Departamento de Arquitectura

Segundo parcial de Sistemas Operativos

1 de julio de 2022

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

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 15 minutos del parcial.

Material:

- El parcial es **SIN** material (no puede utilizarse ningún apunte, libro ni calculadora). Sólo puede tenerse las hojas del parcial, lápiz, goma y lapicera en su banco. Todas sus demás pertenencias debe colocarlas en el piso debajo de su asiento.

Finalización:

- El parcial dura **3 horas**.
- Al momento de finalizar el parcial 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 parcial.

Importante: Debe justificar todas las respuestas. Siempre.

Problema 1 (10 pts)

(a) (3 pts) Explique el algoritmo de reemplazo de páginas Segunda oportunidad (Second chance).

(b) (3 pts) Se tiene un sistema con memoria virtual que utiliza tablas de páginas jerárquicas de 2 niveles, donde las direcciones virtuales son de 16 bits. Las páginas en este sistema tienen un tamaño de 256 bytes y las tablas de primer y segundo nivel tienen el mismo tamaño. Muestre un diagrama de cómo funcionaría este sistema mediante un ejemplo concreto.

(c) (2 pts) Describa brevemente la estructura RAID 5 explique la diferencia con RAID 4.

(d) (2 pts) Explique las diferencias entre los métodos de control de acceso: DAC vs MAC.

Problema 2 (22 pts)

Un sistema operativo administra sus archivos en disco utilizando el método de asignación contigua. Una particularidad de este sistema es que a todos los archivos con un tamaño menor a 5 bloques de memoria, se les asignan 5 bloques de memoria en disco, aunque no almacenen información de interés en ellos (incluso si no almacenan ninguna información). A los archivos con un tamaño superior a 5 bloques de memoria, se les asignan solo los bloques necesarios. Suponga que se dispone de las siguientes estructuras:

```
type block = array [0..511] of byte; // 512 bytes

type dir_entry = Record
    name : array [0..11] of char; // 12 * 8 bits
    type : (file,dir); // 1 bit
    used : boolean; // 1 bit
    inode_num : int16; // 16 bits
    perms : array [0..13] of bit; // 14 bits
End; // 128 bits = 16 bytes


type inode = Record
    inode_num : int16; // 16 bits
    used : boolean; // 1 bit
    start_block: int16; // 16 bits
    block_count: int8; // 8 bits
    type : (file, dir); // 1 bit
    size : int16; // 16 bits
    reserved : array[0..5] of bit; // 6 bits
End; // 64 bits = 8 bytes

type inode_table = array [0..max_inode_on_disk] of inode;

type disk = array [0..max_blocks_on_disk] of block;

type block_used = array [0..max_blocks_on_disk] of boolean;

Var
    TI : inode_table;
    D : disk;
    USED_BLOCKS: block_used;
```

Además, se sabe que:

- El directorio raiz es el inodo número 0 (cero).
- El arreglo `USED_BLOCKS` es una mapa de bits que para cada bloque indica si el mismo está siendo usado o no.
- La tabla de inodos (`TI`), el disco (`D`) y el arreglo `USED_BLOCKS` son globales y se encuentran cargados en memoria principal.
- El campo `size` dentro de la estructura `inode` es el tamaño en bytes del archivo. Este campo no es utilizado para los inode de tipo directorio.

(a) (2 pts) Dado este escenario, mencione una ventaja de asignar siempre al menos 5 bloques de memoria en disco a los archivos nuevos, aunque no sean requeridos.

Solución:
Una ventaja de esto es que se asegura que los archivos pueden aumentar su tamaño hasta 5 bloques (2,5 KB) sin necesidad de ser reubicados.

(b) (2 pts) Indique una desventaja de método planteado en la parte a).

Solución:
Una desventaja es que se tiene fragmentación tanto externa (por ser asignación contigua) como interna (por reservarse 5 bloques aunque no sean necesarios).

(c) (4 pts) Indique cual es el tamaño máximo de un archivo en el sistema planteado.

Solución:
El tamaño máximo de un archivo es el mínimo entre:

- El máximo tamaño que puede indicar el inodo según su variable `size`: $2^{16} - 1$ bytes.
- El máximo tamaño permitido por máxima cantidad de bloques que puede tener asignado el archivo, que según la variable `block_count` del inodo es: $(2^8 - 1) \times 2^9 = 2^{17} - 2^9$ bytes lo cual es mayor que $2^{16} - 1$
- Además, un archivo nunca podrá tener más de `max_blocks_on_disk`-1 bloques, es decir: (`max_blocks_on_disk` $-1) \times 2^9$ bytes.

