Segundo Parcial Junio 2018

Lea detenidamente las siguientes instrucciones. No cumplir los requerimientos puede implicar la pérdida de los puntos del parcial.

Formato

- Indique su nombre completo y número de cédula en cada hoja (no se corregirán las hojas sin nombre, sin excepciones).
- Numere todas las hojas e indique la cantidad total de hojas que entrega 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 el primero de ellos según el orden de hojas.

Dudas

- Sólo se contestarán dudas de letra.
- No se aceptarán dudas en los últimos 15 minutos del parcial.

Material

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

Finalización

- El parcial dura 3 horas 30 minutos.
- Al momento de finalizar el parcial 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 parcial.

Problema 1 (10 puntos) (2.5, 2.5, 2.5, 2.5)

1. Describa brevemente el concepto de ensamblaje dinámico (dynamic linking). ¿Qué ventajas brinda?
2. En un sistema con paginación pura y una estructura de tabla de páginas jerárquica.
   a) Explique la estructura de tabla de páginas jerárquica y qué ventajas brinda.
   b) Describa brevemente qué sucede desde que se presenta una dirección virtual válida hasta que se accede a la memoria física y los componentes que intervienen.
3. Describa brevemente las acciones que realiza la rutina de atención al fallo de página en un sistema con paginación bajo demanda.
4. Describa dos técnicas para la virtualización de instrucciones sensibles y mencione al menos una ventaja de cada una.

Problema 2 (23 puntos) (4, 11, 8)

Se tiene un sistema de archivos híbrido. Por un lado, los directorios son almacenados usando una estrategia indexada simple con soporte para enlaces duros (*hard links*). Por otro lado, los archivos son almacenados usando una estructura FAT. Considere la siguiente estructura de datos:

```
const MAX_BLOQUES = 65536;
const MAX_INODOS = 8192;

type entrada_dir = Record
     usado : boolean; // 1 bit
     nombre : array [0..24] of char; // 25 bytes
     tipo : (file, dir); // 1 bit
     inodo_num : int; // 2 bytes
     fat_inicio : int; // 2 bytes
     tamaño : int; // 2 bytes
     reservado : array [0..5] of bit; // 6 bits
End; // 32 bytes

type inodo = Record
     usado : boolean; // 1 bit
     inodo_num : int; // 2 bytes
     datos : array [0..7] of int; // 16 bytes
     tope : int; // 2 bytes
     referencias : int; // 2 bytes
     reservado : array [0..6] of bit; // 7 bits
End; // 23 bytes

type bloque = array [0..1023] of byte; // 1024 bytes
type mapa_bits = array [0..MAX_BLOQUES-1] of bit;
type inodo_tabla = array [0..MAX_INODOS-1] of inodo;
type fat_tabla = array [0..MAX_BLOQUES-1] of int;

var
  FAT: fat_table;
  IT : inodos_tabla;
  MB : mapa_bits;
```

Notas generales:

- Las variables FAT, IT, y MB son globales.

- En la estructura entrada_dir el campo inodo_num es utilizado únicamente cuando el tipo es dir. El atributo fat_inicio y tamaño son utilizados únicamente cuando el tipo es file.
- El directorio raíz es el inodo número 0.
- En FAT el valor -1 indica el fin de archivo.
- El campo referencias es utilizado para contar la cantidad de hard links.
- En MB el valor 0 indica un bloque libre y 1 indica un bloque ocupado.

Se dispone de los siguientes procedimientos:

- `Procedure leerBlq(blq_num: int, Var buff: bloque) : boolean`

  Lee de disco el bloque blq_num, pasado como parámetro, y carga el contenido leído en el parámetro de salida buff. Retorna el éxito de la ejecución de la operación.

- `Procedure escrBlq(blq_num : int, buff : bloque) : boolean`

  Escribe en el bloque blq_num, pasado como parámetro, la información que se encuentra en el parametro buff. Retorna el éxito de la ejecución de la operación.

