Segunda clase de apoyo de Concurrente

2014 Farfán Jorge | Tracy Christian

Temas a tratar

- ✓ Semáforos
- ✓ Monitores
- ✓ PMA
- ✓ PMS
- ✓ ADA

Semáforos

Un sistema de software esta compuesto por un proceso **Central** y un conjunto de 10 procesos **Periféricos** donde cada uno de ellos realiza una determinada operación especial(cuyo resultado es un valor entero).
El proceso Central debe esperar a que todos los procesos periféricos se hayan iniciado para poder comenzar. Una vez que el proceso Central **comenzó a trabajar**, cada vez que necesita realizar alguna de las 10 operaciones especiales avisa al correspondiente proceso Periférico para que realice el trabajo y espera a que le devuelva el resultado.
Nota: existe una función *int TrabajaProcesoCentral()* que simula el trabajo del proceso central y devuelve un valor entero entre 1 y 10 que indica cual de las 10 operaciones especiales debe ser realizada en ese momento. **(PARCIAL)**

Semáforos

Identificación de semaforos…

```
sem s_pasar:=1;
sem
s_esperando[10]:=([10],0
)
sem
s_despertarCentral:=0;
sem s_resultado:=0;
int cantidad:=0;
int
resultados[10]:=([10],0);
```

Semáforos

```
process Periferico([p=1..10]){
    P(s_pasar);
      cantidad++;
      if(cantidad=10){
         V(s_despertarCentral);
      }
    V(s_pasar);

    while(true){
      P(s_esperando[p]);
      resultados[p]=operacion(p);
       V(s_resultado);
    }}
```

```
process Central(){
var
 int periferico;

    P(s_despertarCentral);
    while(true){
       periferico=TrabajaProcesoCentral();
       V(s_esperando[periferico]);
        P(s_resultado);
       //SE OBSERVA EL RESULTADO
-->resultados[periferico]
    }
}
```

Semáforos

Suponga que se tiene un curso con 50 alumnos. Cada alumno elije una de las 10 tareas para realizar entre todos. Una vez que todos los alumnos eligieron su tarea comienzan a realizarla. Cada vez que un alumno termina su tarea le avisa al profesor y si todos los alumnos que tenían la misma tarea terminaron el profesor les otorga un puntaje que representa el orden en que se terminó esa tarea.

Nota: Para elegir la tarea suponga que existe una función elegir que le asigna una tarea a un alumno (esta función asignará 10 tareas diferentes entre 50 alumnos, es decir, que 5 alumnos tendrán la tarea 1, otros 5 la tarea 2 y así sucesivamente para las 10 tareas). El tiempo en un alumno tarda en realizar la tarea es random. **(PRACTICA)**

Semáforos

Identificación de semaforos…

```
sem s_control:=1;
sem s_esperando =0;
sem s_atender:=1;
sem s_despertarProfe:=0;
sem s_nota:=1;
sem
s_esperandoNota[10]:=([10],0)
int notas[10]:=([10],0);
int grupos[10]:=([10],0);
Int cantidad:=0;
Int t_actual:=0;
```

Semáforos

```
Process Alumno([a=1..50]){
Var
    T,x:int;

T:=elegir(a);
P(s_control);
  cantidad++
  if(cantidad<>50){
      V(s_control);
      P(s_esperando);
   }esle{
      for x=1 to 49 do
          V(s_esperando);
              V(s_control);
   }
P(s_atender);
  t_actual:=T;
  V(s_desperarProfe);
P(s_esperandoNota[T]);
P(s_verNota);
      nota:=notas[T]
V(s_verNota);
}
```

```
Process Profesor(){
Var
    T:int;
    Orden,x,y:int;

Orden:=1;
For x:=1 to 50 do{
    P(s_despertarProfe);
    grupos[t_actual]++;
    if(grupos[t_actual]==5){
      notas[t_actual]:=Orden;
      Orden++;
      For y=1 to 5 do
      V(s_esperandoNota[t_actual]);

    }
    V(s_atender);

}


}
```

Semáforos

Se trata de calcular una cantidad de productos fabricados entre ciertos empleados, en este caso vamos a tener x empleados que fabrican un producto cada uno.
Cuando todos juntos trabajando llegan a la cantidad solicitada(un cantidad de 100 productos), se retiran.
(PARCIAL)

