Programación Concurrente - Practica 4

Tomatis, Vicens, Torchia, Olmos, Manzin, Petraccaro, Traberg, Villareal

(Ejercicio 1)

Consigna:
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 comunicaciones serán necesarios/convenientes para resolver el problema. Luego, resuelva considerando las siguientes situaciones:

a) Existe un único empleado, el cual atiende por orden de llegada.

Ej a

```
chan peticiones(int,text)
chan[N] respuestas(int)
```

Proceso Empleado

```
Process Empleado{
    int cliente
    text peticion
    while (True){
        recv peticiones(cliente,peticion)
        send respuestas[id](peticion)
    }
}
```

Proceso Cliente

```
Process Cliente[id:0..N-1]{
    text repuesta
    send(z,(id,peticion()))
    recv(respuestas[id])
}
```

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

Ej b

```
chan peticiones(int,text)
chan[N] respuestas(int)
type Cliente(int,text)
```

Proceso Empleado

```
Process Empleado[0..E-1]{
    Cliente cliente;
    while True{
        receive peticiones(cliente)
        res=procesar(cliente.peticion)
        send respuestas[cliente.id](res)
    }
}
```

Proceso Cliente

```
Process Cliente[0..N-1]{
    text repuesta
    send peticiones(id,getPeticion())
    receive respuestas[id](respuesta)
}
```

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?

Ej c

```
chan listo(int)
chan peticiones(int,text)
chan[E] trabajo(text)
chan[N] respuestas(int)
```

Proceso Coordinador

```
Process Coordinador{
    int empleado
    cliente(int,text)
    while True{
        receive listo(idEmpleado)
        if (empty(peticiones)){
            send trabajo[idEmpleado](-1,"")
        }else{
            receive peticiones(idCliente,peticion)
            send trabajo[idEmpleado](cliente)
        }
    }
}
```

Proceso Empleado

```
Process Empleado[0..E-1]{
    cliente(int,text)
    while True{
        send listo(id)
        receive trabajo(cliente)
        if (cliente.id<>-1){
            delay(15) -- tareas administrativas durante
 ↪  15 minutos
        }else{
            send respuestas[cliente.id](resolver())
        }
    }
}
```

Proceso Cliente

```
Process Cliente[0..N-1]{
    text repuesta
    send peticiones((id,peticion()))
    receive respuestas[id](respuesta)
}
```

(Ejercicio 2)

Consigna:
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: maximizando la concurrencia
Respuesta:

Ejercicio 2

```
chan listo (int)
chan recibirMinimo (int)
chan entrar[5] (int,double)
chan comprobantes[0..P-1] (text)
chan sali (int)
chan sync
```

Proceso Coordinador

```
Process Coordinador{
    int idCliente, idCajero
    array arrayColas[5] = ([5], 0)
    while (True){
        receive sync
        if (!empty(listo)){
            receive listo (idCliente)
            idCajero = min(arrayColas)
            arrayColas[idCajero]++
            send recibirMinimo[idCliente](idCajero)
        }else{
            receive sali (idCajero)
            arrayColas[idCajero]--
        }
    }

    function min(in arrayColas){
        min=999
        minId=-1
        for i=0 to 4{
            if arrayColas[i]<min{
                min=arrayColas[i]
                minId=i
            }
        }
        return minId
    }
}
```

Proceso Cliente

```
Process Cliente[id:0..P-1]{
    int idCajero
    text comprobante
    send listo (id)
    send sync
    receive recibirMinimo[id](idCajero)
    send entrar[idCajero] (id,pago())
    receive comprobantes[id] (comprobate)
    send sali (idCajero)
    send sync
}
```

Proceso Cajero

```
Process Cajero[0..4]{
    cliente (int, double)
    while True{
        receive entrar[id] (cliente)
        comprobante=generarComprobate(cliente.pago)
        send comprobantes[cliente.id](comprobante)
    }
}
```

(Ejercicio 3)

Consigna:
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.
Respuesta:

Ejercicio 3

```
chan miPedido (int,text)
chan comida[P] (text)
chan cocinando (int,text)
chan pedido[3] (int,text)
```

Ejercicio Cliente

```
Process Cliente[0..P-1]{
    send miPedido (id,generarPedido())
    receive comida[id]
    -- come y se va
}
```

