Práctica 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.

```
chan atender(int)

process Cliente [id: 0.. N-1]{
    text pedido;
    send atender(pedido)
}

process Empleado(){
    text pedido;
    while (true) {
        receive atender(pedido)
        //atiende al cliente
    }
}
```

b. Ídem a) pero considerando que hay 2 empleados para atender, ¿qué debe
modificarse en la solución anterior?

Solo se cambia la cantidad de procesos Empleado.

```
chan atender(int)

process Cliente [id: 0.. N-1]{
    text pedido;
    send atender(pedido)
}

process Empleado(id: 0..1){
    text pedido;
    while (true){
        receive atender(pedido)
        //atiende al cliente
    }
}
```

c. Ídem b) pero considerando que, si no hay clientes para atender, los empleados
realizan tareas administrativas durante 15 minutos. ¿Se puede resolver sin usar
procesos adicionales? ¿Qué consecuencias implicaría?

Puede haber demora innecesaria ya que si dos o más empleados chequean por si el canal
está vacío (supongamos que hay un solo mensaje) con la función empty, a todos les
devolverá que no está vacío, por lo que más de un empleado intentará hacer el receive →
uno lo podrá hacer y el resto se bloqueará en el receive cuando en realidad deberían leer
por 10 minutos

Solución con un proceso adicional

```
chan Reportes(texto);
chan Pedido(int);
chan Siguiente[3](texto);
```

```
Process Coordinador
{ texto Rep;
    int idE;
    while (true){
        receive Pedido ( idE);
        if (empty (Reportes) ) Rep = “VACIO”;
        else
            receive Reportes ( Rep )
        send Siguiente[idE] ( Rep );
    };
}

Process Empleado[id: 0..2]
{
    texto Rep;
    while (true){
         send Pedido(id);
         receive Siguiente[id] ( Rep );
         if (Rep <> “VACIO”) resolver (Rep)
            else delay (600); //lee 10 minutos
        };
}

Process Persona[id: 0..N-1]
{
    texto R;
    while (true){
        R = generarReporteConProblema;
        send Reportes ( R );
        };
}
```

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 pedirCaja(int);
chan caja[1..P];
chan hacerPago[5](id)
chan comprobante[1..P](text);
chan atendido(int);


process Cliente [id: 0.. P-1]{
    int nroCaja;
    elem recibo;
    send pedirCaja(id);
    send hayPedido();
    receive caja[id](nroCaja);
    send hacerPago[nroCaja](pago, id);
    receive comprobante[id](recibo);
}

process Administrador(){
    int idP, idC
    vector contador [5];
    while (true){
        receive hayPedido(pedido);
        if !(empty(pedirCaja)){
            receive pedirCaja(idP);
            minimo= cajaMinimo(contador)
            send caja[idP](minimo)
            contador[minimo]+=1;
        }
        else{
            if !empty(atendido)
                receive atendido(idC);
                contador[idC]-=1;
        }
    }
```

```
}

process Cajero[id: 0..4]{
    int idP
    elem pago
    while (true){
        receive hacerPago(pago, idP);
        send atendido(id);
        send hayPedido();
        send comprobante[idP](hacerComprobante())
    }
}
```

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 pedido(text, int)
chan pedidoPlato(text, int)
chan plato[C](text)
chan vendedorProximo[2](txt, id);
chan vendedorSolicita(id);
```

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

process Vendedor[id: 0.. 2]{
    text pedido;
    int idC;
    while (true){
        send vendedorSolicita(id);
        receive vendedorProximo[id](pedido, idC);
        if (idC != -1 )
            send pedidoPlato(pedido, idC);
        else
            //reponer pack de bebidas
            delay(180)
        }
    }

process Coordinador(){
    text pedidoC;
    int idCliente, idVendedor
    while (true){
        receive  vendedorSolicita(idVendedor) //espera a que
        if (empty(pedido)) -> pedidoC= 'Vacio', idCliente = -
        else
            receive pedido(pedidoC, idCliente);
        send vendedorProximo[idVendedor](pedidoC, idCliente);
    }
}

process Cocinero [id: 0.. 1]{
    text pedido;
    int idC;
    elem platoArmado:...
```

```
    while (true){
        receive pedidoPlato(pedido, idC);
        //realiza pedido
        send plato[idC](platoArmado)
    }
}
```

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.

