Sistemas operativos

Examen Febrero 2021

Bloque 1

Pregunta 1)

¿Cómo se denomina el tipo de E/S que realizan los procesos cuando estos se bloquean hasta que el pedido finalice? ¿Existe otra manera de realizar un pedido de E/S? En caso afirmativo menciónelo y descríbalo.

Pregunta 2)

En el contexto de control de acceso en UNIX, ¿es posible que el UID efectivo sea distinto del real?. Justifique

Pregunta 3)

¿Qué tipo de hipervisor utilizaría si desea crear múltiples máquinas virtuales con diferentes Sistemas Operativos utilizando un software que corre específicamente en un Sistema Operativo?. Mencione y describa brevemente como funciona.

Pregunta 4)

¿Cómo puede una aplicación de usuario obtener o establecer atributos de un archivo?. Describa brevemente cómo funciona.

Bloque 2

Se tiene un sistema operativo multiprogramado, en el cual se dispone de 1 procesador. El planificador del sistema operativo utiliza una estrategia de planificación por prioridad expropiativo. Este sistema maneja hilos con modelo M x 1, utilizando una planificación Round-Robin con quantum de 2ms a nivel de usuario.

Se tiene que el proceso P1 cuenta con 2 hilos (P1H1, P1H2) que ejecutan concurrentemente de forma que un hilo ejecuta el código de R1 y el otro R2. En cambio, P2 cuenta con un único hilo de ejecución y ejecuta el código de R3. En el instante de tiempo inicial (t = 0) la cola de listos contiene a P1 y P2. La prioridad está definida por el tiempo de cómputo restante (más tiempo, más prioridad) y **luego** por **menor** identificador, es decir, P*i* tendrá **mayor** prioridad que P*j* si i < j.

En todos los casos, los empates se resuelven con el mismo criterio que la prioridad.

R1	R2	R3
Ejecuta 2ms	Ejecuta 4ms	Ejecuta 4ms
Bloquea 2ms	Bloquea 4ms	Bloquea 4ms
Ejecuta 2ms	Ejecuta 2ms	Ejecuta 8ms

Se pide:

Dibujar un diagrama de tiempo vs hilos, indicando para cada instante en qué estado se encuentra cada hilo. Utilice los valores E, L1, L2, B, T y - . E corresponde a ejecutando, B que está bloqueado, T que el hilo terminó su ejecución y L para listo siendo el número la posición en la cola. El guión simboliza que el proceso/hilo no está aún en el sistema.

Solución:

	2	4	6	8	10	12	14
P1H1	L2	L1	L1	ER11	B	L2	L1
P1H2	L1	ER21	L2	L1	L1	L1	ER21
P2H1	ER31	L	ER31	B	B	ER33	L
P1	L	E	L	E	B	L	E
P2	E	L	E	B	B	E	L
left p1	10	10	8	8	6	6	6
left p2	12	10	10	8	8	8	6
		menor id proc					menor id proc
	16	18	20	22	24	26	
P1H1	L1	L1	ER13	T	T	T´	
P1H2	B	B	L1	ER23	T	T´	
P2H1	ER33	ER33	L	L	ER33	T´	
P1	B	B	E	E	T	T´	
P2	E	E	L	L	E	T´	
left p1	4	4	4	2	0	0	
left p2	6	4	2	2	2	0	

Bloque 3

Se cuenta con un sistema operativo que utiliza inodos con la siguiente organización en disco y estructuras de datos:

Tabla de inodos: bloque 0..255
Bloques de datos: bloque 256..MAX_BLOQUES

type entrada_dir = Record
    usado: bit // 1 bit. 0 indica que esta libre, 1 que esta usado.
    tipo: bit // 1 bit. 0 indica que es un archivo, 1 que es un directorio.
    num_inodo: int8 // 1 byte. Número de inodo.

```
        nombre: char[126] // 126 bytes. Nombre del archivo/directorio.
        reservado: bit[6] // 6 bits. Reservados para el sistema operativo.
End; // 128 bytes.

type inodo = Record
        usado: bit          // 1 bit. 0 indica que esta libre, 1 que esta usado.
        tipo: bit           // 1 bit. 0 indica que es un archivo,
                // 1 que es un directorio.

        num_inodo: int8      // 1 byte. Numero de inodo.
        tamaño: int16           // 2 bytes. Tamaño del archivo
                                // (solo es valido para archivos).

        cant_bloques: int8   // 1 byte. Cantidad de bloques de datos utilizados.
                // 0 indica que no se tienen datos.

        primerBloque: int16  // 2 bytes. Puntero al primer bloque de datos.
        reservado: bit[6]    // 14 bits. Reservados para el sistema operativo.
End; // 8 bytes.
```

Tenga en cuenta las siguientes consideraciones:

- Cada sector ocupa 2 KB de datos.
- Los datos son dispuestos en disco utilizando Asignacion Continua. La variable primerBloque indica el indice del primer bloque de datos, y la variable cant_bloques el total de bloques utilizados. Si cant_bloques tiene valor 0, la variable primerBloque no tiene sentido.
- MAX_BLOQUES indica la cantidad total de bloques de datos disponible en el sistema operativo.
- Toda operación debe dejar el sistema de archivos en un estado consistente. Pueden utilizar funciones auxiliares. La implementación debe ser eficiente.
- El directorio raíz se encuentra en el inodo con numero 0.