Semáforos

```
sem s_paso:=1;

int cantidad:=0;
```

Identificación de semáforos…

Semáforos

```
Process Empleado([e:1…x]){


P(s_paso);
While(cantidad<>100){

      (cantidad ++;)
       V(s_paso);
 //Fabricando ( delay)
       P(s_paso);

}
 V(s_paso);
}
```

Semáforos

Tips parcial

- ✓Entender el enunciado, e identificar la sección critica
- ✓No dejar semáforos bloqueados
- ✓No usar un **while** cuando se podría usar un **for**(caso de usuarios fijos y que solo piden una sola vez una petición)
- ✓Caso típico de estados ,timer y administrador
- ✓Maximizar la concurrencia=**NO USAR SEMAFOROS DE MAS** (usar un semáforo para determinada acción)

Monitores

En una casa viven una **abuela** y sus **N nietos**. Además la abuela compró caramelos que quiere convidar entre sus nietos. Inicialmente la abuela **deposita** en una fuente X caramelos, luego cada nieto **intenta comer** caramelos de la siguiente manera: si la fuente tiene caramelos el nieto agarra uno de ellos, en el caso de que la fuente esté vacía **entonces se le avisa a la abuela** quien repone nuevamente X caramelos.
Luego se debe permitir que **el nieto que no pudo comer sea el primero en hacerlo**, es decir, el primer nieto que puede comer nuevamente es el primero que encontró la fuente vacía.
NOTA: siempre existen caramelos para reponer. Cada nieto tarda t minutos en comer un caramelo (t no es igual para cada nieto). Puede haber varios nietos comiendo al mismo tiempo. **(PRACTICA)**

Monitores

Identificar los process…

```
process Abuela()  {
var
 int cantidad;

  while(true){
     cantidad:=//la cantidad a  reponer de caramelos
      MAbuela.reponer (cantidad)
    }
}
```

```
process Nieto([n:=1...N]) {

  while(true){

MAbuela.sacarCaramelos();
     delay("Comiendo");
  }
}
```

Monitores

```
Monitor MAbuela
var

caramelos:integer:
=0;

vacio:boolean:=fals
e;
    esperando:cond;
    abuela:cond;
    ultimoNiño:cond;
```

```
procedure sacarCaramelos()  {
    while(vacio){
        wait(esperando);
    }
    if(caramelos>0){
        caramelos--
    }else{
        signal(abuela);
        vacio:=true;
         wait(ultimoNiño);
        caramelos--;
        vacio:=false;
        signalAll(esperando);
  }
}
```

```
procedure reponer(c: in
integer)  {
    caramelos:=c;
    signal(ultimoNiño);
     wait(Abuela);

-Solucion b
  if(caramelos<>0){
      wait(abuela);
  }
  caramelos:=c;
  signal(ultimoNiño);


}
```

Monitores

Suponga una comisión con 50 alumnos. Cuando los alumnos llegan forman una fila, una vez que están los 50 en la fila el jefe de trabajos prácticos les entrega el número de grupo (número aleatorio del 1 al 25) de tal manera que dos alumnos tendrán el mismo número de grupo (suponga que el jefe posee una función DarNumero() que devuelve en forma aleatoria un número del 1 al 25, el jefe de trabajos prácticos no guarda el número que le asigna a cada alumno).
Cuando un alumno ha recibido su número de grupo, busca al compañero que tenga el mismo número de grupo para comenzar a realizar la práctica. Cuando ambos alumnos se encuentran permanecen en una sala realizando la práctica. Al terminar de trabajar, el alumno le avisa al jefe de trabajos prácticos y espera a que su compañero también avise que finalizó.
El jefe de trabajos prácticos, cuando han llegado los dos alumnos de un grupo les devuelve a ambos el orden en que termino el GRUPO (el primer grupo en terminar tendrá como resultado 1, y el último 25). **(PRACTICA)**

Monitores

Identificar los process…

```
process Alumno([a:=1…
50])  {
var
 int orden;
Int G;

 Mjefe.entrar(G);

MGrupo[G].esperarCompa
ñero();

 //Trabajando en la sala;

 Mjefe.salir(G,orden);

//MI ORDEN ES (orden);

}
```

Monitores

Monitor MJefe

```
var
    alumnos:cond;
   cantidad:integer:=0;

grupos[25]:=([25],0);

ordenes[25]:=([25],0);
   res:integer:=0;

compañeros:=cond([2
5]);
```

```
procedure entrar(g:=out integer){

  cantidad++;
  if(cantidad<>50){
     wait(alumnos);
  }else{
     signalAll(alumnos);

  }
  g:=darNumero();

}
```

