Programación Concurrente - Practica 3

Tomatis, Vicens, Torchia, Olmos, Manzin, Petraccaro, Traberg, Villareal

(Ejercicio 1)

Consigna:

```
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;
```

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.

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

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

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?

Respuesta:

a) Parece bien. El primero llega y no entra al while e increrenta cant en 1. A partir de ahí, todos los que lleguen se encolan y se duermen. Cuando el auto salga, disminuye en 1 y hace un signal.

b) Se podria simplificar a un solo procedure cruzar que cruce el puente. No es necesario la variable condition

Ejercicio 1b

```
Monitor Puente{
    Procedure Cruzar(){
        CruzoPuente()
    }
}
Process Auto[a:1..M]{
    Puente.Cruzar()
}
```

c) No y no

(Ejercicio 2)

Consigna:
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.

Respuesta:

Base

```
Monitor Base
    cond espera
    int cant = 0
    Procedure PideAcceso(){
        while (cant == 5){
           wait(espera)
        }
        cant++
    }
    Procedure Libera(){
        cant--
        signal(espera)
    }
End Base;

```

Usuario

```
Process Usuario[id:1..M]
    Base.PideAcceso()
    -- Usa la BD
    Base.Libera()
End Usuario;
```

(Ejercicio 3)

Consigna:
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.

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

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).

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.

Respuesta:

A_Monitor

```
    Monitor Fotocopiadora
        cond espera
        boolean libre=true

        Procedure Usar(){
            while (not libre){ wait(espera)}
            libre = false
        }
        Procedure Dejar(){
            libre = true
            signal(espera)
        }
    End Fotocopiadora
```

A_Process

```
    Process Persona[id:1..M-1]
        Fotocopiadora.Usar()
        Fotocopiar()
        Fotocopiadora.Dejar()
    End Persona
```

Solucion mas facil

Monitor Fotocopiadora

```
Monitor Fotocopiadora{
    Procedure hacerFotocopia(){
        Fotocopiar()
    }
}
```

Proceso Persona

```
Process Persona[id:1..M-1]{
    Fotocopiadora.hacerFotocopia()
}
```

B_Monitor

```
Monitor Fotocopiadora{
    cond espera
    int cantEspera = 0
    boolean libre=true

    Procedure Usar(){
        if (not libre){
            cantEspera++
            wait(espera)
        }
        else{
            libre=False
        }
    }
    Procedure Dejar(){
        if cantEspera > 0{
            cantEspera--
            signal(espera)
        }
        else{
            libre=True
        }
    }
}
```

B_Process

```
    Process Persona[id:1..M-1]
        Fotocopiadora.Usar()
        Fotocopiar()
        Fotocopiadora.Dejar()
    End Persona
```

C_Monitor

```
    Monitor Fotocopiadora
        ColaPrioridad dormidos
        cond espera[M]
        libre = false

        procedure Usar(in edad, id: int){
            if (not libre) {
                Insertar_Ordenado(dormidos, id, edad)
                wait(espera[id])
            } else{
                libre = false
            }
        }

        procedure Dejar(){
            if (empty(dormidos)){
                libre = true
            } else {
                pop(dormidos, id)
                signal(espera[id])
            }
        }
    End Fotocopiadora
```

C_Process

```
    Process Persona[id:1..M-1]
        edad = cuantosaniostengo()

        Fotocopiadora.Usar(edad,id)
        Fotocopiar()
        Fotocopiadora.Dejar()
    End Persona
```

D_Monitor

```
    Monitor Fotocopiadora
        cond espera
        int siguiente = 0
        Procedure Usar(in id: int){
            if (id != siguiente) {
                wait(espera[id])
            }
        }
        Procedure Dejar(){
            siguiente++;
            signal(espera[siguiente])
        }
    End Fotocopiadora
```

D_Process

```
    Process Persona[id:1..M-1]
        Fotocopiadora.Usar(id)
        Fotocopiar()
        Fotocopiadora.Dejar()
    End Persona
```

