Repaso 1er parcial

1e P2)

```
sem detectores = 3

process Persona[id: 0..N-1]{
    int cant = random()
    for i:= 1 to Cant:
        P(detectores)
        //lo usa
        V(detectores)
}
```

2b P3)

```
Monitor motorBD{
    int limite = 5
    int esperando = 0
    cond esperaBD

    procedure entrar(){
        if (limite > 0)
            limite--
        else{
            esperando++
            wait(esperaBD)
        }
```

```
    }


    procedure salir(){
        if (esperando > 0)
            esperando--
        limite++
        signal(esperaBD)
    }
```

3 P2)

```
sem mutex = 1
sem elementos = 5
cola instancias[5]

process Proceso[id:0..P-1]{
    P(elementos)
    P(mutex)
    cola.pop()
    V(mutex)
    //lo usa
    P(mutex)
    cola.push()
    V(mutex)
    V(elementos)
}
```

3e P3

```
Monitor fotocopiadora{
    cond esperaPersona, esperaEmpleado, esperaFotocopiadora
    int esperando = 0
    bool libre = true

    procedure pedirFotocopiadora(){
        signal(esperaPersona)
        esperando++
        wait(esperaEmpleado)
        libre=false
    }

    procedure dejarFotocopiadora(){
        signal(esperaFotocopiadora)
        libre = true
    }

    procedure darFotocopiadora(){
        if (esperando == 0)
            wait(esperaPersona)
        if (!libre)
            wait(esperaFotocopiadora)
        esperando--
        signal(esperaEmpleado)
    }


process persona [id : 0..N-1]{
    fotocopiadora.pedirFotocopiadora()
    //la usa
    fotocopiadora.dejarFotocopiadora()
}

process Empleado{
    for i := 1 to N:
```

```
        fotocopiadora.darFotocopiadora()
}
```

6 P3)

```
Monitor Aula{
    int alumnos = 0
    int nroGrupo
    cond inicioArmadoGrupos, esperaJTP, esperaAlumno


    procedure llegue(){
        alumnos++
        if (cant == 50)
            signal(inicioArmadoGrupos)
        wait(esperaJTP)
    }

    procedure armadoGrupos(){
        wait(inicioArmadoGrupos)
        signall(esperaJTP)
    }

    procedure darNroGrupo(grupo: in int){
        nroGrupo = grupo
        signal(esperaJTP)
        wait(esperaAlumno) //me da duda
    }

    procedure asignarmeGrupo(grupo: out int){
        wait(esperaJTP)
        grupo = nroGrupo
        signal(esperaAlumno) //me da duda
    }
```

```
Monitor Grupo[id: 0..24]{

    int notaGrupo

    procedure darNota(nota: in int){
        notaGrupo = nota
    }

    procedure recibirNotaAlumno(nota: out int){
        nota = notaGrupo
    }
}

Monitor JTP{
    int nroGrupo
    int nota = 25
    int grupos [25] = ([25] 0)

    procedure avisarTarea(id: in int){
        grupos[id]++
        if (grupos[id] == 2){
            Grupo[id].darNota(nota)
            nota--
        }
    }

}

Process alumno [id:0..49]{
    int nroGrupo
    int notaAlumno
    Aula.llegue()
    Aula.asignarmeGrupo(nroGrupo)
    //hace la tarea(hasta aca yo creo q ta bien)
    JTP.avisarTarea(nroGrupo)
```

```
    Grupo[nroGrupo].recibirNotaAlumno(notaAlumno)

}

process jtp{
    Aula.armadoGrupos()
    int nroGrupo
    for i := 1 to 50
        int nroGrupo = AsignarNroGrupo()
        Aula.darNroGrupo(nroGrupo)
```