Dispone de las siguientes funciones (asuma que ya están implementadas y úselas según convenga):

- **int leerBloque(numBloque: int16; buffer: byte[2048])**

  Lee desde el disco el bloque con indice numBloque y lo almacena en la variable buffer. En caso de error, retorna -1.

- **int partirRuta(ruta[]: char; out partes[][126]: char)**
  Dada una ruta hacia un archivo, divide la ruta en partes y carga los nombres obtenidos en la variable partes. Retorna el numero de partes totales.
  Ejemplos:
  - Dada la entrada '/' la función retorna partes [] y el valor 0.
  - Dada la entrada '/home/examen/letra.txt' la función retorna partes ['home', 'examen', 'letra.txt'] y el valor 3.

Si asume algo sobre el sistema que no este dicho en la letra, agréguelo a su respuesta.

a) Indique cuántos inodos entran en un bloque de datos. Indique cuántos entrada_dir entran en un bloque de datos. No es necesario justificar.
b) Implemente la siguiente función:

int buscarInodo(ruta[]: char)

Dada la ruta hacia un archivo/directorio, retorna el numero de inodo del mismo. En caso de error, retorna -1.

Solución:

1)

En un bloque de datos entran $2^{11}/2^{3}$ = 256 estructuras inodo. En un bloque de datos entran $2^{11}/2^{7}$ = 16 estructuras entrada_dir.

2)

```
int buscarInodo(ruta[]: char){
	int actual = 0;
	char[][] partes;
	inodo[256] buffInodo;
	entrada_dir[16] buffEntrada;
	inodo aux;

	int cant = partirRuta(ruta, partes);
	int iter = 0;

	while (iter < cant){
		int bloqueInodo = floor(actual / 256);
		int offsetInodo = actual % 256;

		if (leerBloque(bloqueInodo, buffInodo) < 0)
			return -1;

		aux = buffInodo[offsetInodo];
		if (aux.usado == 0 || aux.tipo == 0)
			return -1;

		bool encontre = false;
		int bloque = 0;

		while (!encontre && bloque < aux.cant_bloques){

			if (leerBloque(aux.primerBloque+bloque, buffEntrada) <0){
				return -1;
			}
```

```
                int ent = 0;
                while (!encontre && ent < 16){
                      if (buffEntrada[ent].usado &&
                          buffEntrada[ent].nombre == partes[iter]){
                            encontre = true;
                            actual = buffEntrada[ent].num_inodo;
                      } else {
                            ent ++;
                      }
                }

                if (!encontre) bloque++;
          }

          if (!encontre) return -1;

          iter++;
     }

     return actual;
```

Bloque 4

Se desea modelar una piscina pública. Dadas las restricciones por COVID solo se permite hasta 25 personas bañándose simultáneamente. Luego de que está llena, las personas que llegan a continuación deben esperar afuera hasta que se libere un lugar. Cada cierto tiempo un empleado debe aspirar la basura del fondo de la piscina y para ello la misma debe estar vacía.

Las personas que se bañan tienen prioridad para entrar sobre el empleado que debe esperar a que nadie quiera usar la piscina.

Se pide:

Implementar las personas y el empleado usando semáforos.

Se dispone de las siguientes funciones auxiliares:

- bañarse(): Ejecutada por las personas para usar la piscina
- limpiar(): Ejecutada por el empleado para aspirar el fondo de la piscina

Solución:

```
var
     cantPersonas: integer;
     mutexPersona, mutexPiscina, cantidad: semaphore;

begin
     cantPersonas := 0;
     init(mutexPersona, 1);
```

```
      init(mutexPiscina, 1);
      init(cantidad, 25);
      cobegin
            empleado();
            persona();
            ...
            persona();
      coend
end


procedure empleado()
      repeat
            P(mutexPiscina);
            limpiar();
            V(mutexPiscina);
            // otras tareas del empleado
      until False
end



procedure persona()
      P(mutexPersona);
      cantPersonas := cantPersonas + 1;
      if cantPersonas = 1 then
            P(mutexPiscina);
      end
      V(mutexPersona);

      P(cantidad);
      bañarse();
      V(cantidad);

      P(mutexPersona);
      cantPersonas := cantPersonas - 1;
      if cantPersonas = 0 then
            V(mutexPiscina);
      end
      V(mutexPersona);
end
```