```
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 pedirCabin
                        receive realizarPago(nroCabina, id
                        send factura[idC](cobrar);
                        receive pedirCabina(idC);
                        send darCabina[idC](nroCabina);

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

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.

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

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 realizarPago) {
            receive realizarPago(nroCabina, idC, pago);
            send factura[idC](cobrar);
            push.cabinaLibre(nroCabina);
        }else{
            if (! empty pedirCabina and !cabinaLibre.vacia
                receive pedirCabina(idC);
                send darCabina[idC](cabinaLibre.pop())
        }
        else
            if (cabinaLibre.vacia() and (!empty pedirCabin
                        receive realizarPago(nroCabina, id
                        send factura[idC](cobrar);
                        receive pedirCabina(idC);
```

```
                        send darCabina[idC](nroCabina);

            }
  }
}
```

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.

```
chan documentosParaImprimir(text);

process Administrativo [id: 0.. N-1]{
    text doc;
    while (true ){
        trabajando..
        send documentosParaImprimir (doc);
    }
}

process Impresora [id: 0..2]{
    text doc;
    while (true){
        receive documentosParaImprimir (doc);
        imprimir(doc);n
    }
}
```

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 documentosParaImprimirAdministrativo(text);
chan documentoParaImprimirDirector(text);
chan hayPedido();

process Administrativo [id: 0.. N-1]{
    text doc;
    while (true ){
        trabajando..
        send documentosParaImprimirAdministrativo (doc);
        send hayPedido();
    }
}

process Impresora [id: 0..2]{
    text doc;
    while (true){
        receive hayPedido();
        if (!empty documentosParaImprimirDirector)
            receive documentosParaImprimirDirector (doc);
        else
            if (!empty documentosParaImprimirAdministrativ
                receive documentosParaImprimirAdministrati
        imprimir(doc);
    }
}

process Director(){
    text doc;
    while (true ){
        trabajando..
        send documentosParaImprimirDireector(doc);
        send hayPedido();
    }
}
```

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

```
chan documentosParaImprimir(text);
chan darDocumento[3](text);
chan pedir();

process Administrativo [id: 0.. N-1]{
    text doc;
    for i:= 1 to 10 do {
        trabajando..
        send documentosParaImprimir (doc);
    }
}

process Impresora [id: 0..2]{
    text doc="";
    send pedir(id);
    receive darDocumento[id](doc);
    imprimir(doc);
    while (doc != "FIN"){
        imprimir(doc);
        send pedir(id);
        receive darDocumento[id](doc);
    }
}

process Administrador (){
        text doc;
        int idI
        for i:= 1 to (10*N){
            receive documentosParaImprimir(doc);
            receive pedir(idI);
            send darDocumento[idI](doc);
        }
```

```
        for i:= 1 to 3 {
                receive pedir(idI)
                send darDocumento[idI]("FIN")
        }
}
```

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.

```
chan documentosParaImprimirAdministrativo(text);
chan documentoParaImprimirDirector(text);
chan hayPedido();
chan pedirDoc(int);
chan darDoc[3](text);

process Administrativo [id: 0.. N-1]{
    text doc;
    for i:= 1 to 10 {
        trabajando..
        send documentosParaImprimirAdministrativo (doc);
        send hayPedido();
    }
}

process Administrador(){
    text doc;
    int idI
    for i:= 1 to (10*N)*2{
        receive hayPedido();
        if (!empty documentosParaImprimirDirector)
            receive documentosParaImprimirDirector (doc);
        else
            if (!empty documentosParaImprimirAdministrativ
```

```
                receive documentosParaImprimirAdministrati
        receive pedirDoc(idI);
        send darDoc[idI](doc);
    }

    for i:= 1 to 3 {
        receive pedirDoc(idI)
        send darDoc[idI]("FIN")
    }
}

process Impresora [id: 0..2]{
    text doc="";
    send pedir(id);
    receive darDocumento[id](doc);
    imprimir(doc);
    while (doc != "FIN"){
        imprimir(doc);
        send pedirDoc(id);
        receive darDoc[id](doc);
    }
}

process Director(){
    text doc;
    for i:= 1 to 10{
        trabajando..
        send documentosParaImprimirDireector(doc);
        send hayPedido();
    }
}
```

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.

