Examen 27 de julio de 2015

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, 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.**

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 4 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 (32 puntos)

1. Defina los siguientes conceptos:
   i. Tiempo de retorno
   ii. Tiempo de espera
   iii. Tiempo de respuesta
   iv. Utilización de CPU
2. Describa las condiciones necesarias para la existencia de *Deadlock.*
3. Describa los algoritmos de planificación de disco
   i. SSTF (*Shortest Seek Time First*)
   ii. C-LOOK.
4. Describa dos métodos de acceso a un archivo.
5. Mencione los servicios básicos que debe brindar un sistema operativo.
6.
   i. Para qué es utilizada la TLB (*Translation Look-aside Buffer*)
   ii. ¿ Qué ventajas agrega el uso de ASID (*Address Space Identifier*) en una TLB ?
7. Describa 3 servicios que brinda el núcleo del sistema operativo para el manejo de Entrada/Salida.
8. Describa el problema de hiperpaginación (*thrashing*) y un método para evitarlo.

Problema 2 (35 puntos)

En un sistema operativo experimental se determinó que dada las características de su utilización la mejor estrategia para implementar su sistema de archivos es una que combine la asignación de bloques en forma de lista (Linked Allocation) y la asignación indexada (Indexed Allocation).

Este sistema de archivos cuenta con la siguiente estructura de datos:

```
type block = array [0..4095] of byte; // 4096 bytes
type dir_entry = Record
     name : Array [0..11] of char;     // 12 byte
     type : (file,dir);                // 1 bit
     used : Boolean;                   // 1 bit
     comienzo_archivo: Integer;        // 4 byte
     tamano_archivo: Integer;          // 4 byte
     id_dueno: Permiso;                // 3 byte
     id_grupo: Permiso;                // 3 byte
     i_node_num : Integer;             // 4 byte
     permisos : array [0..13] of bit; // 14 bits
     End;
type i_node = Record
     i_node_num : Integer;           // 16 bits
     used : Boolean;                 // 1 bit
     data: array [1..15] of Integer; // 60 byte
     tope: Integer;                  // 4 byte
     reserved : array[0..6] of bit;  // 7 bits
     End;
```

```
type inode_table = Array [0..max_i_node_on_disk-1] of i_node;
type fat = array [0..max_blocks_in_disk-1] of integer;
type disk = Array [0..max_blocks_in_disk-1] of block;

Var
      TD : inode_table;
      FAT :fat;
      D : disk;
```

Las características de este sistema de archivos son la siguientes:

- Los archivos son almacenados siguiendo una estrategia de asignación en forma de lista tipo FAT (File Allocation Table), donde el campo `comienzo_archivo` de la estructura `dir_entry` indica el comienzo del archivo.
- La información de los directorios es almacenada utilizando una estrategia de asignación indexada directa, donde el atributo `i_node_num` indica el inodo que almacena la información del directorio.
- Las variables TD, FAT y D son globales.
- En la FAT el valor 0 representa "fin de archivo", el -1 "bloque libre" y el -2 indica que ese bloque está siendo utilizado por un directorio.
- Los archivos de tamaño 0 no reservan ningún bloque de disco.
- El inodo 0 almacena la información del directorio raíz.

Por otra parte se dispone de los siguientes procedimientos:

- `Procedure readBlock(block_num: 0..MAX_BLOCKS_ON_DISK-1; Var buff: block, Var ok: Boolean)`

  Lee de disco el bloque pasado como parámetro y retorna el contenido leído en el parámetro de salida **buff**. En el parámetro **ok** se retorna el éxito de la ejecución de la operación.

- `Procedure writeBlock(block_num : 0..MAX_BLOCKS_ON_DISK-1; buff : block, ok : Boolean)`

  Escribe en el bloque pasado como parámetro la información que se encuentra en el parámetro **buff**. En el parámetro **ok** se retorna el éxito de la ejecución de la operación.

