Departamento de Arquitectura

Examen de Sistemas Operativos

19 de febrero de 2020

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 **3 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 (26 pts)

Conteste justificando cada una de sus respuestas.

(a) I. (4 pts) Explique los conceptos de planificador y despachador. Debe mencionar además las tareas principales de cada uno así como las propiedades deseables para cada uno de estos.

II. (2 pts) Analice el comportamiento de un sistema con planificador Round Robin en el que el *quantum* está definido con un valor mayor al tiempo de la tarea más larga.

(b) I. (2 pts) Explique con un ejemplo el funcionamiento de un hipervisor que ejecute en modo usuario.

II. (2 pts) Realice una comparación de performance de los hipervisores de tipo 1 y tipo 2.

(c) I. (3 pts) Explique detalladamente el concepto de paginación.

II. (3 pts) Muestre con un ejemplo el direccionamiento en paginación con soporte de TLB. Debe detallar todas las secciones del direccionamiento así como mencionar el funcionamiento de todas las estructuras que utilice, especificando el rol de la TLB en dicho mecanismo.

(d) I. (3 pts) Compare las técnicas de planificación de disco SCAN y C-SCAN.

II. (3 pts) Diagrame la planificación con SCAN y C-SCAN para una instancia con las siguientes características: el cabezal comienza posicionado en la pista 60. La cola de solicitudes tiene las pistas: 100 - 198 - 20 - 98 - 8 - 55 - 44.

(e) (4 pts) Mencione, describa y compare los métodos de acceso a los registros de una controladora de entrada/salida.

Problema 2 (41 pts)

(a) (11 pts) Se tiene un controlador para entrar en una sala de servidores. A la sala entran a realizar mantenimiento tanto técnicos de software (TS) como de hardware (TH) y sólo pueden entrar 5 personas al mismo tiempo. los TH tienen prioridad para entrar por sobre los TS, sin embargo dentro de cada sub grupo no existe ningún orden especificado.

Se pide: implemente utilizando semáforos el controlador descrito. Debe implementar las funciones TS y TH. Dispone de la función: `realizar_mantenimiento()`.

Solución:

```
semaphore sala, try, mutexTS, mutexTH;
TS(){
    p(mutexTS);
    p(try);
    p(sala);
    v(try);
    v(mutexTS);
    Realizar_mantenimiento();
    v(sala);
}
TH(){
    p(mutexTH);
    cantTH++;
    if(cantTH == 1)
        p(try);
    v(mutexTH);
    p(sala);
    p(mutexTH);
    cantTH--;
    if(cantTH == 0)
        v(try);
    v(mutexTH);
    Realizar_mantenimiento();
    v(sala);

}
main(){
    init(try, 1);
    init(mutexTH, 1);
    init(mutexTS, 1);
    init(sala, 5)
    int cantTH = 0;
    Cobegin
        TS(); TS(); ... TH(); TH();
    Coend
}
```

(b) (14 pts) Se quiere desarrollar un software que permita distribuir el trabajo entre dos grupos de TI, desarrollo y verificación. Los integrantes del grupo de desarrollo implementan módulos de software y en la medida que culminan dejan los módulos para su verificación. El grupo de verificación

valida, en la medida que tenga módulos disponible, los módulos. Por reglas de la empresa no se puede tener más de 20 módulos disponibles para verificar. En caso de llegar a esta situación, el jefe pasará todos los módulos disponibles a un centro dedicado a la temática y esta es su única tarea. El software además, debe llevar los indicadores de producción, cuántos módulos se implementaron, cuántos se validaron positivamente, cuántos negativamente y cuantos se pasaron al centro especializado.

Se dispone de los siguientes procedimientos y funciones:

- `implementar()` : Función de los desarrolladores, devuelve un módulo.
- `MLPV(m)` : Procedimiento que inserta el módulo `m` en la estructura de módulos listos para verificar.
- `verificar(m)` : Función que verifica el módulo `m`. Devuelve verdadero si se valido el módulo y falso en caso contrario.
- `obtener()` : Función que devuelve el primer módulo disponible para verificar. Si no hay módulos la función falla.
- `a_centro(m)` : Procedimiento invocado por el jefe que entrega al centro especializado el módulo `m` a verificar.

Se pide: implemente con monitores la realidad anterior. Debe incluir los procedimientos Desarrollador, Tester y Jefe. Además debe implementar una función que retorne cada uno de los indicadores en el orden definido en la letra.