No hace

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.

```
process Examinador [id: 0.. R-1]{
    text direccion;
    while (true){
        direccion = detectarError();
        send Administrador!error(direccion);


    }
}

process Administrador(){
    cola direcciones;
    do Examinador[*]?error(direccion) ->  direcciones.
    [] !direcciones.vacia(); Analizador?analizar() -> A
```

```
        od
    }


    process Analizador(){
        text direccion;
        while (true){
            Administrador!analizar();
            Administrador?enviarDireccion(direccion);
            //analiza el error de la direccion recibida
        }
    }
```

c) Modifique el inciso (b) para que el Analizador resuelva los pedidos en el orden
en que se hicieron

```
process Examinador [id: 0.. R-1]{
    text direccion;
    while (true){
        direccion = detectarError();
        send Administrador!error(direccion);


    }
}

process Administrador(){
    cola direcciones;
    do Examinador[*]?error(direccion) ->  direcciones.pus
    [] !direcciones.vacia(); Analizador?analizar() -> Ana
    od
}
```

```
    process Analizador(){
        text direccion;
        while (true){
            Administrador!analizar();
            Administrador?enviarDireccion(direccion);
            //analiza el error de la direccion recibida
        }
    }
```

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 Preparador(){
    while (true){
        adn = generarMuestra()
        Administrador!muestras(adn);
    }

}

process ArmadorSet(){
    elem muestra;
    elem resul;
    cola resultadosArchivados;
    while (true){
        Administrador!Pedido();
        Administrador?enviaMuestra(muestra);
```

```
        Analizador!enviaMuestra(armarSet(muestra));
        Analizador?resultados(resul);
        resultadosArchivados.push(resul);
    }
}

process Analizador (){
    while (true){
        ArmadorSet?enviaMuestra(muestra);
        //analiza la muestra
        ArmadorSet!resultados(resultadosAnalisis);
    }
}

process Administrador (){
    cola colaMuestras;
    do Preparador?muestras(push.colaMuestras);
    [] !colaMuestras.vacia(); ArmadorSet?Pedido() -> ArmadorS
}
```

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.

```
Process Alumno (id: 0.. N-1]{
    //resolver examen
    Administrador!enviarExamen(examen. id);
    Profesor?recibirNota(nota);
}

Process Profesor (){
    elem examen;
    int idA
    for i:= 1 to N {
```

```
        Administrador!pedido();
        Administrador?examen(examen, idA);
        Alumno[idA]!recibirNota(corregirExamen(examen));
    }

}

Process Administrador(){
    cola examenes;
    elem examen;
    int idA;
    for i:= 1 to N*2 {
        if Alumno[*]?enviarExamen(examen, idA); -> examene
        [] !examenes.vacia(); Profesor?pedido() -> Profeso
        od
    }
```

b) Considerando que P>1.

```
Process Alumno (id: 0.. N-1]{
    //resolver examen
    Administrador!enviarExamen(examen. id);
    Profesor[*]?recibirNota(nota);
}

Process Profesor (id: 0.. P-1){
    elem examen;
    int idA
    boolean hayExamenes=true;
    Administrador!pedido(id);
    Administrador? hay(hayExamenes);
    while (hayExamenes){
        Administrador?examen(examen, idA);
        Alumno[idA]!recibirNota(corregirExamen(examen));
        Administrador!pedido(id);
        Administrador? hay(hayExamenes);
    }
```

```
}

Process Administrador(){
    cola examenes;
    elem examen;
    int idA;
    for i:= 1 to (N*2) {
        if Alumno[*]?enviarExamen(examen, idA); -> examene
            [] !examenes.vacia(); Profesor[*]?pedido(idP)
                 Profesor[idP]! hay(true);
                 Profesor[idP]! examen(examenes.pop(examen
                 }
        od
    }
    for i:= 1 to P {
        Profesor[*]?pedido(idP);
        Profesor[idP]!hay(false); //avisa a los profesores
    }
}
```

c) Ídem b) pero considerando que los alumnos no comienzan a realizar su examen hasta
que todos hayan llegado al aula.

