Departamento de Arquitectura

Examen de Sistemas Operativos

18 de diciembre de 2024

Lea detenidamente las siguientes instrucciones. No cumplir los requerimientos puede implicar la pérdida del examen.

- **Indique su nombre completo, número de cédula y número de parcial en cada hoja** (no se corregirán las hojas sin datos). 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**.
- Sólo se contestarán dudas de letra y no se aceptarán dudas en los últimos 30 minutos del examen.
- El examen es **SIN material** y dura **3 horas**. Al momento de finalizar el examen **no se podrá escribir absolutamente nada en las hojas**. El estudiante debe ponerse de pie e ir a la fila de entrega.
- Para aprobar el examen es necesario un mínimo de **60 puntos**.
  Importante: Debe justificar todas las respuestas, siempre.

Problema 1 (30 pts)

(a) I. (2 pts) Explique cómo se implementa el direccionamiento en un sistema con paginación.

> **Solución:** La paginación se implementa a través de la cooperación entre el sistema operativo y el hardware. Cada dirección lógica (virtual) está vinculada por el hardware de paginación a una dirección física. Las direcciones logicas se componen de un número de página y un desplazamiento. El numero de página es un índice de una tabla de paginas y el desplazamiento es la referencia dentro del frame. La tabla de pagina contiene el mapeo entre una página y su frame correspondiente.

II. (3 pts) Expliqué qué problemas de fragmentación evita la estrategia de paginación en un sistema que utiliza swap. Justifique sus respuestas.

> **Solución:** La paginación evita la fragmentación externa porque trabaja con asignaciones de tamaño fijo y elimina la necesidad de compactación de memoria. También resuelve el problema de fragmentación del swap (colocar fragmentos de memoria de diferentes tamaños en el swap), ya que todos los tamaños se corresponden con un frmae. Evita la necesidad de compactación, que en el swap no es aplicable en la práctica por su lentitud.

III. (2 pts) En un sistema con tabla de páginas de tamaño 1KiB, la página 1 tiene asignado el frame número 2, la página 2 tiene asignado el frame 3 y la página 3 tiene asignado el frame 1. ¿Cuál es la dirección física para la dirección virtual (2, 326)? Justifique su respuesta

> **Solución:** La respuesta correcta es c) $3\times1024+326$. Para conocer la dirección física se debe tomar en cuenta el número de página (2), acceder a la tabla para saber el número de frame (3), acceder a la dirección base del frame 3 (3×1024) y sumarle el desplazamiento (326).

(b) I. (3 pts) Describa el algoritmo de planificación de disco SSTF e indique su principal desventaja.

> **Solución:** Shortest Seek Time First (SSTF) elige como próximo pedido a realizar el que genere menos tiempo de búsqueda (seek time). Al considerar el tiempo de búsqueda, mejora sobre FCFS. Su principal desventaja es que puede generar que pedidos nunca sean ejecutados o sean demorados mucho tiempo.

II. (3 pts) ¿Favorece SSTF a los cilindros externos e internos? Justifique su respuesta.

Solución: No, SSTF favorece los cilindros intermedios, porque en esta localización la cabeza se puede mover en ambos sentidos y es más probable encontrar la siguiente petición. En los cilindros externos e internos solo hay una dirección de movimiento.

III. (2 pts) Un disco tiene 200 pistas (numeradas de 0 a 199) y la cola de peticiones de acceso: 81, 142, 86, 172, 89, 145, 97, 170, 125. La posición inicial de la cabeza de lectura/escritura está en la pista 100. ¿Cuál es la longitud media de búsqueda para satisfacer las solicitudes usando SSTF?

Solución:

- Pistas a la que se accede: 97 89 86 81 125 142 145 170 172
- Longitud media: 12
- Número de pistas que se atraviesan: 3 8 3 5 44 17 3 25 2 12

(c) I. (2 pts) Explique para qué sirve el archivo /usr/bin/passwd en un sistema Unix/Linux.

Solución: El archivo /usr/bin/passwd es un programa utilizado para cambiar contraseñas de usuarios en un sistema Unix/Linux.