E_Monitor

```
Monitor Fotocopiadora
    cond espera
    int cantidad=0;

    Procedure Usar(){
        cantidad++;
        signal(hayEsperando)
        wait(espera)
    }
    Procedure Dejar(){
        signal(termine)
    }
    Procedure PermitirSiguiente(){
        if (cantidad==0){
            wait(hayEsperando)
        }
        signal(espera)
        wait(termine)
    }
End Fotocopiadora
```

E_Process

```
Process Persona[id:1..M-1]
    Fotocopiadora.Usar()
    Fotocopiar()
    Fotocopiadora.Dejar()
End Persona
```

```
Process Empleado
    for(i=1 to M){
        Fotocopiadora.PermitirSiguiente()
    }
End Empleado
```

Solucion alternativa

Monitor Entrada

```
Monitor Entrada{
    Procedure llegar(){
        if (not libre){
            esperando++
            wait(espera)
        }else{
            libre=false
        }
    }
    Procedure proximo(){
        if(esperando>0){
            esperando--
            signal(espera)
        }else{
            libre=true
        }
    }
}
```

Proceso Persona

```
Process Persona[id:1..M-1]{
    Entrada.llegar()
    Atencion.fotocopiar()
}
```

Monitor Atencion

```
Monitor Atencion{
    Procedure fotocopiar(){
        Fotocopiar()
        termino=true
        signal(terminoUsar)
    }
    Procedure esperarUso(){
        if(not termino){
            wait(terminoUsar)
        }
    }
}
```

Proceso Empleado

```
Process Empleado{
    for(i=1 to M){
        Entrada.proximo()
        Atencion.esperarUso()
    }
}
```

Ejercicio F

Proceso Fotocopiadora

```
Monitor Entrada{
    Cola<int> colaFotocs
    for i=0 to 9{
        push(colaFotocs,i)
    }

    Procedure llegar(out int idFotoc){
        cantEsperando++
        signal(hayPersonas)
        wait(usar)
        idFotoc=pop(colaFotocs)
    }

    Procedure proximo(){
        if(cantEsperando==0){
            wait(hayPersonas)
        }
        cantEsperando--
        if(empty(colaFotocs)){
            wait(hayFotocs)
        }
        signal(usar)
    }
    Procedure salir(in int idFotoc){
        push(colaFotocs,idFotoc)
        signal(hayFotocs)
    }
}
```

Proceso Persona

```
process Persona[id:0..P-1]{
    Entrada.llegar(idFotoc)
    Atencion[idFotoc].fotocopiar()
    Entrada.salir(idFotoc)
}
```

Proceso Empleado

```
process Empleado{
    for i=0 to P-1{
        Entrada.proximo()
    }
}
```

Monitor Atencion

```
Monitor Atencion[id:0..9]{
    Procedure fotocopiar(){
        Fotocopiar()
    }
}
```

(Ejercicio 4)

Consigna:
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).
Respuesta:

Monitor Puente

```
Monitor Puente{
    int capacidad=50000
    cond esperando
    Procedure Entrar(in int peso){
        if(peso>capacidad or not empty(colaEspera)){
            push(colaEspera,peso)
            wait(espera)
            if(not empty(colaEspera)){
                pesoAux=peek(colaEspera)
                if(pesoAux<capacidad-peso){
                    signal(espera)
                    pop(colaEspera)
                }
            }
        }
        capacidad-=peso
    }

    Procedure Sale(in int peso){
        capacidad+=peso
        if(not empty(colaEspera)){
            pesoAux=peek(colaEspera)
            if(pesoAux<capacidad){
                signal(espera)
                pop(colaEspera)
            }
        }

    }
}
```

Proceso Vehiculo

```
Process Vehiculo[id:0..V-1] {
    Puente.Entrar(this.peso)
    -- pasa
    Puente.Sale(this.peso)
}
```

(Ejercicio 5)

Consigna:
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.

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

Respuesta:

a) .

