P3 – Monitores

CONSIDERACIONES PARA RESOLVER LOS EJERCICIOS:

- **Los monitores utilizan el protocolo signal and continue.**
- **A una variable condition SÓLO pueden aplicársele las operaciones SIGNAL, SIGNALALL y WAIT.**
- **NO puede utilizarse el wait con prioridades.**
- **NO se puede utilizar ninguna operación que determine la cantidad de procesos encolados en una variable condition o si está vacía.**
- **La única forma de comunicar datos entre monitores o entre un proceso y un monitor es por medio de invocaciones al procedimiento del monitor del cual se quieren obtener (o enviar) los datos.**
- **No existen variables globales.**
- **En todos los ejercicios debe maximizarse la concurrencia.**
- **En todos los ejercicios debe aprovecharse al máximo la característica de exclusión mutua que brindan los monitores.**
- **Debe evitarse hacer busy waiting.**
- **En todos los ejercicios el tiempo debe representarse con la función delay.**

1. Se dispone de un puente por el cual puede pasar un solo auto a la vez. Un auto pide permiso para pasar por el puente, cruza por el mismo y luego sigue su camino.

```
Monitor Puente
   cond cola;
   int cant= 0;

   Procedure entrarPuente ()
      while ( cant > 0) wait (cola);
      cant = cant + 1;
   end;

   Procedure salirPuente ()
      cant = cant – 1;
      signal(cola);
   end;
End Monitor;

Process Auto [a:1..M]
   Puente. entrarPuente (a);
   "el auto cruza el puente"
   Puente. salirPuente(a);
End Process;
```

a. ¿El código funciona correctamente? Justifique su respuesta.

No. Al salir del puente decrementa la cantidad, lo que permite que un proceso que no estaba encolado entre, pero a la vez hace el signal, lo que permite que el primero encolado también lo haga, pudiendo tener dos autos a la vez.

b. ¿Se podría simplificar el programa? ¿Sin monitor? ¿Menos procedimientos? ¿Sin variable condition? En caso afirmativo, rescriba el código.

```
Monitor Puente
        Procedure Pasar ()
                Atravesarpuente()
        End;
End;

Process auto [1..A]
        Puente.atravesarpuente();
End;
```

c. ¿La solución original respeta el orden de llegada de los vehículos? Si rescribió el código en el punto b), ¿esa solución respeta el orden de llegada?

La original intenta respetar el orden, pero permite que se metan autos no encolados, supongo que por error. La reescrita no respeta el orden, ya que no solicita esto la consigna.

2. Existen N procesos que deben leer información de una base de datos, la cual es administrada por un motor que admite una cantidad limitada de consultas simultáneas.
a) Analice el problema y defina qué procesos, recursos y monitores serán necesarios/convenientes, además de las posibles sincronizaciones requeridas para resolver el problema.
b) Implemente el acceso a la base por parte de los procesos, sabiendo que el motor de base de datos puede atender a lo sumo 5 consultas de lectura simultáneas. ESTA RARO POR QUE RESPETA EL ORDEN Y NO LO PIDE ASI

```
Monitor BD
Cond cola
Int lectores=0;
Int esperando=0;

Procedure entrar()
        If (cant==5) then
                Esperando:=esperando+1;
                Wait (cola);
        Else
                Cant:=cant+1;
End;

Procedure salir ()
        If (esperando>0) then
                Esperando:= esperando-1;
                Signal (cola);
        Else
                Cant:=cant-1;
End;
End;

Process lector [id: 1..N]
        Monitorbd.entrar();
        //leerinfo
        Monitorbd.salir();
End;
```

3. Existen N personas que deben fotocopiar un documento. La fotocopiadora sólo puede ser usada por una persona a la vez. Analice el problema y defina qué procesos, recursos y monitores serán necesarios/convenientes, además de las posibles sincronizaciones requeridas para resolver el problema. Luego, resuelva considerando las siguientes situaciones:
a) Implemente una solución suponiendo no importa el orden de uso. Existe una función Fotocopiar() que simula el uso de la fotocopiadora.

```
Monitor fotocopiadora
        Procedure pasar()
                Fotocopiar()
        End;
End:

Process persona [id:1..N]
        Fotocopiadora.pasar();
End;
```