Solución:

```
//  Semantica mesa

void desarollador(){
    while(true){
        m = implementar();
        empresa.terminar(m);
    }
}
void verificador(){
    while(true){
        m = empresa.recibir();
        resultado = validar(m);
        empresa.evaluar(resultado);
    }
}
void jefe(){
        empresa.trabajoJefe()
}

monitor empresa{
    int cantModulos=0;
    int cantPendientes=0;
    int cantTercerizados=0;
    int aprobados=0;
    int rechazados=0;
    condition esperandoModulo, esperandoJefe;
    bool lleno = false;
```

```
    void trabajoJefe(){
        if(cantPendientes<20)
            esperandoJefe.wait();
        while(true){
            cantPendientes = 0;
            lleno = false;
            cantTercerizados = cantTercerizados + 20;
            for(int i = 1; i < 21, i++){
                m = obtener();
                tercerizar(m);
            }
            esperandoEspacio.signal();
            esperandoJefe.wait();
        }

    }

    void terminar(modulo m){
        if(cantPendientes == 20){
            esperandoJefe.signal();
            esperandoEspacio.wait();
        }
        cantModulos++;
        cantPendientes++;
        MVLP(m);
        esperandoModulo.signal();

    }
    modulo recibir(){
        if(cantPendientes == 0)
            esperandoModulo.wait();
        cant--;
        return obtener();
    }
    void evaluar(bool resultado){
        if(resultado)
            aprobados++;
        else
            rechazados++
    }
}
```

(c) (16 pts) Se desea tener un sistema para simplificar el orden de atención en las muestras de exámenes. Cada examen tiene un número que lo identifica y consta de tres ejercicios. Para ver los exámenes los estudiantes entran al salón en orden de llegada., donde hay espacio para seis estudiantes. Una vez dentro del salón los ejercicios se muestran en orden, es decir, primero se responden las dudas del ejercicio 1, luego del ejercicio 2 y por último del ejercicio 3. Los docentes que muestran cada ejercicio atenderán, entre los estudiantes que están esperando por ese ejercicio dentro del salón, al primero que ingresó.

Se pide: implementar usando mailboxes la realidad anterior. Debe especificar la semántica de los mailboxes utilizados. Si utiliza estructuras de datos debe especificar su semántica pero no es necesario que las implemente. Puede asumir que existe al menos un docente para cada ejercicio.

Dispone de los siguientes procedimientos y funciones:

- `quiero_ver(ex,ej)` : Función recibe un examen `ex` y un ejercicio `ej` y retorna si el estudiante quiere que se muestre el ejercicio.
- `muestra(ex,ej)` : Procedimiento que muestra el ejercicio `ej` del examen `ex`.

Solución:

```
mailbox of int salon;
mailbox of int[6] numero;
mailbox of array of int[6] mutex;
mailbox of nill[3] bloqueo;
mailbox of nill[6] termine;
mailbox of int salir;
mailbox of int fila;


estudiante(int numExamen){
    receive(salon,id)
        for(int i = 1; i < 4; i++){
            if(quiero_ver(numExamen,i)){
                send(numero[id], numExamen);
                receive(mutex, lista)
                lista[id] = i;
                send(mutex, lista)
                send(bloqueo[i], nill);
                receive(termine[id], msj);
            }
        }
        send(salir, id);
}

admin(){
    int[7] esperando;
    for (int i = 1; i < 7; i++){
        send(salon, i);
        send(fila,i);
        esperando[i] = 0;
    }
    send(mutex,esperando);
    while(true){
        receive(salir, m);
        receive(numero[m], msj)
        receive(mutex, lista)
        for (int i = 1; i < 7; i++){
            receive(fila,c);
```

```
            if(c != m)
                send(fila,c);
        }
        send(fila, m);
        send(mutex, lista);
        send(salon,m);
    }
}

docente(int numEjercicio){
    int numExamen, idEstudiante;
    while(true){
        bool encontre = false;
        receive(bloqueo[numEjercicio], msj);
        receive(mutex,esperando);
        for (int i = 1; i < 7; i++){
            receive(fila, n)
            if(!encontre && esperando[n] == numEjercicio){
                esperando[n] = 0;
                idEstudiante = n;
                receive(numero[n],numExamen)
                encontre = true;
            }
            send(fila,n)
        }
        send(mutex,esperando);
        muestra(numExamen,numEjercicio);
        send(termine[idEstudiante], nill)
    }
}

main(){
    Cobegin
        admin();
        docente(1); docente(2); docente(3); docente(1)...
        estudiante(1)...estudiante(n)
    Coend
}
```

