Departamento de Arquitectura

Examen de Sistemas Operativos

24 de julio de 2019

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 **4 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 (34 pts)

(a) (3 pts) Describa brevemente qué es la Hiperpaginación. ¿Qué efectos puede tener sobre el Sistema Operativo?

(b) En un sistema de paginación bajo demanda.
   I. (3 pts) Explique brevemente un esquema simple para traducir una dirección virtual a una física.
   II. (3 pts) Enumere los elementos de hardware necesarios para su funcionamiento y la función que cumple cada uno.

(c) (3 pts) Describa brevemente las 4 condiciones necesarias para la ocurrencia de deadlock.
   I. (2 pts) ¿En qué condición se convierten en suficientes?
   II. (2 pts) Elija una de las 4 condiciones y mencione una estrategia para evitar que ocurra.

(d) (5 pts) Se dispone de 10 discos para crear un arreglo de discos. Las especificaciones de diseño de la solución establecen que este arreglo debe tolerar la pérdida de hasta dos discos. ¿Qué esquemas de redundancia son adecuados? Descríbalos muy brevemente justificando su respuesta.

(e) Explique brevemente cómo funciona:
   I. (2 pts) una E/S sincrónica.
   II. (2 pts) una E/S asincrónica.

(f) Describa brevemente qué es:
   I. (3 pts) Discretionary Access Control
   II. (3 pts) Mandatory Access Control

(g) (3 pts) Describa brevemente qué es un Hipervisor de Tipo 1.

Problema 2 (33 pts)

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 con índice 0 (cero).
- En FAT el valor -1 indica el fin de archivo.
- El campo referencias es utilizado para contar la cantidad de enlaces.
- En MB el valor 0 (cero) 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:

(a) (4 pts) Dada la estructura del sistema de archivos planteado, mencione y describa brevemente tres problemas de correctitud que pueden darse.

Solución: Por ejemplo:

- Dos archivos comparten la misma cola de la FAT.
- Archivo circular en la FAT.
- El campo tamaño no se corresponde con la cantidad de bloques asignados al archivo.

(b) (8 pts) Implementar una función que busque un directorio o archivo:
`Function search(cam: array of char): entrada_dir;`
Donde `cam` es la ruta completa al elemento del sistema de archivos buscado. Retorna una entrada de directorio que corresponde al elemento buscado y `null` en caso que la operación no se ejecute con éxito.

Solución:

```
function search(cam: array of char): entrada_dir {
  entrada_dir ref;
  array of char elem;
  int i, j, inodo;
  boolean error, encontre=false;
  array [0..31] of entrada_dir buff;

  if(cam == "") return null;
  if(cam == "/"){
    ref.used = true;
    ref.inodo_num = 0;
    return ref;
  }
  inodo = 0;
  parteCamino(cam, elem, cam);
  while (elem!="") {
    if(IT[inodo].usado) {
      i=0; encontre=false;
      while (!encontre and i<IT[inodo].tope) {
        error=leerBlq(IT[inodo].datos[i], buff);
        if(!error) {
          j=0;
          while(!encontre and j<32) {
            if(buff[j].usado and buff[j].nombre==elem) {
              if (cam=="") {
```

```
              return buff[j];
            } else if(buff[j].tipo==dir) {
              encontre = true;
              inodo = buff[j].inodo_num;
              parteCamino(cam, elem, cam);
            } else return null;
          }
          j++;
        }
      } else return null;
      i++;
    }
    if(!encontre) return null;
  }
 }
 return null;
}
```

(c) (6 pts) Implementar una función que reemplace el contenido a un archivo:
`Function zeros(camArchivo: array of char):boolean;`
Donde `camArchivo` es el camino de un archivo cuyo contenido debe ser sobreescrito totalmente con ceros. Retorna `true` si la operación fue ejecutada con éxito. Esta función no crea archivos.

Solución:

```
function zeros(camArchivo: array of char): boolean {
  int ant;
  bloque buff;
  for(int i=0;i<1024;i++) buff[i]=0;

  entrada_dir archivo = search(camArchivo);
  if(archivo==null) return flase;
  if(archivo.tipo!=file) return false;

  ant = archivo.fat_inicio;
  while(ant!=-1){
    if (!escrBlq(ant, buff)) return false;
    ant = FAT[ant];
  }

  return true;
}
```

(d) (15 pts) Implementar una función que mueva un archivo con la siguiente definición:
`Function move(camOrigen: array of char, nomOrigen: array of char, camDestino: array of char, nomDestino: array of char): boolean;`
Donde `camOrigen` es la ruta completa al directorio donde se encuentra el archivo a mover, `nomOrigen` es el nombre del archivo a mover en el directorio origen, `camDestino` es la ruta completa al directorio destino y `nomDestino` es el nombre para el archivo en el destino. Retorna un booleano indicando si la operación fue realizada con éxito. Esta operación no mueve directorios

ni crea directorios.