b) Modifique la solución de (a) para el caso en que se deba respetar el orden de llegada.

```
Monitor fotocopiadora
Boolean libre=true;
Cond cola
```

```
Int esperando=0;

Procedure entrar()
        If (libre=false) then
                Esperando:=esperando+1;
                Wait (cola);
        Else
                Libre=false;
End;

Procedure salir ()
        If (esperando>0) then
                Esperando:= esperando-1;
                Signal (cola);
        Else
                Libre=true;
End;
End;

Process persona [id: 1..N]
        fotocopiadora.entrar();
        //fotocopiar()
        Fotocopiadora.salir();
End;
```

c) Modifique la solución de (b) para el caso en que se deba dar prioridad de acuerdo con la edad de cada persona (cuando la fotocopiadora está libre la debe usar la persona de mayor edad entre las que estén esperando para usarla).

```
Monitor fotocopiadora
Boolean libre=true;
Cond espera[n];
Int esperando=0;
Int id;
colaOrdenada Cola;

Procedure entrar(id, edad)
        If (libre=false) then
                EncolarConPrioridad (cola, id, edad); //encolar se asume que funciona, no se implementa
                Wait (espera[id]);
        Else
                Libre=false;
End;

Procedure salir ()
        If (cola.isempty()=false) then
                Desencolar (cola, id)
                Signal (espera[id]);      //no puede hacerle signal a una cola normal por que sacaria el 1º
        Else
                Libre=true;
End;
End;

Procedure persona [id: 1..N]
        Int edad;
```

```
        fotocopiadora.entrar(id, edad);
        //fotocopiar()
        Fotocopiadora.salir();
End;
```

d) Modifique la solución de (a) para el caso en que se deba respetar estrictamente el orden dado por el identificador del proceso (la persona X no puede usar la fotocopiadora hasta que no haya terminado de usarla la persona X-1).
e) Modifique la solución de (b) para el caso en que además haya un Empleado que le indica a cada persona cuando debe usar la fotocopiadora.
f) Modificar la solución (e) para el caso en que sean 10 fotocopiadoras. El empleado le indica a la persona cuál fotocopiadora usar y cuándo hacerlo.

4. Existen N vehículos que deben pasar por un puente de acuerdo con el orden de llegada. Considere que el puente no soporta más de 50000kg y que cada vehículo cuenta con su propio peso (ningún vehículo supera el peso soportado por el puente).

```
Monitor adminPuente
Pesoactual=0;
Cola cola;
Cond espera;

        Procedure llegar(id, pesoauto)
                if (pesoactual+pesoAuto <= 50000) then
                        encolar (cola, id, pesoAuto);
                        wait(espera);
                else
                        pesoactual:= pesoactual + pesoAuto;

        Procedure salir(){
                Boolean ok = true;
                Int peso_aux;
                pesoactual:= pesoactual – pesoAuto;
                while(cola.empty()=false and ok=true) then
                        top(cola, peso_aux);     //devuelve el primer valor de la cola pero sin desencolarlo
                        if(pesoactual+peso_aux<=50000) then
                                desencolar(cola, id, peso_aux);
                                pesoactual:=pesoactual+peso_aux;
                                signal(espera);
                        else
                                ok = false;
                end;

Procedure auto [id: 1..n]
        Int Pesoauto;
        Adminpuente.llegar (id, pesoauto);
        //pasar el puente;
        Adminpuente.salir();
```

5. En un corralón de materiales se deben atender a N clientes de acuerdo con el orden de llegada. Cuando un cliente es llamado para ser atendido, entrega una lista con los productos que comprará, y espera a que alguno de los empleados le entregue el comprobante de la compra realizada.
a) Resuelva considerando que el corralón tiene un único empleado.
Monitor Admin

```
Cola C;
Cond espera;
Cond HayPedido;
String comprobante[C];

Procedure Pedido (IdC, lista, comprob)
        encolar (C, idC, lista);        //Los encolas si o si, para respetar el orden. El empleado debe ser
        signal (HayPedido);             //otro proceso para que no trabe el monitor
        wait (espera);
        c = comprobante [idC];

Procedure Siguiente (IdC; lista)
        if (empty (C)) then
                wait (HayPedido);
        desencolar (C, IdC, lista));

Procedure Resultado (IdC; comprob)
        comprobante[IdC] = comprob;
        signal (espera);
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Process empleado
        String lista, comprobante;
        int aux;
        for i:=1 to C do
                Admin.Siguiente(aux, lista);
                comprobante = generarcomprobante(lista);
                Admin.Resultado (aux, comprobante);

Process Cliente [id: 1..C]
        String lista, comprobante;
        //hacer lista L
        corralon.Pedido(id, Lista, comprobante);
```