II. (2 pts) ¿Qué significa s en los permisos de usuario? ¿Por qué usarlo en /usr/bin/passwd?

Solución: La s en los permisos de usuario indica que el bit SetUID está activado para el archivo. Esto significa que el programa se ejecutará con los permisos del propietario del archivo (en este caso, root), independientemente del usuario que lo ejecute.

III. (2 pts) Explique los riesgos de utilizar el mecanismo de la parte ii).

Solución: Si el programa tiene errores o vulnerabilidades, un atacante podría aprovechar el bit SetUID para escalar privilegios y obtener acceso completo al sistema como root.

IV. (2 pts) Mencione dos estrategias para mitigar el riesgo del mecanismo de la parte iii).

Solución:

- Limitar el uso del bit SetUID a programas estrictamente necesarios.
- Auditar regularmente programas SetUID instalados en el sistema.

(d) Sea un sistema operativo que usa planificación Round Robin con cuanto 4. Cuatro procesos llegan al sistema (proceso, tiempo de llegada, duración): (P1, 0, 8), (P2, 1, 6), (P3, 2, 6), (P4, 3, 4).

I. (3 pts) Presente la secuencia de ejecución de los cuatro procesos.

Solución: Secuencia de ejecución:

- Primer ciclo: Cada proceso ejecuta hasta 4 unidades de tiempo (o menos si su tiempo de ejecución restante es menor).
  - P1: Ejecuta 4 unidades (quedan 4).
  - P2: Ejecuta 4 unidades (quedan 2).

- P3: Ejecuta 4 unidades (quedan 6).
- P4: Ejecuta 4 unidades (finaliza).

- Segundo ciclo: Se procesan los procesos restantes.
  - P1: Ejecuta 4 unidades (finaliza).
  - P2: Ejecuta 2 unidades (finaliza).
  - P3: Ejecuta 4 unidades (quedan 2).
- Tercer ciclo: Se procesa el último proceso restante.
  - P3: Ejecuta 2 unidades (finaliza).

II. (2 pts) Calcule los tiempos promedio de espera y de retorno para cada proceso.

Solución: Cálculos:
Tiempo de finalización (TF):

- P1: 20
- P2: 22
- P3: 28
- P4: 16

Tiempo de retorno (TR): TR = TF − Tiempo de llegada

- P1: 20 - 0 = 20
- P2: 22 - 1 = 21
- P3: 28 - 2 = 26
- P4: 16 - 3 = 13

Tiempo de espera (TE): TE = TR − Tiempo de ejecución

- P1: 20 - 8 = 12
- P2: 21 - 6 = 15
- P3: 26 - 10 = 16
- P4: 13 - 4 = 9

Promedios:

- Promedio TR = $(20 + 21 + 26 + 13)/4 = 20$
- Promedio TE = $(12 + 15 + 16 + 9)/4 = 13$

III. (2 pts) Explique qué sucede si se modifica el cuanto: i) a un valor muy pequeño. ii) a un valor muy grande.

Solución:

- Si el cuanto es demasiado pequeño, se incrementa el número de interrupciones, lo que puede causar una sobrecarga en el sistema debido al tiempo perdido en el cambio de contexto.
- Si el cuanto es demasiado grande, el algoritmo se comporta más como First Come, First Serve (FCFS), lo que puede aumentar el tiempo promedio de espera para procesos cortos.

Problema 2 (40 pts)

Considere un sistema de elecciones por grupos. Para votar se debe pasar por tres lugares, en orden: la fila para pasar al salón de votación, el salón de votación y el cuarto secreto. No puede haber más de 10 votantes a la vez en el salón de votación.

En la fila hay adultos y adultos mayores, los adultos mayores tienen prioridad para entrar al salón de votación. Una vez que se entra al salón de votación, el presidente de mesa verifica que el votante esté en el padrón electoral. Si no está debe retirarse del salón de votación. Si está en el padrón queda en un grupo en espera para votar. Cuando el grupo llega a 10 personas, sus integrantes quedan habilitados para ingresar al cuarto secreto. Dentro del cuarto secreto solo puede haber un votante por vez. A medida que los votantes se van del cuarto secreto pueden entrar otros votantes de su grupo y también pueden entrar otros votantes al salón, que si están en el padrón formarán otro grupo.

