MEMORIA DE PRÁCTICAS PCD

PORTADA

Asignatura: PCD
Convocatoria: Junio 2023

Profesora: Raquel Martínez España

Alumnos:
Balsalobre Rodríguez, Miguel G2.2
Matas Conesa, Christian G2.3

EJERCICIO 1

•Recursos no compartibles. Los que se hayan identificado en el problema.

- La matriz A: esta matriz es un recurso no compartible, y nos aseguramos de que no se sobrescriba por los dos hilos, porque sólo la utilizamos para leer la matriz y usamos una matriz local para escribir el resultado.
- La pantalla: utilizamos la variable l (ReentranLock) antes de imprimir por pantalla y la liberamos cuando terminemos de usarla.

• Condiciones de sincronización. Las detectadas en el problema.

Hemos utilizado los objetos de cerrojo (ReentrantLock) para proteger el acceso a la pantalla en las funciones sumarMatriz() y multiplicarMatriz(). Además, el programa utiliza cobegin y coend para asegurar que los procesos Hilo1 y Hilo2 se ejecuten concurrentemente.

• Pseudocódigo.

```
Program Ejercicio1

Process MatrizCompartida

     const

          n=3;

     type

          matriz=array[1..n][1..n] of integer;

     var

          A:matriz;

          l:mutex;

     //La función sumarMatriz muestra por pantalla "A + A" y
     en la siguiente línea sus valores, después muestra "2A" y
     por último muestra por pantalla el resultado de la suma
     de las dos matrices iguales.

     //Se realiza la suma de las matrices (A + A) y se guarda
     su resultado en la matriz local B, luego entre cerrojos
     imprimimos por pantalla (para que esté protegida).

     function sumarMatriz();

     //La función multiplicarMatriz almacena en una matriz
     auxiliar B el producto de dos matrices iguales, muestra
     por pantalla "A x A" y en la siguiente línea sus valores,
```

```
     después muestra "A^2" y por último muestra por pantalla
     los valores de la matriz B.

     //Se realiza la multiplicación de las matrices (A * A) y
     se guarda su resultado en la matriz local B, luego entre
     cerrojos imprimimos por pantalla (para que esté
     protegida).

     function multiplicarMatriz();

Process Hilo1

     var c:MatrizCompartida;

     //Hilo1 llama a la función sumarMatriz con la variable c

Process Hilo2

     var c:MatrizCompartida;

     //Hilo2 llama a la función multiplicarMatriz con la
     variable c

Process Ejemplo

     begin

          var

               c:MatrizCompartida;

               a:Hilo1;

               b:Hilo2;

          cobegin

               ejecutar hilo a;

               ejecutar hilo b;

          coend;

          write("FIN DEL PROGRAMA");

     end;
```

• Cuestiones planteadas en los ejercicios

a) Antes de usar la clase ReentrantLock el problema es que los dos hilos escriben en la pantalla a la misma vez, lo que causa que las matrices se mezclen entre ellas.

La salida es una mezcla entre la matriz A^2, la matriz 2A, etc…

b) Después de usar la clase ReentrantLock la salida ya es correcta y se imprimen todas las matrices correctamente por pantalla, esto se debe a que con la clase ReentrantLock protegemos los bucles que imprimen la matriz y mientras que una matriz se imprime la otra espera.

c) Primero imprime las 10 veces un hilo y luego el otro, algunas veces imprime primero el que suma la matriz y otras veces el que la multiplica, por lo que no sigue un orden. Es correcto que el orden varíe porque depende de qué hilo llega primero.

d) Al principio y al final de cada función usamos un println que indica la acción que se está realizando, si cuando imprimimos por pantalla salen los dos println sin que se mezclen con los de otra función es porque está funcionando bien. También se podrían añadir más matrices y realizar más operaciones con ellas para aumentar la complejidad del programa y comprobar si sigue funcionando correctamente.

EJERCICIO 2

•Recursos no compartibles. Los que se hayan identificado en el problema.

