UNIVERSIDAD NACIONAL DE LA PLATA

Programacion Concurrente - Resolucion Examenes

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

(Parcial MC - 2020 - 1 - Tema 6)

Consigna:
Resolver con SEMÁFOROS el siguiente problema. En una empresa hay UN Coordinador y 30 Empleados que formarán 3 grupos de 10 empleados cada uno. Cada grupo trabaja en una sección diferente y debe realizar 345 unidades de un producto. Cada empleado al llegar se dirige al coordinador para que le indique el número de grupo al que pertenece y una vez que conoce este dato comienza a trabajar hasta que se han terminado de hacer las 345 unidades correspondientes al grupo (cada unidad es hecha por un único empleado). Al terminar de hacer las 345 unidades los 10 empleados del grupo se deben juntar para retirarse todos juntos. El coordinador debe atender a los empleados de acuerdo al orden de llegada para darle el número de grupo (a los 10 primeros que lleguen se le asigna el grupo 1, a los 10 del medio el 2, y a los 10 últimos el 3). Cuando todos los grupos terminaron de trabajar el coordinador debe informar (imprimir en pantalla) el empleado que más unidades ha realizado (si hubiese más de uno con la misma cantidad máxima debe informarlos a todos ellos). Nota: maximizar la concurrencia; suponga que existe una función Generar que simula la elaboración de una unidad de un producto.
Respuesta:

Variables compartidas

```
sem mutexCola = 1, numeroDado[30] = 0, salida[3] = 0, terminaron = 0, llegoEmpleado = 0
Cola<int> colaLlegada
int asignarGrupo[30]=([30] -1), esperando[3] = 0, producidos[30]=([30] 0)
```

Proceso Empleado

```
sem produccionGrupo[3]=([3] 1)
sem mutexEsperando[3]=([3] 1)
sem salida[3]=([3] 0)
int cantidad[3]=([3] 0)
int esperando[3]=([3] 0)
Process Empleado[id=0..29]{
    P(mutexCola)
    push(colaLlegada,id)
    V(mutexCola)
    V(llegoEmpleado)
    P(numeroDado[id])
    int numeroGrupo=asignarGrupo[id]
    P(produccionGrupo[numeroGrupo])
    while(cantidad[numeroGrupo]<345){
        cantidad[numeroGrupo]++
        V(produccionGrupo[numeroGrupo])
        Generar()
        producidos[id]++
        P(produccionGrupo[numeroGrupo])
    }
    V(produccionGrupo[numeroGrupo])

    P(mutexEsperando[numeroGrupo])
    esperando[numeroGrupo]++
    if (esperando[numerogrupo]==10){
        for i to 10 {
            V(salida[numeroGrupo])
        }
        V(terminaron)
    }
    V(mutexEsperando[numeroGrupo])
    P(salida[numeroGrupo])
    -- se van
}
```

Proceso Coordinador

```
Process Coordinador{
    for i = 0 to 2{
        for j = 1 to 10{
            P(llegoEmpleado)
            P(mutexCola)
            pop(colaLLegada, id)
            V(mutexCola)
            asignarGrupo[id] = i
            V(numeroDado[id])
        }
    }
    -- terminan todos
    for i = 0 to 2{
        P(terminaron)
    }
    int max=-1
    Cola<int> colaMaximos
    for i=0 to 29{
        if(producidos[i]>max){
            max=producidos[i]
            -- vaciarla
            while(not empty(colaMaximos)) {
                pop(colaMaximos)
            }
            push(colaMaximos,i)
        }
        if (producidos[i]==max){
            push(colaMaximos,i)
        }
    }
    imprimir("cantidad maxima: "+max)
    imprimir("empleados con cantidad maxima:")
    while (not empty(colaMaximos)){
        imprimir(pop(colaMaximos))
    }
}
```

Consigna:
2. Resolver con MONITORES la siguiente situación. En la guardia de traumatología de un hospital trabajan 5 médicos y una enfermera. A la guardia acuden P Pacientes que al llegar se dirigen a la enfermera para que le indique a que médico se debe dirigir y cuál es su gravedad (entero entre 1 y 10); cuando tiene estos datos se dirige al médico correspondiente y espera hasta que lo termine de atender para retirarse. Cada médico atiende a sus pacientes en orden de acuerdo a la gravedad de cada uno. **Nota:** maximizar la concurrencia.
Respuesta:

