloqueInd, ok);
                if(ok) {
                    for(int i = 0;i < in.indirecto_tope && entry < 0
                        && ok; i++) {
                        leerBloque(D, bloqueInd[i], bloque, ok);
                        if(ok)
                        {
                            buscarEntradaDir(bloque, nombre, entry)
                        }
                    }
                }
            }
            if(!ok || entry<0) {
                ok = false;
            } else {
                nroInodo = bloque[entry].inodo_num;
            }
        }
    }

    procedure obtenerInodo(char[] ruta, int &inodo, bool &ok) {
        char pila[MAX_INODOS][];
        pilaCant = 0;
        char[] dir;
        char[] nombre;
        obtenerNombre(ruta, nombre);
        obtenerDirPadre(ruta, dir);
        if(dir = "" and nombre == ""/) {
            inodo = 0;
        } else {
            do {
                pila[pilaCant] = nombre;
                pilaCant++;
                ruta = dir;
                obtenerNombre(ruta, nombre);
                obtenerDirPadre(ruta, dir);
            } while (dir = "" and nombre != "/")

            inodoPadre=0;
            for (int i=pilaCant-1; i>=0 && ok; i--) {
                buscarInodo(inodoPadre, nombre[i], inodo, ok);
                if (ok) inodoPadre = inodo;
```