- panel1, panel2, panel3, panel4 y panel5: son objetos del tipo Panel que representan los paneles que se van a imprimir (hacen la función de la pantalla y por eso hay que protegerlos).

• Condiciones de sincronización. Las detectadas en el problema.

- semP1: es un semáforo binario que se usa para garantizar que sólo un hilo a la vez acceda a escribir en el panel1.
- semP2P3: este semáforo permite que en el caso que un hilo tenga mucho que escribir, mientras esté escribiendo no sea sobrescrito por otro hilo.
- semP4P5: este semáforo permite que en el caso que un hilo tenga mucho que escribir, mientras esté escribiendo no sea sobrescrito por otro hilo.
- mutexP2P3: este semáforo de 2 bits se usa para coordinar el acceso a los paneles p2 y p3
- mutexP4P5:este semáforo de 2 bits se usa para coordinar el acceso a los paneles p4 y p5.
- p2, p3, p4, p5: son variables enteras que indican si el panel está libre (valor 0) o ocupado (valor 1).

• Pseudocódigo.

```
Program Ejercicio2

     Process Hilo

          for i:=0 to 30 do:

               //Se crean dos variables con números aleatorios
               y se elige la operación de forma aleatoria.

               int x = aleatorio
               int y = aleatorio
               int r = x op y

               if (r multiplo 5)

                    wait (semP1)

                    escribir (panel1)

                    signal (semP1)

               else if (r par)

                         //Mientras que p == null, es decir,
                    ningún panel este ocupado, se busca que
                    panel está libre y se le asigna a p.
```

```
        int p = null
        while(p == null)
             wait(mutexp2p3)
             if(p2 == libre)
                  p2 = ocupado
                  p = p2
             else if(p3 == libre)
                  p3 = ocupado
                  p = p3
             signal(mutexp2p3)

             wait(semp2p3)

             //Semp2p3 es para evitar que se
             sobreescriba, quizas un hilo tiene
             que escribir mucho y mientras en el
             proceso llega otro y sobreescribe


             Imprimir(p)

             //P indica que panel se escribe
             signal(semp2p3)

        //LIBERAR PANEL
        wait(mutexp2p3)
        if(p==2)
             p2 = libre
        else
             p3 = libre
        signal(mutexp2p3)

else

             //Hace lo mismo que el anterior, pero
        para los  resultados impares con los
        paneles p4 y p5.

        int p = null
        while(p == null)
             wait(mutexp4p4)
             if(p4 == libre)
                  p4 = ocupado
                  p = p4

             else if(p5 == libre)
                  p5 = ocupado
                  p = p5
             signal(mutexp4p5)

             wait(semp4p5)
```

```
                    //Este semaforo para evitar que se
               sobreescriba, quizas un hilo tiene que
               escribir mucho y mientras esta en el
               proceso llega otro y sobreescribe


                    Imprimir(p)
                    signal(semp4p5)

               //LIBERAR PANEL
               wait(mutexp4p5)
               if(p==4)
                    p4 = libre
               else
                    p5 = libre
               signal(mutexp4p5)

Process Ejemplo

begin

     var

          panel1:Panel

          panel2:Panel;

          panel3:Panel;

          panel4:Panel;

          panel5:Panel;

          semP1:semaphore(1)

          semP2P3:semaphore(2)

          semP4P5:semaphore(2)

          mutexP2P3:semaphore(1)

          mutexP4P5:semaphore(1)

          p2, p3, p4, p5:int

     cobegin

          for i:=1 to 51 do:

               //Se crean 50 hilos dentro de este bucle.

     coend;
```

```
end;
```

• Cuestiones planteadas en los ejercicios.

a)Escoger el panel que tienen que usar (cuando no tengan que modificar la misma variable local para bloquearlo, por ejemplo, las variables p2 y p3 que están controladas por el mismo semáforo de 1 bit) y escribir en él siempre que no tengan que utilizar el mismo.