Solución:

```
Function move(camOrigen: array of char, nomOrigen: array of char;
camDestino: array of char, nomDestino: array of char):boolean {
  array [0..31] of entrada_dir bufforig, buffdst;
  int iorig, jorig, idst, jdst;
  int ifree, jfree;
  int i,j;
  boolean encontre, error;

  if(nomOrigen=="" or nomDestino=="") return false;

  entrada_dir dirOrigen = search(camOrigen);
  entrada_dir dirDestino = search(camDestino);

  if(dirOrigen==null or dirDestino==null) return false;
  if(dirOrigen.tipo!=dir or dirDestino.tipo!=dir) return false;

  iorig=0;
  encontre = false;
  while (!encontre and i<IT[dirOrigen.inodo].tope) {
    error=leerBlq(IT[dirOrigen.inodo].datos[iorig], bufforig);
    if(!error) {
      jorig=0;
      while(!encontre and jorig<32) {
        if(bufforig[jorig].usado and bufforig[jorig].tipo==file
        and bufforig[jorig].nombre==nomOrigen) {
          encontre=true;
        } else
          jorig++;
      }
    } else
      return false;
    if(!encontre)
      iorig++;
  }
  if(!encontre) return false;

  ifree=-1; jfree=-1;
  idst=0;
  while (i<IT[dirDestino.inodo].tope) {
    error=leerBlq(IT[dirDestino.inodo].datos[idst], buffdst);
    if(!error) {
      jdst=0;
      while(j<32) {
        if(buffdst[jdst].usado buffdst[jdst].nombre==nomDestino) {
          return false;
        } else
          if(!buffdst[jdst].usado and ifree==-1) {
```

```
            ifree=idst;
            jfree=jdst;
          }
        jdst++;
      }
    } else
      return false;
    idst++;
  }

  if(ifree==-1) {
    if(IT[dirDestino.inodo].tope<8) {
      i=0;
      while(mapa_bits[i]!=0 and i<MAX_BLOQUES-1) i++;
      if(mapa_bits[i]!=0)
        return false;
      else {
        for(j=0;j<32;j++) buffdst[j].usado=false;
        ifree=IT[dirDestino.inodo].tope;
        jfree=0;
        mapa_bits[i]=1;
        TI[dirDestino.inodo].tope++
        TI[dirDestino.inodo].datos[ifree]=i;
      }
    } else return false;
  }

  buffdst[jfree].usado = true;
  buffdst[jfree].nombre = nomDestino;
  buffdst[jfree].tipo = bufforig[iorig].tipo;
  buffdst[jfree].inodo_num =  bufforig[iorig].inodo_num;
  buffdst[jfree].fat_inicio =  bufforig[iorig].fat_inicio;
  buffdst[jfree].tamaño =  bufforig[iorig].tamaño;
  buffdst[jfree].reservado =  bufforig[iorig].reservado;

  bufforig[iorig].usado = false;

  if(escrBlq(TI[dirDestino.inodo].datos[ifree], buffdst))
    return false;
  return (!escrBlq(IT[dirOrigen.inodo].datos[iorig], bufforig));
}
```

Problema 3 (33 pts)

En una planta nuclear trabajan 4 empleados encargados de controlar el núcleo del reactor. La planta cuenta con una sala de control donde hay 4 paneles que son utilizados por estos empleados para realizar los controles necesarios. Cada empleado necesita 2 paneles de forma exclusiva para realizar sus tareas. Los paneles son utilizados por cada empleado de la siguiente forma:

- El empleado 0 utiliza el panel 0 y 1.
- El empleado 1 utiliza el panel 1 y 2.

- El empleado 2 utiliza el panel 2 y 3.
- El empleado 3 utiliza el panel 3 y 0.

En la planta trabaja también un supervisor que cada cierto tiempo ingresa a la sala de control para verificar que los paneles estén funcionando correctamente. El supervisor supervisa de a dos paneles por vez. Mientras el supervisor realiza su tarea, los empleados no podrán usar los paneles que están siendo supervisados. En caso de encontrar alguna anomalía, el supervisor debe reparar los paneles. Para esto, el supervisor debe tener acceso exclusivo a la sala por lo que debe esperar a que todos los empleados terminen de trabajar y dejen la sala. El supervisor tiene prioridad sobre los empleados a la hora de realizar sus tareas.