Monitor Atencion

```
Monitor Atencion[id:0..4]{
    Procedure Encolarse(in int id,gravedad){
        -- gravedad criterio de ordenamiento
        InsertarOrdenado(colaPacientes,id,gravedad)
        signal(hayPacientes)
        wait(atendido[id])
    }
    Procedure EsperarPaciente(out int idPaciente){
        if(empty(colaPacientes)){
            wait(hayPacientes)
        }
        idPaciente=pop(colaPacientes)
    }
    Procedure LiberarPaciente(int idPaciente){
        signal(atendido[idPaciente])
    }
}
```

Monitor Enfermera

```
Monitor Enfermera{
    Procedure Entrar(out int idMedico, gravedad){
        -- how
        idMedico=ObtenerMedico()
        gravedad=ObtenerGravedad()
    }
}
```

Proceso Paciente

```
Process Paciente[id:0..P-1]{
    Enfermera.Entrar(idMedico,gravedad)
    Atencion[idMedico].Encolarse(id,gravedad)
}
```

Proceso Medico

```
Process Medico[id:0..4]{
    while(true){
        Atencion[id].EsperarPaciente(idPaciente)
        -- esta siendo atendido
        Atender(idPaciente)
        Atencion[id].LiberarPaciente(idPaciente)
    }
}
```

Consigna:

2. Se tienen N procesos que comparten el uso de una CPU.

Un proceso, cuando necesita utilizar la CPU, le pide el uso, le proporciona el trabajo que va a ejecutar y el tiempo que insume ejecutar el trabajo. Luego, el proceso se queda dormí do hasta que la CPU haya finalizado de ejecutar su trabajo momento en que es despertado y prosigue su ejecución normal.

La CPU funciona de la siguiente manera. Cada vez que el proceso i (i:1..N) pide CPU esta lo pone en el lugar iesimo de una lisia de procesos esperando para ejecutar. La CPU recorre Circularmente la lista de procesos esperando a ser ejecutados (del 1 al N) T sí él proceso esta lista lo saca de ía lisia y lo ejequia durante un lapso de 10 mis (o un lapso menor, si el tiempo de ejecución del proceso es.menor a 10 mis). Una vez que ejecutó un proceso, si todavía le queda al proceso tiempo de.ejecución lo vuelve a poner en su lugar en Ía lista y pasa al siguiente: Si el proceso terminó de ejecutar su trabajo, la CPU lo despierta para que continúe su ejecución y continúa con el siguiente proceso de la lista. Cuando la lisia está vacío, ía CPU no debe hacer nada, simplemente debe esperar a que algún proceso ingrese a la lista de procesos en espera de ejecución, para empezar a trabajar.

Tenga en cuenta que existe la función delay(x) que retarda un proceso durante x mis. La ejecución de un proccsd puede ser modelizada utilizando esta función,

No hagan suposiciones acerca de ios tiempos de ejecución de los procesos Modelice utilizando Monitores.

Respuesta:

Monitor AccesoCPU

```
Monitor AccesoCPU{
    (text,int) trabajos[N] = ([N]("",-1))
    int cantprocesos = 0, tiempo = 0, i = 0
    cond termine[N]

    Procedure Encolar(in text trabajo, in int tiempo, in int id)
    {
        trabajos[id]= (trabajo,tiempo)
        signal(hayprocesos)
        cantprocesos++
        wait(termine[id])
    }

    Procedure solicitarTrabajo(out int id, out int tiempo)  {
        if (cantprocesos==0){
            wait(hayProcesos)
        }
        encontre=false
        while (not encontre){
            nom,tiempoproc=trabajos[i]
            if tiempoproc != -1{
                id = i
                if tiempoproc > 10 {
                    tiempo=10
                }
                else{
                    tiempo=tiempoproc
                }
                encontre=true
            }
            i++
            if i == N{i=0}
        }
    }

    Procedure devolverTrabajo(in int id, in int tiempo)
    {
        nom,tiempoproc=trabajos[id]
        if tiempoproc - tiempo  == 0{
            signal(termine[id])
            trabajos[id] = ("",-1)
            cantprocesos--
        }else{
            trabajos[id] = (nom,tiempoproc - tiempo)
        }

    }

}
```

Proceso Proceso

```
Process Proceso[id:0..N-1]{
    text trabajo = ObtenerNombre()
    int tiempo = CalcularTiempo()
    AccesoCPU.Encolar(trabajo,tiempo,id)
    -- se termina el proceso
}

