Notas para Parcial Memoria Distribuida

Notas de PMS

Sintaxis

> No se declaran los canales, ya que por cada par de procesos hay un canal Implícito.

```
// PARA ENVIO DE MENSAJES

ProcessDestino!(parametro)
ProcessDestino!PuertoOpcional(parametro)

ProcessOrigen[num]!PuertoOpcional(id,parametro)
//En este caso no podemos usar el comodin para recibir, debemos enviarlo a un
numero especifico. (Arreglo de procesos para identificarlos)



// PARA LA RECEPCIÓN DE MENSAJES

ProcessOrigen?(parametro)
ProcessOrigen?PuertoOpcional(parametro)

ProcessOrigen[num]?PuertoOpcional(id,parametro)
ProcessOrigen[*]?PuertoOpcional(id,parametro)
// En la recepción de mensajes puede usarse un comodin para recibir de
cualquier proceso con el cual tenga comunicación.
```

> IMPORTANTE :
> El proceso que envía el mensaje se queda esperando a que el proceso receptor reciba el mensaje.

GUARDAS :

```
// En las guardas solo pueden usarse  sentencias de recepción (No de envio)
cola buffer;
DO
    Testeo?reporte(Resultado) -> buffer.push(Resultado);
```

```
    [] not Empty(buffer); Manteni?pedido() -> Manteni!sig(buffer.pop());
OD
```

> IMPORTANTE :
> Para cortar el DO debemos tener todas las guardas en false.

USO DEL ADMIN :

> Hacemos el uso del admin cuando queremos que no se queden esperando a que otro proceso reciba los datos y pueda o no seguir trabajando. El admin recibe los datos de un proceso guardándolos en una cola y los manda cuando el otro proceso los necesita.

EJERCICIO 2

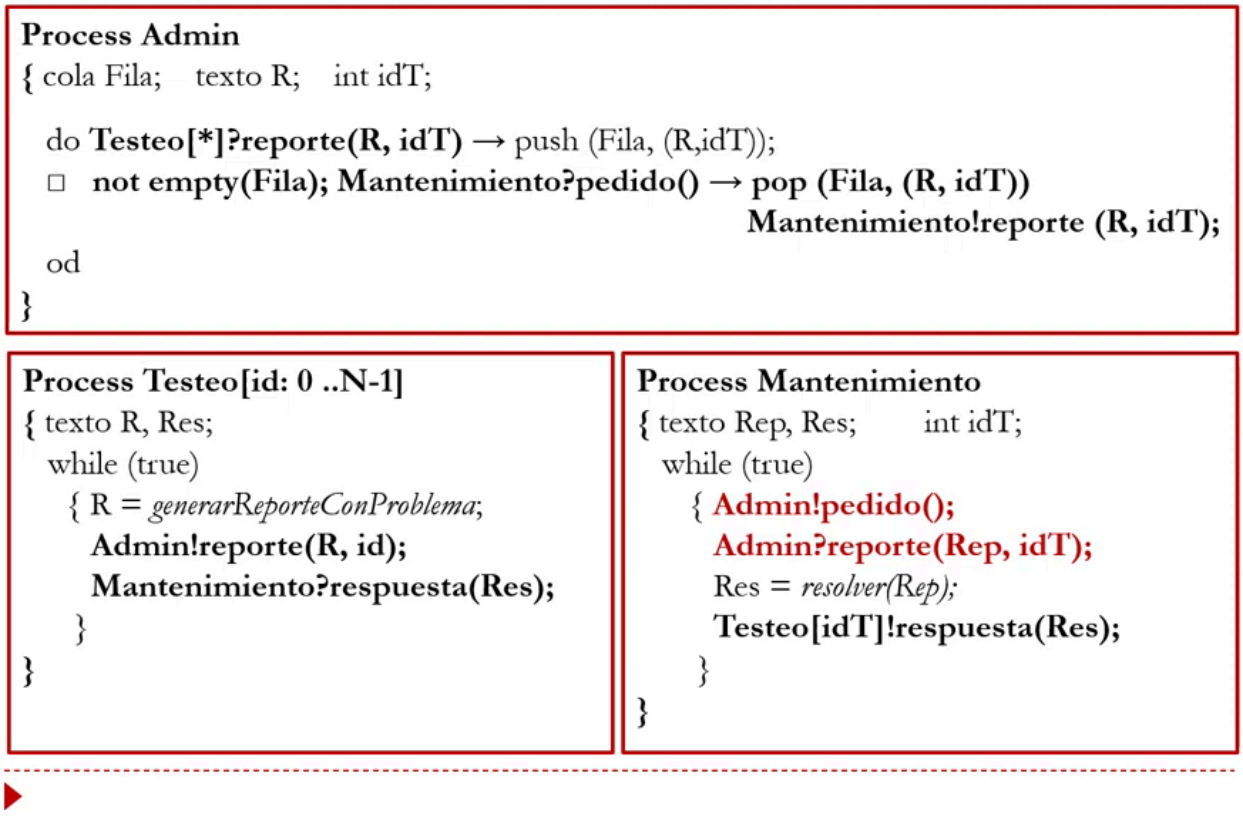
La segunda guarda debe tener una condición booleana, ya que no se puede procesar el pedido de *Mantenimiento* si no hay reportes pendientes. Por lo tanto debo demorar la recepción de ese mensaje hasta estar en condiciones de entregarle un reporte.

```
Process Admin
{ cola Buffer;
  texto R;
  do Testeo?reporte(R) → push (Buffer, R);
  □ not empty(Buffer); Mantenimiento?pedido() →
                Mantenimiento!reporte (pop (Buffer));
  od
}
```

```
Process Testeo
{ texto R, Res;
  while (true)
    { R = generarReporteConProblema;
      Admin!reporte(R);
    }
}

Process Mantenimiento
{ texto Rep, Res;
  while (true)
    { Admin!pedido();
      Admin?reporte(Rep);
      Res = resolver(Rep);
    }
}
```

