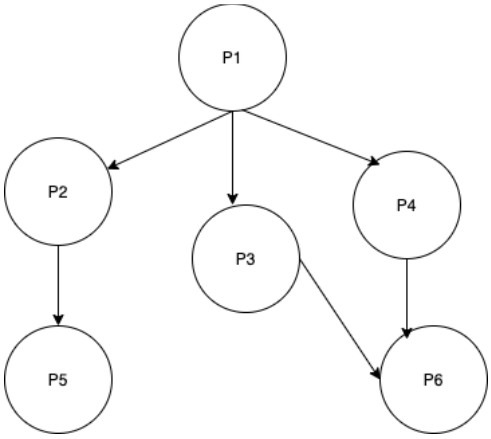
Bloque 1

a) El siguiente grafo de precedencia se puede representar con COBEGIN-COEND? En caso afirmativo de una representación, en caso contrario utilice una primitiva que si pueda.



Solución

Si, se puede

```
BEGIN
P1
COBEGIN
  BEGIN
    P2
    P5
  END
  BEGIN
    COBEGIN
          P3
          P4
    COEND
    P6
  END
COEND
END
```

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

Ver teo

c) ¿Cómo puede una aplicación de usuario solicitar o liberar un dispositivo, así como obtener acceso a memoria compartida ?. Describa brevemente como funciona.

Ver teo

Bloque 2
Se tiene un sistema con un único procesador sobre el que ejecuta un sistema operativo con un planificador Round Robin con un cuanto de 10 ms y un modelo de hilos Nx1. Inicialmente en el sistema no se tiene ningún proceso de usuario ejecutando y en tiempo t=0 se lanza la ejecución del siguiente programa de usuario:

```
begin
   pid = fork()
   if(pid == 0){
      Ejecuta 5ms
      Bloquea 10ms
      Ejecuta 10ms
   }else{
       create_thread(proc1)
       create_thread(proc2)
   }
 end
```

```
procedure proc1()
begin
   Ejecuta 5ms
   Bloquea 5ms
   Ejecuta 10ms
end

procedure proc2()
begin
   Ejecuta 5ms
   Bloquea 5ms
   Ejecuta 5ms
end
```

Se sabe que la operación fork() es estilo Unix y al igual que en Unix, luego de un fork() el proceso hijo siempre tiene un único hilo de ejecución. La operación create_thread() crea un nuevo hilo de ejecución y comienza su ejecución en el procedimiento recibido como argumento. Salvo que se indique explícitamente, el tiempo de ejecución de todas las funciones es de 5 ms (fork, create_thread, condición del if, asignación, etc). Tenga en cuenta la composición de estas operaciones, e.g. a = a+1 tiene 5ms por la suma y 5ms por la asignación. La planificación definida para los hilos a nivel de usuario es SJF expropiativo. En caso de empates gana el que tenga menor id, o sea el más antiguo.

Se pide:
a) Realice un diagrama de planificación (**tiempo vs hilos**), comenzando en el tiempo t = 0, indicando el estado de cada uno de los hilos y su posición en cada una de las colas.

b) Indique el tiempo de espera para el primer proceso del sistema. Exprese el resultados en ms.
Indique el tiempo de retorno para el segundo proceso del sistema. Exprese el resultados en ms.

Solución

a)

	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80
P1H1	E (fork)	E (asig)	L	L	E (if)	E (thead1)	L	E (thead2)	T							
P1H2	-	-	-	-	-	-	L	L	E	B	L	L	L(B)	L	E	E
P1H3	-	-	-	-	-	-	-	-	L	L(B)	L	E	B	E	T	
P2H1	-	L	E (asig)	E (if)	L	L	E	B	B	E	E	T				

b)