```
            }
        }
    }
```

Problema 3 (20 pts)

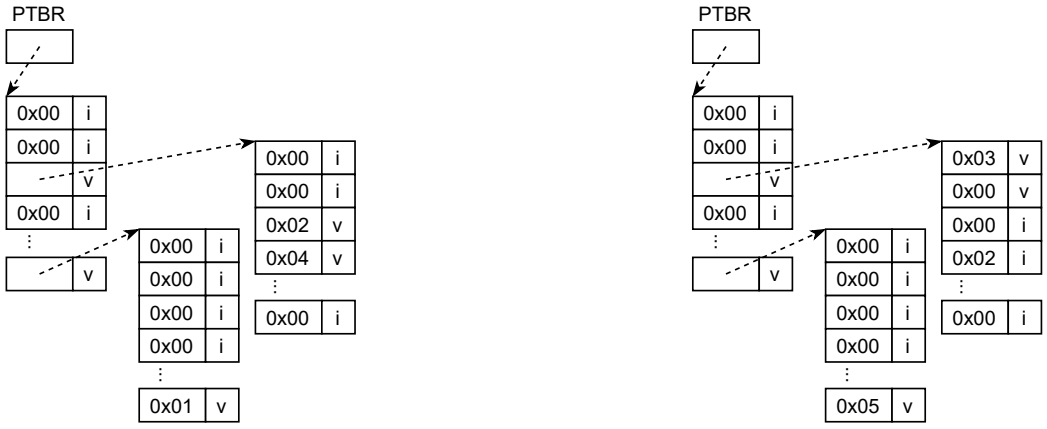
Sea un Sistema Operativo (SO) con un planificador Round-Robin (RR) que tiene un quantum de 3 unidades de tiempo. Este SO ejecuta sobre una arquitectura computacional con un único procesador y una MMU que utiliza memoria virtual con direcciones de 16 bits (i.e. $0\text{x}0000$ a $0\text{x}\text{FFFF}$) y paginación bajo demanda de dos niveles. Cada nivel de la tabla de páginas es direccionada usando 4 bits de la dirección virtual mientras que los restantes 8 bits son usados para el desplazamiento. El SO implementa un algoritmo de asignación global de 6 marcos en memoria principal para uso de los procesos de usuario. Los marcos se identifican dentro del rango $0\text{x}00$ a $0\text{x}05$ y el SO usa un algoritmo de reemplazo First In, First Out (FIFO) para manejarlos.

La arquitectura ofrece dos tipos de instrucciones a los procesos de usuario:

- Para manipular u operar con registros de la CPU que se identifican con la letra `O` (e.g. operaciones aritméticas, lógicas, etc.). Siempre requieren una única unidad de tiempo para completarse.
- Para aceder a memoria principal que se identifican con la letra `M`. Estas van seguidas de la dirección virtual de memoria accedida y requieren una unidad de tiempo para completarse. Un proceso será terminado por el SO si la página accedida no pertenece a su espacio de memoria asignado.

Además, el SO ofrece una llamada al sistema que permite el acceso a dispositivos de E/S y que se identifica con la letra `B`. Esta llamada bloquea el proceso que la invoca y va seguida de la cantidad de unidades de tiempo que requiere para terminar (e.g. `B(5)` indica que el proceso debe esperar *en total* 5 unidades de tiempo).

Suponga que los procesos P1 y P2 son los únicos procesos en ejecución en el sistema. El espacio de memoria virtual asignado a P1 es $[0\text{x}2100 - 0\text{x}23\text{FF}]$ y $[0\text{x}\text{FF}00 - 0\text{x}\text{FFFF}]$, y a P2 es $[0\text{x}2000 - 0\text{x}23\text{FF}]$ y $[0\text{x}\text{FF}00 - 0\text{x}\text{FFFF}]$. En un instante dado de tiempo (t=0), las tablas de páginas de P1 y P2 se encuentran tal como muestra la Figura 1 (las tablas comienzan con el índice 0 y el último luego de los puntos es el 15)



(a) Proceso P1 (b) Proceso P2

Figura 1: Tabla de páginas de los procesos P1 y P2 en el instante t=0

La cola FIFO mantenida por el algoritmo de reemplazo del SO contiene las siguientes páginas en t=0 (de más reciente a la izquierda a más antigua a la derecha):

Pág.	Proc.	Pág.	Proc.	Pág.	Proc.	Pág.	Proc.	Pág.	Proc.	Pág.	Proc.
0xFF	P1	0x20	P2	0x21	P2	0xFF	P2	0x23	P1	0x22	P1

Dadas la siguiente secuencia de invocaciones de P1 y P2:

Programa de P1	Programa de P2
M(0x220F) O O O M(0x22F1) M(0x2014) B(10) M(0xFFFC)	M(0xFFFE) O M(0x2210) O B(2) M(0x208D)

Sabiendo que P1 se encuentra primero en la cola de listos en t=0 y que los cambios de contexto son instantáneos y no consumen unidades de tiempo.

Se pide:

(a) (15 pts) Realice un esquema que, para cada proceso en cada instante del tiempo, muestre (al menos): su estado, que invocación está realizando, y que página es reemplazada y por cuál.

Solución:

t	P1			P2		
	Estado	Op	Mem	Estado	Op	Mem
0	E	M(0x220F)		L		
1	E	O		L		
2	E	O		L		
3	L			E	M(0xFFFE)	
4	L			E	O	
5	L			E	M(0x2210)	out 0x22 P1 in 0x22 P2
6	E	O		L		
7	E	M(0x22F1)	out 0x23 P1 in 0x22 P1	L L		
8	E	M(0x2014)	seg. fault	L		
9	T			E	O	
10	T			E	B(2)	
11	T			E	B(2)	
12	T			E	M(0x208D)	
13	T			T		

(b) (5 pts) Muestre el estado de las tablas de páginas de P1 y P2 al finalizar su ejecución.

Solución:

Las tablas de páginas de primer nivel no cambian. Se muestran a continuación las de segundo nivel que corresponden a los desplazamientos `0x2` y `0xF` de las tablas de primer nivel.

P1				P2			
0x2		0xF		0x2		0xF	
0x00	i	0x00	i	0x03	v	0x00	i
0x00	i	0x00	i	0x00	v	0x00	i
0x04	v	0x00	i	0x02	v	0x00	i
0x04	i	0x00	i	0x02	i	0x00	i
0x00	i	0x00	i	0x00	i	0x00	i
		...					
0x00	i	0x00	i	0x00	i	0x00	i
0x00	i	0x01	v	0x00	i	0x05	v

Problema 4 (35 pts)

Con motivo de las fiestas, Papá Noel y los duendes se disponen a entregar regalos a los niños. Los niños eligen el regalo que quieren y mandan una carta al polo norte. En el polo norte trabajan 6 duendes. La carta se asigna a un duende que este esperando para que esté confeccione el regalo. Si no hay duendes esperando cuando se recibe la carta, se descarta la misma y el niño se queda sin regalo.

Los regalos se van agregando a una bolsa. La bolsa solo puede ser accedida por un duende a la vez. Cuando la misma alcanza 100 regalos, se debe despertar a Papá Noel para que reparta los mismos. Dependiendo de que tan cansado esté, Papá Noel puede solicitar la ayuda de uno de los duendes. Mientras Papá Noel reparte los regalos, los duendes pueden seguir trabajando.

Se pide implementar, usando mailboxes, los procesos `Niño`, `PapaNoel` y `Duende`. Se pueden usar procesos auxiliares.

Dispone de los siguientes procedimientos auxiliares:

- `escribir_carta(): Carta`. Ejecutado por un niño para escribir la carta a enviar al servidor.
- `confeccionar_regalo(c: Carta): Regalo`. Ejecutado por un duende para crear un regalo.
- `repartir_regalos(regalos: array[100] of Regalo)`. Ejecutado por Papá Noel para repartir los regalos. Recibe un arreglo con 100 regalos.
- `estoy_cansado(): Boolean`. Ejecutada por Papá Noel. Retorna True si necesita la ayuda de un duende.
- `ayudar()`: Ejecutada por un duende para ayudar a Papá Noel. Puede asumir que esta función termina al mismo tiempo que `repartir_regalos`.
- `recibir_regalo()`. Ejecutada por el Niño. Esta función se bloquea y termina cuando Papá Noel le entrega el regalo.

Solución:

