Programación concurrente – Tres Arroyos

Monitores

Inconvenientes de semáforos:

Exclusión mutua y sincronización mezcladas.
Mal uso de P y V.

Definición:

Es una estructura que encapsula datos y provee un conjunto de procedimientos de manera que un solo proceso a la vez puede acceder llamando a cualquiera de ellos.

Convenciónes:

- Utilizar m_nombreDelMonitor;
- Utilizar c_nombreDeVariableCondicion;
- Definicion del Monitor:

Simple:

```
Monitor Mnombre
End Monitor;
```

Arreglo de monitores:

```
Monitor Mnombre[1..N]
End Monitor;
```

- Definicion de variables condición:
  Simple: Condition c_nombre;
  Arreglo: Condition c_nombre[1..N]

Exclusión mutua y sincronización:

Exclusión mutua:

Exclusión mutua implícita. Cola asociada al monitor. NO HAY VARIABLES GLOBALES TODO DATO COMPARTIDO SE ENCAPSULA.

```
Process  Proceso[p:1..P]
        MRegionCritica.usarRegionCritica();
End Proceso;

Monitor MRegionCritica
        Procedure usarRegionCritica()
                "usa RC"
        end usarRegionCritica;
End Monitor;
```

Sincronización:

Sincronización mediante variables de condición.
Operaciones: wait, signal, signalAll.
Signal sin efecto si no hay procesos dormidos.
Cola asociada a las variables condición.
Funcionamiento modos signal and wait y signal and continue.

Ejemplo: el proceso P1 espera al proceso P2 para continuar su ejecución.

```
Process  P1
        MSincronizar.esperar();
End Proceso;

Process  P2
        MSincronizar.seguir();
End Proceso;

Monitor MSincronizar
Condition c;
Boolean llegoP2=false;
        Procedure esperar()
                If not (llegoP2)
                        wait(c);
                End if;
        End  esperar;

        Procedure seguir()
                llegoP2=true;
                signal(c);
        End seguir;
End Monitor;
```

Todo proceso dormido que recibe un signal se despierta y se encola en la cola del monitor, cuando le toca continuar su ejecución lo hace en la instrucción siguiente al wait en que estaba dormido previamente.

Maximización de la concurrencia:

Ejemplo: existen N procesos niños y M procesos niñas, existe además un auto y una muñeca, los niños compiten por jugar con el auto y las niñas compiten por jugar con la muñeca, ambos utilizan el juguete un tiempo y lo liberan.

Error común: Usar un monitor para todo.

```
Process Niño[n:1..N]
        MJuguetes.jugarAuto();
End Niño;
```

```
Process Niña[n:1..M]
        MJuguetes.jugarMuñeca();
End Niña;
```

```
Monitor MJuguetes
        Procedure jugarAuto()
                "Juega"
        End jugarAuto;

        Procedure jugarMuñeca()
                "Juega"
        End jugarMuñeca;

End Monitor;
```

Solucion: El hecho de que un niño use el auto no debiera interferir en que una niña use la muñeca porque son eventos independientes. Se debe usar un monitor para cada interacción.

```
Process Niño[n:1..N]
        MAuto.jugar();
End Niño;
```

```
Process Niña[n:1..M]
        MMuñeca.jugar();
End Niña;
```

```
Monitor MAuto
        Procedure jugar()
                "Juega"
        End jugarAuto;
End Monitor;
```

```
Monitor MMuñeca
        Procedure jugar()
                "Juega"
        End jugarAuto;
End Monitor;
```

Obtener valores del monitor

Es posible obtener valores de las variables del monitor mediante pasaje de parámetros por referencia.

Error común: Usar funciones.

Ejemplo a continuación de productor-consumidor.

Orden de ejecución de los procesos en el llamado a un monitor

Siempre hay que tener en cuenta el orden en que los procesos llegan al llamar a un monitor.
Ejemplo a continuación de productor-consumidor, un productor llega antes que el consumidor o al revés.

Comunicación entre procesos utilizando monitores, modelo Productor - Consumidor:

Buffer con capacidad "infinita"

```
Process Productor
Var ítem:tipoItem;

        "produce item"
        MBuffer.depositar(item);
End Productor;

Process Consumidor
Var ítem:tipoItem;

        MBuffer.retirar(item);
        "consume item"
End Consumidor;

Monitor MBuffer
Var buffer:Queue;
    c:Condition;

        Procedure depositar(item)
                push(buffer,item);
                signal(c);
        End depositar;

        Procedure retirar(var item)
                If empty(buffer)
                        wait(c);
                Item = pop(buffer);
        End retirar;
End Monitor;
```

Errores comunes:

Busy waiting del consumidor si el buffer esta vacio, se suele entrar y salir a cada rato del monitor.

```
Process Consumidor
        While true()
                While not hayItem
                        M_buffer.consumir(hayItem,item);
                End while;
                "consumir item"
        End while;
```

```
        End Consumidor;

        Monitor M_buffer
                …
                Procedure consumir(Boolean hayItem, item)
                        If isEmpty(buffer)
                                hayItem=false;
                        else
                                hayItem=true;
                                item=pop(buffer);
                        End if;
                End consumer;
                …
        End Monitor;
```

Productores o consumidores de cantidad de ítems finita.

