PMA

Hacer otra cosa si no hay mensajes

```
process Buffer{
    while(true){
        receive listo(idManager);
        if (!empty(peticiones)){
            receive peticiones(idWorker,msg);
        }else{
            msg="";
        }
        send trabajo[idManager](idWorker,msg);
    }
}

process Manager[id:0..M-1]:
    while(true):
        send listo(id)
        receive trabajo(idWorker,msg)
        if(msg=""):
            delay(5) # hace otra cosa
        else:
            res=resolver(msg)
            send respuesta[idWorker](res)

process Cliente[id:0..C-1]:
    send peticiones(id,msg)
    receive respuesta[id](res)
```

Multiples procedures en un "monitor"

deterministico con prioridad

```
process Manager{
    while(true){
        receive sync
        if (!empty(operacion1)){ // prioridad
            receive operacion1(params)
            // hacer algo
        }else{ // seria de la operacion 2
            receive operacion2(params)
            // hacer algo
        }
    }
}
// puede ser de otra forma el envio
// lo importante son los syncs desp de operacion
proccess Worker[id:0..W-1]{
    send operacion1
    send sync
    send operacion2
```

```
    send sync
}

// utilizando manejo de recursos seria este en Buffer
while (true){
    receive sync
    if (!empty(liberar) or empty(colaLibres)){
        receive liberar(idWorker,idRecurso)
        colaLibres.push(idRecurso)
    }else {
        receive adquirir(idWorker)
        recursoLibre=pop(colaLibres)
        send respuesta[idWorker](recursoLibre)
    }
}
```

deterministico sin prioridad

```
process Manager{
    while(true){
        receive peticion(idWorker,tipoOp,datoOp)
        if (tipoOp=="operacion1"){
            receive operacion1()
            // hacer algo con datoOp
        }else if{ // seria de la operacion 2
            receive operacion2()
            // hacer algo con datoOp
        }
    }
}
// puede ser de otra forma el envio
// lo importante son los syncs desp de operacion
proccess Worker[id:0..W-1]{
    send peticion(id,"operacion1",5);
    send peticion(id,"operacion2",5);
}
```

no deterministico

```
process Manager{
    while(true){
        if not empty(operacion1) =>
            receive operacion1(params)
        [] not empty(operacion2) =>
            receive operacion2(params)
        fi
    }
}
proccess Worker[id:0..W-1]{
    send operacion1
    send operacion2
}
```

PMS

Buffer orden de llegada

```
process Buffer{
    do Worker[*]?peticion(msg)
        => push(peticiones,msg)
    [] not empty(peticiones);Analizador[*]?(id)
        => Analizador[id]!(pop(peticiones))
}
```

Utilizar un recurso con exclusion mutua

sin orden de llegada

```
process Worker[id:0..M-1]{
    Manager!pedir(id);
    Manager?usar();
    -- usar
    Manager!liberar();
}

process Manager{
    Worker[*]?pedir(id);
    Worker[id]!usar();
    Worker[id]?liberar();
}
```

con orden de llegada

```
process Worker[id:0..M-1]{
    Manager!pedir(id);
    Manager?usar();
    -- usar
    Manager!liberar();
}

process Manager{
    do libre; Worker[*]?pedir(id)
        -> libre=false; Worker[id]!usar()
    [] not libre; Worker[*]?pedir(id)
        -> push(pedidos,id)
    [] empty(pedidos); Worker[*]?liberar()
        -> libre=true;
    [] not empty(pedidos); Worker[*]?liberar()
        -> Worker[pop(pedidos)]!usar()
}
```

Barreras

Barrera tipo ring

```
if id!=0{
    Worker[N-id]?()
    Worker[id+1 mod N]!()
}else{
    Worker[id+1 mod N]!()
    Worker[N-id]?()
}
```

Barrera con fors

```
if id==0{
    for (int i=1;i<n;i++){
        Worker[i]?();
    }
    for (int i=1;i<n;i++){
        Worker[i]!();
    }
}else{
    Worker[id]?();
}
```

Barrera con dos procesos

```
process Coordinador[id:0..C-1]{
    // si hay mas de un coordinador
    if id==0{ // si hay uno no es necesario el if
        for (int i=1;i<n;i++){
            Worker[*]?permiso();
        }
        for (int i=1;i<n;i++){
            Worker[i]!entrar();
        }
    }
}
process Worker[]{
    Coordinador[0]!permiso() // si hay mas de un coord
    Coordinador[0]?entrar() // el indice
}ecialista is
```

Ada

Limitar por cantidades

```
```

```
cantidadTotal:Integer=N;
loop
    select // puede ser que sea implicita y no sea necesario
        accept salir(cantidad:IN Integer) do // el parametro
            cantidadTotal:=cantidadTotal+cantidad;
        end salir;
    or when (cantidadTotal>=X) =>
        accept entrarConCantX();
        cantidadTotal:=cantidadTotal-X;
    end select;
end loop;
```

Prioridades por proceso

```
loop
    select
        accept conPrioridad();
    or when (conPrioridad'count=0) =>
        accept sinPrioridad();
    end select;
end loop;
```

Esperar N minutos o irse

```
select
    Manager.tarea(); // puede ser un accept
or delay N*60  // no puede haber un else antes o despues
    null;
end select;
```

Si inmediatamente no responde irse

```
select
    Manager.tarea(); // puede ser un accept
or
    Manager.tarea2(); // puede ser un accept
else // no puede haber un or delay antes o despues
    null;
end select;
```

Hacer una accion por X segundos

```
task Timer body is
   accept empezarTimer();
   delay X
   Manager.terminoTimer();
end Timer;
```

```
task Manager body is
    Timer.empezarTimer();
    while seguir loop
        select
            accept terminoTimer();
            seguir:=false;
        or when(terminoTimer'count=0) =>
            // realizar otra accion: ej enviar/aceptar msg
        end select;
    end loop;
end Manager;
```

Barrera

```
// N workers
accept ident...

Manager.llegue()
Manager.termino()

// manager
for i in 1..N loop
    accept llegue();
end loop;
// hacer algo [OPCIONAL]
for i in 0..N-1 loop
    accept termino()
end loop;
```

N workers y M managers

```
// es necesario un buffer que coordine
// [BUFFER]
accept mensajeServ(idWorkerOut:OUT Integer; peticionOut:OUT text) do
    accept mensajeWork(idWorkerIn:IN Integer; peticionIn:IN text) do
        idWorkerOut:=idWorkerIn;
        peticionOut:=peticionIn;
    end mensajeWork;
end mensajeServ;


// [WORKER]
Buffer.mensajeWork(id,peticion);
accept respuesta(res:IN text);

// [MANAGER]
Buffer.mensajeServ(idWorker,peticion);
res=HacerAlgo(peticion);
workers[idWorker].respuesta(res);
```