Monitores

```
procedure salir(g:in integer,orden: out integer){

 grupos[g]++;
 if(grupos[g]=2){
      res:=miOrden(g);
      ordenes[g]:=res;
      signal(compañeros[g]);
      orden:=res;
 }else{
      wait(compañeros[g]);
      orden:=ordenes[g];
  }

}
}
```

Monitores

Monitor
Mgrupo([m:=1…50/2])
var
    cantidad:integer:=0;
    amigos:cond;

```
procedure esperarCompañero()  {

  cantidad++;
  if(cantidad=2){
      cantidad:=0;
      signal(amigos);
  else{
       wait(amigos);
  }
  }
  }
}
```

Monitores

En una casa de pastas se realiza la venta de las mismas.Para comprar los clientes deben respetar el orden de llegada , ademas se pueden atender 5 personas a la vez**.(PARCIAL)**

Monitores

Identificar los process…

```
process Cliente([c:=1…N]) {

 MCasaDePastas.quieroComprar();

 delay(x);//comprando pastas

 MCasaDePastas.terminoCompra( );

}
```

Monitores

Monitor MJefe

```
var
 cantidad:integer:=0;

esperando:integer:=
0;
 esperando:cond;
```

```
procedure quieroComprar(){

   if(cantidad+1==6){
      esperando++;
      wait(esperando);
   }else{
      cantidad++
   }


}
```

```
procedure terminoCompra(){

 if(esperando>0){
    esperando--;
    signal(esperando);
 }else{
    cantidad--
 }

}


}
```

Monitores

En un Crucero por el Mediterráneo hay 200 personas que deben subir al barco por medio de 10 lanchas con 20 lugares cada una. Cada persona sube a la lancha que le corresponde. Cuando en una lancha han subido sus 20 personas durante 5 minutos navega hasta el barco. Recién cuando han llegado las 10 lanchas al barco se les permite a las 200 personas subir al barco.
Nota: suponga que cada persona llama a la función int NúmeroDeLancha () que le devuelve un valor entre 0 y 9 indicando la lancha a la que debe subir. Maximizar la
concurrencia. **(PARCIAL)**

Monitores

Tips parcial

✓Identificar quienes piden realizar acciones, y quien las administra.

✓Tratar de no demorar con un delay a un monitor, si el ejercicio no lo pide explícitamente.

✓**Caso especial** de dormir un monitor dentro de otro monitor, hay casos en los que se puede hacer.

✓Agrupar process para que trabajen juntos, lo que significa que no pueden irse hasta que todos terminen y la tarea este hecha.

AHI ESTA EL TRABAJO EN GRUPO

✓**Maximizar concurrencia**=usar monitores para diferentes acciones

PMA

Se tiene una tabla distribuida entre **10 procesos** (cada proceso tiene 10000 registros).

La tabla contiene el nombre y la dirección de cada persona .Se debe encontrar la dirección de "Juan Pérez"(seguro esta en la tabla, y solo una vez).

Ni bien se encuentra el dato todos los procesos deben dejar de buscar (PARCIAL)

PMA

```
chan terminarDebuscar();
process control([p:1...10])  {
Var
 x:int;
 nombre:string;
 dir :string;

      x=0;
      while(q_empty(terminarDebuscar) and (x<10000)){
       nombre:= dameNombreDeLaTabla()//METODO QUE ME DA UN NOMBRE
       dir:=dameDireccionDeLaTabla();
       if(nombre=="Juan Perez"){
          //LO ENCONTRE!!!
          send terminarDebuscar();
       }
        x++;
      }
}
```

PMA

Para una aplicación de venta de pasaje se tiene **3 servidores** replicados para mejorar la eficiencia en la atención .
Existen **N clientes** que hacen alguna de estas dos solicitudes:compra de un pasaje o devolución de un pasaje.Las solicitudes se deben atender dando prioridad a las solicitudes de compra.