Ejemplo: Se tienen P procesos que entre todos realizan N tareas, cada proceso si hay tarea para hacer toma una tarea y la realiza, cada proceso puede tardar un tiempo distinto en realizar cada tarea.

```
Process Proceso[p:1..P]
Var miTarea:integer;

        Mtarea.obtenerTarea(t);
        While (t<>0)
                "Realiza la tarea"
                Mtarea.obtenerTarea(t);
        End while;
End Proceso;

Monitor Mtarea
Var tareas:integer=N ;

        Procedure obtenerTarea(var t:integer)
                t=tareas;
                if tareas>0
                        tareas=tareas – 1;
        End obtenerTarea;
End Monitor;
```

Error comun: Inconsistencia.

```
Process Proceso[p:1..P]
Var miTarea:integer;
    ok:boolean;

        Mtarea.hayTarea(ok);
        While (ok)
                Mtarea.obtenerTarea(t);
                "Realiza la tarea"
                Mtarea.hayTarea(ok);
        End while;
End Proceso;
```

```
Monitor Mtarea
Var tareas:integer=N ;

        Procedure hayTarea(var ok:boolean)
                if tareas>0
                        ok=true;
                else
                        ok=false;
        End obtenerTarea;

        Procedure obtenerTarea(var t:integer)
                t=tareas;
                tareas=tareas – 1;
        End obtenerTarea;
End Monitor;
```

Monitores como controladores de acceso y competencia entre procesos encolados por exclusión mutua y los despertados mediante signal:

Los monitores proveen exclusión mutua implícita pero hay ocasiones que es deseable que más de un proceso acceda a un recurso por lo tanto no se puede encapsular el recurso dentro del monitor, en este caso el monitor se debe utilizar como controlador de acceso.

Ejemplo: Supongamos que hay un recurso al que pueden acceder a lo sumo 2 procesos a la vez, un monitor para controlar el posible acceso podria ser:

Error común:

4 procesos p1, p2, p3 y p4 que hacen lo siguiente:

```
Process pi[i= 1..4]
 miMonitor.pedir();
  //utilizar recurso
 miMonitor.liberar();
end pi;

Monitor MiMonitor
var C:condition;
cantidadDeProcesos:integer = 0;

 procedure pedir()

   if (cantidadDeProcesos == 2)
    wait(c);
   end if;
   cantidadDeProcesos++;

 end pedir;

 procedure liberar()
```

```
  cantidadDeProcesos--;
  signalAll(c);

 end liberar;

end miMonitor;
```

Se ejecutan con el siguiente orden:

p1 entra al monitor llamando al procedimiento ingresar, como cantidadDeProcesos = 0 no ingresa al if, luego incrementa la variable cantidadDeprocesos dejandola en 1.
p2 hace lo mismo que p1 dejando la variable cantidadDeProcesos = 2.
P3 ingresa al monitor llamando al procedimiento ingresar, como cantidadDeProcesos = 2, ingresa al if y se queda dormido en la cola asociada a la variable c.
p2 llama a salir, decrementa cantidadDeProcesos dejandola en 1, hace un signalAll(c) despertando a p3 y deja el monitor.
Mientras p2 esta ejecutando el procedimiento salir hace una llamada al monitor p4, como el monitor esta siendo utilizado por p2 entonces p4 debe esperar en la cola asociada al monitor.
p3 se despierta pero p4 esta esperando entrar al monitor, por lo tanto p3 y p4 deberan competir por el acceso a este.
Si p4 le gana el acceso al proceso p3, ejecuta ingresar y como cantidadDeProcesos = 1 no entra al if y luego incrementa cantidadDeProcesos dejandola en 2.
p3, luego que p4 ejecuta ingresa al monitor continuando desde la linea siguiente a donde se habia dormido, despues del wait(c) en el procedimiento ingresar, incrementa la variable cantidadDeProcesos dejandola en 3 ERROR!!!, al recurso lo están utilizando 3 procesos p1, p4 y p3 cuando en realidad solo debería haber a lo sumo 2.