Semáforos

Resolver los problemas siguientes:
a) En una estación de trenes, asisten P personas que deben realizar una carga de su tarjeta SUBE
en la terminal disponible. La terminal es utilizada en forma exclusiva por cada persona de acuerdo
con el orden de llegada. Implemente una solución utilizando únicamente procesos Persona. Nota:
la función UsarTerminal() le permite cargar la SUBE en la terminal disponible.

```
bool libre = true
cola fila
sem mutex = 1
sem sig[N] = ([N] 0)

process Persona[id:0..N-1]{
    P(mutex)
    if (libre)
        libre = false
        V(mutex)
```

```
    else
        fila.push(id)
        V(mutex)
        P(sig[id])
    UsarTerminal()
    P(mutex)
    if (!fila.vacio())
        V[fila.pop()]
    else
        libre = true
    V(terminal)
}
b) Resuelva el mismo problema anterior pero ahora considerando q
Las personas realizan una única fila y la carga la realizan en l
Recuerde que sólo debe emplear procesos Persona. Nota: la funci
cargar la SUBE en la terminal t.
```

```
sem sig [N] = ([N] 0)
cola personasEspera [N]
personas[N]
sem mutex = 1
terminales [T]
cola idTerminales


process Persona[id:0..N-1]{
    P(mutex)
    if (!idTerminales.vacio()){
        personas[id] = idTerminales.pop()
        V(mutex)
    }
    else{
        personasEspera.push(id)
        V(mutex)
        P(sig[id])
```

```
    }



}
```

Monitores

Resolver el siguiente problema. En una elección estudiantil, se utiliza una máquina para voto
electrónico. Existen N Personas que votan y una Autoridad de Mesa que les da acceso a la máquina
de acuerdo con el orden de llegada, aunque ancianos y embarazadas tienen prioridad sobre el resto.
La máquina de voto sólo puede ser usada por una persona a la vez. Nota: la función Votar() permite
usar la máquina.

```
Monitor Mesa{
    cola personas
    cond cola[N]
    bool libre = true
    esperando = 0

    procedure pedir(edad: in int, embarazada: in boolean, id: in
        if (!libre){
            esperando++
            personas.insertarOrdenado(edad,embarazada,id)
            wait(cola[id])
        }
        else
            libre = false
    }
```

```
    procedure dejar(){
        if (esperando > 0)
            esperando--
            signal(cola[personas.pop()])
        else
            libre = true
    }

    procedure darMesa(){



}

process Persona[id: 0..N-1]{
    AutoridadMesa.pedir(id.edad,id.embarazada,id)
    Votar()
    AutoridadMesa.dejar()
}

process AutoridadMesa{
    for i:= 1 to N
        Mesa.darMesa()
}
```

SEMAFOROS

Implemente una solución para el siguiente problema. Un sistema debe validar un conjunto de 10000
transacciones que se encuentran disponibles en una estructura de datos. Para ello, el sistema dispone
de 7 workers, los cuales trabajan colaborativamente validando de a 1 transacción por vez cada uno.
Cada validación puede tomar un tiempo diferente y para realizarla los workers disponen de la
función Validar(t), la cual retorna como resultado un número entero entre 0 al 9. Al

finalizar el
procesamiento, el último worker en terminar debe informar la cantidad de transacciones por cada
resultado de la función de validación. Nota: maximizar la concurrencia

```
sem validador = 1
transacciones[10000]
cantTransacciones = 0
total  = 0
cantWorker = 0
process workers [id:0..6]{
    P(validador)
    cantWorker++
    while (cantTransacciones < 10000){
        total += Validar(transacciones.pop())
        cantTransacciones++
        V(validador)
        P(validador)
    }
    V(Validador)
    P(Validador)
    if (cantWorker == 7)
        print(total)
    V(Validador)
}
```

MONITORES

Resolver el siguiente problema. En una empresa trabajan 20 vendedores ambulantes que forman 5
equipos de 4 personas cada uno (cada vendedor conoce previamente a qué equipo pertenece). Cada equipo se encarga de vender un producto diferente. Las personas de un equipo se deben juntar antes de comenzar a trabajar. Luego cada integrante del equipo trabaja independientemente del resto vendiendo ejemplares del producto correspondiente. Al terminar cada integrante del grupo debe

conocer la cantidad de ejemplares vendidos por el grupo. Nota: maximizar la concurrencia.

```
Monitoe equipo[id:0..4]{
    wait equipoListo
    int cantTrabajos = 0
    int cant = 0

    procedure llegue(){
        cant++
        if (cant == 4)
            signal_all(equipoListo)
        else
            wait(equipoListo)
    }
    procedure sumarTrabajos(cant: in int)
        cantTrabajos += cant


process Vendedor[id:0..19]{
    int idEquipo
    int cantTrabajos = 0
    equipo[idEquipo].llegue()
    //trabajar(cantTrabajos)
    equipo[idEquipo].sumarTrabajos(cantTrabajos)
}
```