Nota:suponga que cada cliente llama a la funcion TipoSolicitud() que le devuelve el tipo de solicitud a realizar**.Maximizar la concurrencia**. (PARCIAL)

PMA

```
//Declaramos los canales
Chan devolucion(int c);
Chan compra(int c);
Chan respuesta[c]:=([c], string mensaje);
Chan pedirSolicitud(int s);
Chan solicitud[3](int cli, string tipo)
```

```
Process Cliente([c:=1…N])  {
Var
    tipo:string;

Tipo=tipoDeSolicitud(c);
If(tipo="Devolucion"){
   send devolucion(c);
}
else{
   send compra(c);
}
Receive respuesta[c]
(mensaje);

//IMPRIMIR mensaje
}
```

PMA

```
Process Servidor([s:=1...3])  {
Var
     cli:int;
     tipo:string;

While (true){
    send pedirSolicitud(s);
    receive solicitud[S](cli,tipo);
    if(tipo=“Compra”){
        //Compra
    }else{
        //Devolucion
    }
    send respuesta[cli](“Se atendio”+tipo)
}
}
```

PMA

```
Process Intermedio()  {
Var
    cli,s:int;
    tipo:string;

While (true){
  if(not q_empty(compra) and (not q_empty(pedirSolicitud)))
    //Prioridad
     receive pedirSolicitud(s);
     receive compra(cli);
     send solicitud[s](cli,”Compra”)
  [](  (q_empty(compra) and not q_empty(devolucion)) and not
q_empty(pedirSolicitud))
     receive pedirSolicitud(s);
     receive devolucion(cli);
     send solicitud[s](cli,”Devolucion”)
}}
```

PMA

En una sala de baile deben entrar como mínimo dos bailarines. Existen cuatro tipos de bailarines los de danza (A), los de tango (B), los de salsa (C) y los de rock (D). En la sala siempre debe haber un bailarín A y uno D. Además la cantidad de bailarines A debe ser mayor que la cantidad de bailarines B y que la cantidad de bailarines C. Dentro de la sala los bailarines bailan 5 minutos y luego deben intentar retirarse de la sala. Modelice el problema utilizando PMA.

PMA

//Declaramos los canales

```
Chan entrarA(int a);
Chan entrarB(int b);
Chan entrarC(int c);
Chan entrarD(int d);

Chan pasarA[a]();
Chan pasarB[b]();
Chan pasarC[c]();
Chan pasarD[d]();
```

```
Chan salirA(int a);
Chan salirB(int b);
Chan salirC(int c);
Chan salirD(int d);

Chan saliendoA[a]();
Chan saliendoB[b]();
Chan saliendoC[c]();
Chan saliendoD[d]();
```

PMA

```
Process Danza([a:=1...N]) {

 send entrarA(a);
 receive pasarA[a]();
 delay(5´); //Bailando
 send salirA(a);
 receive saliendoA[a]
();
}
Process Tango([b:=1...N]) {

send entrarB(b);
 receive pasarB[b]();
 delay(5´); //Bailando
 send salirB(b);
 receive saliendoB[b]
();
```

```
Process Rock([d:=1...
N]) {

send entrarD(d);
 receive pasarD[d]();
 delay(5´); //Bailando
 send salirD(d);
 receive saliendoD[d]
();
}
Process
 Salsa([c:=1...N]) {

 send entrarC(c);
  receive pasarC[c]();
  delay(5´); //Bailando
  send salirC(c);
  receive saliendoC[c]
 ();
```

PMA

```
Process Sala() {
Var

cant_a,cant_b,cant_c,cant_c:integer;
    idA,idB,idC,idC:integer;
Begin
       cant_a,cant_b,cant_c,cant_c:=0;
       idA,idB,idC,idC:=0;
       receive entrarA(idA);
       cant_a++;
       receive entrarD(idD);
       cant_d++;
       send pasarA[idA]();
       send pasarD[idD]();
```

```
While(True){
//Entradas!!
  if(not q_empty(entrarA))
       receive entrarA(idA);
       cant_a++;
       send pasarA[idA]();
  [](not q empty(entrarD))
       receive entrarD(idD);
       cant_a++;
       send pasarD[idD]();
  []( (not q_empty(entrarB)  and  (cant_b+1
<cant_a)  )
       receive entrarB(idB);
       cant_b++;
       send pasarB[idB]();
  [](not q_empty(entrarC)  and  (cant_c+1
<cant_a)  )
       receive entrarC(idC);
       cant_c++;
       send pasarC[idC]();
```

