Practica 4

PMA

1. Suponga que N clientes llegan a la cola de un banco y que serán atendidos por sus
   empleados. Analice el problema y defina qué procesos, recursos y canales/comunicaciones
   serán necesarios/convenientes para resolverlo. Luego, resuelva considerando las siguientes
   situaciones:
   a. Existe un único empleado, el cual atiende por orden de llegada.
   b. Ídem a) pero considerando que hay 2 empleados para atender, ¿qué debe modificarse en la solución anterior?
   c. Ídem b) pero considerando que, si no hay clientes para atender, los empleados
   realizan tareas administrativas durante 15 minutos. ¿

a)

```
chan mesa(int)

Process cliente[id:0..N-1]{
    send mesa(id)
}

process Empleado{
    int idC
    while(True){
        receive mesa(idC)
        //atender(idc)
    }
}
```

b)

```
chan mesa(int)

Process cliente[id:0..N-1]{
    send mesa(id)
}

process Empleado[id:0..1]{
    int idC
    while(True){
        receive mesa(idC)
        //atender(idc)
    }
}
```

c)

```
chan antencion(int) //canal donde los clientes piden atencion
chan pedido(int) //canal donde los empleados solicitan recibir
chan siguiente[2](int) //canal donde el admin avisa a los emplea

Process cliente[id:0..N-1]{
    send pedido(id)
}

process Admin{
    int idE, idC

    while(true){
        receive pedido(idE)
        if (empty(atencion))
            idC = -1
        else
            receive atencion(idC)
        send siguiente[idE](idC)
    }
```

```
}

process empleado[id:0..1]{
    int idC
    while(True){
        send pedido(id)
        receive siguiente[id](idC)
        if (idC == -1)
            sleep(15)
        else
            //atender(idC)
    }
}
```

2- Se desea modelar el funcionamiento de un banco en el cual existen 5 cajas para realizar
pagos. Existen P clientes que desean hacer un pago. Para esto, cada una selecciona la caja
donde hay menos personas esperando; una vez seleccionada, espera a ser atendido. En cada
caja, los clientes son atendidos por orden de llegada por los cajeros. Luego del pago, se les
entrega un comprobante. Nota: maximizar la concurrencia

```
chan comprobante[P](txt)
chan pedido[P](int)
chan llegada(int)
chan salida(int)
chan siguiente[5](txt,int)
chan huboAccion()


process cliente[id:0..P-1]{
    int caja
    text comprobante
    send llegada(id)
```

```
    send huboAccion()
    receive pedido[id](caja)
    send siguiente[caja](pago,id)
    receive comprobante[id](comprobante)


}


process caja[id:0..4]{
    while(true){
        text pago
        int id
        receive siguiente[id](pago,id)
        pago = //hacer comprobante
        send comprobante[id](pago)
        send huboAccion()
        send salida(id)

}

process admin{
    int cantEnCajas[5] = ([5] 0)
    int idClient, idCaja, cajaMin
    while(true){
        receive huboAccion()
        if(!empty(llegada)) { // si llego algun cliente
            receive llegada(idClient)
            cajaMin = min(cantEnCajas) //retorna la caja con me
            cantEnCajas[cajaMin]++
            send pedido[idClient](cajaMin)
        }
        else
            if(!empty(salida)){//si alguno ya termino su atenci
                receive salida(idCaja)
                cantEnCajas[idCaja]--
```

```
        }
    }

}
```

3- Se debe modelar el funcionamiento de una casa de comida rápida, en la cual trabajan 2
cocineros y 3 vendedores, y que debe atender a C clientes. El modelado debe considerar
que:

- Cada cliente realiza un pedido y luego espera a que se lo entreguen.
- Los pedidos que hacen los clientes son tomados por cualquiera de los vendedores y se
  lo pasan a los cocineros para que realicen el plato. Cuando no hay pedidos para atender,
  los vendedores aprovechan para reponer un pack de bebidas de la heladera (tardan entre
  1 y 3 minutos para hacer esto).
- Repetidamente cada cocinero toma un pedido pendiente dejado por los vendedores, lo
  cocina y se lo entrega directamente al cliente correspondiente.
  Nota: maximizar la concurrencia.

```
chan plato[C](text)
chan siguiente[3](int,text)
chan pedido(int,text)
chan espera(int)
chan llegada(int,text)

process cliente[id:0..C-1]{
    text pedido
    send pedido(id,pedido)
    receive plato[id](pedido)
}
```

```
process admin{
    int idC, idV
    text pedido
    while(true){
        receive espera(idV)
        if (empty(pedido))
            pedido = 'basura'
            idC = -1
        else
            receive pedido (idC, pedido)
        send siguiente[idV](idC,pedido)
    }
}

process vendedor[id:0..2]{
    int idC
    text pedido
    while(true){
        send espera(id)
        receive siguiente[id](idC,pedido)
        if (idC <> -1)
            send llegada(idC,pedido)
        else
            delay(180)
    }
}

process cocinero [id:0..1]{
    int idC
    text plato,pedido
    while (true){
        receive llegada(idC,pedido)
        plato = //hacerPlato(pedido)
```

```
        send plato[idC](plato)
    }
```