EJERCICIO 3



Notas de PMA

> En PMA se deben declarar los canales. En esta forma de comunicación los canales deben declararse.

> Dentro de los procesos de ser necesario podemos manejar estructuras de datos.

> Los cortes de acá son con texto "corte" en los canales ya que no hay un do ..

Sintaxis

```
chan nombreCanal(tipoDato)

process Cliente[id:1..N]{
    send atencion(id)
}

process Empleado{
int idA;
    while (true){
        recive atencion(idA)
        //Atiende al cliente
```

```
    }
}
```

```
// Acá se puede usar el if no deterministico...
// Dentro de un for o while

if (condicion){

}
[] (condicion2){

}
```

```
// Para verificar cosas o tener prioridades debemos fijarnos las colas vacias
y los canales vacios.
// Ya que si primero tenemos a gente con prioridad y en otra cola tenemos
gente sin prioridad, se deja pasar a alguien sin prioridad si no esta vacia la
cola y el no hay nadie esperando lo de prioridad.
```

> Los procesos pueden hacer cosas mientras no realizan alguna tarea :

```
chan atencion(int)
chan pedido(int)
chan siguiente[3](int)

process Cliente[id:1..N]{
    send atencion(id)
}

process Empleado[Id:1..2{
int idA;
    while (true){
        send pedido(id) //El empleado pide el sig al coordinador y le aviso
quien soy
        receive siguiente[id](idA) //Espero el sig (que me avisen) en MI canal
con MI ID
        if (idA == "vacio"){
            Delay(900)
        }else{
            //Atiende al cliente
        }
    }
}
```

```
process Administrador{
int idA,id;
    receive pedido(idA) //Se  demora en el canal esperando solicitud del
empleado
    if (Empty(atencion)){  //Si el canal del cliente esta vacio
        id = "vacio"       //Setea el id con vacio
    }else{
        receive atencion(id) //Recibo el id y lo seteo en la variable id
    }
    send siguiente[idA](id)  //Envio al empleado quiere laburar la
informaación cual sea el caso en el que estamos
}
```

> El admin se usa para poder comunicar más de un proceso por el cual se usa el isEmpty() para ver quien fue el que mando el mensaje:

```
chan espera(int)
chan respuesta(txt)

chan aviso()
chan pedidoCliente(int)
chan avisoDeCaja(int)

int vectorContador([5] 0)
chan numeroCaja[P](int)

chan esperarAtencion[5](int)
chan esperarComprobante[P](txt)

process cliente[id: 1..P]{
int cola; txt comprobante;
    send aviso() //Se hace un signal para poder avisarle al admin
    send viso(id) //Se le pide una caja al admin
    receive numeroCaja[id](cola) //Espera el numero de cola

    //Se comunica directo con la caja a partir de ahora
    send esperarAtencion[cola](id); //Le manda a la caja y espera
    receive esperarComprobante[id](comprobante);
    //Se retira
}

process caja[id:1..5]{
int idProx; txt comprobante;
    receive esperarAtencion[id](idProx) //Espera el proximo cliente
```

```
    comprobante = GenerarComprobante(id);
    send esperarComprobante[idProx](comprobante)

    //Ahora debemos comunicarnos con el administrador
    send aviso() //Se hace un signal para poder avisarle al admin
    send avisoCaja(id); //Le mandamos nuestro id para que nos reste uno en la
cant de personas esperando en caja
}

process administrador{
int idAux, idC, colaMin
    receive aviso() //Se depierta
    //Ahora debe fijarse quien es el que me desperto
    if ( isEmpty(pedidoCliente) ){ //Si el cliente no mando nada es de la caja
        receive avisoDeCaja(idAux);
        vectorContador[idAux] --; //Decrementa la cantidad de clientes en la
cola de la caja
    }else{ //Es un pedido de un cliente para una caja
        receive aviso(idC);
        colaMin = Min(vectorContador); //El numero de cola con menor cant de
gente
        vectorContador[colaMin] ++; //Sumo uno para el nuevo cliente
        send numeroCaja[P](colaMin); //Le envia al cliente el numero de caja
donde ir
    }

}
```