```
Process Alumno (id: 0.. N-1]{
    Administrador!llegue();
    Administrador?comenzar();
    //resolver examen
    Administrador!enviarExamen(examen. id);
    Profesor[*]!recibirNota(nota);
}

Process Profesor (id: 0.. P-1){
    elem examen;
    int idA
```

```
    boolean hayExamenes=true;
    Administrador!pedido(id);
    Administrador? hay(hayExamenes);
    while (hayExamenes){
        Administrador?examen(examen, idA);
        Alumno[idA]!recibirNota(corregirExamen(examen));
        Administrador!pedido(id);
        Administrador? hay(hayExamenes);
    }

}

Process Administrador(){
    cola examenes;
    elem examen;
    int idA;
    for i:= 1 to N {
        Alumno[*]?llegue();
    }
    for i:= 1 to N {
        Alumno [i]!comenzar();
    }

    for i:= 1 to N {
        do Alumno[*]?enviarExamen(examen, idA); -> examene
            [] !examenes.vacia(); Profesor[*]?pedido(idP)
                 Profesor[idP]! hay(true);
                 Profesor[idP]! examen(examenes.pop(examen
                 }
        od
    }
    for i:= 1 to P {
        Profesor[*]?pedido(idP);
        Profesor[idP]!hay(false); //avisa a los profesores
    }
}
```

Nota: maximizar la concurrencia; no generar demora innecesaria; todos los procesos deben
terminar su ejecución

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.

```
Process Persona [id: 0.. P-1]{
    Emppleado!solicitar(id);
    Empleado?usarSimulador();
    //usa el simulador
    Empleado!retirarse();
}

Process Empleado(){
    int idP;
    while (true){
        Persona[*]?solicitar(idP);
        Persona[idP]!usarSimulador();
        Persona[idP]?retirarse();
    }
}
```

b) Modifique la solución anterior para que el empleado considere el orden de llegada para
dar acceso al simulador.

```
Process Persona [id: 0.. P-1]{
    Administrador!solicitar(id);
    Empleado?usarSimulador();
```

```
        //usa el simulador
        Empleado!retirarse();
    }

    Process Administrador(){
        int idP;
        cola espera;
        do Persona[*]?solicitar(idP) -> espera.push(id
P);
            []!espera.vacia() ; Empleado?pedido() -> Emp
leado!siguiente(espera.pop);
        od
    }

    Process Empleado(){
        int idP;
        while (true){
            Administrador!pedido();
            Administrador?siguiente(idP);
            Persona[idP]!usarSimulador();
            Persona[idP]?retirarse();
        }
    }
```

Nota: cada persona usa sólo una vez el simulador.

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.. P-1]{
        Administrador!solicitar(id);
        Maquina?usarMaquina();
        //usa el simulador
```

```
        Maquina!retirarse();
    }

    Process Administrador(){
        int idE;
        cola espera;
        do Espectador[*]?solicitar(idE) -> espera.push(idE
            []!espera.vacia() ; Maquina?pedido() -> Maquin
        od
    }

    Process Maquina(){
        int idP;
        while (true){
            Administrador!pedido();
            Administrador?siguiente(idE);
            Espectador[idE]!usarMaquina();
            Persona[idE]?retirarse();
        }
    }
    -------------------------Otra solución----------------
    Process Espectador [id: 0.. P-1]{
        Administrador!solicitar(id);
        Administrador?usarMaquina();
        //usa el simulador
        Administrador!retirarse();
    }

    Process Administrador(){
        int idE;
        cola espera;
        bool libre = true
        do Espectador[*]?solicitar(idE) ->
                if(libre){
                    libre = false
                    Espectador[idE]!usarMaquina()
                } else {
                    espera.push(idE)
```

```
                }
                espera.push(idE);
            [] Espectador[*]?retirarse() ->
                    if (!espera.vacia()){
                        Espectador[espera.pop()]!usarMaqui
                    }
                    else
                        libre=true
        od
    }
```