b) semP1 - SEMÁFORO PARA EL PANEL 1 (MÚLTIPLO DE 5), para que no acceda un hilo mientras otro está escribiendo.
semP2P3 - SEMÁFORO DE DOS BITS PARA EL PANEL 2 Y 3 (RESULTADOO IMPAR Y NO ES MULTIPLO DE 5), para que solo puedan entrar dos hilos a escribir, uno en el panel 2 y otro en el 3.
semP4P5 - SEMÁFORO DE DOS BITS PARA EL PANEL 4 Y 5 (RESULTADOO PAR Y NO ES MULTIPLO DE 5), para que solo puedan entrar dos hilos a escribir, uno en el panel 4 y otro en el 5.
mutexP2P3 - SEMÁFORO DE UN BIT PARA EVITAR QUE DOS O MÁS HILOS ACCEDAN A LA MISMA VARIABLE GLOBAL A LA VEZ (PARA LOS PANELES 2 Y 3), este semáforo protege las variables globales p2 y p3 para que no puedan acceder dos hilos o más a la vez y sobrescribir la variable.
mutexP4P5 - SEMÁFORO DE UN BIT PARA EVITAR QUE DOS O MÁS HILOS ACCEDAN A LA MISMA VARIABLE GLOBAL A LA VEZ (PARA LOS PANELES 4 Y 5), este semáforo protege las variables globales p4 y p5 para que no puedan acceder dos hilos o más a la vez y sobrescribir la variable.

c) Sí pueden escribir varios hilos a la vez (máximo 5), siempre que no sea en el mismo panel, porque cada panel es como si fuera una pantalla propia.

EJERCICIO 3

•Recursos no compartibles. Los que se hayan identificado en el problema.

- Los surtidores (surtidores[]): sólo un cliente a la vez puede utilizar cada surtidor
- Los túneles de lavado (tuneles[]): Al igual que los surtidores, sólo un cliente puede utilizar un túnel de lavado a la vez
- El array de tiempo de espera (tiempoEspera[]): Cada vez que un cliente llega se le asigna un tiempo de espera en función de la cantidad de clientes que están delante de él en la cola.
- La pantalla (pantallaOcupada): Si la pantalla está ocupada, ningún otro cliente puede utilizarla para ver su tiempo de espera.

• Condiciones de sincronización. Las detectadas en el problema.

En el problema de los surtidores, se deben garantizar las siguientes condiciones de sincronización:

Exclusión mutua: Solo un vehículo puede utilizar un surtidor en un momento dado. Si un vehículo está utilizando un surtidor, los demás vehículos deben esperar hasta que el surtidor esté disponible.

Asignación justa: Los vehículos deben ser atendidos en orden de llegada. No se permite que un vehículo salte a la cabeza de la cola y utilice el surtidor antes que los demás vehículos que están esperando.

Tiempo límite: Si un vehículo espera demasiado tiempo en la cola, debe ser atendido de inmediato en cuanto el surtidor esté disponible. De lo contrario, el vehículo puede abandonar la estación de servicio sin recibir servicio y esto no es deseado.

En el problema de los túneles de lavado, también se deben garantizar las condiciones de exclusión mutua y asignación justa, de manera que solo un vehículo puede utilizar un túnel de lavado en un momento dado y los vehículos deben ser atendidos en orden de llegada. Además, se debe garantizar que el túnel de lavado esté disponible solo si el vehículo ha sido atendido en los surtidores y se debe controlar el tiempo de espera del vehículo en la cola del túnel de lavado.