Se pide:

1. ¿Cuál es el tamaño máximo que puede tener un archivo en este sistema?. Justifique su respuesta.
2. Implementar una función que renombra un archivo o directorio. El cabezal de la función debe ser el siguiente:

   ```
   Procedure rename(i_padre: int; v_nom: array of char; n_nom: array of char; Var ok: boolean);
   ```

   El parámetro **i_padre** es el inodo del directorio que contiene el archivo o directorio a renombrar, el parámetro **v_nom** es el nombre actual del archivo/directorio y **n_nom** es el nuevo nombre del archivo/directorio. El parámetro **ok** se retorna el éxito de la operación.
3. Implementar la función deleteFile que borre un archivo en el sistema. El cabezal de la función debe ser el siguiente:

```
Procedure deleteFile(i_padre: int; nom: array of char; Var ok: boolean);
```

El parámetro **i_padre** representa el inodo del directorio que contiene el archivo a borrar y el parámetro **nom** representa el nombre del archivo. En el parámetro **ok** se retorna el éxito en la ejecución de la operación.

Solución:

1) Suponemos que tenemos un único archivo, f, en el disco. El disco tendrá el primer inodo asignado al directorio raíz. El directorio raíz deberá tener al menos un bloque asignado donde existirá un dir_entry con el archivo f. Entonces como el disco ya tiene un bloque asignado al directorio raíz, quedan max_blocks_in_disk-1 disponibles para asignar al archivo f. Por otro lado el tamaño de un archivo se almacena en un campo de integer de 4 bytes, entonces no podrán existir archivos más grandes que 2^32-1 bytes.

Por lo que el tamaño máximo (en bytes) que puede tener un archivo es:

min { (max_blocks_in_disk-1) * 4096, 2^32-1 }

2)

Un record dir_entry ocupa 12 + 12 + 6 + 2 = 32 bytes. En total, pueden haber hasta 4096/32 = 128 dir_entry por bloque.

```
Procedure rename(i_padre: int; viejo_nombre: array of char, nuevo_nombre: array of char; ok: boolean) {
	if (!TD[i_padre].used) {
		ok = false;
		return;
	}
	if(nuevo_nombre == viejo_nombre){
		ok = true;
		return;
	}

	dir_entry found_buff[128];
	dir_entry aux_buff[128];
	int found_bloq = -1;
	bool conflicto = false;

	for (int bloq = 0; bloq < TD[i_padre].tope && !conflicto; bloq++) {
		readBlock(TD[i_padre].data[bloq], aux_buff, ok);
		if (!ok) return;
		for (int j=0; j < 128 && !conflicto; j++) {
			if (aux_buff[j].used) {
				if (aux_buff[j].name == viejo_nombre) {
					found_bloq = TD[i_padre].data[bloq];
					found_buff = aux_buff;
					found_buff[j].name = nuevo_nombre;
				} else if (aux_buff[j].name == nuevo_nombre) {
					conflicto = true;
				}// end if
			}// end if
		}// end for
	}// end for
```

```
      if (found_bloque > -1 && !conflicto) {
            writeBlock(found_bloq, found_buff, ok);
      } else {
            ok = false;
      }
}
```

3)

```
Procedure deleteFile(i_padre: int; nom: array of char; Var ok : Boolean) {
  int siguiente, bloq, aux_siguiente;
  bool encontrado = false;
  dir_entry buff[128];

  if (!TD[inodo_padre].used) {
      ok = false;
      return;
  }

  for (bloq=0; bloq<TD[i_padre].tope && !encontrado; bloq++) {
    readBlock(TD[i_padre].data[bloq], buff, ok);

    if (!ok) {
      return;
    } else {
      for (int j=0; j<128 && !encontrado; j++) {
        if (buff[j].used && buff[j].type == file && buff[j].name == nom) {
            encontrado = true;
            siguiente = buff[j].comienzo_archivo;
            buff[j].used = false;
            writeBlock(TD[i_padre].data[bloq], buff, ok);
            if (!ok) return;
        }// end if
      }// end for
    }// end if
  }// end for

  if (encontrado) {
    while (siguiente != 0) {
      aux_siguiente = FAT[siguiente];
      FAT[siguiente] = -1;
      siguiente = aux_siguiente;
    }
  } else {
      ok = false;
  }
}
```

Problema 3 (33 puntos)

Se desea modelar un juego de búsqueda del tesoro con dos equipos. El juego se desarrolla en un edificio con 100 habitaciones y un cofre. Los 10 integrantes del equipo de escondedores se encargan de esconder tesoros en las habitaciones del edificio. Por otro lado los 30 integrantes del equipo de buscadores se encargan de buscarlos y guardalos en el cofre que tiene capacidad para 8 tesoros. En el cofre se pueden guardar hasta 4 tesoros a la vez. El primer integrante del equipo de buscadores que, luego de guardar el tesoro, determina que el cofre está lleno deberá hacer sonar la sirena para llamar al tesorero. El tesorero espera hasta recibir un aviso y entonces contabiliza los premios y luego retira los tesoros del cofre.