SEMAFORO

Implemente una solución para el siguiente problema. Se debe simular el uso de una máquina
expendedora de gaseosas con capacidad para 100 latas por parte de U usuarios. Además, existe un
repositor encargado de reponer las latas de la máquina. Los usuarios usan la

máquina según el orden
de llegada. Cuando les toca usarla, sacan una lata y luego se retiran. En el caso de que la máquina se
quede sin latas, entonces le debe avisar al repositor para que cargue nuevamente la máquina en forma
completa. Luego de la recarga, saca una botella y se retira. Nota: maximizar la concurrencia; mientras
se reponen las latas se debe permitir que otros usuarios puedan agregarse a la fila.

```
sem mutex  = 1
sem sig[U] = ([U] 0)
int latas = 100
sem noHayLata, hayLata = 0
cola espera
bool libre = true

process Usuario[id:0..U-1]{
    P(mutex)
    if (!libre)
        espera.push(id)
        V(mutex)
        P(sig[id])
    else
        libre = false
        V(mutex)

    if (latas > 0)
        //toma lata
        latas--
    else
        V(noHayLata)
        P(hayLata)
        //toma lata
        latas--
```

```
        """if (latas == 0)
            V(noHayLata)
            P(hayLata)
        //toma lata
        latas--"""

    P(mutex)
    if (!espera.vacia()){
        V(sig[espera.pop()])
    else
        libre = true
    V(mutex)
}

process Reponedor{
    while(true)
        P(noHayLata)
        //carga latas
        latas= 100
        V(hayLata)
}
```

Monitores

Resolver el siguiente problema. En una montaña hay 30 escaladores que en una parte de la subida
deben utilizar un único paso de a uno a la vez y de acuerdo con el orden de llegada al mismo. Nota:
sólo se pueden utilizar procesos que representen a los escaladores; cada escalador usa sólo una vez
el paso

```
Monitor Paso{
    cond miTurno
    libre = true
    esperando = 0

    procedure llegar(){
        if (!libre)
            esperando++
            wait(miTurno)
        else
            libre=false
    }

    procedure salir(){
        if (esperando > 0)
            esperando--
            signal(miTurno)
        else
            libre = true
    }

    process escalador[id:0..29]{
        Paso.llegar()
        //usa el paso
        Paso.salir()
    }
```

P3 5)

1. En un corralón de materiales se deben atender a N clientes de acuerdo con el orden de llegada.
   Cuando un cliente es llamado para ser atendido, entrega una lista con los productos que
   comprará, y espera a que alguno de los empleados le entregue el

comprobante de la compra
realizada.
a) Resuelva considerando que el corralón tiene un único empleado.

```
Monitor corralon{
    esperando = 0
    cond esperaEmpleado, esperaCliente

    procedure llegar(){
            esperando++
            signal(esperaCliente)
            wait(esperaEmpleado)
            esperando--
  }

    procedure atender(){
        if (esperando == 0)
            wait(esperaCliente)
        signal(esperaEmpleado)
    {

}

Monitor Escritorio{
    cond comprobante, lista, finComprobante
    listaProductos=null, comprobante=null text

    procedure darLista(lista: in text){
        listaProductos = lista
        signal(lista)
    {

    procedure hacerComprobante()
        if(listaProductos==null)
            wait(lista)
```

```
        comprobante= lohace(listaProductos)
        signal(comprobante)
        wait(finComprobante)

    procedure recibirComrpobante(c: out text)
        if(comprobante==null)
            wait(finComprobante)
        c = comprobante
        signal(finComprobante)
        listaProductos=null
        comprobante=null
}

process Cliente[id:0..N-1]{
    lista, comprobante text
    Corralon.llegar()
    Escritorio.darLista(lista)
    Escritorio.recibirComprobante(comprobante)
}

process empleado{
    for i := 1 to N
        Corralon.atender()
        Escritorio.hacerComprobante()
}
```

b) Resuelva considerando que el corralón tiene E empleados (E > 1). Los empleados no
deben terminar su ejecución

```
Monitor corralon{
    cola empleados
    esperando = 0
    eLibres = E
    cond esperaEmpleado, esperaCliente
```

```
    procedure espera(id : out int){
        if (eLibres > 0)
            eLibres--
        else{
            esperando++
            wait(esperaEmpleado)
        }
        id = empleados.pop()
    }

    procedure proximo(id: in int){
        empleados.push(id)
        if (esperando > 0){
            eLibres--
            esperando--
            signal (esperaCliente)
        }
        else
            eLibres++
    }

}
```