PMA

```
//Salidas!!(continuacion del while!!)
  if(not q_empty(salirB))
        receive salirB(idB);
        cant_b--;
        send saliendoB[idB]();
  [](not q empty(salirC))
        receive salirC(idC);
        cant_c--;
        send saliendoC[idC]();
  []( (not q_empty(salirA)  and  (cant_a-1 >cant_b)  and (cant_a-1>cant_c) and (cant_a-1>=
1) and (cant_d>= 1) )
        receive salirA(idA);
        cant_a--;
        send saliendoA[idA]();
  [](not q_empty(entrarD)  and  (cant_d-1 >=1) and (cant_a>= 1)  )
        receive salirD(idD);
        cant_d--;
        send saliendoD[idD]();

}
```

PMA

Tips parcial

✓Usar las propias colas de mensajes , para poder usarlas en las guardas del intermedio.

✓Muchos usuarios y muchos admins, usar siempre un intermedio

✓Muchos usuarios y un solo admin, no usar intermedio

✓En el caso de que un process se tenga que quedar esperando, primero manda sus datos(id) y luego se queda esperando a ser atendido

✓Manejar bien las prioridades solicitadas

✓**Maximizar concurrencia**=QUE TODOS TRABAJEN A LA PAR(CASO TIPICO DE INTERMEDIO)

PMS

Se debe administrar el acceso para usar un determinado **servidor** donde no se permita a **mas de 10 usuarios** trabajando al mismo tiempo por cuestiones de rendimiento. Existen **N usuarios** que solicitan acceder al servidor, esperan hasta que se le da acceso para trabajar en el y luego salen del mismo.

Nota: suponga que existe una función TrabajarEnServidor que llaman los usuarios para representar que esta trabajando dentro del servidor**(PARCIAL)**

PMS

```
process usuario([u:1...10]) {

    while(true){
        Central! pedirPermiso();
        TRABAJAR EN EL SERVIDOR!!
        Central!meVoy();
    }
}
```

```
process Central() {
var
 int cantidad;

  cantidad:=10;
  while(true){
      if(cantidad>0,usuario[*]?pedirPermiso()
          cantidad--;
      [](true,usuario[*]?meVoy())
          cantidad++;
   }

}
```

PMS

En el sistema de administracion de alumnos trabajan 5 empleados que estan continuamente trabajando. Cuando les surge algun problema, envian un requerimiento al coordinador y continuan trabajando.El coordinador lee los requerimientos en el orden en que los recibio y resuelve el problema.
(PARCIAL)

PMS

```
process Empleados([e:1...5]) {
Var
    problema:string;

    while(true){
       problema="Se produjo un
problema"
       Intermedio!
dejarProblema(problema);
        //SEGUIR TRABAJANDO!!
    }
}
```

```
process Coordinador() {
var
 p:string;
   while(true){
      Intermedio!atender();
      Intermedio?resolver(p);
      //Resolviendo el problema(p);

    }

}
```

PMS

```
process Intermedio()  {
Var
    problemas:queue of string;
    p,pro:string;

    while(true){
       if(Empleados[*]?dejarProblema(p))
      q_push(problemas,p);
       [](not q_empty(problemas),Coordinador?atender())
      pro:=q_pop(problemas);
      Coordinador!resolver(pro);
    }
}
```

PMS

En un edificio existen 3 Porteros y P Personas. Las personas dejan reclamos en la oficina de los porteros que deben ser atendidos por cualquiera de ellos. Cada portero está continuamente trabajando, si hay algún reclamo pendiente lo atiende, y sino realiza un recorrido por los pisos durante 10 minutos.

Nota: la persona no debe esperar a que el reclamo sea atendido, ni se le debe avisar que se resolvió. **(PARCIALITO)**

PMS

Process Persona([per:1...N])
Var
problema:string;

while(true){
reclamo="Se presento un reclamo";
Intermedio! dejarReclamo(reclamo);
//Continuar!!
}
}

{**process Portero([por:1...3])** {
var
p,e:string;

while(true){
Intermedio!atender(por);
Intermedio?resolver(e,p);
if(e="Atender"){
//Atender reclamo(p);
}else{
//Delay(10´); revisando los pisos
}
}
}

PMS

```
process Intermedio()  {
Var
    reclamos:queue of string;
    r,re:string;

    while(true){
       if(Persona[*]?dejarReclamo(r))
      q_push(reclamos,r);
       [](not q_empty(reclamos),Portero[*]?atender(p))
      re:=q_pop(reclamos);
      Portero[p]!resolver(“Atender”,re);
       [](  q_empty(reclamos),Portero[*]?atender(p))
      Portero[p]!resolver(“No atender”,null);
    }
}
```