Si pasados 20 minutos de la última interacción con el cuarto secreto hay menos de 10 personas (pero al menos una) en el salón, el grupo de votantes en el salón puede pasar al cuarto secreto y votar aunque no se llegue al mínimo de 10. Si llegan más votantes, formarán otro grupo.

Implemente las tareas fila, votante, presidente de mesa y cuarto secreto utilizando ADA. No se pueden usar tareas auxiliares. Se dispone de los siguientes procedimientos:

- soyAdultoMayor(): boolean, ejecutado por el votante para saber si es adulto mayor.
- miIdentificador() : int, ejecutado por el votante para obtener su identificador de votante.
- verificarId (int identificador) : boolean, ejecutado por el presidente de mesa para verificar si el identificador de votante está en el padrón.
- votar() : void, ejecutado por el votante para votar.

Solución:

```
program ej3 is

task fila is
    entry entrar_adulto_mayor();
    entry entrar_adulto();
    entry salir();
end fila;

task body fila is
    int cant = 0;
    loop
        select
            when (cant < 10) =>
                accept entrar_adulto_mayor();
                cant = cant + 1;
        or
            when (cant < 10 and entrar_adulto_mayor'count == 0) =>
                accept entrar_adulto();
                cant = cant + 1;
```

```
        or
            accept salir();
            cant = cant - 1;
        end select;
    end loop;
end fila;

task type votante is
end votante;

task body votante is
    if (soy_adulto_mayor()) then
        fila.entrar_adulto_mayor();
    else
        fila.entrar_adulto();

    boolean exito;
    presidente_mesa.verificar_identificador(mi_identificador(), exito);
    if (not exito) then
        fila.salir();
    else begin
        cuarto_secreto.quiero_votar();
        cuarto_secreto.voy_a_votar();
        votar();
        cuarto_secreto.termine_votar();
        fila.salir();
    end;
end votante;

task presidente_mesa is
    entry verificar_identificador(in id : int, out exito : boolean);
end presidente_mesa;

task body presidente_mesa is
    loop
        accept verificar_identificador(id, exito) do
            exito = verificar_id(id);
        end verificar_identificador;
    end loop;
end presidente_mesa;

task cuarto_secreto is
    entry quiero_votar();
    entry voy_a_votar();
    entry termine_votar();
end cuarto_secreto;

task body cuarto_secreto is
    boolean votando = false;
    boolean pasar_a_votar = false;
    int esperando = 0;
    loop
        select
            when (not pasar_a_votar) =>
                accept quiero_votar();
                esperando = esperando + 1;
                if (esperando == 10) then
                    pasar_a_votar = true;
        or
            when (not votando and pasar_a_votar) =>
                accept voy_a_votar();
                votando = true;
        or
            accept termine_votar();
```

```
            votando = false;
            esperando = esperando - 1;
            if (esperando == 0) then
                pasar_a_votar = false;
        or
            delay 1200;
            if (esperando > 0) then
                pasar_a_votar = true;
        end select;
    end loop;
end cuarto_secreto;

begin
end;

end ej3;
```

Problema 3 (30 pts)

Un sistema de archivos utiliza asignación indexada de dos niveles y administra su espacio libre con un mapa de bits. Sean las estructuras de datos:

```
const MAX_BLOQUES = 65536;   // Max. bloques en el disco
const MAX_INODOS = 8192;     // Max. numero de inodos

type bloque = array[0..4095] of byte;
type mapa_bits = array[0..(MAX_BLOQUES-1)] of bit;

type entrada_dir = record
    usado: boolean;                 // 1 bit
    nombre: array[0..59] of char;   // 60 B
    tipo: (archivo, directorio);    // 1 bit
    inodo_num: int16;               // 2 B
    reservado: array [0..13] of bit;
end; // 64 B

type inodos_tabla = array[0.. MAX_INODOS-1] of inodo;
type disco = array [0..( MAX_BLOQUES-1)] of bloque;
```

```
type inodo = record
    usado: boolean;          // 1 bit
    inodo_num: int16;        // 2 B
    es_dir: boolean;         // 1 bit
    tam: int32;              // 4 B, tamano del archivo
    directos: array [0..6] of int16; // 7 punteros (14 B)
    directos_tope: int16;    // 2 B
    indirecto: int16;        // 2 B, puntero a indirecto
    indirecto_tope: int16;   // 2 B
    reservado: array [0..45] of bit;
end; // 32 B

var
    IT : inodos_tabla; // Tabla de inodos
    MB: mapa_bits;     // Mapa de bits
    D : disco;         // Disco
```