```

Proceso CPU

```
Process CPU{
    while(true){
       AccesoCPU.solicitarTrabajo(id,tiempo)
       delay(tiempo)
       AccesoCPU.devolverTrabajo(id,tiempo)
    }
}
```

Consigna:

1. Suponga un juego donde hay 30 competidores. Cuando los jugadores llegan avisan al encargado, una vez que están ios 30 el encargado del juego les entrega un numero aleatorio del 1 al 15 de tal manera que dos compelí dores tendrán el mismo numero. (Suponga que existe una función DarNumeroQ que devuelve en forma aleatoria un numero de! 1 aí 15, el encargado no guarda el numero que les asigna a los competidores). Una vez que ya se entregaron los 30 números/los competidores buscaran concurrentemente su compañero que tenga el mismo numero (tenga en cucntn que pueden empezar a buscar cuando lodos los competidores tengan cf numero no antes; Además la búsqueda de un jugador no interfiere con la búsqueda de otros que tengan distinto numero). Cuando los competidores se encuentran permanecen en una snla durante 15 minutos y dejan de jugar. Luego cada uno de los competidores avisa al encargado que termino de jugar y espero a que su compañero (el que tenia ci mismo numero) también avise que finalizo para luego irse ambos; el encargado cuando llega ei segundo competidor Ies devuelve a ambos el resultado que obtuvieron que es el orden en que so van, (Los primeros en irse; tendrán como resultado 1, los últimos i 5). Para modelizar el tiempo utilice la función Delay(x) que produce un retardo de x minutos.

Respuesta:

Variables Compartidas

```
sem esperandoLlegada = 0, llego=0, esperandoLlegada = 0,
numeroAsignado = 0, mutexllegada = 1, entraralmostrador = 0, esperandoLugar = 0
sem espcompañero[N/2] = ([N/2]0)
sem mutexesperando[N/2] = ([N/2]1)
sem mutexafuera[N/2] = ([N/2]1)
int arrayEquipos[N]
int lugares[N/2]
int compañerosAfuera[N]
int cantencontrados[N/2] = ([N/2]0)

```

Proceso Proceso

```

Process Jugador[id:0..N-1]{
    int compañero
    V(llego)
    P(esperandoLlegada)
    P(numeroAsignado)
    int numeroGrupo=array_equipos[id]
    for i=1 to N{
        if numeroGrupo==array_equipos[i] and i<>id{
            compañero=i
        }
    }
    P(mutexesperando[numeroGrupo])
    cantencontrados[numeroGrupo]++
    if cantencontrados[numeroGrupo]==2{
        V(mutexesperando[numeroGrupo])
        for i=1 to 2{
            V(espcompañero[numeroGrupo])
        }
    }else{
        V(mutexesperando[numeroGrupo])
    }
    P(espcompañero[numeroGrupo])

    delay(15)

    -- Comienza proceso de salida
    P(mutexAfuera[numeroGrupo])
    companierosAfuera[numeroGrupo]++
    V(mutexAfuera[numeroGrupo])
    P(mutexLlegada)
    idlugar=numeroGrupo
    V(leAviso)
    P(esperandoLugar)
    -- Se van

}

```

Proceso Encargado

```
Process Encargado{

    for i=1 to N{
        P(llego)
    }
    for i=1 to N{
        V(esperandoLlegada)
    }
    for i=1 to N{
        array_equipos[i]=asignarNumero()
    }
    for i=1 to N{
        V(numeroAsignado)
    }
    for i=1 to N{
        P(leAviso)
        if companierosAfuera[idLugar]==2{
            lugares[idlugar]=i
            for i=1 to 2{
                V(esperandolugar)
            }
        }
        V(mutexLlegada)
    }
}
```

Consigna:
Se debe controlar el acceso a una impresora. Existen dos tipos de procesos trabajadores, A procesos de tipo 1 y B procesos de tipo 2 que trabajan y al final del día por única vez intentan imprimir un resumen de la siguiente manera: Proceso tipo 1: espera hasta poder imprimir el resumen. Proceso tipo 2: intenta imprimir su resumen, si no lo logro en 2 minutos, se retira sin imprimirlo. Además existen E técnicos encargados del mantenimiento de la impresora, donde cada uno podrá acceder a la impresora si no hay ningún otro proceso (trabajador o técnico) usándola. Al acceder detecta si hay algún error y lo resuelve (tarda 2 minutos en realizar esto). Para que un proceso trabajador pueda imprimir debe asegurarse de que la impresora esté libre (ningún trabajador o técnico debe estar usando la impresora). Al acceder a la impresora imprime durante cierto tiempo.