PMS

Prioridades

Para las prioridades en pms recordar :

If(**CONDICION BOOLEAN,MENSAJE**)

```
If(procesoA[*]?mensaje1(idA))
      q_push(colaA,idA)
[](procesoB[*]?mensaje2(idB))
      q_push(ColaB,idB)
[]( (not empty(colaB)),admin?pedir())
      Aca esta la prioridad!!!
[]( (empty(ColaB)and not q_empty(colaA)), admin?pedir())
```

Si hubiera muchos admins, se aplicaria lo mismo

PMS

Tips parcial

✓Usar bien el concepto de bloqueo , conjuntamente con las guardas.

✓Saber lo tipos de estado de las guardas al tener la condición boolean en true o false, y si llego o no llego el mensaje receptor que se va usar.

✓Muchos usuarios y muchos admins, usar un intermedio

✓Muchos usuarios y un admin, no usar un intermedio.

✓Usar bien las prioridades usando colas, conjuntamente con los mensajes solicitados

✓**Maximizar concurrencia**=USAR INTERMEDIOS Y AVECES NO

ADA

Se debe modelar el siguiente problema .En una clínica existe un **médico** de guardia que recibe continuamente peticiones de atención de las **E enfermeras** que trabajan en su piso y de las **P personas** que llegan a la clínica ser atendidos. Cuando una persona necesita que la atiendan espera **a lo sumo 5 minutos** a que el médico lo haga, si pasado ese tiempo no lo hace**, espera 10 minutos** y vuelve a requerir la atención del médico. Si no es atendida tres veces, se enoja y se retira de la clínica.
Cuando una enfermera requiere la atención del médico, si este no lo atiende inmediatamente **le hace una nota** y se la deja en el **consultorio** para que esta resuelva su pedido en el momento que pueda (el pedido puede ser que el médico le firme algún papel). Cuando la petición ha sido recibida por el médico o la nota ha sido dejada en el escritorio, continúa trabajando y haciendo más peticiones.
El médico atiende los pedidos **dándoles prioridad a los enfermos** que llegan para ser atendidos. Cuando atiende un pedido, recibe la solicitud y la procesa durante un cierto tiempo. Cuando está libre aprovecha a procesar las notas dejadas por las enfermeras.
MAXIMICE LA CONCURRENCIA. **(PRACTICA)**

ADA

```
Task Type Persona is
    entry
identificacion(idPersona:IN
integer);
end Persona;
personas:=array [1..N]of Persona;
Task Body Persona is
VAR
    cant,idPersona:=integer;
    atendido:=Boolean;
BEGIN


    atendido:=False;
    cant:=0;
    While(cant<3)and(!atendido){
        Select

Medico.solicitarAtencionPaciente( );
            atendido:=True;
        or delay 5
            delay 10
                cant++;
        end Select;
    }
END Persona;
```

ADA

```
Task Type Enfermera is
end Enfermera;
    enfermeras:=array [1...N] of Enfermera;
Task Body Enfermera is
VAR
BEGIN
    Loop
        Select
            Medico.pedirAtencionEnfermera("Pide su atencion con algo para
firmar");
        else
            Consultorio.DejarNota("Mi nueva nota");
        end Select;
    end Loop;
end Enfermera;
```

ADA

```
Task Consultorio is
   entry DejarNota(nota:IN String);
   entry HayNota(res:OUT Boolean);
   entry TomarNota(nota:OUT
String);
end Consultorio;
Task Body Consultorio is
VAR
  n:String;
  notas:queue of String;
  res:Boolean;
  nota:String;
BEGIN
   res:=False;
   Loop
      Select
         accept DejarNota(nota:IN String)do
            n:=nota;
            q_push(notas,n);
         end DejarNota;
      or
         accept HayNota(res:OUT Boolean)do
            res:=notas.q_empty();
         end HayNota;
      or
         accept TomarNota(nota:OUT String)do
            nota:=q_pop(notas);
         end TomarNota;
      end Select;
   end Loop;
end Consultorio;
```