- `Procedure parteCamino(camino: array of char, var base: array of char, var resto: array of char);`

  Retorna la primera parte de la ruta en el parámetro base, y las restantes partes en resto. Por ejemplo: `parteCamino('/users/so/a.txt', 'users', '/so/a.txt'),`

  `parteCamino('/a.txt', 'a.txt', '')`

  `parteCamino('/', '', '')`

Se pide:

1. Se cuenta con un sistemas de archivos con al menos el siguiente contenido (donde todo lo que tiene extensión es un archivo y lo que no es un directorio):

   `/home/sistoper/tareas/informe.pdf` tamaño: 2 KB

   `/home/sistoper/tareas/labA.txt` tamaño: 5000 bytes

   `/home/2018/`

   a. ¿Cuál es la cantidad mínima de bloques e inodos que se tiene ocupados?

   b. Se crea un *hard link* al directorio `/home/sistoper` en la ruta `/home/2018/sistoper`. ¿Qué cambios se producen en el sistema de archivos? ¿Cuál es ahora la cantidad mínima de bloques e inodos ocupados? Justifique.

2. Implemente una función que busque un elemento de tipo directorio dentro del sistema de archivos y retorne el número de su inodo:

   `function buscarDir(cam: array of char): int;`

   Donde cam es la ruta completa al elemento que se desea buscar. La función retorna el número de inodo del directorio que contiene el elemento buscado. En caso de no encontrase el elemento o producirse un error debe retornar -1.

3. Implemente una función que verifique un archivo en el sistema de archivos (sin tomar en cuenta el resto de los archivos del sistema):

   `function chequearArchivo(archivo: entrada_dir): int;`

   Donde archivo es la entrada de directorio del archivo que se desea verificar. La función retorna 1 en caso que el archivo se encuentre en estado correcto y -1 en caso de que el archivo esté corrupto o en caso de error.

Solución

1)
a)
Se tienen 4 directorios diferentes (/home, /home/sistoper, /home/sistoper/tareas, /home/2018) y el directorio raíz, por lo tanto se tienen al menos 5 inodos ocupados.

El directorio /home tiene como contenido 2 subdirectorios, por lo que con 1 bloque de espacio es suficiente. Algo similar sucede con el directorio raíz (1 subdirectorio), /home/sistoper (1 subdirectorio) y /home/sistoper/tareas (2 archivos). Todos ocupan al menos 1 bloque. El único directorio vacío es /home/2018 que no precisa ningún bloque. En total se usan 4 bloques en directorios. Los archivos ocupan 2 KB (2 bloques) y 5000 bytes (5 bloques). En total el sistema tiene al menos 11 bloques usados.

b)
Se agrega un bloque al directorio /home/2018. Se agrega una entrada de directorio a ese bloque recien asignado con la entrada_dir del hard link y se incrementa en 1 la cantidad de referencias del inodo del directorio /home/sistoper.

Es decir que no hay ningún cambio en la cantidad de inodos y la cantidad de bloques usados se incrementa en 1 y pasa a ser 12.

2)
```
entrada_dir buscarEnDir(int inodo, string elem) {
    entrada_dir[32] buff;

    if !IT[inodo].usado
        return null;

    for (int i=0; i<IT[inodo].tope; i++) {
        if !leerBlq(IT[inodo].datos[i], buff)
            return null;
        for (int j=0; j<32; j++) {
            if buff[j].usado == true and buff[j].nombre == elem
                return buff[j];
        }
    }
}

int buscarDir(string cam) {
    int inodo_act = 0;
    entrada_dir entrada;
    string primero, resto;

    parteCamino(cam, primero, resto);
    if primero == ""
        return inodo_act;
    while resto != "" {
        entrada = buscarEnDir(inodo_act, primero);
        if entrada == null || entrada.tipo != dir
            return -1;
        inodo_act = entrada.inodo_num;
        cam = resto;
        parteCamino(cam, primero, resto);
```

```
    }
    entrada = buscarEnDir(inodo_act, primero);
    if entrada != null
        return inodo_act;
    else
        return -1;
}
```