• **Pseudocódigo.**

```
Program Ejercicio3
Process Monitor


l:mutex
colaSurtidores:Condition
surtidoresDisponibles: int
surtidores: boolean[]
tunel1:Condition
tunel2:Condition
tunel3:Condition
tunel4:Condition
tunel5:Condition


tuneles: boolean[]
tiempoEspera: boolean[]
pantalla:Condition
pantallaOcupada: boolean
surtidor: int
tunel:int


// Si no hay surtidores disponibles se espera y cuando se
libere uno se le asigna el surtidor
Function repostar
while (surtidoresDisponibles == 0){
     colaSurtidores.delay()
}
surtidoresDisponibles--
```

```
for (int i = 0; i < surtidores.length; i++) {
     if (surtidores[i]) {
          Coche.surtidor = i
          surtidores[i] = false
          break
     }
}
//Devuelve el surtidor en el que va a repostar
return surtidor


// Libera el surtidor que estaba usando
Function terminarRepostar ( int surtidor)

surtidoresDisponibles++
surtidores[Coche.surtidor] = true
colaSurtidores.resume()


//Busca la cola con menos tiempo de espera
Function buscarCola (int lav, boolean haRepostado)
tiempos: int[]
while (pantallaOcupada) {
     pantalla.delay()
}
pantallaOcupada = true
for (int j = 0; j < 5; j++){
```

```
          tiempos[j] = tiempoEspera[j]

     }


     if (!haRepostado){  // Si no ha repostado se va
directamente a la cola 5 y se suma el tiempo          tunel =
5         de espera de su lavado

          tiempoEspera[4] += lav

     } else {

          cola:int

          min:int

          //asignar cola con menor tiempo de espera

          return tiempos //las 5 primeras posiciones son los
tiempos de cada cola y la última es el túnel
asignado

     }


     // Se espera hasta que el tunel que se le asignó esté
libre y lo ocupa

     Function entrarLavar (int tunel)


     pantallaOcupada = false

     pantalla.resume()

     switch (tunel) {

     case 1: {

          while (!tuneles[0]) {

               tunel1.delay()

          }

          tuneles[0] = false

          break

     }
```

```
case 2: {
     while (!tuneles[1]) {
          tunel2.delay()
     }
     tuneles[1] = false
     break
}
case 3: {
     while (!tuneles[2]) {
          tunel3.delay()
     }
     tuneles[2] = false
     break
}
case 4: {
     while (!tuneles[3]) {
          tunel4.delay()
     }
     tuneles[3] = false
     break
}
case 5: {
     while (!tuneles[4]) {
          tunel5.delay()
     }
     tuneles[4] = false
     break
```

```
}


// Libera el tunel que estaba usando
Function salirLavar (int tunel, int lavado)

switch (tunel) {
case 1: {
     tuneles[0] = true
     tiempoEspera[0] = tiempoEspera[0] - lavado;
     tunel1.resume()
     break
}
case 2: {
     tuneles[1] = true
     tiempoEspera[1] = tiempoEspera[1] - lavado;
     tunel2.resume()
     break
}
case 3: {
     tuneles[2] = true
     tiempoEspera[2] = tiempoEspera[2] - lavado;
     tunel3.resume()
     break
}
case 4: {
     tuneles[3] = true
```

```
        tiempoEspera[3] = tiempoEspera[3] - lavado;
        tunel4.resume()
        break
}
case 5: {
        tuneles[4] = true
        tiempoEspera[4] = tiempoEspera[4] - lavado;
        tunel5.resume()
        break
}
}
```

```
Process Coche

private int id;
private Monitor monitor;
private int surtidor;
private int tunel;
private int[] array;
private int dinero;
private int tiempo;
private int lavado;
private boolean haRepostado = false;
const BASICO 50;
const NORMAL 55;
const PREMIUM 60;
```

```
Function imprimirPantalla
if (haRepostado){
     //imprime
} else {
     //imprime
}

Process run
//Elige un número aleatorio de 1 a 10 y si es menor o igual que 7 repostará, si no solo lavará el coche.
Random rand = new Random()
int n = rand.nextInt(10) + 1

if (n > 3) {
     haRepostado = true
     surtidor = monitor.repostar()
     sleep(tiempo)
     monitor.terminarRepostar(surtidor)
     array = monitor.buscarCola(lavado, haRepostado)
     tunel = array[5]
     imprimirPantalla()
     monitor.entraLavar(tunel)
     sleep(lavado)
     monitor.salirLavar(tunel, lavado)
} else {
     array = monitor.buscarCola(lavado, haRepostado);
     tunel = array[5]
     imprimirPantalla()
```

```
        monitor.entraLavar(5)
        sleep(lavado)
        monitor.salirLavar(5, lavado)
    }


    Process Main
    Monitor monitor
    Thread[100] coche


    cobegin
        for i:=1 to 101 do:
            //Se crean 100 hilos dentro de este bucle.
    coend


    imprime("FIN DEL PROGRAMA")
```