b) Resuelva considerando que el corralón tiene E empleados (E > 1).

```
Monitor Admin
Cola C;
Cond espera[N];
Cond HayPedido;
String comprobante[C];

Procedure Pedido (IdC, lista, comprob)
        encolar (C, idC, lista);
        signal (HayPedido);
        wait (espera[idC]);             //debido a que puede ser que un servidor termine y entregue el
        c = comprobante [idC];          //resultado mas rápido que otro, asi que asegura entregar
                                        //al correcto y no al primero
Procedure Siguiente (IdC; lista)
        while (empty (C)) then
                wait (HayPedido);
        desencolar (C, IdC, lista));

Procedure Resultado (IdC; comprob)
        comprobante[IdC] = comprob;
        signal (espera[idC]);
```

```
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Process empleado
        String lista, comprobante;
        int aux;
        for i:=1 to C do
                Admin.Siguiente(aux, lista);
                comprobante = generarcomprobante(lista);
                Admin.Resultado (aux, comprobante);

Process Cliente [id: 1..C]
        String lista, comprobante;
        //hacer lista L
        corralon.Pedido(id, Lista, comprobante);
```

6. Existe una comisión de 50 alumnos que deben realizar tareas de a pares, las cuales son corregidas por un JTP. Cuando los alumnos llegan, forman una fila. Una vez que están todos en fila, el JTP les asigna un número de grupo a cada uno. Para ello, suponga que existe una función AsignarNroGrupo() que retorna un número "aleatorio" del 1 al 25. Cuando un alumno ha recibido su número de grupo, comienza a realizar su tarea. Al terminarla, el alumno le avisa al JTP y espera por su nota. Cuando los dos alumnos del grupo completaron la tarea, el JTP les asigna un puntaje (el primer grupo en terminar tendrá como nota 25, el segundo 24, y así sucesivamente hasta el último que tendrá nota 1). Nota: el JTP no guarda el número de grupo que le asigna a cada alumno.

```
process alumno()[idA:1..50]{
   int idG;
   int t;

   organizarInicio.iniciarAlumno(idA, idG);
   random(t);
   #Delay(t) #Hace la tarea
   grupo[idG].entregar()
}

process JTP(){
   organizarInicio.iniciarJTP();
   entregas.esperandoGrupos()
}

Monitor organizarInicio
   int cantAlumnos;
   cond esperandoATodos[1..50]
cond JTPespera;
   cola 50alumnos;
   int numG;

procedure iniciarAlumno(in idA; out idG){
      cantAlumnos++;
      50alumnos.push(idA)
      if cantAlumnos < 50{
         wait(esperandoATodos[idA]);}
      else{
         signal(JTPespera);
      idG = numG;
      signal(JTPespera);
```

```
    }

    procedure iniciarJTP(){
        if cantAlumnos < 50{
            wait(JTPespera);
}
        for i: 1..50{
        idA= cola.pop();
        numG= DarNumero();
        signal(esperandoATodos[idA]);
        wait(JTPespera);
}
    }
}



Monitor entregas{
    int vectorEntregas[1..25] = 0
    int nota=25;
    int notas[1..25];
queue colaEntregas;
cond esperandoNota[1..25];
    procedure darNota(out notaEntregar){
        notaEntregar = nota;
        nota--;
    }

    produre avisarEntregaGrupal(in idG, out notaFinal){
        colaEntregas.push(idG)
        signal(esperaJTP);
        wait(esperandoNota[idG]);
        notaFinal=notas[idG];
    }


    prodecure esperandoGrupos(){
        while cantEquiposTerminados < 25{
if colaEntregas.empty()
wait(esperaJTP);
idG=colaEntregas.pop()
        vectorEntregas[idG]++;     #representa cant de alumnos x grupo que entregaron
    if vectorEntregas[idG] == 2{
    cantEquiposTerminados ++;
    notas[idG]=darNota()
    signal_all(esperandoNota[idG]);
    }
}
    }
}



# Opcion2 :No respeta el enunciado pero maximiza la concurrencia
```

```
process alumno()[idA:1..50]{
   int idG;
   int t;

   JTP.iniciarAlumno(idG);
   random(t);
   #Delay(t) #Hace la tarea
   grupo[idG].entregar()
}


Monitor JTP{
   int vectorEntregas[1..25] = 0
   int nota=25;

   procedure DarNumero(out grupo)
      #viene implementado

   procedure iniciarAlumno(in idA; out idG){
      cantAlumnos++;
      if cantAlumnos < 50{
         wait(esperandoATodos);
      else{
         signal_all(esperandoATodos);
      idG=DarNumero(idG)
      }

   procedure darNota(out notaEntregar){
      notaEntregar = nota
      nota--
   }
}

Monitor grupo[id=1..25]{
   int cantAlumnos=0; #sorry
   cond esperarNota;
   int notaGrupo;

   procedure entregar(out nota)
cantAlumnos++;
      if cantAlumos == 2 { # no cumple el enunciadoo (el jtp deberia regular que lleguen los 2)
JTP.darNota(id,notaGrupo)
signal(esperarNota)
}
wait(esperarNota)
nota=notaGrupo
}
```

```java
Monitor Jefe{
   int notas[25] = [25](0);
   int notas_cant[25] = [25](0);
```

```
    cond vc_espera_nota[25];
    int nota_actual = 25;
    int cant = 0;
    cond vc_espera;

    Procedure recibir_numero(nro : out int){
      cant++;
      if(cant == 50){
        signal_all(vc_espera);
      }else{
        wait(vc_espera);
      }

      nro = DarNumero();
    }

    Procedure corregir(nro_grupo : in int, nota : out int){
      notas_cant[nro_grupo]++;
      if(notas_cant[nro_grupo] == 2){
        notas[nro_grupo] = nota--;
        signal(vc_espera_nota[nro_grupo]);
      }else{
        wait(vc_espera_nota[nro_grupo])
      }

      nota = notas[nro_grupo];
    }
  }

  Process Alumno[id: 1..50]{
    int nro_grupo;
    Jefe.recibir_numero(nro_grupo);
    Jefe.corregir(nro_grupo, nota)
  }
```