Nota: también se considera una solución correcta si se retorna entrada.inode_num en lugar de inodo_act

3)

```
int chequearArchivo(entrada_dir archivo) {
    /* Chequear:
    - Tipo correcto
    - Valor del campo usado
    - Tamaño del archivo
    - Dependencias circulares
    - Referencia FAT dentro de rango
    - Uso de mapa de bits
    - Error de lectura física
    */
    bit chq_circular[0..MAX_BLOQUES-1] = {0...0};
    bloque buff;

    if !archivo.uso or archivo.tipo != file
        return -1

    int tamaño_fisico = 0;
    int blq_actual = archivo.fat_inicio;
    while blq_actual != -1 {
        if blq_actual < 0 || blq_actual > MAX_BLOQUES-1
            return -1
        if chq_circular[blq_actual] == 1
            return -1
        chq_circular[blq_actual] = 1

        tamaño_fisico += 1024;

        if MB[blq_actual] == 0
            return -1
        if !leerBlq(blq_actual, buff)
            return -1

        blq_actual = FAT[blq_actual];
    }

    if 0 <= tamaño_fisico-archivo.tamaño < 1024
        return 1
    else
        return -1;
}
```

Problema 3 (30 puntos) (14,16)

Se desea modelar una fábrica de bebidas energizantes. La misma posee 20 máquinas que generan el líquido y lo almacenan en un tanque con capacidad para 50 litros. Además se tienen 5 máquinas embotelladoras que toman líquido del tanque, lo guardan en latas y luego las tapan y les ponen una etiqueta.

La fábrica dispuso que un inspector controle la bebida producida. Para esto tomará una muestra del contenido del tanque en forma periódica para su análisis. En caso de encontrar algún problema deberá detener todo el sistema hasta solucionar el inconveniente. El inspector tiene prioridad sobre cualquiera de las máquinas.

El tanque tiene una sola boca de entrada y salida de líquido por lo que no se podrá tener más de una máquina (o inspector) poniendo o sacando líquido a la vez.

Se dispone de las siguientes funciones auxiliares:

- generarBebida()

  Ejecutada por las máquinas generadoras para producir 500 ml de bebida

- descargarEnTanque()

  Ejecutada por las máquinas generadoras de líquido para poner lo producido en el tanque

- llenarLata()

  Ejecutada por las embotelladoras para llenar una lata de 500 ml

- taparEtiquetarLata()

  Ejecutadas por las embotelladoras para tapar y etiquetar la lata llena

- tomarMuestra()

  Ejecutada por el inspector para tomar una muestra para analizar (cantidad despreciable)

- procesarSolucionar()

  Ejecutada por el inspector para analizar la muestra y solucionar el problema encontrado en caso de ser necesario

- otrasTareas()

  Ejecutada por el inspector mientras no está analizando muestras

Se pide:

1. Modelar las máquinas generadora, embotelladora y el inspector usando monitores.
2. Modelar las máquinas generadora, embotelladora y el inspector usando semáforos.

En ninguno de los dos casos se pueden usar tareas auxiliares.