Respuesta:
– Preguntar

Proceso Empleado1

```
Process Empleado1[id:0..A-1]{
    bool ocupado
    acceso.ocupar(ocupado)
    -- imprimir
    acceso.liberar()
}
```

Proceso Empleado2

```
Process Empleado2[id:0..B-1]{
    EstadoT2[id].llego()
    acceso.ocuparT2(sigueEsperando,id)
    if(sigueEsperando){
        -- imprimir
    }
    acceso.liberar()
}
```

Proceso Acceso

```
monitor Acceso{
    bool libre=false
    procedure ocupar(){
        while not libre{
            wait(impresoraLibre)
        }
        libre=false
    }
    procedure ocuparT2(out bool sigueEsp,in int id){
        while not libre{
            wait(impresoraLibre)
        }
        libre=false
        EstadoT2[id].sigueEsperando(sigueEsp)
    }
    procedure liberar(){
        libre=true
        signal_all(impresoraLibre)
    }

}
```

Monitor EstadoT2

```
monitor EstadoT2[id:0..B-1]{
    bool seFue=false
    bool llego[B]=([false] B)
    cond llegar[B]
    Procedure iniciar(){
        if(not llego[id]){
            wait(llegar[id])
        }
    }

    Procedure timeout(){
        seFue=true
    }

    Procedure llego(){
        llego[id]=true
        signal(llegar[id])
    }

    Procedure sigueEsperando(out bool esp){
        esp=not seFue
    }
}
```

Proceso Timer

```
process Timer[id:0..B-1]{
    EstadoT2[id].iniciar()
    delay 2 min
    EstadoT2[id].timeout()
}
```

Consigna:
monitores

Respuesta:
– Preguntar

Proceso Carrera

```
Monitor Carrera{
    procedure largada(){
        listo++
        if listos==C{
            signalall(largar)
        }else{
            wait(largar)
        }
    }
    procedure llegarAlPuente(){
        if !libre{
            cantEsperando++
            wait(esperandoTurno)
        }
        else{
            libre=False
        }
    }
    procedure salirAlPuente(){
       if cantEsperando==0{
            libre=True
       }else{
            cantEsperando--
            signal(esperandoTurno)
       }
    }
}
```

Proceso Corredor

```

Process Corredor[id:0..C-1]{
    carrera.largada()
    -- correr
    carrera.llegarAlPuente()
    -- cruza
    carrera.salirAlPuente()
    -- correr
}

```

Proceso Acceso

```
monitor Acceso{
    bool ocupado
    procedure ocupar(ocupado){
        while ocupado{
            wait(impresoraLibre)
        }
        ocupado=True
    }
    procedure liberar(){
        ocupado=False
        signalall(impresoraLibre)
    }
}
```

Proceso Acceso

```
monitor Tiempo{

    -- implementar array de ticks
    procedure timeout(out bool esperando){
        if tick>120{
            esperando=False
        }
        else{
            esperando=True
        }
    }
    procedure aumentarTicks(){
        tick++ -- esto lo llama el os atraves de
    ↪   interrupciones
    }
}
```

Consigna:
monitores

Respuesta:

Proceso Acceso

```
    sem mutexprimero = 1, mutexllegada = 1, barrera =
↪   0, empezarexamen = 0,
    corremutex = 1, mutexcola = 1

```

Proceso Acceso

```
Process Alumno{
    P(mutexllegada)
    personas++
    if personas==P+3{
        V(mutexllegada)
        for i=1 to P+3{
            V(barrera)
        }
    }else{
        V(mutexllegada)
    }
    P(barrera)
    P(empezarexamen)
    examen
    P(mutexcola)
    push(colaterminados)
    V(mutexcola)
    V(entrego)
    P(esperanota[id])
}
```