4- Simular la atención en un locutorio con 10 cabinas telefónicas, el cual tiene un empleado
que se encarga de atender a N clientes. Al llegar, cada cliente espera hasta que el empleado
le indique a qué cabina ir, la usa y luego se dirige al empleado para pagarle. El empleado
atiende a los clientes en el orden en que hacen los pedidos. A cada cliente se le entrega un
ticket factura por la operación.
a) Implemente una solución para el problema descrito.
b) Modifique la solución implementada para que el empleado dé prioridad a los que
terminaron de usar la cabina sobre los que están esperando para usarla.
Nota: maximizar la concurrencia; suponga que hay una función Cobrar() llamada por el
empleado que simula que el empleado le cobra al cliente.

a)

```
chan pedirCabina(int);
chan darCabina[N](int);
chan realizarPago(int, int, txt);
chan factura[N](text);
chan hayAviso(boolean);

process Cliente [id: 0.. N-1] {
    int cabina;
    text ticket;
    send pedirCabina (id);
    send hayAviso();
    receive darCabina[id](cabina);
    //usa la cabina enviada por el canal
    send realizarPago (id, cabina);
```

```
    send hayAviso();
    receive factura[id](factura;
}

process Empleado(){
    int idC = -1;
    int nroCabina;
    cola cabinaLibre;
    text pago;
    while (true){
        receive hayAviso();
        if (! empty pedirCabina and !cabinaLibre.vacia()){
            receive pedirCabina(idC);
            send darCabina[idC](cabinaLibre.pop())
        }
        else
            if (cabinaLibre.vacia() and (!empty pedirCabina)){
                        receive realizarPago(nroCabina, idC, pag
                        send factura[idC](cobrar);
                        receive pedirCabina(idC);
                        send darCabina[idC](nroCabina);

            }
            else {
                        receive realizarPago(nroCabina, idC, pag
                        send factura[idC](cobrar);
                        push.cabinaLibre(nroCabina);
                        }
  }
}
```

b)

```
chan llegoCliente(int)
chan terminoCliente(int,int)
chan comprobante[N](text)
chan recibirCabina[N](int)

process cliente[id:0..N-1]{
    int idC
    text comprobante
    send llegoCliente(id)
    receive recibirCabina[id](idC)
    //usarCabina(idC)
    send terminoCliente(idC,id)
    receive comprobante[id](comprobante)
}

process empleado{
    boolean cabinas[10] = ([10] false)
    int idCliente, cabina
    while(true){
        if (!(empty(terminoCliente))){
                receive terminoCliente(idCliente,cabina)
                cabinas[cabina] = false
                comprobante = cobrar(idCliente)
                send comprobante[idCliente](comprobante)
        }
        else
            if (!empty(llegoCliente)) and (hayCabinaLibre(cabina
                receive llegoCliente(idCliente)
                cabina = obtenerCabinaLibre(cabinas)
                cabinas[cabina] = true
                send recibirCabiba[idCliente](cabina)
            }
  }
```

5- Resolver la administración de 3 impresoras de una oficina. Las impresoras son usadas por N
administrativos, los cuales están continuamente trabajando y cada tanto envían documentos
a imprimir. Cada impresora, cuando está libre, toma un documento y lo imprime, de
acuerdo con el orden de llegada.
a) Implemente una solución para el problema descrito.
b) Modifique la solución implementada para que considere la presencia de un director de
oficina que también usa las impresas, el cual tiene prioridad sobre los administrativos.
c) Modifique la solución (a) considerando que cada administrativo imprime 10 trabajos y
que todos los procesos deben terminar su ejecución.
d) Modifique la solución (b) considerando que tanto el director como cada administrativo
imprimen 10 trabajos y que todos los procesos deben terminar su ejecución.
e) Si la solución al ítem d) implica realizar Busy Waiting, modifíquela para evitarlo.
Nota: ni los administrativos ni el director deben esperar a que se imprima el documento