Tanto los buscadores como los escondedores estarán continuamente seleccionando habitaciones para buscar y esconder respectivamente.

Los integrantes del equipo de escondedores no podrán estar en una habitación si hay integrantes del equipo de buscadores para que éstos no vean los escondites y viceversa. Además, para evitar peleas sobre los escondites, no podrá haber más de un escondedor a la vez en una habitación.

Se pide: Implementar las tareas buscador, escondedor y tesorero usando semáforos. No se permiten tareas auxiliares.

Se tienen las siguientes funciones auxiliares:

- elegir_hab(): 1..100 — Ejecutada por los jugadores para seleccionar una habitación
- esconder(hab: integer) — Ejecutada por los escondedores para esconder un tesoro. Recibe como parámetro el número de habitación.
- buscar(hab: integer): tesoro — Ejecutada por los buscadores. Retorna un tesoro o NULL si no encuentra nada. Recibe como parámetro el número de habitación.
- guardar(tesoro) — Ejecutada por los buscadores para guardar el tesoro en el cofre.
- sirena() — Ejecutada por los buscadores para hacer sonar la sirena
- contabilizar_y_retirar() — Ejecutada por el tesorero para contabilizar los premios y retirarlos.

Solución

```
mutex: Array[1..100] of Semaforo
ocupado: Array[1..100] of Semaforo
sem_depositar_tesoro: Semaforo
sem_entrega_tesoro: Semaforo
sem_tesorero: Semaforo
mutex_cant_tesoros: Semaforo
cant_buscadores: Array[1..100] of integer
cant_tesoros : integer;

escondedor() {
      while true {
            habitacion = elegir_hab();
            P(ocupado[habitacion])
            esconder(habitacion);
            V(ocupado[habitacion]);
      }
}

buscador() {
      while(true) {
            habitacion = elegir_hab();
            P(mutex[habitacion]);
            cant_buscadores[habitacion]++;
            if(cant_buscadores[habitacion] == 1) {
                  P(ocupado[habitacion]);
            }
            V(mutex[habitacion];

            tesoro = buscar(habitacion);

            P(mutex[habitacion]);
            cant_buscadores[habitacion]--;
            if (cant_buscadores[habitacion] == 0) {
                  V(ocupado[habitacion]);
            }
            V(mutex[habitacion]);

            if (tesoro != null) {
                  P(sem_entrega_tesoro); //Inicializado en +8
                  P(sem_depositar_tesoro); //Inicializado en +4
                  guardar(tesoro);
                  P(mutex_cant_tesoros);
                  cant_tesoros++;
                  if(cant_tesoros == 8) {
                        sirena();
                        V(sem_tesorero);
                        cant_tesoros = 0;
                  }
                  V(mutex_cant_tesoros);
                  V(sem_depositar_tesoro);
            }
      }
}
```

```
tesorero() {
      while (true){
            P(sem_tesorero);
            contabilizar_y_retirar();
            for(int i=1 to 8) {
                  V(sem_entrega_tesoro);
            }
      }
}

main(){
      init(sem_tesorero, 0);
      init(mutex_cant_tesoros, 1);
      init(sem_depositar_tesoro, 4);
      init(sem_entrega_tesoro, 8);
      for i=0 to 100 {
            cant_buscadores[i] = 0
            init_sem(ocupado[i], 1)
            init_sem(mutex[i], 1)
      }
      cant_tesoros = 0;

      cobegin
            tesorero();
            buscador();
            ... // 28 buscadores más
            buscador();
            escondedor();
            ... // 8 escondedores más
            escondedor();
      coend
}
```