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.

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

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.

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:=false;
   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

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

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**.

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

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

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

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.

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
   accept identificacion(id:IN integer)do
      idPersona:=id;
   end identificacion;

   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;
   end Loop;
end Medico;
```

ADA

```
VAR
   x:int;

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

ADA

Casos

Select con accept

Select
when *accept* E1()do
end E1;
Or
*accep*t 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