ADA

```
Task Medico is
   entry solicitarAtencionPaciente();
   entry
pedirAtencionEnfermera(Pet:OUT
String);

end Medico;

Task  Body Medico is
VAR
   Resultado:Boolean;
   nota:String;
BEGIN
   Resultado:=False;
   Loop
      Select
         //Prioridad
          accept
solicitarAtencionPaciente()do
             delay(tiempo para atenderlo);
          end solicitarAtencionPaciente;
      or
         when(solicitarAtencionPaciente
´count=0)==>
             accept
pedirAtencionEnfermera(Pet:OUT String)do
                Pet:=Firmando peticion:
             end pedirAtencionEnfermera;
       else
         Select
            Consultorio.HayNota(Resultado);
             if(Resultado){
               Consultorio.TomarNota(nota);
                //atendiendo la nota sacada;
            }
         else
            null;
         end Select;
      end Select;
```

ADA

```
VAR
   x:int;

For x:1 to  N
      personas[x].identificacion(x);
endfor
```

ADA

En una casa viven una **abuela** y sus **N nietos**. Además la abuela compró caramelos que quiere convidar entre sus nietos.
Inicialmente la abuela **deposita** en una fuente X caramelos, luego cada nieto **intenta comer** caramelos de la siguiente manera: si la fuente tiene caramelos el nieto agarra uno de ellos, en el caso de que la fuente esté vacía **entonces se le avisa a la abuela** quien repone nuevamente X caramelos.
Luego se debe permitir que **el nieto que no pudo comer sea el primero en hacerlo**, es decir, el primer nieto que puede comer nuevamente es el primero que encontró la fuente vacía.
NOTA: siempre existen caramelos para reponer. Cada nieto tarda t minutos en comer un caramelo (t no es igual para cada nieto). Puede haber varios nietos comiendo al mismo tiempo.
(PRACTICA)

ADA

```
Task Type TNieto is

end Nieto;

nietos:=array [1..N]of
Nieto;

Task Body TNieto is
VAR
    caramelo:string;
BEGIN
   caramelo:=“”;
   loop{
      Tfuente.quieroCaramelo(caramelo);
      //Comiendo caramelo(caramelo)
   }

END Persona;
```

ADA

```
Task  TAbuela is

    entry reponerFuente(c out
integer);
end Abuela;
```

```
Task Body TAbuela is
VAR
```

```
BEGIN

  loop{
      accept reponerFuente(c:out
integer)do
            c:=//Repone caramelos!!
      end reponerFuente;
  }

END TAbuela;
```

ADA

```
Task  TFuente is
    entry quieroCaramelo(caramelo :out
string);
end Abuela;
```

```
Task Body TFuente is
VAR
    cantidad:integer;
```

```
BEGIN
    integer:=0;
  loop{
          accept quieroCaramelo(caramelo:out
string)do
        if(cantidad=0)then
          TAbuela.reponerFuente(cantidad);
       endIf;
       cantidad--;
       caramelo=“El caramelo”;
          end quieroCaramelo;
  }
END Persona;
```

ADA

Casos

Select con accept

```
Select
      when accept E1()do

      end E1;
Or
      accept E2()do

      end E2;
Or
      accept E3()do

      end E3;
End Select;
```

Solo se puede usar 1 de los dos(NO LOS DOS)

+ or delay

//aca adentro puedo agregar Select(es otro cuerpo separado e independiente)

+ else

//lo mismo que el or delay

ADA

Casos

Select con call

Select
otraTarea.E1();

+ or delay
//aca adentro puedo agregar Select(es otro cuerpo separado e independiente)

+ else
//lo mismo que el or delay

End Select;

Solo se puede usar 1 llamado
Solo se permite ->Un Else o un Or delay

ADA

Tips parcial

- ✓ Muchas tareas de distinto tipo, que quieran usar una tarea en especial solo la podran usar cuando una tarea la deje libre.
- ✓ Siempre hacer el cuerpo del accept, para tomar variables y poder usarlas en la tarea una vez finalizada el accept
- ✓ Los llamados entre tareas pueden traer deadlock
- ✓ Caso especial de tarea Timer
- ✓ Usar bien *rendevouz*
- ✓ **Maximizar concurrencia**=(rendevouz+evitar deadlock+identificadores para las tareas+bloquear a los demas, para usar la seccion critica con el accept)
- ✓ Para el caso de prioridades se usan los when y muchos or (**POR QUE PUEDE HABER MUCHAS TAREAS**) ,pero si solo fueran dos tareas, un ELSE funciona tambien

FIN
CLASE DE APOYO DE
CONCURRENTE 2014

Farfán Jorge | Tracy Christian