Problema 3 (33 pts)

Se tiene un sistema de archivos que utiliza una estrategia indexada simple como muestra la siguiente estructura de datos:

```
1   const MAX_BLOQUES = 65536;
2   const MAX_INODOS = 8192;
3   type entrada_dir = Record
4     usado : boolean;                  // 1 bit
5     nombre : array [0..22] of char;   // 23 bytes
6     tipo : (file, dir);               // 1 byte
7     inodo_num : int;                  // 2 bytes
8     size : int;                       // 2 bytes
9     ext: array[0..3] of Char;         // 3 bytes
10    reservado : array [0..6] of bit; // 7 bits
11  End; // 32 bytes
12
13  type inodo = Record
14    usado : boolean;                  // 1 bit
15    inodo_num : int;                  // 2 bytes
16    datos : array [0..8] of int;      // 18 bytes
17    tope : int;                       // 2 bytes
18    reservado : array [0..6] of bit; // 7 bits
19    permisos: int;                    // 2 bytes
20  End; // 25 bytes
21
22  type bloque = array [0..1023] of byte; // 1024 bytes
23  type mapa_bits = array [0..MAX_BLOQUES-1] of bit;
24  type inodo_tabla = array [0..MAX_INODOS-1] of inodo;
25
26  var
27    IT : inodos_tabla;
28    MB : mapa_bits;
```

Notas generales:

- Las variables IT, y MB son globales.
- El directorio raíz es el inodo con índice 0 (cero).
- 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 `true` en caso de que la operación se ejecute con éxito y `false` en caso contrario.
- `Procedure escrBlq(blq_num : int, buff : bloque) : boolean`
  Escribe en el bloque `blq_num`, pasado como parámetro, la información que se encuentra en el parámetro `buff`. Retorna `true` en caso de que la operación se ejecute con éxito y `false` en caso contrario.
- `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('/', '', '')
```

- `Procedure bloque(MB: mapa_bits, IT: inodos_tabla) : int`
  Retorna el índice del primer bloque libre en el mapa de bits MB.

Se pide:

(a) (4 pts) Dada la estructura del sistema de archivos planteado, justifique la cantidad máxima de archivos o directorios que se pueden almacenar: en el sistema y en el directorio raíz.

(b) (7 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 buscado. Retorna una entrada de directorio que corresponde al elemento buscado y `null` en caso que la operación no se ejecute con éxito.

Solución:

```
entrada_dir search(char* cam){
  entrada_dir retorno;
  char* base;
  int i, j, inodo;
  bool encontre = false;
  entrada_dir[32] buff;


  if(cam == "")
    return null;
  if (cam == "/"){
    retorno.tipo = dir;
    retorno.usado = true;
    retorno.inodo_num = 0;
    return retorno;
  }
  inodo = 0;
  parteCamino(cam, base, cam);
  while(elem != ""){
    if(!IT[inodo].usado || IT[tipo != dir])
      return null;
    i=0;
    encontre = false;
    while(!encontre && i<IT[inodo].tope){
      if(!leerBlq(IT[inodo].datos[i], buff)){
        j = 0;
        while(!encontre && j<32){
            string name;
            if(buff[j].ext != null)
                name = buff[j].nombre + "." + buff[j].ext
            else
                name = buff[j].nombre
          if(buff[j].usado and name == base){
            if(cam == "")
              return buff[j];
            encontre = true;
```

```
            inodo = buff[j].inodo_num;
            parteCamino(cam, base, cam);
          }
          j++;
        }
      }
      i++;
    }
    if(!encontre)
      return null;
  }
  return null
}
```

(c) (10 pts) Implementar una función que renombre un archivo o directorio:
`Function rename(cam: array of char, nombre: array of char, ext: array of char, nuevoNombre: array of char, nuevaExt: array of char): ok;`
Si `ext == null`, el tipo a renombrar es Directorio, en caso contrario es del tipo Archivo. `cam` es la ruta completa al directorio del sistema que contiene el elemento a renombrar. Retorna `true` en caso de que la operación se ejecute con éxito y `false` en caso contrario.

Solución:

```
bool rename(char* cam, char* nombre, char* ext, char* nuevoNombre,
char* nuevaExt){
  entrada_dir padre = search(cam);
  if(padre == null || padre.tipo == archivo)
    return false;
  entrada_dir[32] buff, buffEsc;
  int i=0, j=0, iesc, jesc;
  bool encontre = false;
  while(i<IT[padre.inodo_num].tope){
    if(leerBlq(IT[padre.inodo_num].datos[i],buff)){
      j=0;
      while(j<32){
          if(buff[j].nombre == nuevoNombre && buff[j].ext ==
          nuevaExt)
              return false;
        if(!encontre && buff[j].nombre == nombre && buff[j].ext
        == ext){
          encontre = true;
          jesc = j;
          iesc = i;
          buffEsc = buff;
        }
      }
    }

  }
  if(!encontre)
```

```
    return false;
  buff[j].nombre = nuevoNombre;
  buff[j].ext = nuevaExt;
  if(!escrBlq(IT[padre.inodo_num].datos[i],buff))
    return false;
  return true;

}
```

(d) (12 pts) Implementar una función que mueva completamente un archivo o directorio:
`Function move(cam: array of char, nombre: array of char, ext: array of char, camDestino: array of char): ok;`
Si `ext == null`, el tipo es Directorio, en caso contrario es de tipo Archivo. `cam` es la ruta completa al directorio del sistema donde se encuentra el elemento de nombre: `nombre`. Retorna `true` en caso de que la operación se ejecute con éxito y `false` en caso contrario. Puede asumir que para el caso de un directorio, no se invocará con un `camDestino` contenido en él.

Solución:

```
bool move(cam, char* nombre, char* ext, char* camDestino){
  entrada_dir[32] bufforig , buffdst;
  int iorig , jorig , idst , jdst;
  int ifree , jfree;
  int i,j;
  bool encontre;

  if(nombre = "")
    return false;
  entrada_dir dirOrigen = search(camOrigen);
  entrada_dir dirDestino = search(camDestino);
  if(dirOrigen == null || dirOrigen.tipo!=dir || dirDestino==null
  || dirDestino.tipo != dir)
    return false;


  iorig =0;
  encontre = false;
  while (!encontre && i<IT[dirOrigen.inodo_num].tope) {
    if(leerBlq(IT[dirOrigen.inodo_num].datos[iorig], bufforig)) {
      jorig =0;
      while (!encontre && jorig <32) {
        if(bufforig[jorig].usado && bufforig[jorig].nombre ==
        nomOrigen && (ext == null || ext ==
        bufforig[jorig].ext))
          encontre=true;
        else
          jorig ++;
      }
    }
    if(!encontre)
```

```
    iorig ++;
}
if(!encontre)
  return false;

encontre = false;
idst = 0;
while(idst<IT[dirOrigen.inodo_num].tope){
  if(leerBlq(IT[dirOrigen.inodo_num].datos[idst], buffdst)){
    jdst = 0;
    while(jdst<32){
      if (buffdst[jdst].usado && buffdst[jdst].nombre ==
      nomOrigen && (ext == null || ext == buffdst[jdst].ext)){
        return false;
      }
      if(!encontre && buffdst[jdst].usado){
        i = idst
        j = jdst
      }
    jdst++;
    }
  }
  idst++;
}


if(!encontre && IT[dirDestino.inodo_num].tope<8){
    int nuevo = bloque(MB,IT);
    if(nuevo == -1)
      return false;
    MB[nuevo] = 1;
    for(int k = 1, k<32, k++){
      buffdst[k].usado = false;
    }
    i=IT[dirDestino.inodo_num].tope;
    j=0;
    IT[dirDestino.inodo_num].tope++;
    IT[dirDestino.inodo_num].datos[i] = nuevo;
}

buffdst[j].usado = true;
buffdst[j].nombre = nomOrigen;
buffdst[j].tipo = bufforig[jorig].tipo;
buffdst[j].inodo_num = bufforig[jorig].inodo_num;
buffdst[j].size = bufforig[jorig].size;
buffdst[j].ext = ext;
buffdst[j].reservado = bufforig[jorig].reservado;
bufforig[jorig].usado = false;
if(!escrBlq(IT[dirDestino.inodo_num].datos[i],buffdst)  ||
!escrBlq(IT[dirOrigen.inodo_num].datos[iorig]), bufforig)
```

```
    return false;
  return true;
}
```