7. En un entrenamiento de fútbol hay 20 jugadores que forman 4 equipos (cada jugador conoce el equipo al cual pertenece llamando a la función DarEquipo()). Cuando un equipo está listo (han llegado los 5 jugadores que lo componen), debe enfrentarse a otro equipo que también esté listo (los dos primeros equipos en juntarse juegan en la cancha 1, y los otros dos equipos juegan en la cancha 2). Una vez que el equipo conoce la cancha en la que juega, sus jugadores se dirigen a ella. Cuando los 10 jugadores del partido llegaron a la cancha comienza el partido, juegan durante 50 minutos, y al terminar todos los jugadores del partido se retiran (no es necesario que se esperen para salir).

```
monitor equipo
int equipos[4]= ([4] 0)
cond miEquipo[4]
cond esperaraquipo[4]
int esperandorival:=0;
cola colaparajugar;

procedure llegar (miEquipo)
        equipos[miEquipo]=equipos[miEquipo]+1
        if (equipos[miEquipo] < 5)
                wait(esperarEquipo[miEquipo])
        else
```

```
                esperandoRival:= esperandoRival+1
                colaParaJugar.push(miEquipo);

monitor equipo[id:1..4] {
        canchaAsignada = 0;
        int cantjugadores:=0;

        procedure llegar (int out: miCancha) {
                cantjugadores:=cantjugadores+1
                if (cantjugadores = 5) then
                        match.matchear(miCancha)
                        canchaAsignada = miCancha
                        signal_all(esperarEquipo)
                else
                        wait(esperarEquipo)
                        miCancha = canchaAsignada

monitor match {
        cond esperarrival[0..1]
        int cantEquipo = 0;
        procedure matchear (int out cancha) {
                        cantEquipo:=cantequipo+1;
                        if (cantEquipo < 3) then                //son los 2 primeros
                                cancha = 1
                                if (cantEquipo=1)               //solo llego 1ero
                                        wait(esperarRival[0])
                                else                            //si no llegaron los 2
                                        signal(esperarRival[0])
                        else
                                cancha = 2
                                if (cantEquipo=3)               //si llegaron 3 equipos
                                        wait(esperarRival[1])
                                 else                           //si ya llegaron los 4
                                        signal(esperarRival[1])

monitor cancha[id:1..2] {
        cantJugadores = 0
        cond jugarPartido
        cond esperarJugadores

        procedure esperar () {
                if(cantJugadores<10)
                        wait(jugarPartido)

        procedure jugar ()
                cantJugador:=cantjugadores+1;
                if (cantJugadores=10)
                        signal(jugarPartido)
                wait(esperarJugadores)

        procedure finPartido(){
                signal_all(esperarJugadores)
```

