Departamento de Arquitectura

Examen de Sistemas Operativos

26 de julio de 2022

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 (24 pts)

(a) Suponga un SO que utiliza paginación, que da servicio a 40 usuarios. Cada uno de los usuarios ejecuta un editor de texto que consiste en un ejecutable de 150 KB de código, 50 KB de espacio de datos y 50KB de stack. Las páginas de este sistema son de 50 KB.

I. (5 pts) Cuánta memoria es necesaria para atender a los 40 usuarios si:
i) El código del editor de texto no se puede compartir entre los procesos.
ii) El código del editor de texto se puede compartir entre los procesos.
No tome en cuenta la memoria requerida para las tablas de páginas. Justifique su respuesta en ambos casos.

II. (3 pts) Explique **brevemente** cómo es el mecanismo para implementar memoria compartida en este tipo de sistema.

(b) Dado el modelo de hilos (threads) Mx1

I. (4 pts) Describa brevemente sus principales características.

II. (4 pts) Suponga que el proceso P1 dispone de 3 hilos y uno de ellos realiza una E/S. ¿Qué sucede con el resto de los hilos? Justifique su respuesta.

(c) Describa brevemente

I. (3 pts) Las principales características de una System Call.

II. (3 pts) Qué pasos implica la llamada a una System Call.

III. (2 pts) Las maneras de enviar/recibir parámetros en una System Call.

Problema 2 (36 pts)

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

```
type block = array [0..511] of byte; // 512 bytes
type dir = array[0..FILES_PER_DIR-1] of file;
type fs = array[0..MAX_DIRS-1] of dir;

type file = record
    usado: bit // entrada usada o no (1 bit)
    nombre: array [0..15] of char       // nombre del elemento (16 bytes)
    ext: array[0..2] of char //extension, solo valido si es archivo (3 bytes)
    inicio: (0..MAX_BLOCKS_1)   // direccion de comienzo (4 bytes)
    tipo: {archivo,directorio}  //1 bit
    fs_index: (0..MAX_DIRS-1)   //indice en fs, solo valido si es dir (4 bytes)
    tamanio: integer            // numero en bytes (4 bytes)
    reservado: array[0..5] of bit // espacio reservado por el sistema (6 bits)
end

type fat = array[0..MAX_BLOCKS-1] of (-2..MAX_DIRS-1)
type disk = array[0..MAX_BLOCKS-1] of block

fs FS
fat FT
disk D
```

En este sistema el tamaño mínimo de los archivos es de 5 bloques y los archivos son creados con estos ya asignados. Se sabe que:

- Las variables FS, FT y D son globales.
- La variable FT siempre se encuentra cargada en memoria con el contenido de la FAT. No se debe modelar su persistencia en memoria.
- Los datos de los archivos son alojados en D usando la estructura de la FAT.
- El directorio raiz se guarda en el índice 0 del FS.
- En la FAT el valor -1 representa "fin de archivo" y el -2 "bloque libre".

Se dispone de los siguientes procedimientos:

- `procedure partirCamino(char[] camino, out: char[][] ruta)`: Retorna en el parámetro ruta el camino partido. Ej: obtenerCamino('/home/sistoper/a.txt') retorna ['home', 'sistoper', 'a.txt']
- `procedure partirNombre(char[] camino, out: char[] nombre, out: char[] ext)`: Retorna en el parámetro nombre el nombre del archivo y en el parámetro ext su extensión. Ej: partirNombre('a.txt') retorna 'a' en nombre y 'txt' en ext.

Se pide:

(a) (8 pts) Implemente una función que dado un directorio padre busque dentro de este un archivo o directorio específico. En caso de encontrarlo retorna la posición del archivo/directorio en el directorio y falla en caso contrario.

```
search_in_dir(dir dir_padre, char[0..15] nom, char[0..2]
ext, bool esArchivo, out: int pos, out: bool ok):
```

Donde:

- dir_padre es el directorio en el que se debe buscar
- nom es el nombre del archivo/directorio a buscar y ext es la extensión en caso de que sea un archivo
- esArchivo es un booleano que determina si se busca un archivo o directorio
- ok define si la operación fue completada con éxito

Solución:

```
search_in_dir(dir dir_padre, char[0..15] nom, char[0..2] ext,
bool esArchivo, out: int pos, out: bool ok){
    {archivo, directorio} tipo;
    if(esArchivo)
        tipo = 'archivo';
    else
        tipo = 'directorio';

    ok=false;
    for(pos=0; pos<FILES_PER_DIR;pos++){
        if(dir_padre[pos].usado && dir_padre[pos].nombre == nom &&
        dir_padre[pos].ext == ext && dir_padre[pos].tipo = tipo){
            ok= true;
            break;
        }
    }
}
```

(b) (10 pts) Implemente una función que busque un archivo en todo el sistema de archivos a partir de una ruta completa. Esta función retorna la entrada correspondiente al archivo (no su posición) en caso de que lo encuentra y falla en caso contrario. Esta función también falla cuando la ruta brindada en "camino" es de un directorio.

```
search_file(char[] camino, out: file archivo, out: bool ok):
```

Donde:

- camino es la ruta completa desde la raíz al archivo que se busca
- archivo es la entrada file a la que corresponde camino
- ok define si la operación fue completada con éxito

Solución:

```
search_file(char[] camino, out: file archivo, out: bool ok){
    char[][] directorios;
    procedure partirCamino(camino, directorios);
    int ultima = size(directorios)-1;
    char[] nom, ext;
    partirNombre(directorios[ultima], nom, ext);
    ok=true;
    int dir=0, buscado;
    for (int i = 0; i < ultima; ++i){
        search_in_dir(FS[dir], directorios[i], "", false, buscado, ok)
        if(!ok) return;
        dir = FS[dir][buscado].fs_index;
    }
```

```
    search_in_dir(FS[dir], nom, ext, true,buscado,ok);
    if(ok)
        archivo = fs[dir][buscado];
}
```

(c) (12 pts) Implemente una función que elimine un archivo de un directorio. Esta función recibe el índice del directorio en el filesystem donde se encuentra el archivo a borrar. La función falla si el archivo no existe.