Proceso Acceso

```
Process Profesor{
    P(mutexprimero)
    if primero == -1{
        primero=id
    }
    V(mutexprimero)
    P(mutexllegada)
    personas++
    if personas==P+3{
        V(mutexllegada)
        for i=1 to P+3{
            V(barrera)
        }
    }else{
        V(mutexllegada)
    }
    P(barrera)
    if primero == id{
        for i=1 to P{
            V(empezarexamen)
        }
    }
    P(corremutex)
    while(corregidos < P){
        corregidos++
        V(corremutex)
        P(mutexcola)
        pop(cola,idalumno)
        V(mutexcola)
        nota[idalumno]=corregir(idalumno)
        V(esperarnota[idalumno])
        P(corremutex)
    }
    V(corremutex)
}
```

(parcial 2021)

Resolver con SEMÁFOROS el siguiente problema. Simular un examen técnico para concursos Nodocentes en la Facultad, en el mismo participan 100 personas distribuidas en 4 concursos (25 personas en cada concurso) con un coordinador en cada una de ellos. Cada persona ya conoce en que concurso participa. El coordinador de cada concurso espera hasta que lleguen las 25 personas correspondientes al mismo, les entrega el examen a resolver (el mismo para todos los de ese concurso), y luego corrige los exámenes de esas 25 personas de acuerdo al orden en que van entregando. Cada persona al llegar debe esperar a que su coordinador (el que corresponde a su concurso) le dé el examen, lo resuelve, lo entrega para que su coordinador lo evalúe y espera hasta que le deje la nota para luego retirarse. Nota: maximizar la concurrencia; sólo usar los procesos que representes a las personas y a los coordinadores; todos los procesos deben terminar.

Proceso Concursante

```
process Concursante[0..99]{
    P(mutexLlegar[aula])
    llegue[aula]++
    if llegue[aula]==25{
        V(listos[aula])
    }
    V(mutexLlegar[aula])
    P(tengoExamen[aula])
    -- resolver
    P(mutexEntrega[aula])
    push(entregas[aula],id)
    V(entregue[aula])
    V(mutexEntrega[aula])
    P(devolucion[id])
    minota = notas[id]
}
```

Proceso Coordinador

```
process Coordinado[0..3]{
    P(listos[aula])
    for i=0 to 24{
        -- entrego examen
        V(tengoExamen[aula])
    }
    for i=0 to 24{
        P(entregue[aula])
        alumno = pop(entregas[aula])
        notas[alumno]=correccion(alumno)
        V(devolucion[alumno])
    }
}
```

(parcial 2021 2)

Resolver con MONITORES el siguiente problema. Se debe modelar el funcionamiento de un Complejo de Canchas de Paddle que posee 10 canchas, adonde acuden 40 personas para jugar. Cuando una persona llega busca el número de cancha a la cual debe ir, y se dirige a ella. Cuando a una cancha han llegado las 4 personas correspondientes, se juega el partido durante 60 minutos, y al terminar el mismo las 4 personas se retiran. El número de cancha se asigna a los jugadores de acuerdo al orden de llegada (los 4 primeros a la cancha 1, los siguientes 4 a la 2 y así sucesivamente). Nota: maximizar la concurrencia.

Proceso Jugadores

```
process Jugador[id:0..39]{
    Entrada.llegar(idCancha)
    Paddle[idCancha].entrar()
}
```

Proceso Cancha

```
process Cancha[id:0..9]{
    Paddle[id].iniciar()
    delay(60min)
    Paddle[id].terminar()
}
```

Monitor Entrada

```
Monitor Entrada{
    int cant=0
    Procedure llegar(out int idCancha){
        idCancha=cant/4
        cant++
    }
}
```

Monitor Paddle

```
Monitor Paddle[id:0..9]{
    Procedure iniciar(){
        if(not canchaLlena){
            wait(iniciar)
        }
    }
    Procedure entrar(){
        cant++
        if(cant==4){
            canchaLlena=true
            signal(iniciar)
        }
        wait(termino)
    }
    Procedure terminar(){
        signal_all(termino)
    }
}
```

(ejercicio 2021 semaf)

Resolver con SEMÁFOROS el siguiente problema. Se debe simular el funcionamiento de una mesa de votación en una elección donde hay 2 listas candidatas (A y B). A la misma acuden 300 personas para votar en el cuarto oscuro (se dispone de una función obtenerVoto() que retorna la lista a votar: A o B). Además está el Presidente de la Mesa que es quien habilita a las personas a pasar al cuarto oscuro de a una a la vez de acuerdo al orden de llegada; y cuando todas las personas han votado determina cual es la lista ganadora (A o B). Nota: sólo se deben usar los procesos que representes a las personas y al Presidente de Mesa.