Monitor Corralon

```
Monitor Corralon{
    cond espera
    int esperando = 0
    bool libre = true
    Procedure Entrar(){
        if(not libre){
            esperando++
            wait(espera)
        }else{
            libre=false
        }
    }

    Procedure Salir(){
        if (esperando>0){
            esperando--
            signal(espera)
        }else{
            libre=true
        }
    }
}
```

Monitor Atencion

```
Monitor Atencion{
    Procedure EntregarLista(in text lista, out text
    ↪ comprobante){
        listaProductos=lista
        entrego=true
        signal(hayListas)
        wait(entregoComprobante)
        comprobante=comprobCompartido
        signal(agarroComprobante)
    }

    Procedure EsperarLista(out text lista){
        if(not entrego){
            wait(hayListas)
        }
        lista=listaProductos
    }
    Procedure EntregarComprobante(in text comprobante){
        comprobCompartido=comprobante
        signal(entregoComprobante)
        wait(agarroComprobante)
        entrego=false -- resetear variable
    }
}
```

Proceso Cliente

```
Process Cliente[id:0..C-1]{
    Corralon.Entrar()
    Atencion.EntregarLista(this.lista,comprobante)
    Corralon.Salir()
}
```

Monitor Empleado

```
Process Empleado{
    for i=1 to C{
        Atencion.EsperarLista(lista)
        comprobante=generarComprobante(lista)
        Atencion.EntregarComprobante(comprobante)
    }
}
```

b) .

Monitor Entrada

```
Monitor Entrada{
    int cantLibres=0
    Procedure Entrar(out int idEmpleado){
        if(cantLibres==0){
            esperando++
            wait(esperar)
        }else{
            cantLibres--
        }
        idEmpleado=pop(empleadosLibres)
    }

    Procedure Proximo(in int idEmpleado){
        push(empleadosLibres,idEmpleado)
        if(esperando>0){
            esperando--
            signal(esperar)
        }else{
            cantLibres++
        }

    }
}
```

Proceso Empleado

```
Process Empleado[id:0..E-1]{
    while(true){
        Entrada.Proximo(id)
        Atencion[id].EsperaLista(lista)
        -- verifica la lista
        comprobante = GenerarComprobante(lista)
        Atencion[id].EntregarComprobante(comprobante)
    }
}
```

Monitor Atencion

```
Monitor Atencion[id:0..E-1]{
    Procedure EntregarLista(in text lista, out text
    ↪ comprobante){
        listaCompartida=lista
        entrego=true
        signal(entregoLista)
        wait(entregoComprobante)
        comprobante=comprobanteCompartido
        signal(agarroComprobante)
    }

    Procedure EsperaLista(out text lista){
        if(not entrego) wait(entregoLista)
        lista=listaCompartida
    }

    Procedure EntregarComprobante(in text comprobante){
        comprobanteCompartido=comprobante
        signal(entregoComprobante)
        wait(agarroComprobante)
        entrego=false
    }
}
```

Proceso Cliente

```
Process Cliente[id:0..C-1]{
    Entrada.Entrar(idEmpleado)
    Aten-
    ↪ cion[idEmpleado].EntregarLista(lista,comprobante)
}
```

(Ejercicio 6)

Consigna:
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.
Respuesta:

Entrada

```
Monitor Entrada{
    Procedure Entrar(){
        cantidad++;
        if (cantidad==50){
            signal(pasar)
        }
        wait(barrera)
    }

    Procedure Avisar(){
        if(cantidad<50){
            wait(pasar)
        }
        signal_all(barrera)
    }
}
```

Grupos

```
Monitor Grupos{
    Procedure DarNumero(in int numero){
        push(colaNumeros,numero)
        signal(hayNumeros)
    }
    Procedure ObtenerNumero(out int numero){
        if(empty(colaNumeros)){
            wait(hayNumeros)
        }
        numero = pop(colaNumeros)
    }
}
```