Hay C chicos y hay una bolsa con caramelos limitada a N caramelos administrada por UNA abuela. Cuando todos los chicos han llegado llaman a la abuela, y a partir de ese momento ella N veces selecciona un chico aleatoriamente y lo deja pasar a tomar un caramelo

```
sem sig[C] = ([C] 0)
sem mutex = 1
sem barrera = 0
int cant = 0
sem meToca = 0
sem avisaAbuela = 0
seguir = true
```

```
process chico[id:0..C-1]{
    P(mutex)
    cant++
    if (cant == N){
        for i := 1 to N:
            V(barrera)
        V(avisaAbuela)
    }
    V(mutex)
    P(barrera)
    P(sig[id])
    while (seguir){
        //tomo caramelo
        V(avisaAbuela)
        //comer caramelo
        P(sig[id])
    }


}

process abuela{
    P(avisaAbuela)
    for i := 1 to N
        V(sig[random()]
        P(avisaAbuela)
    seguir = false
    for i := 0 to C
        V(sig[i])
}
```

Se debe simular un partido de fútbol 11. Cuando los 22 jugadores llegaron a la cancha juegan durante 90 minutos y luego todos se retiran.

```
Monitor Cancha{
    int jugadores = 0
    cond inicio,esperaJugador

    procedure llegue(){
        jugadores++
        if (jugadores == 22)
            signal(inicio)
        wait(esperaJugador)
    }

    procedure inicio(){
        if (jugadores < 22)
            wait(inicio)
    }

    procedure fin(){
        signal_all(esperaJugador)
    }

process Jugador[id:0..21]{
    Cancha.llegue()
}

process Partido{
    Cancha.inicio()
    delay(90)
    Cancha.fin()
}
```

En una empresa de genética hay N clientes que envían secuencias de ADN para que sean analizadas y esperan los resultados para poder continuar. Para resolver estos análisis la empresa cuenta con 1 servidores que resuelve los pedidos de acuerdo al orden de llegada de los mismos.

```
sem mutex = 1
sem avisarSV = 0
cola clientes
vector adns [N]

process Cliente[id:0..N-1]{
    P(mutex)
    clientes.push(id)
    V(mutex)
    adns[id] = darSecuencia()
    V(avisarSV)
    P(sig[id])
}

process sv{
    id
    for i := 1 to N{
        P(mutex)
        P(avisarSV)

        adns[id] = aplicarResultado()
        V(sig[id])
    }
```

```
Monitor empresa{

    esperando = 0
    cond esperaCliente, esperaSV
    vector secuencias[N]
    cola clientes

    procedure llegue(id: in int, secuencia: in text ){
```

```
        esperando++
        secuencias[id] =secuencia
        clientes.push(id)
        signal(esperaCliente)
        wait(esperaSV)
    }

    procedure atender(sec : out text){
        if (esperando > 0)
            esperando--
        else
            wait(esperaCliente)
        sec = secuencias[clientes.pop()]
        signal(esperaSV)
    }

Monitor Terminal{

    secuencia text
    cond SecLista, SecEspera

    procedure esperarSecuencia(sec: out text){
        wait(secLista)
        sec = secuencia
        signal(SecEspera)
    }

    procedure cargarSecuencia(sec: in text){
        secuencia = sec
        signal(secLista)
        wait(SecEspera)
    }
}

process Cliente[id:0..N-1]{
    sec = generarSec()
```

```
    Empresa.llegue(id,sec)
    Terminal.esperarSecuencia(sec)
}

process sv{
    sec text
    while (true)
        Empresa.atender(sec)
        sec = aplicarFunciona(sec)
        Terminal.cargarSecuencia(sec)
```

En una empresa de genética hay N clientes que envían secuencias de ADN para que sean analizadas y esperan los resultados para poder continuar. Para resolver estos análisis la empresa cuenta con 2 servidores que van alternando su uso para no exigirlos de más (en todo momento uno está trabajando y el otro descansando); cada 5 horas cambia en servidor con el que se trabaja. El servidor que está trabajando, toma un pedido (de a uno de acuerdo al orden de llegada de los mismos), lo resuelve y devuelve el resultado al cliente correspondiente. Cuando terminan las 5 horas se intercambian los servidores que atienden los pedidos. Si al terminar las 5 horas el servidor se encuentre atendiendo un pedido, lo termina y luego se intercambian los servidores.

```
sem mutex  = 1

process Cliente[id:0..N-1]{
    P(mutex)
```

Monitores:

Resolver el siguiente problema con **MONITORES**. Simular la atención en una Salita Médica para vacunar contra el coronavirus. Hay ***UNA enfermera*** encargada de vacunar a ***30 pacientes***, cada paciente tiene un turno asignado (valor entero entre 1..30 ya conocido por el paciente). La enfermera atiende a los pacientes de acuerdo al turno que cada uno tiene asignado. Cada paciente al llegar espera a que sea su turno y se dirige al consultorio para que la enfermera lo vacune, y luego se retira. ***Nota:*** suponer que existe una función *Vacunar()* que simula la atención del paciente por parte de la enfermera. Todos los procesos deben terminar.