Por lo tanto el máximo tamaño de un archivo es el mínimo entre $2^{16} - 1$ bytes y (`max_blocks_on_disk` $-1) \times 2^9$ bytes.

(d) (14 pts) Implementar una función que crea un soft link a un archivo/directorio dentro de un directorio padre.

```
procedure createSoftLink(referenced: dir_entry; dir_inode: int16; var ok: boolean)
```

Donde:

- `referenced` es el `dir_entry` que se debe almacenar dentro del directorio padre y que apunta al archivo para el cual se está creando el soft link.
- `dir_inode` es el número de inodo del directorio padre.
- `ok` es verdadero si la operación salió bien.

Asuma que cuenta con las siguientes funciones para leer/escribir en el disco:

- `procedure readBlock(D: disk; index_block: int16; var buffer: block)` Lee el bloque `index_block` del disco `D` y carga el contenido leído en el parámetro de salida `buffer`. Esta operación no puede fallar.
- `procedure writeBlock(D: disk; index_block: int16; buffer: block)` Escribe al bloque `index_block` del disco `D` la información que se encuentra en la variable `buffer`. Esta operación no puede fallar.

Observaciones:

- Si no es posible almacenar el nuevo `dir_entry` (`referenced`) sin reubicar los bloques del directorio, la función `createSoftLink` debería fallar.

Solución:

```
void createSoftLink(dir_entry referenced,int16 dir_inode,bool &ok) {
  inode padre = inode_table[dir_inode];
  bool encontre = false;
  int16 indice_bloque_libre = 0;
  int16 indice_entrada_libre = 0;
  dir_entry dir_entry_buff[32];

  ok = false;

  if (!padre.used || padre.type != dir) return;
  if (!dir_entry.used) return;

  if (padre.block_count == 0) {
    for (int i = 0; i < USED_BLOCKS && !encontre; i++) {
      if (!USED_BLOCKS[i]) {
        encontre = true;
        indice_bloque_libre = 0;
        indice_entrada_libre = 0;
        padre.start_block = i;
        padre.block_count = 1;
        USED_BLOCKS[i] = true;

        // Inicializar el buffer en 0, en especial los Used
        dir_entry_buff = {0};
      }
    }
  } else {
    for (int i = 0; i < padre.block_count; i++){
      readBlock(D, padre.start_block + i, dir_entry_buff);

      for (int j = 0; j < 32; j++) {
        if (!encontre && !dir_entry_buff[j].used){
          encontre = true;
          indice_bloque_libre = i;
          indice_entrada_libre = j;
        }
        if (dir_entry_buff[j].name == referenced.name &&
            dir_entry_buff[j].type == referenced.type &&
            dir_entry_buff[j].used)
          // Colisión de nombres! No pueden haber dos archivos
          // con el mismo nombre en el directorio.
          return;
      }
    }

    if (!encontre) {
      if (USED_BLOCKS[padre.start_block+padre.block_count])  {
        // Se requiere extender pero el siguiente bloque no esta libre
```

```
        return;
      } else {
        USED_BLOCKS[padre.start_block+padre.block_count] = true;
        indice_bloque_libre = padre.block_count;
        indice_entrada_libre = 0;
        padre.block_count++;
        encontre = true;

        // dejar el buffer en 0, en especial los .used
        dir_entry_buff = {0};
      }
    } else {
      readBlock(
        D,padre.start_block+indice_bloque_libre,dir_entry_buff);
    }
  }

  dir_entry_buff[indice_entrada_libre] = referenced;
  writeBlock(D,padre.start_block+indice_bloque_libre,dir_entry_buff);
  ok = true;

  return;
}
```

Problema 3 (32 pts)

(a) (27 pts) Se desea modelar una ferretería que cuenta con: 6 empleados que atienden a los clientes que llegan, un depósito y una única caja con un POS operado por el mismo empleado que atiende al cliente (solo se puede cobrar a un cliente por vez). Los empleados reciben los pedidos de los clientes, los retiran del depósito y luego los cobran en la caja. En la ferretería puede haber a lo sumo 5 clientes. Cuando está llena los clientes que llegan esperan afuera.