Tareas

```
Monitor Tareas{
    int notaGeneral=25
    Procedure EsperarCorrecion(out int nota,in int
    ↪ idGrupo){
        if (llego[idGrupo]){
            push(terminaron,idGrupo)
            signal(hayEntregados)
        }else{
            llego[idGrupo]=true
        }
        wait(corregido[idGrupo])
        nota=notaGrupo[idGrupo]
    }
    Procedure CorregirGrupo(){
        if(empty(terminaron)){
            wait(hayEntregados)
        }
        idGrupo=pop(terminaron)
        notaGrupo[idGrupo]=notaGeneral
        notaGeneral--
        signal_all(corregido[idGrupo]
    }
}
```

Alumno

```
A=50
Process Alumno[id:0..A-1]{
    Entrada.Entrar()
    int numeroGrupo
    Grupos.ObtenerNumero(numeroGrupo)
    -- hacer tarea
    int nota
    Tareas.EsperarCorrecion(nota,numeroGrupo)
}
```

JTP

```
Process JTP{
    Entrada.Avisar()
    for i=1 to 50{
        Grupos.DarNumero(AsignarNroGrupo())
    }
    for i=1 to 25{
        Tareas.CorregirGrupo()
    }
}
```

(Ejercicio 7)

Consigna:
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).
Respuesta:

Monitor Cancha

```
Monitor Cancha[id:0..1]{
    Procedure Entrar(){
        cant++
        if(cant==10) signal(inicio)
        wait(espera)
    }

    Procedure Iniciar(){
        if(cant<10) wait(inicio)
    }

    Procedure Terminar(){
        signal_all(espera)
    }
}
```

Proceso Partido

```
Process Partido[id:0..1]{
    Cancha[id].Iniciar()
    delay(50)
    Cancha[id].Terminar()
}
```

Monitor Entrada

```
Monitor Entrada{
    Procedure Llegar(in int numeroEquipo, out int
    ↪ numeroCancha){
        cant[numeroEquipo]++
        if(cant[numEquipo]==5){
            cantEquipos++;
            cancha[numEquipo]=cantCanchasUsadas
            if(cantEquipos==2){
                cantCanchasUsadas++
            }
            signal_all(espera[numEquipo])
        }else{
            wait(espera[numEquipo])
        }
        numeroCancha=cancha[numEquipo]
    }
}
```

Proceso Jugador

```
Process Jugador[id:0..19]{
    Entrada.llegar(DarEquipo(),numeroCancha)
    Cancha[numeroCancha].Entrar()
}
```

(Ejercicio 8)

Consigna:
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.
Respuesta:

Monitor Carrera

```
Monitor Carrera{
    int llegaron = 0
    cond esperandoCorrer
    Process Llegue(){
        llegaron++
        if (llegaron == C){
            signal_all(espera)
        } else{
            wait(espera)
        }
    }
}
```

Monitor UsarMaquina

```
Monitor UsarMaquina{
    cond espera
    int esperando = 0
    bool libre = true
    Procedure Entrar(){
        if(not libre){
            esperando++
            wait(espera)
        }else{
            libre=false
        }
    }

    Procedure Salir(){
        if (esperando>0){
            esperando--
            signal(espera)
        }else{
            libre=true
        }
    }
}
```

Monitor Maquina

```
Monitor Maquina{
    int botellas = 20
    cond noHayBotellas, hayBotellas
    Procedure Reponer(){
        if (botellas!=0){
            wait(noHayBotellas)
        }
        signal(hayBotellas)
        botellas=20
    }
    Procedure DameBotella(){
        if (botellas == 0){
            signal(noHayBotellas)
            wait(hayBotellas)
        }
        botellas--
    }
}
```

Proceso Corredor

```
Process Corredor[0..C-1]{
    Carrera.Llegue()
    -- correr
    UsarMaquina.Entrar()
    Maquina.DameBotella()
    UsarMaquina.Salir()
}
```

Proceso Repositor

```
Process Repositor{
    while(true){
        Maquina.Reponer()
    }
}
```