Proceso Persona

```
process Persona[id:0..299]{
    P(mutexEspera)
    push(colaEsperar,id)
    V(mutexEspera)
    V(hayGente)
    P(esperar[id])
    contadorVotos[obtenerVoto()]++
    V(entregoVoto)
}
```

Proceso Presidente

```
process Presidente{
    for i=0 to 299{
        P(hayGente)
        P(mutexEspera)
        idPers=pop(colaEsperar)
        V(mutexEspera)
        V(esperar[idPers])
        P(entregoVoto)
    }
    if(contadorVotos["A"]<contadorVotos["B"]){
        imprimir("gano B")
    }else if(contadorVotos["A"]>contadorVotos["B"]){
        imprimir("gano A")
    }else{
        imprimir("empataron")
    }
}
```

Proceso Persona

```
process Persona[id:0..39]{
    Complejo.llegar(idCancha)
    Cancha[idCancha].jugar()
    -- se van
}
```

Monitor Complejo

```
Monitor Complejo{
    int cant=0
    Procedure llegar(out int idCancha){
        -- division enteros
        idCancha=cant/4
        cant++
    }
}
```

Monitor Cancha

```
Monitor Cancha[id:0..9]{
    int cant=0
    Procedure jugar(){
        cant++
        if (cant!=4){
            wait(esperando)
        }else{
            signal_all(esperando)
            delay(60min)
        }
    }
}
```

(Ejercicio 1 Redictado ATIC 2021)

Resolver el siguiente problema con MONITORES. En un examen de la secundaria hay un preceptor y una profesora que deben tomar un examen escrito a 45 alumnos. El preceptor se encarga de darle el enunciado del examen a los alumnos cundo los 45 han llegado (es el mismo enunciado para todos). La profesora se encarga de ir corrigiendo los exámenes de acuerdo al orden en que los alumnos van entregando. Cada alumno al llegar espera a que le den el enunciado, resuelve el examen, y al terminar lo deja para que la profesora lo corrija y le dé la nota. Nota: maximizar la concurrencia; todos los procesos deben terminar su ejecución; suponga que la profesora tienen una función corregirExamen que recibe un examen y devuelve un entero con la nota.

Proceso Alumno

```
process Alumno[id:0..44]{
    Entrada.entrar(examen,id)
    -- Hace el examen
    Resolucion.entregar(examen, nota)
}
```

Monitor Resolucion

```
Monitor Resolucion{
    Procedure entregar(in text examen, out int nota){
        push(colaExamenes,examen)
        signal(entregoExamen)
        wait(corrigieron)
        nota=notaCompartida
        signal(agarroNota)
    }

    Procedure esperarExamen(out text examen){
        if(empty(colaExamenes)){
            wait(entregoExamen)
        }
        examen=pop(colaExamenes)
    }

    Procedure entregarNota(in int nota){
        notaCompartida=nota
        signal(corrigieron)
        wait(agarroNota)
    }
}

```

Monitor Entrada

```
Monitor Entrada{
    Procedure entrar(out text miExamen,in int id){
        cantAlumnos++
        if(cantAlumnos==45){
            signal(llegaronTodos)
        }
        wait(esperarPreceptor[id])
        miExamen=examenes[id]
    }

    Procedure entregarExamenes(){
        if(cantAlumnos<45){
            wait(llegaronTodos)
        }
        for i=0 to 44{
            examenes[i]=ObtenerExamen()
            signal(esperarPreceptor[i])
        }
    }
}
```

Proceso Profesora

```
process Profesora{
    for i=0 to 44{
        Resolucion.esperarExamen(examen)
        nota = corregirExamen(examen)
        Resolucion.entregarNota(nota)
    }
}
```

Proceso Preceptor

```
process Preceptor{
    Entrada.entregarExamenes()
}
```

(Conferencistas)

Resolver con MONITORES el siguiente problema. En una sala se juntan 20 conferencistas y un coordinador para una conferencia internacional. Cuando todos han llegado a la sala (los 20 conferencistas y el coordinador) el coordinador abre la sesión con una presentación de 30 minutos, y luego cada conferencista realiza su presentación de 10 minutos, de a uno a la vez y de acuerdo al orden en que llegaron a la sala. Cuando todas las presentaciones terminaron, las personas (conferencistas y coordinador) se retiran.