Soluciones:

Con while:

```
Monitor MiMonitor
var C:condition;
cantidadDeProcesos:integer = 0;

 procedure pedir()

  while (cantidadDeProcesos == 2)
   wait(c);
  end if;
  cantidadDeProcesos++;

 end pedir;

 procedure liberar()

  cantidadDeProcesos--;
  signalAll(c);

 end liberar;
```

```
end miMonitor;
```

Con passing the condition:

```
Monitor MiMonitor
var C:condition;
cantidadDeProcesos:integer = 0;
espera:integer=0;

 procedure pedir()

  if (cantidadDeProcesos == 2)
        espera++;
        wait(c);
        espera--;
  else
        cantidadDeProcesos++;
  end if;
 end pedir;

 procedure liberar()

  if espera > 0
        signal(c);
  else
    cantidadDeProcesos--;
 end if;

 end liberar;

end miMonitor;
```

Barreras:

Ejemplo: existen N autos que deben esperarse para largar una carrera.

```
Process Auto[a:1..N]
        MLargada.barrera();
        …
End Auto;

Monitor MLargada
Var cant:integer=0;
    c:condition;

        Procedure barrera()
                cant=cant + 1;
                if (cant==N)
                        signalAll(c);
```

```
                    Else
                        wait(c);
                End if;
        End barrera;
End Monitor;
```

Grupos

Armado

Ejemplo: Se tienen N jugadores que deben agruparse en equipos de E jugadores cada uno (E múltiplo de N).

```
Process jugador[j:1..N]
Var miEquipo:integer;

        MAsignarEquipos.obtenerEquipo(miEquipo);
        …
End Jugador;

Monitor MAsignarEquipos
Var nroEquipo:Integer=1;
    cant:integer=0;

        Procedure obtenerEquipo(var equipo)
                equipo=nroEquipo;
                cant = cant + 1;
                if (cant==E)
                        cant=0;
                        nroEquipo= nroEquipo + 1;
                end if;
        End obtenerEquipo;
End Monitor;
```

Interacción entre integrantes de cada grupo

Ejemplo: Siguiendo con el ejemplo anterior, una vez que los jugadores ya conocen su equipo deberían obtener el numero de camiseta.

Error común: usar un monitor para todos los grupos.

```
Process jugador[j:1..N]
Var miEquipo:integer;
    miNumero:integer;

        MAsignarEquipos.obtenerEquipo(miEquipo);
        MAsignarNumeros[miEquipo].obtenerNumero(miNumero);
        …
End Jugador;

Monitor MAsignarNumeros
Var numero:integer = 1;
```

```
        Procedure obtenerNumero(var miNumero)
                miNumero = numero;
                numero = numero + 1;
        End obtenerNumero;
End Monitor;
```

Solucion: usar un monitor para cada grupo.

```
Process jugador[j:1..N]
Var miEquipo:integer;
    miNumero:integer;

        MAsignarEquipos.obtenerEquipo(miEquipo);
        MAsignarNumeros[miEquipo].obtenerNumero(miNumero);
        …
End Jugador;

Monitor MAsignarNumeros[1..N/E]
Var numero:integer = 1;

        Procedure obtenerNumero(var miNumero)
                miNumero = numero;
                numero = numero + 1;
        End obtenerNumero;
End Monitor;
```

Competencia entre grupos

Ejemplo: Continuando con el ejemplo anterior, una vez que los jugadores tienen su numero se disponen a jugar contra otro equipo un partido de futbol, para eso deben conocer la cancha donde deben jugar.

Error común: hacer que todos los integrantes del equipo vayan a buscar una cancha.

Solucion: hacer que un solo integrante del equipo vaya a buscar una cancha mientras que los demás se quedan esperando.

```
Process jugador[j:1..N]
Var miEquipo:integer;
    miNumero:integer;
    miCancha:integer;

        MAsignarEquipos.obtenerEquipo(miEquipo);
        MAsignarNumeros[miEquipo].obtenerNumero(miNumero);
        MBuscarCancha[miEquipo].obtenerCancha(miCancha);
        …
End Jugador;

Monitor MBuscarCancha[1..N/E]
Var cancha:integer;
    cant:integer=0;
    c:condition;
```

```
        Procedure obtenerCancha(var miCancha)
                cant=cant + 1;
                if (cant==E)
                        MAsignarCancha.obtenerCancha(cancha);
                        signalAll(c);
                  Else
                        wait(c);
                End if;
                miCancha = cancha;
        End obtenerCancha;
End Monitor;

Monitor MAsignarCancha
Var cancha:integer=1;
    cant:integer=0;

        Procedure obtenerCancha(var miCancha)
                miCancha=cancha;
                cant=cant + 1;
                if (cant==2)
                        cant=0;
                        cancha=cancha + 1;
                End if;
        End obtenerCancha;
End Monitor;
```

Llamados de monitor a monitor

Ventajas: a veces es mas fácil llamar de un monitor a otro que entrar al primero, salir para entrar al segundo y competir nuevamente para volver a entrar al primero.