Ejercicio Coordinador

```
Process Coordinador{
    int idCliente, idVendedor
    text orden
    while True{
        receive listo (idVendedor)
        if (empty(miPedido)){
            send pedido[idVendedor] (-1,"")
        }else{
            receive miPedido (idCliente, orden)
            send pedido[idVendedor] (idCliente, orden)
        }
    }
}
```

Ejercicio Vendedor

```
Process Vendedor[0..2]{
    int idCliente
    text pedido
    while True{
        send listo (id)
        receive pedido[id] (idCliente,pedido)
        if (idCliente<>-1){
            send cocinando (idCliente,pedido)
        }else{
            delay(random(1..3))
        }
    }
}
```

Ejercicio Cocinero

```
Process Cocinero[0..1]{
    int idCliente
    text pedido, comida
    while True{
        receive cocinando (idCliente,pedido)
        comida = haciendoElPedido(pedido)
        send entrega[idCliente] (comida)
    }
}
```

(Ejercicio 4)

Consigna:
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, pero siempre dando prioridad a los que terminaron de usar la cabina. A cada cliente se le entrega un ticket factura. Nota: maximizar la concurrencia; suponga que hay una función Cobrar() llamada por el empleado que simula que el empleado le cobra al cliente.
Respuesta:

Ejercicio 4

```
chan sync
chan listo(int)
chan atencion[N](int)
chan liberar(int,int)
chan comprobantes[N](text)
```

Proceso Cliente

```
process Cliente[id:0..N-1]{
    send listo(id)
    send sync
    receive atencion[id](idCabina)
    -- usar cabina
    send liberar(id,idCabina)
    send sync
    receive comprobantes[id](comprobante)
}
```

Proceso Empleado

```
process Empleado{
    while (true){
        receive sync
        if (!empty(liberar) or empty(colaLibres)){
            receive liberar(idCliente,idCabina)
            colaLibres.push(idCabina)
            comprobante=generarComprobante()
            send comprobantes[idCliente](comprobante)
        }else {
            receive listo(idCliente)
            cabinaLibre=pop(colaLibres)
            send atencion[idCliente](cabinaLibre)
        }
    }
}
```

(Ejercicio 5)

Consigna:
Resolver la administración de las impresoras de una oficina. Hay 3 impresoras, N usuarios y 1 director. Los usuarios y el director 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, pero siempre dando prioridad a los pedidos del director. Nota: los usuarios y el director no deben esperar a que se imprima el documento.
Respuesta:

Canales Ejercicio 5

```
chan sync
chan mailDirector
chan mailUsuario
chan impresoras(int)
chan copias[3]
```

Proceso Director

```
process Director[id:N]{
    while(true){
        -- trabaja
        send mailDirector(pedido)
        send sync
    }
}
```

Proceso Usuario

```
process Usuario[id:0..N-1]{
    while(true){
        -- trabaja
        send mailUsuario(pedido)
        send sync
    }
}
```

Proceso Coordinador

```
process Coordinador{
    while (true){
        receive impresoras(idImpresora)
        receive sync
        if (!empty(mailDirector)){
            receive mailDirector(pedido)
        }else {
            receive mailUsuario(pedido)
        }
        send copias[idImpresora](pedido)
    }
}
```

Proceso Impresora

```
process Impresora[id:0..2]{
    while (true){
        send impresoras(id)
        receive copias[id](pedido)
        Imprimir(pedido)
    }
}
```

(Ejercicio 1)

Consigna:
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 resolver el problema.

b) Implemente una solución con PMS.

Respuesta:

Proceso Examinador

```
Process Examinador[0..R-1]{
virus sospecha
    while True{
        sospecha = buscarVirus()
        Buffer! virus(sospecha)
    }
}
```

Proceso Analizador

```
virus trabajo
Process Analizador{
    while true{
        Buffer!()
        Buffer?(trabajo)
        analizo(trabajo)
    }
}
```

Proceso Buffer

```
cola analizar
virus sospecha
Process Buffer{
    while True{
        do
        [] Examinador[*]? virus(sospecha) -> push(analizar, sospecha)
        [] not empty(analizar);Analizador?() -> Analizador!(pop(analizar))
        od
    }
}
```

(Ejercicio 2)

Consigna:
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.