```
# Utilizamos mailboxes infinitos con recibir bloqueante
buzon_del_servidor: mailbox of Carta
duendes_disponibles: mailbox of int
cant_duendes_disponibles: mailbox of int
```

```
# En este mailbox se trancan los duendes
# esperando que les digan si tienen que ayudar
# o armar un regalo
duende_esperando: array[6] of mailbox of Boolean

# mailbox donde el servidor le pasa la carta
duende_espera_carta: array[6] of mailbox of Carta

cant_regalos: mailbox of int
bolsa_regalos: mailbox of Regalo
duerme_papanoel: mailbox of NIL

# Para saber si el nino debe esperar regalo o no
mtx_nino_servidor: mailbox of NIL
nino_recibe_regalo: mailbox of Boolean

procedure Niño()
    begin
        var c: Carta;
        var recibo: Boolean;
        c = escribir_carta();

        recibir(mtx_nino_servidor, NIL);
        enviar(buzon_del_servidor, c);

        recibir(nino_recibe_regalo, recibo);
        enviar(mtx_nino_servidor, NIL);

        if (recibo)
            recibir_regalo();
        end if
    end

procedure PapaNoel()
    begin
        var regalos: array[100] of Regalo;
        var r: Regalo;
        var cant: int;
        var cansado: Boolean;
        var duende: int;

        while (true)
            recibir(duerme_papanoel, NIL);

            for i=1 to 100
                recibir(bolsa_regalos, r);
                regalos[i] = r;
            end for

            recibir(cant_regalos, cant);
```

```
            enviar(cant_regalos, cant - 100);

            cansado = estoy_cansado();
            if (cansado)
                # recibimos este mailbox solo para no interferir
                # con el acceso del servidor a duendes_disponibles
                recibir(cant_duendes_disponibles, cant);
                recibir(duendes_disponibles, duende);
                enviar(cant_duendes_disponibles, cant - 1);
                enviar(duende_esperando[duende], True);
            end if

            repartir_regalos(regalos);
        end while
    end

procedure Duende(var id: int)
    begin
        var c: Carta;
        var tengo_que_ayudar: Boolean;
        var r: Regalo;
        var cant_r, cant: int;

        while (true)
            recibir(duende_esperando[id], tengo_que_ayudar);
            if (tengo_que_ayudar)
                ayudar();
            else
                recibir(duende_espera_carta[id], c);
                r = confeccionar_regalo(c);

                # metemos el regalo en la bolsa
                enviar(bolsa_regalos, r);
                recibir(cant_regalos, cant_r);
                cant_r = cant_r + 1;

                #si tenemos mas de 100 regalos, despertamos a papa noel
                if (cant_r >= 100)
                    enviar(duerme_papanoel, NIL);
                enviar(cant_regalos, cant_r);

            # una vez que el duende termina con sus tareas,
            # se vuelve a poner disponible
            enviar(duendes_disponibles, id);

            # actualizamos la cantidad despues para
            # evitar deadlock con papanoel
            recibir(cant_duendes_disponibles, cant);
            enviar(cant_duendes_disponibles, cant + 1);
```

```
        end while
    end

procedure Servidor()
    begin
        var duende, cant: int;
        var c: Carta;

        while (true)
            # recibimos una carta
            recibir(buzon_del_servidor, c);

            # si hay duendes esperando, tomamos uno
            # y le enviamos la carta
            recibir(cant_duendes_disponibles, cant);
            if (cant > 0)
                enviar(nino_recibe_regalo, True);
                recibir(duendes_disponibles, duende);
                enviar(cant_duendes_disponibles, cant - 1);
                enviar(duende_esperando[duende], False);
                enviar(duende_espera_carta[duende], c);
            else
                # en caso de no haber duendes esperando,
                # se descarta la carta
                enviar(nino_recibe_regalo, False);
                enviar(cant_duendes_disponibles, cant);
            end if
        end while
    end

begin
    for i=1 to 6
        enviar(duendes_disponibles, i);
    end for
    enviar(cant_duendes_disponibles, 6);
    enviar(cant_regalos, 0);

    cobegin
        PapaNoel();
        Duende(1); Duende(2); Duende(3); Duende(4); Duende(5); Duende(6);
        Servidor();
        Niño(); ... Niño();
    coend
end
```