En caso de que el producto a retirar sea muy pesado, el empleado necesitará la ayuda de otro empleado para que lo ayude a cargarlo. Para esto recurrirá a uno de los empleados que se encuentre libre. En caso de no haber ninguno esperará hasta que se libere alguno. Los empleados que terminan de cobrar deben darle prioridad a ayudar a los que ya tomaron pedidos en lugar de tomar ellos pedidos nuevos.

Se pide implementar, usando mailboxes, los procesos `empleado` y `cliente`. No se pueden usar procesos auxiliares.

Dispone de los siguientes procedimientos auxiliares:

- `obtener_pedido(): Pedido`. Ejecutado por los clientes para hacer un pedido.
- `es_pesado(p: Pedido): boolean`. Ejecutado por los empleados para saber si precisan ayuda.
- `obtener_artículo(p: Pedido)`. Ejecutado por los empleados para sacar los artículos pedidos del depósito. Si se necesita ayuda no se podrá ejecutar hasta conseguir un ayudante.
- `ayudar()`. Ejecutado por los empleados para ayudar a otro. Esta función termina junto con `obtener_artículo` cuando se termina el trabajo.
- `pagar()`. Ejecutado por los clientes para pagar el pedido.

Solución:

```
var
  { controla cantidad de clientes }
  entrada: mailbox of nil;

  { los clientes encolan sus pedidos }
  cliente_pedido: mailbox of (Pedido, integer);

  { donde esperan los clientes a terminar de ser atendidos }
  cliente_espera: array [1..5] of mailbox of nil;

  { donde espera el empleado a que el cliente pague }
  empleado_pagar: mailbox of nil;

  { guarda la cantidad de empleados esperando ayuda }
  cant_empleados_esperando_ayuda: mailbox of integer;

  { esperan los empleados a que alguien los ayude }
  empleados_esperando_ayuda: mailbox of nil;

  { mutex para la caja }
  caja: mailbox of nil;

Procedure cliente()
var p: Pedido;
    pos: integer;
begin
  p := obtener_pedido();
  pos := receive(entrada);
  send(cliente_pedido, (p, pos));
  send(empleados_libres, nil);
  receive(cliente_espera[pos])
  pagar();
  send(empleado_pagar, nil);
  send(entrada, pos);
end

Procedure empleado()
var p: Pedido
    empleados_ayuda, id_cli: integer
begin
  while true do
    receive(empleados_libres);
    empleados_ayuda := receive(cant_empleados_esperando_ayuda);
    if empleados_ayuda > 0 then
      send(cant_empleados_esperando_ayuda, empleados_ayuda - 1);
      send(empleados_esperando_ayuda, nil);
      ayudar();
```

```
    else { si llega al else tiene que haber clientes esperando }
      send(cant_empleados_esperando_ayuda, empleados_ayuda);
      (p, id_cli) := receive(cliente_pedido);
      if es_pesado(p) then
        empleados_ayuda := receive(cant_empleados_esperando_ayuda);
        send(cant_empleados_esperando_ayuda, empleados_ayuda + 1);
        send(empleados_libres, nil);
        receive(empleados_esperando_ayuda);
      end
      obtener_articulo(p);
      receive(caja);
      send(cliente_espera[id_cli], nil);
      receive(empleado_pagar);
      send(caja, nil);
    end
  end
end

begin
  for i := 1 to 5 do
    send(entrada, i);
  end
  send(caja, nil);
  send(cant_empleados_esperando_ayuda, 0);
  cobegin
    empleado();
    empleado();
    empleado();
    empleado();
    empleado();
    empleado();
    cliente();
    ...
    cliente();
  coend
end
```

(b) (5 pts) Si se cambiara la letra anterior para permitir entrar a 10 clientes a la ferretería, ¿qué problema nuevo aparecería y qué estrategia usaría para resolverlo? Justifique.

Solución: Al permitir entrar a 10 clientes puede pasar que los 6 empleados comienzen a atender a uno de ellos y todos precisen ayuda por lo que el sistema quedaría en deadlock. Una posible forma de arreglarlo es no dejar que más de 5 empleados atiendan clientes al mismo tiempo. Es un problema y solución similares a los filósofos.