*-

```
process personas [id:0..19]
        miEquipo = DarEquipo();
        equipo[miEquipo].llegar(miCancha)
        cancha[miCancha].jugar()

process partido [id:1..2]
        cancha[id].esperar()
        delay(50)  //jugarPartido
        cancha[id].finPartido()


==================================OTRA OPCION
monitor entrenamiento[1..4]
        int equipo:=0;
        cond conformación;

        procedure llegar(OUT int nrocancha){
                equipo:=equipo+1;
                if(equipo == 5)
                        signal_all(conformacion);
                        administracion.pedir_cancha(cancha);
                        nrocancha = cancha;
                else{
                        wait(conformacion);
                        nrocancha = cancha;

Monitor administracion
int cancha_disponible = 1 equiposlistos = 0;

        procedure pedir_cancha(OUT int cancha)
                equiposlistos:=equiposlistos+1;
                cancha = cancha_disponible;
                if(equiposlistos == 2){
                        cancha_disponible:=cancha_disponible+1;
                        equiposlistos = 0;

Monitor canchas[1..2]           ESTO ESTA RARO. NUNCA DESPIERTA A LOS QUE SE DURMIERON EN JUGAR

        int jugadores:=0
        procedure llegar()
        jugadores:=jugadores+1;
        if(jugadores == 10)
                        delay(50);//no hay problema que este demorando aca por que no compite con nadie
para usarlo
                else
                        wait(jugar);

process jugador(1..20){
        nro = DarEquipo();
        entrenamiento[nro].llegar(asignada);
        cancha[asignada].llegar();
```

8. Se debe simular una maratón con C corredores donde en la llegada hay UNA máquina expendedoras de agua con capacidad para 20 botellas. Además, existe un repositor encargado de reponer las botellas

de la máquina. Cuando los C corredores han llegado al inicio comienza la carrera. Cuando un corredor termina la carrera se dirigen a la máquina expendedora, espera su turno (respetando el orden de llegada), saca una botella y se retira. Si encuentra la máquina sin botellas, le avisa al repositor para que cargue nuevamente la máquina con 20 botellas; espera a que se haga la recarga; saca una botella y se retira. Nota: mientras se reponen las botellas se debe permitir que otros corredores se encolen.

```
Monitor carrera
Cant_corredores=:0;
        Procedure largada ()
                Cant_corredores:=cant_corredores+1;
                If (cant_corredores=c) then
                        Signalall(barrera);
                Else
                        Wait(barrera);

Monitor expendededora
        Boolean haybotellas:=true
        Int botellas:=20;
        Cond reponer;
        Cond esperar;
        Cola c;

Procedure Tomarbotella
        if (notlibre) then
                encolarme(c, id)
        else
                If (not haybotellas) then
                        //agarrar una
                        Botellas:=botellas-1;

                Else
                        Haybotellas:=false;
                        Signal (reponer);
                        Await (esperar);
                        //agarrar una
                        Botellas:=botellas-1;
                        Haybotellas:=true;


Process Corredor [id: 1..C]
        Carrera.largada ()
        //correrrcarrera
        Expendedora.tomarbotella()
        //irse
```

```
OPCION A =
process corredor [id: 0..N] {
   Carrera.iniciar()
   // Corren carrera
   Fila.llegar()
   Maquina.AgarrarBotella()
}
```

```
process reponedor {
   cantB = 20
   while(true) {
      Fila.proximo()
      cantB--
      if(cantB==0){
         Maquina.reponer()
         cantB=20
      }
   }
}

-------------------------------------------
process corredor [id: 0..N] {
   Carrera.iniciar()
   // Corren carrera
   Fila.llegar()
   Maquina.AgarrarBotella()
   Fila.salir()
}
process reponedor {
   for i= 1..N/20 {
      Maquina.reponer()
   }
}
```