• Cuestiones planteadas en los ejercicios.
a) Monitor usando las clases Condition y ReentrantLock, hemos usado este tipo porque así podemos usar una condición para cada túnel y así simular las diferentes colas de espera.

b) Usando una condición para bloquear la pantalla cuando esté siendo utilizada, esta condición se controla mediante una variable que indica lo que debe realizar la condición. Cuando la variable está a true es porque la pantalla se está utilizando por lo que esta se bloquea.

c) Cinco conditions para los túneles(una para cada túnel), una condition para repostar y una condition para controlar la pantalla.

EJERCICIO 4

• **Recursos no compartibles. Los que se hayan identificado en el problema.**
- La pantalla. Debe ser controlada por un buzón para que solo un hilo pueda escribir en ella en un mismo instante.

• **Condiciones de sincronización. Las detectadas en el problema.**
- Los buzones pantalla, registro, votar, listaCandidatos, usuario para la clase Usuario.
- Los buzones pantalla, registro, voto, servidor, admin, candidatos, buzonUsuarios de la clase Servidor.
- Los buzones admin, servidor de la clase Admin.

• **Pseudocódigo.**

```
Program Ejercicio4


Process Usuario

int id, token, voto

Buzon pantalla, registro, votar, listaCandidatos, usuario

boolean decidoVotar = true

boolean estoyRegistrado = false

boolean heVotado = false

boolean pidoListaCandidatos = true

int[] paquete

ArrayList<String> lista

Selector s


send(registro, id)

// Mando la petición para registrarme, si hay menos de
100 usuarios registrados se genera un token (en este caso es
el mismo que la id)

// sino se manda un -1 por el buzón indicando que no se ha
podido registrar.

select()

     or:
```

```
          pantalla.receive()

          imprimir("Soy el usuario " + id + " y no me he
podido registrar.")

          send(pantalla, “”)

          or:

          receive(usuario, token)

          if (token != -1)

               estoyRegistrado = true

               receive(pantalla)

               imprimir("Soy el usuario " + id + " y me he
registrado correctamente, y este es mi token: " + token + ".")

               send(pantalla, “”)

           else

               receive(pantalla)

               imprimir("Soy el usuario " + id + " y no me he
podido registrar.")

               send(pantalla, “”)

     if (pidoListaCandidatos)     // El usuario puede pedir la
lista de candidatos siempre que quiera.

          send(listaCandidatos, id);

          switch

               or:

               receive(usuario, lista)


     if (estoyRegistrado && decidoVotar)   // Si el usuario
está registrado y quiere votar, se comprueba si ha pedido la
lista para ver a los candidatos.

          if (lista.vacia())               // Si no la tiene,
la pide.

               send(listaCandidatos, id)

               receive(usuario, lista)
```

```
     // Vota a un candidato de forma aleatoria mandando un
paquete con su token (es el mismo que su id) y su voto.

          voto = numeroAleatorio(1-5);

          paquete[0] = token

          paquete[1] = voto


          send(votar,paquete)

          select (s.selectOrTimeout(1000))

          or:

                    heVotado = receive(usuario)

       // Dependiendo de si la votación ha sido exitosa se
imprime una de las dos opciones.

               if (heVotado)

                    receive(pantalla)

                    imprimir(he votado a…)

                    send(pantalla,“”)

               else

                    receive(pantalla)

                    imprimir(no he podido votar)

                    send(pantalla,“”)


end select;


     if (estoyRegistrado && !decidoVotar)   // Si el usuario
pudo registrarse, pero no ha querido votar.

          receive(pantalla)

          imprimir(me abstengo)
```

```
          send(pantalla,"")


     if (!estoyRegistrado && decidoVotar)    // Si el usuario
