Departamento de Arquitectura

Examen de Sistemas Operativos

10 de diciembre de 2018

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 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.

Pregunta 1 (32 pts)

Conteste justificando brevemente cada una de las preguntas.

(a) (4 pts) Describa 3 métodos para la interacción entre la controladora y un dispositivo durante una operación de Entrada/Salida.

(b) Sobre los llamados a sistema:

i. (1 pt) ¿Cuál es su propósito?

ii. (3 pts) Dado el siguiente código escrito en lenguaje C

```
main ()
{
    char * buffer;
    int fd_r, fd_w, count;
    buffer = (char *) malloc(256*sizeof(char));
    if ((fd_r = openread("read.txt") == -1) || (fd_w = openwrite("write.txt") == -1))
        exit 1;
    while (count = read(fd_r,buffer,256))
        write(fd_w,buffer,count);
    closeread(fd_r);
    closewrite(fd_w);
    exit 0;
}
```

¿Cuáles invocaciones a procedimientos son un llamado a sistema (system call) o desembocan en uno?

(c) (4 pts) Compare ventajas/desventajas entre un sistema operativo monolítico frente a uno en capas.

(d) (4 pts) En un planificador RR, qué implicancias tiene agrandar o disminuir el tiempo de quantum.

(e) (4 pts) En general los sistemas operativos brindan un servicio de buffering para el subsistema de entrada/salida, ¿qué motiva brindar este servicio?

(f) (4 pts) ¿Qué beneficio brinda la multiprogramación? Discuta según el sistema sea monoprocesador o multiprocesador.

(g) (4 pts) ¿Por qué en el PCB (Process Control Block) se reserva lugar para los registros de la CPU?

(h) (4 pts) En un sistema con paginación. ¿Qué ventaja/desventaja tiene agregar más jerarquías de tablas de página?

Pregunta 2 (34 pts)

Un sistema de archivos utiliza una estrategia indexada multinivel de dos niveles y un mapa de bits para administrar el espacio libre del disco. En la estructura indexada se dispone de 7 bloques de primer nivel y 1 bloque de indirección simple. 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..1023] of byte;  // 1024 bytes
type mapa_bits = array [0..MAX_BLOQUES-1] of bit;
type entrada_dir = Record
    usado : boolean;                    // 1 bit
    nombre : array [0..123] of char;    // 124 bytes
    es_dir : boolean;                   // 1 bit
    inodo_num : int16;                  // 2 bytes
    permisos : array [0..13] of bit;    // 14 bits
End; // 128 bytes

type inodo = Record
    usado : boolean;                    // 1 bit
    inodo_num : int16;                  // 2 bytes
    es_dir : boolean;                   // 1 bit
    tamaño : int32;                     // 4 bytes
    directo : array [0..6] of int16;    // 14 bytes
    directo_tope : int16;               // 2 bytes
    indirecto : int16;                  // 2 bytes
    indirecto_tope : int16;             // 2 bytes
    reservado : array [0..45] of bit;   // 46 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 es el directorio raíz.
- las entradas de los directorios son almacenadas utilizando un array de entradas.
- en el mapa de bits, los bloques libres son marcados con valor 1 (true).

Se dispone de los siguientes procedimientos:

- `Procedure leerBloque(d : disco; bloque_num : 0..MAX_BLOQUES-1; var buffer : array [0..1023] of byte; var ok : boolean);`
  Lee desde el disco `d` el bloque con índice `bloque_num` en la variable `buffer`. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.
- `Procedure escribirBloque(d : disco; block_num : 0..MAX_BLOQUES-1; buffer : array [0..1023] of bytes; var ok : boolean);`
  Escribe en el bloque con índice `bloque_num` del disco `d` la información que se encuentra en la variable `buffer`. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.
- `Procedure obtenerNombreDirectorio(camino : array of char; var dir : array of char);`
  Retorna en el parámetro `dir` el nombre del directorio que contiene el archivo referenciado en el parámetro `camino`.
  Por ejemplo: `obtenerNombreDirectorio('/home/sistoper/a.txt') = '/home/sistoper'`
- `Procedure obtenerNombreBase(camino : array of char; var base : array of char);`
  Retorna en el parámetro `base` el nombre del archivo o directorio referenciado en el parámetro `camino`.
  Por ejemplo: `obtenerNombreBase('/home/sistoper/a.txt') = 'a.txt'`

Se pide:

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

(b) (5 pts) Implemente una función que encuentre y retorne un bloque libre en el disco. La función a implementar tiene la siguiente firma:
`Procedure buscarBloqueLibre(var bloque : 0..MAX_BLOQUES-1; var ok : boolean);`
El parámetro `bloque` es el número de bloque libre encontrado. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.