`rm_file(int fs_index, char[0..15] nom, char[0..2] ext, bool ok):`

Donde

- fs_index es el índice del directorio en el que está el archivo a borrar
- nom y ext son el nombre y extensión del archivo a buscar
- ok define si la operación fue completada con éxito

Solución:

```
rm_file(int fs_index, char[0..15] nom, char[0..2] ext, bool ok){
    int pos;
    search_in_dir(FS[fs_index], nom, ext, true, pos, ok)
    if(!ok) return;

    FS[fs_index][pos].usado=false;
    pos = FS[fs_index][pos].inicio
    while(pos != -1){
        int temp = FAT[pos];
        FAT[pos] = -2;
        pos = temp;
    }
}
```

(d) (6 pts) Implemente una función que cree un nuevo archivo. Esta función recibe el índice del directorio en el filesystem donde se debe crear el archivo.

`new_file(int fs_index, char[0..15] nom, char[0..2] ext, out: bool ok):`

Donde

- fs_index es el índice del directorio en el que debe crearse el archivo
- nom y ext son el nombre y extensión del archivo a crear
- ok define si la operación fue completada con éxito

Solución:

```
new_file(int fs_index, char[0..15] name, char[0..2] ext,out: bool ok){

    search_in_dir(FS[fs_index], nom, ext, true, NULL, ok);
    if(ok){
        ok=false
        return;
    }
    int i
```

```
    for ( i = 0; i < FILES_PER_DIR; ++i){
        if(!FS[fs_index][i].usado)
            break;
    }
    if(i == FILES_PER_DIR){
        ok=false;
        return;
    }
    bool primero = true;
    int bloques=0, j;
    int[5] bloques_disponibles;

    for (j = 0; bloques < 5 && j<MAX_BLOCKS; ++i){
        if(FAT[j] ==-2){
            bloques_disponibles[bloques] = j;
            bloques++;
        }
    }

    if(bloques<5){
        ok=false
    }else{
        ok=true;
        FS[fs_index][i].usado= true;
        FS[fs_index][i].nombre = nom;
        FS[fs_index][i].ext = ext;
        FS[fs_index][i].tipo = archivo;
        FS[fs_index][i].tam = 5*512;
        FS[fs_index][i].inicio = bloques_disponibles[0];
        for (int k = 1; k < 5; k++){
            FAT[bloques_disponibles[k-1]] = bloques_disponibles[k];
        }
        FAT[bloques_disponibles[4]] = -1;
    }
}
```

Nota:

En las partes (b), (c) y (d) puede usar sin implementar las funciones de partes anteriores.

Problema 3 (40 pts)

(a) (40 pts) Se considera un puente levadizo de una única vía por el que cruzan vehículos y barcos. En cuanto al cruce por parte de vehículos:

- Pueden haber varios vehículos a la vez en el puente.

En cuanto al cruce por parte de barcos:

- Los barcos tendrán prioridad para el cruce, debiendo poder hacer el cruce tan pronto como se haya vaciado de vehículos y elevado el puente.
- Solamente se admitirá el cruce de un único barco a la vez.
- En caso de presentarse cola de barcos, se desea evitar que el puente se eleve y descienda para cada uno de ellos.

Se pide:

Utilizando monitores implementar los procedimientos Barco y Vehículo. Dispone de las siguientes funciones:

- `levantar_puente()`: ejecutada para levantar el puente
- `bajar_puente()`: ejecutada para bajar el puente
- `cruzar()`: ejecutada en el contexto de los barcos y vehículos para el cruce

Solución:

```
auto(){
    puente.iniciarCruce();
    cruzar();
    puente.terminarCruce();
}

barco(){
    puente.iniciarCruceBarco();
    cruzar()
    puente.terminarCruceBarco();
}

monitor puente{
    //asumo que el puente está bajo al comienzo
    bool puenteBajo=true;
    int cantBarcos =0
    condition esperaBarco, barcosCruzando;
    iniciarCruce(){
        if(cantBarcos > 0)
            barcosCruzando.wait()
        if(!puenteBajo){
            puenteBajo=true;
            bajarPuente();
        }
        autosCruzando++
        barcosCruzando.signal()
        }
    terminarCruce(){
        autosCruzando--
        if(autosCruzando==0){
            esperaBarco.signal()
        }
    }
    iniciarCruceBarco(){
        cantBarcos++
        if(cantBarcos>1 || autosCruzando>0)
            esperaBarco.wait();
        if(puenteBajo){
            puenteBajo=false;
            subirPuente();
        }
    }

    terminarCruceBarco(){
        cantBarcos--
        if(cantBarcos==0)
            barcosCruzando.signal()
        else
            esperaBarco.signal()
    }
```

```
    main(){}
}
```