Respuesta:

Proceso Preparador

```
Process Preparador{
    while True{
        muestra = prepararMuestra()
        Buffer!(muestra)
    }
}
```

Proceso Analista

```
Process Analista{
    while True{
        Armador?(muestra)
        resultado = analizarMuestra(muestra)
        Armador!(resultado)
    }
}
```

Proceso Armador

```
Process Armador{
    While True{
        Buffer!()
        Buffer?(muestra)
        muestra = armoMuestra(muestra)
        Analista!(muestra)
        Analista?(respuesta)
        Archivar(respuesta)
    }
}
```

Proceso Buffer

```
Process Buffer{
    while True{
        do Preparador?(muestra) -> push(muestras, muestra)
        [] if not empty(muestras);Armador?() -> Armador!(pop(muestras)
    }
}
```

(Ejercicio 3)

Consigna:
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 y no generar demora innecesaria.
Respuesta:

Proceso BufferResolucion

```
Process BufferResolucion{
    while True{
        do Alumno[*]?(resolucion,id) -> push(resoluciones, resolucion, id)
        [] not empty(resoluciones);Profesor[*]?(idP) -> Profesor[idP]!(pop(resoluciones,id))
    }
}
```

Proceso Alumno

```
Process Alumno[id:0..N-1]{
    if id!=0{
        Alumno[N-id]?()
        Alumno[id+1 mod N]!()
    }else{
        Alumno[id+1 mod N]!()
        Alumno[N-id]?()
    }
    resolucion = resolverExamen(examen)
    BufferResolucion!(resolucion,id)
    Profesor[*]?(correcion)
}
```

Proceso Profesor

```
Process Profesor[id:0..P-1]{
    while True{
        BufferResolucion!(id)
        BufferResolucion?(resolucion,idA)
        correccion = corregirExamen(resolucion)
        Alumno[idA]!(correccion)
    }
}
```

Proceso BufferResolucion con otra barrera

```
Process BufferResolucion{
    while True{
        do Alumno?(resolucion) -> push(resoluciones, resolucion)
        [] not empty(resoluciones);Profesor?() -> Profesor!(pop(resoluciones)
    }
}
```

Proceso Alumno con otra barrera

```
Process Alumno[0..N-1]{
    Profesor[0]!()
    Profesor[0]?()
    resolucion = resolverExamen(examen)
    BufferResolucion!(resolucion)
    BufferCorrecion!()
    BufferCorreccion?(correcion)
}
```

Proceso Profesor

```
Process Profesor[id:0..P-1]{
    if id==0{
        for i=0 to N-1{
            Alumno?()
        }
        for i=0 to N-1{
            Alumno!()
        }
    }
    while True{
        BufferResolucion!()
        BufferResolucion?(resolucion)
        correccion = corregirExamen(resolucion)
        BufferCorrecion!(correccion)
    }
}
```

Proceso BufferCorreccion

```
Process BufferCorreccion{
    while True{
        do Profesor?(correccion) -> push(correcciones, correccion)
        [] not empty(correcciones);Alumno?() -> Alumno!(pop(correcciones)
    }
}
```

(Ejercicio 4)

Consigna:
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. El empleado deja usar el simulador a las personas respetando el orden de llegada. Nota: cada persona usa sólo una vez el simulador.

Respuesta:

Persona

```
Process Persona [id: 0..P-1]{
    Admin! pedir(id)
    Empleado? libre()
    -- Usar simulador
    Empleado! irse()
}
```

Empleado

```
Process Empleado{
    int idPersona
    while (true){
        Admin! siguiente()
        Admin? pasar(idPersona)
        Persona[idPersona]! libre()
        Persona[idPersona]? irse()
    }
}
```

Admin

```
Process Admin{
    int idPersona
    cola fila
    do
        [] Persona[*]? pedir(idPersona) ->
            push (fila, idPersona)
        [] not empty(fila); Empleado?siguiente() ->
            Empleado!pasar(pop(fila))
    od
}
```

(Ejercicio 5)

Consigna:
En un estadio de fútbol hay una máquina expendedora de gaseosas que debe ser usada por E Espectadores de acuerdo al 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.
Respuesta:

Proceso Espectador

```
Process Espectador[id:0..E-1]{
    Admin!entrar(id)
    Admin?usar()
    -- usar
    Admin!salir()
}
```

Proceso Admin

```
Process Admin{
    libre = true
    cola
    do
        [] libre; Espectador[*]?entrar(id) -> libre=false; Espectador[id]!usar()
        [] not libre; Espectador[*]?entrar(id) -> push(cola,id)
        [] empty(cola); Espectador[*]?salir -> libre=true
        [] not empty(cola); Espectador[*]?salir -> Espectador[pop(cola)]!usar()
    od
}
```