(c) (12 pts) Implemente una función que retorne en la variable `inodo_num` el número de inodo correspondiente al camino absoluto `camino`. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito. La función a implementar tiene la siguiente firma:
`Procedure obtenerInodo(camino : array of char; var inodo_num : integer; var ok : boolean);`

(d) (12 pts) Implemente una funcion que dado un inodo perteneciente a un directorio, retorne la cantidad de archivos y directorios que se encuentran dentro de ese inodo.
`Procedure obtenerCantidad(inodo_num : integer; var ok : boolean, var cantidad_archivos: integer, var cantidad_directorios: integer);`
Nota: La funcion **NO** debe ser implementada de forma recursiva.

Solución: a) Un archivo no puede tener más de 2^{32} bytes porque está limitado por el campo tamaño del inodo, i.e. no puede tener más de 4 GB. Tampoco puede tener más de 65534 bloques porque es el máximo que soporta el disco menos la raíz, i.e. no puede tener más de 67 MB. Además, la cantidad máxima de bloques que se le pueden asignar a un archivo son 7 (directos) más la cantidad de bloques

indirectos. Cada referencia a un bloque requiere un `int16` (2 bytes), es decir que en el bloque indirecto (1 KB) pueden haber 512 referencias a bloques. En total, un archivo puede tener $512 + 7$ bloques asignados. Es decir que su tamaño máximo podrá ser 519 KB (el mínimo entre todos sus topes).

b)

```
Procedure buscarBloqueLibre(var bloque: 0..MAX_BLOQUES-1; var ok: boolean) {
  ok = false;
  for (int i=0; i<MAX_BLOQUES; i++) {
    if (MB[i]==1) {
      bloque = i;
      ok = true;
      return;
    }
  }
}
```

c)

```
Procedure buscarEntrada(nombre: array of char; bloqueid: int16;
  var inodo_num: integer; var ok: boolean) {
  entrada_dir buffer[8];
  inodo_num = -1;
  ok = true;

  leerBloque(D, bloqueid, buffer, ok);
  if (!ok) return;

  for (int j=0; j<8; j++) {
    if (buffer[j].usado and buffer[j].nombre == nombre) {
      inodo_num = buffer[j].inodo_num;
      return;
    }
  }
}

Procedure obtenerInodo(camino : array of char; var inodo_num : integer;
  var ok : boolean) {
  string pila[MAX_INODOS];
  int pilaCant = 0;
  int16 indblq[512], foundinode;
  string nombre,resto;
  bool encontrado = true;
  ok = true;
  inodo_num = 0;

  if (camino == "") {
    ok = false;
    return;
  }
  if (camino == "/") return;
```

```
  do {
    obtenerNombreBase(camino, nombre);
    pila[pilaCant] = nombre;
    pilaCant++;
    obtenerNombreDirectorio(camino, resto);
    camino = resto;
  } while (resto != "")

  while (pilaCant > 0 && ok) {
    encontrado = false;
    for (int i=0; i<IT[inodo_num].directo_tope && !encontrado; i++) {
      buscarEntrada(pila[pilaCant-1], IT[inodo_num].directo[i], foundinode, ok);
      if (!ok) return;
      if (foundinode>=0) encontrado=true;
    }
    if (IT[inodo_num].indirecto_tope>0 && !encontrado) {
      leerBloque(D, IT[inodo_num].indirecto, indblq, ok);
      if (!ok) return;
      for (int i=0; i<IT[inodo_num].indirecto_tope && !encontrado; i++) {
        buscarEntrada(pila[pilaCant-1], indblq[i], foundinode, ok);
        if (!ok) return;
        if (foundinode>=0) encontrado=true;
      }
    }

    if (encontrado) {
      pilaCant--;
      inode_num = foundinode;
      if (pila[pilaCant] > 0 and IT[inodo_num].es_dir != true) {
        ok = false;
      }
    } else {
      ok = false;
    }
  }
}
```