a)

```
chan pedidos(doc)

process impresora[id:0..2]{
    text doc
    while (true){
        receive pedido(doc)
        //imprimir(doc)
    }
}

process administrativo[id:0..N-1]{
```

```
    text doc
    while (True){
        //trabajar()
        //genararDoc(doc)
        send pedidos(doc)
    }
}
```

b) Modifique la solución implementada para que considere la presencia de un director de
oficina que también usa las impresas, el cual tiene prioridad sobre los administrativos.

```
chan hayPedido(boolean) //canal para evitar busy waiting
chan impresoraLibre(int) // canal donde cada impresora avisa q e
chan pedidoAdmin(doc) //canal donde los admins envian sus docs
chan pedidoDirec(doc) //canal donde los direc envian sus docs
chan avisarImpresora[3](doc) //canal para indicarle a la impreso



process administrador[id:0..N-1]{
    text doc
    while (true){
        //trabajar()
        //generarDoc(doc)
        send pedidoAdmin(doc)
        send hayPedido(true)
    }

process director{
    text doc
    while (true){
        //trabajar()
        //generarDoc(doc)
        send pedidoDirec(doc)
```

```
        send hayPedido(true)
    }
}

process Coordinador{
        boolean ok
        text doc
        int id
        while (True){
            receive impresoraLibre(id)
            receive hayPedido(ok)
            if (!empty(pedidoDirec))
                receive pedidoDirec(doc)
            else
                receive pedidoAdmin(doc)
            send avisarImpresora[id](doc)
        }
}

process impresora[id:0..2]{
    text doc
    while (true){
        send impresoraLibre(id)
        receive avisarImpresora[id](doc)
        //imprimirDoc(doc)
    }
}
```

c) Modifique la solución (a) considerando que cada administrativo imprime 10 trabajos y
que todos los procesos deben terminar su ejecución.

```
chan pedidos[3](doc)
chan avisarImpresoras(int)
chan hayImpresora(boolean)
chan arrancaImpresora[3](boolean)
chan sigue[3](boolean)
chan hayDoc(boolean)

process impresora[id:0..2]{
    text doc
    boolean ok
    boolean seguir = true
    while (seguir){
        send avisarImpresoras(id)
        send hayImpresora(true)
        receive sigue[id](seguir)
        if (seguir)
            receive arrancaImpresora[id](ok)
            receive pedidos[id](doc)
            //imprimir(doc)
    }

}

process coordinador{
    int idImpresora
    text doc
    boolean ok
    for i := 1 to N*10{
        receive Haypedido(ok,idImpresora)
        send sigue[idImpresora](true)
        send arrancaImpresora[idImpresora](true)
    }
    for i:= 0 to 2{
        send sigue[i](false)
    }
```

```
}


process administrativo[id:0..N-1]{
    text doc
    int idImpresora
    boolean ok
    for i:= 1 to 10{
        //trabajar()
        //genararDoc(doc)
        receive hayImpresora(ok)
        receive avisarImpresoras(idImpresora)
        send pedidos[idImpresora](doc)
        send Haypedido(true,idImpresora)
    }

}



---------------------------

process impresora[id:0..2]{
    text doc
    while (!cantidadTotal()'empty()){
        receive pedido(doc,seguir)
        if (seguir)
            //imprimir(doc)
    }
    for
}

proces coordinador
    for 1..10*N:
```

```
        // Recibe pedidos de impresora y les da los documentos,
        // al pedido 10*N devuelve -1 o false
```

d) Modifique la solución (b) considerando que tanto el director como cada administrativo
imprimen 10 trabajos y que todos los procesos deben terminar su ejecución.

```
```