Desventajas:

- Esperas innecesarias.
- Posibles bloqueos usándolo con sincronización: un wait en el segundo monitor bloquea al primero y nadie más podrá entrar.
- Bloqueo absoluto si un proceso llama de un monitor a otro monitor y vuelve a llamar desde ahí al primero.

Notar en el ejemplo anterior que el llamado de monitor a monitor no provoca ningún inconveniente, dado que, en ese momento, todos los procesos que podrían entrar al monitor MBuscarCancha estarán dormidos en la variable condición de ese mismo monitor. Si se solucionara sin el llamado de monitor a monitor, entonces, el último jugador debería entrar al monitor para saber que él debe ser el que tiene que ir a buscar la cancha, salir y verificar en el código del proceso si es el que tiene que ir a buscar la cancha o es un proceso que se había dormido esperando la cancha, si es el que tiene que ir a buscar la cancha debería entrar al otro monitor que le asigne la cancha y luego volver a despertar a sus compañeros.

Combinación de barreras con creación de grupos

Error comun: Usar un solo monitor para asignación y la barrera, no maximiza la concurrencia. Un proceso que ya tiene su numero de grupo y esta esperando a sus compañeros no debería entorpecer la asignación de grupo de otro proceso.

Signals selectivos:

Existen N alumnos que reciben una tarea, cada alumno demora un tiempo en resolverla, cuando terminan se van retirando en orden de acuerdo al numero de tarea (no se puede retirar el alumno con la tarea i si no se retiro el i-1).

```
Process Alumno[a:1..N]
        miTarea= obtenerTarea();
        "realiza tarea";
        M_FinTarea.retirarse(miTarea);
End process;
```

Desde el próximo a retirarse:

```
Monitor M_FinTarea
        Condition c[1..N];
        Proximo:integer=1;

        Procedure retirarse(int idTarea)
                If proximo <> idTarea
                        Wait(c[idTarea]);
                End if;

                If idTarea < N //Por si es el ultimo
                        proximo++;
                        Signal(c[proximo]);
                End if;
        End Procedure;
End Monitor;
```

Desde el que se retiro:

```
Monitor M_FinTarea
        Condition c[1..N];
        seRetiro:integer=1;

        Procedure retirarse(int idTarea)
                If seRetiro <> idTtarea - 1
                        Wait(c[idTarea]);
                End if;

                seRetiro++;

                If idTarea < N //Por si es el ultimo
                        Signal(c[idTarea+1]);
                End if;
        End Procedure;
```

```
End Monitor;
```

Errores comunes:

Hacer una barrera esperando a todos y dejar esperando procesos que pueden retirarse.
Hacer un signalAll y despertar a todos, vuelven a competir por el monitor, hay que tener cuidado con el código, pueden despertar procesos que no se tienen que ir.

Relojes:

Ejemplo: Se tiene un banco con una sola caja, los clientes llegan se encolan y esperan ser atendidos, si pasados 15 minutos un cliente no fue atendido se retira.

```
Process Persona[p:1..N]
Estado:string;

        M_cola.encolarse(p);
        M_reloj[p].contar();
        M_persona[p].esperar(estado);
        If estado = “atencion”
                “Será atendido”
        End if;
End Process;

Process Reloj[r:1..N]
        M_reloj[r].esperaContar();
        Delay(15);
        M_persona[r].avisoReloj();
End Process;

Process Empleado
estado:string;
cantClientes:integer=0;

        While (cantClientes < N)
                M_cola.obtenerCliente(cliente);
                M_persona[cliente].avisoAtencion(estado);
                If estado = “atencion”
                        “Será atendido”
                End if;
                cantClientes++;
        End While;
End Process;

Monitor M_cola
Cola:queue;
c:condition;

        Procedure encolarse(int persona)
```

```
            Push(cola,persona);
            Signal(c);
        End procedure;

        Procedure ObtenerCliente(int cliente)
            If empty(cola)
                Wait(c);
            cliente=pop(cola);
        End procedure;
End Monitor;

Monitor M_reloj[1..N]
c:condition;
llegoCliente:boolean=false;

        Procedure contar()
            Signal(c);
            llegoCliente=true;
        End Procedure;

        Procedure esperaContar()
            If not llegoCliente
                Wait(c);
        End procedure;
End Monitor;

Monitor M_persona[1..N]
Estado:string = “”;
c:condition;

        Procedure esperar(String estadoPersona)
            If estado = “”
                wait(c);
            estadoPersona=estado;
        End procedure;

        Procedure avisoReloj()
            If estado = “”
                estado=”reloj”;
                signal(c);
            end if;
        End procedure;

        Procedure avisoAtencion(estadoCliente)
            If estado = “”
                estado=”atencion”;
                signal(c);
            end if;
            estadoCliente=estado;
        End procedure;

End Monitor;
```