Proceso Coordinador

```
Process Coordinador{
    Conferencia.llegarConf()
    delay(30)
    Conferencia.Esperar()
}
```

Proceso Conferencista

```
Process Conferencista{
    Conferencia.llegar()
    delay(10)
    Conferencia.Esperar()
}

```

Monitor Conferencia

```

    Procedure llegarConf(){
        if (cantConf < 20){
            wait(llegaronTodos)
            }
        cantconf++
    }

    Procedure llegar(){
        cantConf++
        if cantConf == 20{
            signal(llegaronTodos)
        }
        wait(turno)
    }

    Procedure Esperar{
        cantconf--
        if cantconf == 0{
            signal_all(termino)
        }else{
            signal(turno)
            wait(termino)
        }
    }

```

Proceso Coordinador

```
Process Coordinador{
    Conferencia.llegarConf()
    delay(30)
    Conferencia.Presentar()
}
```

Proceso Conferencista

```
Process Conferencista{
    Conferencia.llegar()
    delay(10)
    Conferencia.Terminar()
}

```

Monitor Conferencia

```
Monitor Conferencia{
    Procedure llegarConf(){
        if (cantConf < 20){
            wait(llegaronTodos)
        }
    }

    Procedure llegar(){
        cantConf++
        if cantConf == 20{
            signal(llegaronTodos)
        }
        wait(turno)
    }

    Procedure Presentar(){
        for i=1 to 20{
            signal(turno)
            wait(termino)
        }
        signal_all(terminaronTodos)
    }

    Procedure Terminar(){
        signal(termino)
        wait(terminaronTodos)
    }
}
```

(ParcialMonitores.rtf)

Resolver este ejercicio con Semáforos o Monitores. En un centro oftalmológico hay 2 médicos con diferentes especialidades. Existen N pacientes que deben ser atendidos. Para esto algunos de los pacientes puede ser atendido indistintamente por cualquier médico y otros solo por uno delos médicos en particular. Cada paciente saca turno con cada uno de los médicos que lo puede atender y espera hasta que llegue el turno con uno de ellos, espera a que termine de atenderlo y se retira. Nota: suponga que existe la función ElegirMedico() que retorna 1,2 o 3. (1 indica que solo se atiende con el medico 1, 2 que solo se atiende con el medico 2 o 3 que puede ser atendido indistintamente por cualquier medico)

(semaforos)

Proceso Paciente

```
Process Paciente[0..P-1]{
    opcion=ElegirMedico()
    if(opcion==3){
        P(mutexCola[opcion])
        if (cantPersonas[1]<cantPersonas[2]){
            V(mutexCola[opcion])
            opcion=1
        }else{
            V(mutexCola[opcion])
            opcion=2
        }
    }
    P(mutexCola[opcion])
    push(cola[opcion],id)
    cantPersonas[opcion]++
    V(mutexCola[opcion])
    P(esperando[id])

}

```

Proceso Medico

```
Process Medico[id:1..2]{
    while(true){
        P(haygente[id])
        P(mutexCola[id])
        idPaciente=pop(cola[id])
        cantPersonas[id]--
        V(mutexCola[id])
        V(esperando[idPaciente])
    }
}
```

(monitores)

Monitor Llegada

```
Monitor Llegada{
    Procedure Llegar(in int opcion, out int idMedico){
        if(opcion==3){
            if (esperando[1]<esperando[2]){
                opcion=1
            }else{
                opcion=2
            }
        }
        if(not libre[opcion]){
            esperando[opcion]++
            wait(esperar[opcion])
        }else{
            libre[opcion]=false
        }
    }
    Procedure proximo(in int idMedico){
        if(esperando[idMedico]>0){
            esperando[idMedico]--
            signal(esperar[idMedico])
        }else{
            libre[idMedico]=true
        }
    }
}
```

Proceso Paciente

```
Process Paciente[0..P-1]{
    Llegada.llegar(idMedico)
    Atencion[idMedico].atenderse()
}
```

Proceso Medico

```
    while(true){
        Llegada.proximo()
        Atencion[id].esperarPaciente()
        -- atender
        Atencion[id].retirarPaciente()
    }
```

Monitor Llegada

```
Monitor Atencion[id:1..2]{
        Procedure atenderse(){
            llego=true
            signal(llegar)
            wait(atendido)
        }

        Procedure esperarPaciente(){
            if(not llego){
                wait(llegar)
            }
        }
        Procedure retirarPaciente(){
            signal(atendido)
        }
}
```