PMS

1. Suponga que existe un antivirus distribuido que se compone de R procesos robots
   Examinadores y 1 proceso Analizador. Los procesos Examinadores están buscando
   continuamente posibles sitios web infectados; cada vez que encuentran uno avisan la
   dirección y luego continúan buscando. El proceso Analizador se encarga de hacer todas las
   pruebas necesarias con cada uno de los sitios encontrados por los robots para determinar si
   están o no infectados.
   a) Analice el problema y defina qué procesos, recursos y comunicaciones serán
   necesarios/convenientes para resolverlo.
   b) Implemente una solución con PMS sin tener en cuenta el orden de los pedidos.
   c) Modifique el inciso (b) para que el Analizador resuelva los pedidos en el orden
   en que se hicieron

b)

```
process examinador[id:0..R-1]{
    web sitio
    while (true){
        sitio = //encontrarSitio()
        admin!avisar(sitio)
    }
}

process admin{
    cola buffer
    web sitio
    do examinador[*]?avisar(sitio) -> push (buffer,sitio);
    [] not empty(buffer); analizador?pedido() -> analizador!rec
    od


process analizador{
    web sitio
    while (true){
        admin!pedido()
        admin?recibir(sitio)
        //hacerPruebas(sitio)
```

c)

```
process examinador[id:0..R-1]{
    web sitio
    while (true){
        sitio = //encontrarSitio()
        admin!avisar(sitio)
    }
}

process admin{
    cola buffer
```

```
    web sitio
    do examinador[*]?avisar(sitio) -> push (buffer,sitio);
    [] not empty(buffer); analizador?pedido() -> analizador!rec
    od


process analizador{
    web sitio
    while (true){
        admin!pedido()
        admin?recibir(sitio)
        //hacerPruebas(sitio)
```

2- En un laboratorio de genética veterinaria hay 3 empleados. El primero de ellos continuamente prepara las muestras de ADN; cada vez que termina, se la envía al segundo
empleado y vuelve a su trabajo. El segundo empleado toma cada muestra de ADN preparada, arma el set de análisis que se deben realizar con ella y espera el resultado para
archivarlo. Por último, el tercer empleado se encarga de realizar el análisis y devolverle el
resultado al segundo empleado.

```
process empleado1{
    text muestra
    while (true){
        muestra = //prepararMuestra()
        admin!enviarMuestra(muestra)
    }
}

process empleado2{
```

```
    text muestra
    while (true){
        admin!avisar()
        admin?recibirMuestra(muestra)
        //armarSet(muestra)
        empleado3!enviarSet(muestra)
        empleado3?recibirResultado(muestra)
    }
}

process admin{
    cola buffer
    text muestra
    do empleado1?enviarMuestra(muestra) -> buffer.push(muestra)
    [] not empty(buffer); empleado2?avisar() -> empleado2!recibi
    od
}

process empleado3{
    text muestra
    while (true){
        empleado2?enviarSet(muestra)
        //aplicarAnalisis(muestra)
        empleado2!recibirResultado(muestra)
    }
}
```

3- En un examen final hay N alumnos y P profesores. Cada alumno resuelve su examen, lo
entrega y espera a que alguno de los profesores lo corrija y le indique la nota. Los profesores corrigen los exámenes respetando el orden en que los alumnos van entregando.
a) Considerando que P=1.
b) Considerando que P>1.
c) Ídem b) pero considerando que los alumnos no comienzan a realizar su examen

hasta
que todos hayan llegado al aula.
Nota: maximizar la concurrencia; no generar demora innecesaria; todos los procesos deben
terminar su ejecución

a)

```
process alumno[id:0..N-1]{
    text examen
    int nota
    examen = //realizarExamen()
    admin!entregarExamen(examen,id)
    profe?recibirNota(nota)
}

process admin{
    int id
    text examen
    cola buffer
    int cant = 0

    do cant<N; alumno[*]?entregarExamen(examen,id) -> buffer.pus


    [] not empty(buffer); profe?esperarExamen() -> profe!recibi

    od
}

process profe{
    text examen
    int idAlumno
    int nota
    for i:= 1 to N{
```

```
        admin!esperarExamen()
        admin?recibirExamen(examen,idAlumno)
        nota = //corregir(examen)
        alumno[idAlumno]!recibirNota(nota)
    }
}
```

b)

```
process alumno[id:0..N-1]{
    text examen
    int nota
    examen = //realizarExamen()
    admin!entregarExamen(examen,id)
    profe[*]?recibirNota(nota)
}

process admin{
    int idAlumno, idProfe
    int cant = 0
    text examen
    cola buffer


    do  cant<N; alumno[*]?entregarExamen(examen,id) ->  buffer.

    [] not empty(buffer) ; profe[*]?esperarExamen(idProfe) -> pr

    od

    for i := 1 to P{
```

```
            profe[*]?esperarExamen(idProfe)
            profe[idProfe]!recibirExamen('vacio',-1,false)
        }



        od
    }

    process profe[id:0..P-1]{
        text examen
        int idAlumno
        int nota
        bool sigue = true
        while (sigue){
            admin!esperarExamen(id)
            admin?recibirExamen(examen,idAlumno,sigue)
            if (sigue)
                nota = //corregir(examen)
                alumno[idAlumno]!recibirNota(nota)
        }
    }
```