Los empleados y el supervisor alternan sus tareas en la sala de control con otras tareas de la planta.

Se dispone de los siguientes métodos:

- `void realizar_controles(int p1, int p2)`: Ejecutado por un empleado para controlar el núcleo del reactor utilizando los paneles `p1` y `p2`.
- `int que_verif()`: Ejecutado por el supervisor para obtener un panel para verificar. Garantiza que dos invocaciones sucesivas genera dos números diferentes.
- `bool verif_paneles(int p1, int p2)`: Ejecutado por el supervisor para verificar los paneles `p1` y `p2`. Devuelve `true` si no se encuentran anomalías en los paneles y `false` en caso contrario.
- `void reparar_paneles()`: Ejecutado por el supervisor para reparar los paneles.
- `void otras_tareas()`: Ejecutado por los empleados y el supervisor para realizar otras tareas que no involucren la sala de control.

Se pide modelar empleados y supervisor utilizando monitores.

Solución:

```
void empleado(int nroEmp){
  int panel1 = nroEmp;
  int panel2 = (nroEmp + 1) % 4;
  // intercambio para siempre pedir primero el más chico
  ordenar(panel1, panel2);
  while (true){
    monitor.entrar_sala_emp();
    monitor.pedir_panel_emp(panel1);
    monitor.pedir_panel_emp(panel2);
    realizar_controles(panel1, panel2);
    monitor.liberar_panel_emp(panel2);
    monitor.liberar_panel_emp(panel1);
    monitor.liberar_sala_emp();
    otras_tareas();
  }
}

void supervisor(){
  while (true){
    int panel1 = que_verif();
    int panel2 = que_verif();
```

```
    // intercambio para siempre pedir primero el más chico
    ordenar(panel1, panel2);
    monitor.pedir_panel_sup(panel1);
    monitor.pedir_panel_sup(panel2);
    bool ok = verif_paneles(panel1, panel2);

    if (!ok) {
      monitor.evacuar_empleados();
      reparar_paneles();
      monitor.liberar_sala_supervisor();
    }
    monitor.liberar_panel_sup(panel2);
    monitor.liberar_panel_sup(panel1);

    otras_tareas();
  }
}

function ordenar(int &p1, int &p2){
  if (p1 < p2){
    int temp = p2;
    p2 = p1;
    p1 = temp;
  }
}

monitor sala_control{
  condition cond_sala_evacuada_supervisor;
  condition cond_supervisor_espera_sala_vacia;
  condition cond_emp_espera_panel[4], cond_sup_espera_panel[4];

  bool sala_evacuada_supervisor = false;
  int empleados_sala = 0;
  bool paneles_libres[4] = { true, true, true, true };
  int empleados_esperando_panel[4] = { 0, 0, 0, 0 };
  bool supervisor_esperando_panel[4] = { false, false, false, false };

  procedure entrar_sala_emp() {
    if (sala_evacuada_supervisor){
      cond_sala_evacuada_supervisor.wait();
      cond_sala_evacuada_supervisor.signal();
    }
    empleados_sala++;
  }

  procedure salir_sala_emp() {
    empleados_sala--;

    if (sala_evacuada_supervisor and empleados_sala==0)
```

```
      cond_supervisor_espera_sala_vacia.signal();
  }

  procedure pedir_panel_emp(int p) {
    if (!paneles_libres[p]){
      cond_emp_esperando_panel[p].wait;
    }
    paneles_libres[p] = false;
  }

  procedure liberar_panel_emp(int p){
    paneles_libres[p] = true;

    if (supervisor_esperando_panel[p]) // prioridad supervisor
      cond_sup_espera_panel[p].signal();
    else
      cond_emp_esperando_panel[p].signal();
  }

  procedure pedir_panel_sup(int p) {
    if (!paneles_libres[p]){
      supervisor_esperando_panel[p]=true;
      cond_sup_espera_panel[p].wait;
      supervisor_esperando_panel[p]=false;
    }
    paneles_libres[p] = false;
  }

  procedure liberar_panel_sup(int p) {
    paneles_libres[p] = true;
    cond_emp_esperando_panel[p].signal();
  }

  procedure evacuar_empleados(){
    sala_evacuada_supervisor = true;

    if (empleados_sala>0) {
      cond_supervisor_espera_sala_vacia.wait();
    }
  }

  procedure liberar_sala_supervisor(){
    sala_evacuada_supervisor = false;
    cond_sala_evacuada_supervisor.signal();
  }
}

begin
  cobegin
    empleado(0);
```

```
    empleado(1);
    empleado(2);
    empleado(3);
    supervisor();
  coend;
end
```