El inodo número 0 corresponde al directorio raíz y en el mapa de bits (MB), el valor 1 indica un bloque ocupado y 0 uno libre. Se dispone de las siguientes funciones auxiliares (no se deben re-implementar):

- leerBloque(d: disco; bloquenum: int; var buffer: bloque; var ok: boolean), lee el bloque bloquenum del disco d y lo almacena en buffer. Retorna true si la operación fue exitosa.
- escribirBloque(d: disco; bloquenum: int; buffer: bloque; var ok: boolean), escribe el contenido de buffer en el bloque bloquenum del disco d. Retorna true si la operación fue exitosa.
- nombreDirPadre(camino: array of char; var dir: array of char), dado un camino absoluto, retorna en dir el camino del directorio padre. Por ejemplo: nombreDirPadre('/home/user/file.txt', dir) = '/home/user'.
- nombreBase(camino: array of char; var base: array of char) - dado un camino absoluto, retorna en base el nombre del archivo o directorio base. Por ejemplo: nombreBase('/home/user/file.txt', base) = 'file.txt'.

(a) (4 pts) Determine cuántas entradas de directorio pueden almacenarse en un solo bloque. Justifique su respuesta.

Solución: Cada entrada de directorio ocupa 64 bytes, mientras que cada bloque tiene un tamaño de 4096 bytes. Por lo tanto:
$\frac{2^{12} \text{ bytes}}{2^6 \text{ bytes/entrada}} = 2^6 = 64$ entradas por bloque.

(b) (6 pts) Determine el tamaño máximo que puede tener un archivo utilizando el sistema de archivos planteado. Justifique su respuesta, considerando los bloques directos e indirectos.

Solución: El inodo dispone de:

- 7 punteros directos, cada uno referencia un bloque de 4096 bytes.
- 1 puntero indirecto que referencia un bloque con 2048 ($2^{12}/2^1 = 2^{11}$) punteros. Cada uno de esos 2048 punteros referencia un bloque de 4096 bytes.

Por lo tanto:
Tamaño por bloques directos $= 7 \times 2^{12}$ bytes $= 7 \times 2\text{KB}$
Tamaño por bloques indirectos $= 2^{11} \times 2^{12}$ bytes $= 2^{23}$ bytes $= 8\text{MB}$
Suma total:
$(7 + 2^{11}) \times 2^{12}$ bytes $< 2^{24}$ bytes $= 16\text{MB}$ bytes
El tamaño máximo de un archivo dada la estructura de datos utilizada (inodo.tam) es de $(2^{32} - 1)$ bytes $= 4\text{GB} - 1$ byte.
La máxima canitdad de bloques del sistema es de 2^{16} bloques. Por la tanto, el tamaño máximo de un archivo es de $2^{16} \times 2^{12}$ bytes $= 2^{28}$ bytes $= 256\text{MB}$.
El menor de los tres valores es el que se toma como tamaño máximo, por lo tanto el tamaño máximo de un archivo es de $(7 + 2^{11}) \times 2^{12}$ bytes.

(c) (8 pts) Implemente la función no recursiva darInodo(camino: array of char; var inodo: int; var ok: boolean); que dado un camino absoluto a un archivo retorna el inodo correspondiente y en ok si existe o no el archivo. Puede utilizar buscarEnDir(inodoDir: int; nombre: array of char; var inodoEncontrado: int; var ok: boolean); que busca el nombre dado en el directorio referenciado por inodoDir.