d)

```
Procedure obtenerTamanio(inodo_num: integer; var ok: boolean,
  var cantidad_archivos: integer, var cantidad_directorios: integer) {
  int16 pila[MAX_INODOS];
  int pilaCant = 0;
  int16 indblq[512];
  entrada_dir buffer[8];

  if (!IT[inodo_num].usado || !IT[inodo_num].es_dir) {
    ok = false;
    return;
  }

  cantidad_archivos = 0;
```

```
  cantidad_directorios = 0;

  do {
    for (int i=0; i<IT[inodo_num].directo_tope; i++) {
      leerBloque(D, IT[inodo_num].directo[i], buffer, ok);
      if (!ok) return;
      for (int j=0; j<8; j++) {
        if (buffer[j].usado) {
          if (buffer[j].es_dir) {
            cantidad_directorios++;
            pila[pilaCant]=buffer[j].inodo_num;
            pilaCant++;
          } else {
            cantidad_archivos++;
          }
        }
      }
    }
    if (IT[inodo_num].indirecto_tope>0) {
      leerBloque(D, IT[inodo_num].indirecto[j], indblq, ok);
      if (!ok) return;
      for (int k=0; k<IT[inodo_num].indirecto_tope; k++) {
        leerBloque(D, indblq[k], buffer, ok);
        if (!ok) return;
        for (int l=0; l<8; l++) {
          if (buffer[l].usado) {
            if (buffer[l].es_dir) {
              cantidad_directorios++;
              pila[pilaCant]=buffer[l].inodo_num;
              pilaCant++;
            } else {
              cantidad_archivos++;
            }
          }
        }
      }
    }

    if (pilaCant > 0) inode_num = pila[pilaCant-1];
    pilaCant--;
  } while (pilaCant >= 0)
}
```

Pregunta 3 (34 pts)

Se desea modelar una chivitería que tiene 5 vendedores, y vende chivitos con carne y dos aderezos a elección entre 10 disponibles. Los aderezos están numerados del 1 al 10.

Los clientes que ingresan se forman en dos filas (embrazadas y otros) por orden de llegada, hasta que un vendedor pueda atender al primero, y así sucesivamente. Las embarazadas tienen prioridad. La cantidad máxima de clientes no es conocida.

La chivitería tiene un recipiente por cada aderezo. Cada recipiente sólo puede ser usado por un vendedor a la vez. El vendedor debe usar simultáneamente los recipientes de los dos aderezos requeridos al momento de armar el chivito para poder mezclarlos.

Se pide: Modelar en Ada las tareas Cliente y Vendedor.

Se dispone de los siguientes procedimientos:

`armar_chivito(integer, integer)`
: Ejecutado por el vendedor para armar el chivito

`servir_chivito()`
: Ejecutado por el vendedor para darle el chivito al cliente

`estoy_embarazada():boolean`
: Ejecutado por el cliente para saber si está embarazada

`que_aderezo():integer`
: Ejecutado por el cliente para obtener aderezo. El cliente la llama dos veces y nunca retorna repetidos para ese cliente.

`comer()`
: Ejecutado por el cliente para comer el chivito

Nota: Se pueden utilizar tareas auxiliares

Solución:

```
task type Cliente;

task body Cliente is
  aderezo1, aderezo2, vendedor: integer;
begin
  aderezo1:=que_aderezo();
  aderezo2:=que_aderezo();

  if estoy_embarazada() then
    Filas.EMBARAZADAS(vendedor);
  else
    Filas.OTROS(vendedor);
  end if;

  V(vendedor).PEDIDO(aderezo1,aderezo2);
  comer();
end Cliente;

task type Vendedor is
  entry SET_ID(I: integer);
  entry PEDIDO(aderezo1: integer, aderezo2: integer);
```

```
end Vendedor;

task body Vendedor is
  id: integer;
begin
  accept SET_ID(I: integer) do
    id:=I;
  end SET_ID;
  loop
    Filas.VENDEDOR_LIBRE(id);
    accept PEDIDO(aderezo1: integer, aderezo2: integer) do
      if aderezo1 < aderezo 2 then
        Aderezos(aderezo1).USAR;
        Aderezos(aderezo2).USAR;
      else
        Aderezos(aderezo2).USAR;
        Aderezos(aderezo1).USAR;
      end if
      armar_chivito(aderezo1,aderezo2);
      Aderezos(aderezo1).LIBERAR;
      Aderezos(aderezo2).LIBERAR;
      servir_chivito();
    end PEDIDO;
  end loop;
end Vendedor;

task Filas is
  entry EMBARAZADAS(vendedor: integer);
  entry OTROS(vendedor: integer);
  entry VENDEDOR_LIBRE(id: integer);
end Filas;

task body Filas is
begin
  loop
    accept VENDEDOR_LIBRE(id: integer) do
    select
      when EMBARAZADAS'count=0 =>
        accept OTROS(vendedor: out integer) do
          vendedor:=id;
        end OTROS;
      or
        accept EMBARAZADAS(vendedor: out integer) do
          vendedor:=id;
        end EMBARAZADAS;
      end select;
    end VENDEDOR_LIBRE;
  end loop;
end Filas;

task type Aderezo is
```

```
  entry USAR();
  entry LIBERAR();
end Aderezo;

task body Aderezo is
begin
  loop
    accept USAR;
    accept LIBERAR;
  end loop;
end Aderezo;

V: array(1..5) of Vendedor;
Aderezos: array(1..10) of Aderezo;
H: integer;

begin
  for H in 1..10 loop
    V(H).SET_ID(H);
  end loop;
end;
```