quería votar, pero no pudo registrarse.

          receive(pantalla)

          imprimir(no he podido votar)

          send(pantalla,"")


Process Servidor


     ArrayList<String> listaCandidatos

     int[] votosCandidatos

     int[] conjuntoUsuarioRegistrados

     Buzon[] buzonUsuarios

     Set<Integer> conjuntoTokens

     Selector s

     Buzon pantalla, registro, voto, servidor, admin,
candidatos

          int usuariosRegistrados

          boolean fin

          boolean votacionAbierta


     while (!fin)

          select (s.selectOrTimeout(1000))

          or:

               receive(registro, id)

               if (usuariosRegistrados < 100)    // Si hay
menos de 100 usuarios registrados se registra a el usuario y
se le genera un token.
```

```
                int token = id
                conjuntoTokens.add(token)
                send(buzonesUsuarios[id], token)

conjuntoUsuarioRegistrados[usuariosRegistrados] = token
                usuariosRegistrados++
                if (usuariosRegistrados == 50) // Si ya hay 50
usuarios registrados se abre la votación.
                     votacionAbierta = true
                     receive(pantalla)
                     imprimir("Votación abierta: ")
                     send(pantalla, “”)
                     send(admin, “”)
                else
                     send(buzonesUsuarios[id],-1)


          or:
                     receive(candidatos, id)

send(buzonesUsuarios[id2],listaCandidatos)// Se le manda la
lista de candidados al usuario.


          or:
                receive(voto, paquete);  // Recibe el voto del
usuario y lo guarda.
                int id3 = paquete[0]
          // Si la votación ha sido exitosa devuelve true, si
no devuelve false.
                if (votacionAbierta)
                     if (contiene(conjuntoTokens(id3)))
                               votosCandidatos[paquete[1]]++
```

```
                        send(buzonesUsuarios[id3], true)
                    else
                        send(buzonesUsuarios[id3], false)

                else
                    send(buzonesUsuarios[id3], false)

          or:
     servidor.receive()//Recibe el mesanje del admin para
cerrar la votación.
               votacionAbierta = false;
               fin = true
end select;
     int max = -1
     String candidatoGanador = ""

     // Si hay mas de 25 votos en total, la votación es válida
y se muestra al ganador por pantalla.

     int totalVotos = 0
     for (int i = 0; i < votosCandidatos.length; i++)
          if (votosCandidatos[i] > max)
               max = votosCandidatos[i];
               candidatoGanador = listaCandidatos.get(i)
          totalVotos += votosCandidatos[i]

     if (totalVotos >= 25)
          receive(pantalla)
```

```
            imprimir(ganador)
            send(pantalla,"")
      else
            imprimir(votacion no valida)

process Admin

      Buzon admin, servidor

      receive(admin)
      sleep(1000)
      send(servidor, "Cerrar votación")

process Main

      send(pantalla,"")
      buzones
      servidor([]buzon, registro, votar, pantalla, admin,
adminCerrar, lista)
      admin(Admin, admincerrar)

      for(i = 1 to 120)
            new usuario(buzon[i], registro, pantalla, votar,
            lista)
            usario.start

      for(i = 1 to 120)
            usuario.join
```

```
servidor.join

admin.join
```

• Cuestiones planteadas en los ejercicios.

a) Usando un buzón por el que el usuario le manda la petición para conseguir la lista de candidatos y el servidor se la proporciona.
b) También se usa un buzón, antes de escribir por pantalla tienes que recibirla (receive(pantalla)) y cuando terminas la mandas (send(pantalla, “”)).
c) El servidor después de recibir su voto, le manda al usuario (por su buzón personal) true si la votación fue admitida o false si fue rechazada (dependiendo si la votación estaba abierta o cerrada.