c)

```
process alumno[id:0..N-1]{
    text examen
    int nota
    admin!llegue()
    admin?espero()
```

```
    examen = //realizarExamen()
    admin!entregarExamen(examen,id)
    profe[*]?recibirNota(nota)
}


process admin{
    int idAlumno, idProfe
    int cant = 0
    text examen
    cola buffer

    for i:= 1 to N{
        alumno[*]?llegue()
    }

    for i:= 0 to N-1{
        alumno[i]!espero()
    }


    do  cant<N; alumno[*]?entregarExamen(examen,id) ->  buffer.p


    [] not empty(buffer) ; profe[*]?esperarExamen(idProfe) -> pr

    od

    for i := 1 to P{
        profe[*]?esperarExamen(idProfe)
        profe[idProfe]!recibirExamen('vacio',-1,false)
    }



    od
```

```
}

process profe[id:0..P-1]{
    text examen
    int idAlumno
    int nota
    bool sigue = true
    while (sigue){
        admin!esperarExamen(id)
        admin?recibirExamen(examen,idAlumno,sigue)
        if (sigue)
            nota = //corregir(examen)
            alumno[idAlumno]!recibirNota(nota)
    }
}
```

4- En una exposición aeronáutica hay un simulador de vuelo (que debe ser usado con
exclusión mutua) y un empleado encargado de administrar su uso. Hay P personas que
esperan a que el empleado lo deje acceder al simulador, lo usa por un rato y se retira.
a) Implemente una solución donde el empleado sólo se ocupa de garantizar la exclusión
mutua.
b) Modifique la solución anterior para que el empleado considere el orden de llegada para
dar acceso al simulador.
Nota: cada persona usa sólo una vez el simulador.

a)

```
process persona[id:0..P-1]{
    empleado!pedir(id)
    empleado?recibirPase()
    //jugar()
    empleado!fin()
}

process empleado{
    int id
    for i := 1 to P{
        persona[*]?pedir(id)
        persona[id]!recibirPase()
        persona[id]?fin()
    }
}

---------------------------------------
process persona[id:0..P-1]{
    empleado!pedir()
    //jugar()
    empleado!fin()
}

process empleado{
    int id
    for i := 1 to P{
        persona[*]?pedir()
        persona[id]?fin()
    }
}
```

b)

```
process persona[id:0..P-1]{
    admin!pedir(id)
    empleado?recibirPase()
    //jugar()
    empleado!fin()
}

process admin{
    cola buffer
    int id
    do persona[*]?pedir(id) -> buffer.push(id)
    [] not empty(buffer); empleado?esperarPersona() -> empleado
    od
}

process empleado{
    int id
    for i := 1 to P{
        admin!esperarPersona()
        admin?recibirPersona(id)
        persona[id]!recibirPase()
        persona[id]?fin()
    }
}
```

5- En un estadio de fútbol hay una máquina expendedora de gaseosas que debe ser usada por
E Espectadores de acuerdo con el orden de llegada. Cuando el espectador accede a la
máquina en su turno usa la máquina y luego se retira para dejar al siguiente. Nota: cada
Espectador una sólo una vez la máquina

```
process espectador [id:0..E-1]{
    admin!pedir(id)
    maquina?recibirAcceso()
    //usarMaquina()
    maquina!fin()
}

process admin{
    cola buffer
    int id
    do espectador[*]?pedir(id) -> buffer.push(id)
    [] not empty(buffer); maquina?esperarEspectador() -> maquina
    od
}

process maquina{
    int id
    for i := 1 to E{
        admin!esperarEspectador()
        admin?recibirEspectador(id)
        espectador[id]!recibirPase()
        espectador[id]?fin()
    }
}

-----------------------------------

process espectador [id:0..E-1]{
    admin!pedir(id)
    admin?permiso()
    //usarMaquina()
    admin!fin()
}

process admin{
```

```
    cola buffer
    int id
    bool libre=true
    do espectador[*]?pedir(id) ->
        if(libre){
            libre=false
            espectador[id]!permiso()
            }
            else{
                buffer.push(id)
            }
    [] espectador[*]?fin() ->
        if(buffer not empty){
            espectador[buffer.pop()]!permiso()
        else {
            libre=true
        }
    od
}
```