Solución:

```
Pasos:
1. Si el camino es "/", el inodo es 0.
2. En caso contrario, se descompone el camino en su directorio padre y base.
3. Se resuelve iterativamente desde la raíz, navegando por cada componente
del camino hasta llegar al archivo final.

class stack<T> {
    private:
        T[] data;
        int top;
        int size;
    public:
        stack(int size = 100) {
            top = -1;
            this.size = size;
            data = new T[size];
        }
        ~stack() {
            delete[] data;
        }
        void push(T item) {
            top++;
            data[top] = item;
        }
        T pop() {
```

```
            T item = data[top];
            top--;
            return item;
        }
        T top() {
            return data[top];
        }
        bool empty() {
            return top == -1;
        }
        bool full() {
            return top == size - 1;
        }
};

Procedure darInodo(camino: array of char; var inodo: int; var ok: boolean);
var
    dir, base: array of char;
    inodoActual, inodoEncontrado: int;
    okBusqueda: boolean;
begin
    if (camino = "/") then begin
        inodo := 0;
        ok := true;
        exit;
    end;

    // Obtener nombre base y directorio padre
    nombreBase(camino, base);
    nombreDirPadre(camino, dir);

    // Si el padre es "/" o es vacío, el padre es inodo 0
    if (dir = "") or (dir = "/") then begin
        inodoActual := 0;
        ok := true;
    end
    else begin
        // Navegar el camino padre hasta llegar al inodo final del directorio
        // padre
        // Se resuelve iterativamente, separando las partes del camino

        var pila = new stack<array of char>();
        var parte: array of char;
        while(dir != "/") do begin
            nombreBase(dir, parte);
            pila.push(parte);
            nombreDirPadre(dir, dir);
        end;

        inodoActual := 0; // empezar desde raiz
        ok := true;
        while (!pila.empty()) and ok do begin
            buscarEnDir(inodoActual, pila.top(), inodoEncontrado, okBusqueda);
            pila.pop();
            if not okBusqueda then begin
                ok := false;
```

```
            end else begin
                inodoActual := inodoEncontrado;
            end;
        end;
        if not ok then begin
            delete pila;
            exit;
        end;
    end;

    // Ya en el directorio padre, buscar el archivo base
    buscarEnDir(inodoActual, base, inodoEncontrado, okBusqueda);
    if okBusqueda then begin
        inodo := inodoEncontrado;
        ok := true;
    end else
        ok := false;

    delete pila;
end;
```

(d) (12 pts) Implemente una función no recursiva contarArchivos(camino: array of char; var cantArchivos: int; var ok: boolean); que dado un camino absoluto a un directorio cuenta la cantidad de archivos que se encuentran directamente en el directorio.

Solución:

```
Pasos:
1. Obtener el inodo del directorio a partir del \texttt{camino} usando
\texttt{darInodo}.
2. Verificar que \texttt{inodo} sea un directorio.
3. Leer los bloques directos e indirectos del inodo del directorio.
4. Contar cuántas entradas de tipo archivo están \texttt{usado=true}.
5. Devolver el resultado en \texttt{cantArchivos}.

Procedure contarArchivos(camino: array of char; var cantArchivos: int;
var ok: boolean);
var
    in: inodo;
    inodoDir: int;
    bloqueInd: array[0..2047] of int16;
    buff: bloque;
    entradas: array[0..63] of entrada_dir; // 64 entradas por bloque
    leerOk: boolean;
    i, j: int;
begin
    // Obtener el inodo del directorio
    darInodo(camino, inodoDir, ok);
    if not ok then exit;

    in := IT[inodoDir];
    if (not in.usado) or (not in.es_dir) then begin
        ok := false;
        exit;
    end;
```

```
    cantArchivos := 0;

    // Leer bloques directos
    for i := 0 to in.directos_tope - 1 do begin
        leerBloque(D, in.directos[i], buff, leerOk);
        if not leerOk then begin
            ok := false; exit;
        end;
        entradas := array of entrada_dir(buff);
        for j := 0 to 63 do
            if (entradas[j].usado) and (entradas[j].tipo = archivo) then
                cantArchivos := cantArchivos + 1;
    end;

    // Leer bloques indirectos
    if (in.indirecto_tope > 0) then begin
        leerBloque(D, in.indirecto, buff, leerOk);
        if not leerOk then begin
            ok := false; exit;
        end;
        bloqueInd := array of int16(buff);

        for i := 0 to in.indirecto_tope - 1 do begin
            leerBloque(D, bloqueInd[i], buff, leerOk);
            if not leerOk then begin
                ok := false; exit;
            end;
            entradas := array of entrada_dir(buff);
            for j := 0 to 63 do
                if (entradas[j].usado) and (entradas[j].tipo = archivo) then
                    cantArchivos := cantArchivos + 1;
        end;
    end;

    ok := true;
end;
```