20
50

Bloque 3

```
const MAX_BLOQUES = 32768;
const MAX_INODOS = 4096;
type entrada_dir = Record
        usado : boolean; // 1 bit
        nombre : array [0..22] of char; // 23 bytes
        ext: array[0..2] of Char; // 3 bytes
        size : int; // 2 bytes
        tipo : (file, dir); // 1 byte
        inodo_num : int; // 2 bytes
        reservado : array [0..6] of bit; // 7 bits
End; // 32 bytes

type inodo = Record
        usado : boolean; // 1 bit
        inodo_num : int; // 2 bytes
        datos : array [0..8] of int; // 18 bytes
        tope : int; // 2 bytes
        reservado : array [0..6] of bit; // 7 bits
End; // 23 bytes

type bloque = array [0..1023] of byte; // 1024 bytes
type mapa_bits = array [0..MAX_BLOQUES-1] of bit;
type inodo_tabla = array [0..MAX_INODOS-1] of inodo;

var
IT : inodo_tabla;
MB : mapa_bits;
```

a) ¿Cuál es la cantidad máxima de archivos que pueden guardarse en el sistema?
b) ¿Cuál es el tamaño máximo que puede tomar un archivo?
c) Implemente la funcion delete_file que dado un número de inodo de un directorio elimina el archivo con nombre name y extención ext del mismo. La función retorna True si encuentra y borra el archivo y False en caso contrario.
Puede asumir que inodoNum está en uso y es ocupado por un directorio. No es necesario comprobarlo.

Se dispone de las funciones 'leer_bloque(int nro bloque): bloque' y 'escribir_bloque(int nro_bloque, bloque blk)' las cuales permiten leer y escribir un bloque del disco. Puede asumir que dichas funciones no fallan.

Solución:

```
function delete_file(inodoNum: int, name: array of char, ext: array of char): bool
       inodo in = IT[inodoNum]
       for(int i =0; i < in.tope ; i++){
              entrada_dir[] blk = (entrada_dir[]) leer_bloque(in.datos[i]);
              for(int j=0; j<32; j++){ // 32 entradas dir en cada bloque de 1KB.
                     entrada_dir e = blk[j]
                     if (e.usado && e.tipo==‘file’ && e.nombre == name && e.ext ==ext){
                            e.usado=false;
                            inodo a_liberar = IT[e.inodo_num]
                            for(int k=0; k < a_liberar.tope;k++){
                                   MB[a_liberar.datos[k]] = 0;
                            }
                            escribir_bloque(in.datos[i], blk);
                            return true;
                     }
              }
       }
       return false;
end function
```

Bloque 4

En un local comercial se dispone de una Narguile (pipa de agua) que tiene capacidad para 8 personas a la vez. Los clientes que asisten al local para experimentar la experiencia son considerados en forma individual, es decir van utilizando el dispositivo en orden de llegada y en la medida que tengan lugar. El cliente, antes de empezar a utilizar la Narguile consulta si hay suficiente tabaco para su uso. En caso que no alcance el tabaco disponible el cliente le pide al empleado que recargue. Para recargar el Narguile el empleado debe esperar que culminen los clientes que ya lo están utilizando.

Se dispone de las siguientes funciones:

completar(), Este procedimiento es ejecutada por el empleado para recargar la Narguile.
hay_tabaco(): boolean, Esta función es ejecutada por los clientes y retorna si hay suficiente tabaco. No puede ser ejecutada por dos o más clientes a la vez.
fumar(), Es ejecutado por los clientes.

Se pide: Implementar los procedimientos Cliente y Empleado utilizando monitores. Especifique la semántica de los monitores.

```
Monitor pipa {

integer: cant_fumar
boolean: rellenar
condition: cnd_empleado , cnd_cliente , cnd_relleno

 void entrar () {
  if (cant_fumar == 8 or rellenar)
        cnd_cliente.wait()
  if (!hay_tabaco ()) {
        rellenar = True
        if (cant_fumar == 0) {
                cnd_empleado.signal()
                cnd_relleno.wait()
        }
  }
  cant_fumar++
  if (cant_fumar < 8)
        cnd_cliente.signal()
 }

 void termine_fumar (){
  cant_fumar--
  if (cant_fumar == 0 && rellenar)
        cnd_empleado.signal()
  if  (!rellenar)
        cnd_cliente.signal()
 }

 void empleado_inicio (){
  if (!rellenar || cant_fumar > 0)
        cnd_empleado.wait()
 }

 void empleado_fin (){
    rellenar = False
    cnd_relleno.signal()
  }
```

```
 void main (){
   cant_fumar = 0
   rellenar = False
 }
}

 void cliente (){
  pipa.entrar()
  fumar ()
  pipa.termine_fumar()
 }

void empleado (){
 while True {
         pipa.empleado_inicio()
         completar ()
         pipa.empleado_fin()
  }
}
```