Solución

a)

```
semaforo bebida, capacidad, tanque, mutexProdEmb;

void maquinaProductora() {
	while (true) {
		generarBebida();
		P(capacidad);
		P(mutexProdEmb);
		P(tanque);
		descargarEnTanque();
		V(tanque);
		V(mutexProdEmb);
		V(bebida);
	}
}
void maquinaEmbotelladora() {
	while (true) {
		P(bebida);
		P(mutexProdEmb);
		P(tanque);
		llenarLata();
		V(tanque);
		V(mutexProdEmb);
		V(capacidad);
		taparEtiquetarLata();
	}
}
void inspector() {
	while (true) {
		P(tanque);
		tomarMuestra();
		procesarSolucionar();
		V(tanque);
		otrasTareas();
	}
}
```

```
void main() {
	init(capacidad, N);
	init(bebida, 0);
	init(tanque, 1);
	init(mutexProdEmb, 1);
	cobegin
		maquinaProductora(); maquinaProductora(); maquinaProductora(); maquinaProductora();
		maquinaProductora(); maquinaProductora(); maquinaProductora(); maquinaProductora();
		maquinaProductora(); maquinaProductora(); maquinaProductora(); maquinaProductora();
		maquinaProductora(); maquinaProductora(); maquinaProductora(); maquinaProductora();
		maquinaProductora(); maquinaProductora(); maquinaProductora(); maquinaProductora();
		maquinaEmbotelladora(); maquinaEmbotelladora(); maquinaEmbotelladora();
		maquinaEmbotelladora(); maquinaEmbotelladora();
		inspector();
	coend
}
```

b)

```
monitor tanque = {
	condition esperaProd, esperaEmb, esperaInspector;
	bool prioridadInspector, libre;
	int cantBebida, cantEsperaProd, cantEsperaEmb;
	void startDescargar() {
		if (!libre || cantBebida == 100 || prioridadInspector) {
			cantEsperaProd++;
			esperaProd.wait()
			cantEsperaProd--;
		}
		libre = false;
	}
	void endDescargar() {
		cantBebida++;
		libre = true;
		if (prioridadInspector) {
			esperaInspector.signal();
		} else if (cantEsperaProd > 0 && cantBebida < 100) {
			esperaProd.signal();
		} else if (cantEsperaEmb > 0) {
			esperaEmb.signal();
		}
	}
```

```
void startLlenar() {
        if !libre || cantBebida == 0 || prioridadInspector {
                cantEsperaEmb++;
                esperaEmb.wait();
                cantEsperaEmb--;
        }
        cantBebida--;
        libre = false;
}
void endLlenar() {
        libre = true;
        if (prioridadInspector) {
                esperaInspector.signal();
        } else if (cantEsperaProd > 0) {
                esperaProd.signal();
        } else if (cantEsperaEmb > 0 && cantBebida > 0) {
                esperaEmb.signal();
        }
}
void trabajar() {
        if (!libre) {
                prioridadInspector = true;
                esperaInspector.wait();
                prioridadInspector = false;
        }
        tomarMuestra()
        solucionarProblema();
        if (cantEsperaProd > 0 && cantBebida < 100) {
                esperaProd.signal();
        } else if (cantEsperaEmb > 0 && cantBebida > 0) {
                esperaEmb.signal();
        }
}
{       prioridadInspector=false;
        libre=true;
        cantBebida=0;
        cantEsperaProd=0;
        cantEsperaEmb=0;    }
```

```
}
void embotelladora() {
        while (true) {
                tanque.startLlenar();
                llenarLata();
                tanque.endLlenar();
                taparEtiquetarLata();
        }
}
void productora() {
        while (true) {
                generarBebida();
                tanque.startDescargar();
                descargarBebida();
                tanque.endDescargar();
        }
}
void inspector() {
        while (true) {
                tanque.trabajar();
                otrasTareas();
        }
}
void main() {
        cobegin
                maquinaProductora(); maquinaProductora(); maquinaProductora(); maquinaProductora();
                maquinaProductora(); maquinaProductora(); maquinaProductora(); maquinaProductora();
                maquinaProductora(); maquinaProductora(); maquinaProductora(); maquinaProductora();
                maquinaProductora(); maquinaProductora(); maquinaProductora(); maquinaProductora();
                maquinaProductora(); maquinaProductora(); maquinaProductora(); maquinaProductora();
                maquinaEmbotelladora(); maquinaEmbotelladora(); maquinaEmbotelladora();
                maquinaEmbotelladora(); maquinaEmbotelladora();
                inspector();
        coend
}
```