Para que quede más clara la explicación supondremos que los turnos son de 0..29

```
Monitor salita{

    cond cola[N]
    int esperando = 0
    cond llegue

    procedure llegue(id: in int){
        esperando++
        signal(llegue)
        wait(cola[id])

    }

    procedure atender(id: in int){
        if(esperando == 0)
            wait(llegue)
        signal(cola[id])
        esperando--
```

```
    }
}


process Paciente[id:0..29]{
        salita.llegue(id)
        Vacuna.sentarme()
}

process Enfermera{
    for i := 0 to 29
        salita.atender(i)
```

Resolver con **Semáforos** el siguiente problema. En una clínica hay un médico que debe atender a 20 pacientes de acuerdo al turno de cada uno de ellos *(no puede atender al paciente con turno i+1 si aún no atendió al que tiene turno i)*. Cada paciente ya conocen su turno al comenzar *(valor entero entre 0 y 19, o entre 1 y 20, como les resulte más cómodo)*, al llegar espera hasta que el médico lo llame para ser atendido, se dirige al consultorio y luego espera hasta que el médico lo termine de atender para retirarse. ***Nota:*** los únicos procesos que se pueden usar son los que representen a los pacientes y al médico; se debe asegurar que nunca haya más de un paciente en el consultorio; no se puede usar el ID del proceso como turno.

```
sem sig[20] = ([20] 0)

process Paciente[id:0..19]{

    int miTurno = darTurno()
    P(sig[miTurno])
    P(medico)
    //siendo atendido
    V(medico)
```

```
    V[sig[miTurno]
}

process Medico{
    turno = -1
    for i := 1 to 20
        turno++
        V(sig[turno])
        V(medico)
}
```

sem sig[20] = ([20] 0)

process Paciente[id:0..19]{

```
int miTurno = darTurno()
P(sig[miTurno])
P(medico)
//siendo atendido
V(medico)
V[sig[miTurno]
```

}

process Medico{
turno = -1
for i := 1 to 20
turno++
V(sig[turno])
V(medico)
}

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.

```
Monitor aula{

    cond esperaEnunciado, recibeEnunciado
    examen text
    alumnos = 0
    listo = false

    procedure llegue(){
        alumnos++
        if (alumnos == 45)
            signal(inicioReparto)
        wait(barrera)
    }

    procedure esperar45(){
        if (alumnos < 45)
            wait(inicioReparto)
        signall(barrera)
    }

    procedure darEnunciado(enunciado: in text){
        examen = enunciado
        listo = true
        signal(esperaEnunciado)
```

```
        wait(recibeEnunciado)
    }

    procedure recibirEnunciado(enun: out text){
        if (!listo)
            wait(esperaEnunciado)
        enun = examen
        listo = false
        signal(recibeEnunciado)
    }

Monitor Mesa{

    examenMesa text
    notaMesa int
    listo = false
    cond esperaExamen, esperaNota, reciboNota

    procedure entregaExamen(examen: in text, nota: out int){
        examenMesa = examen
        listo = true
        signal(esperaExamen)
        wait(esperaCorrecion)
        nota = notaMesa
        signal(reciboNota)
    }

    procedure agarrarExamen(examen: out text){
        if (!listo)
            wait(esperaExamen)
        examen = examenMesa
    }

    procedureDarNota(nota: in int){
        notaMesa = nota
        signal(esperaCorreccion)
```

```
        wait(reciboNota)
    }


process
```

Resolver el siguiente problema con ***SEMÁFOROS***. Simular la atención de una Planta Verificadora de Vehículos, donde se revisan cuestiones de seguridad de vehículos de a uno por vez. Hay ***N vehículos*** que deben ser verificados, donde algunos son autos y otros ambulancias. Antes de ser verificados, los autos deben hacer el pago correspondiente en la Caja de la Planta, donde le entregarán un recibo de pago. Las ambulancias no pagan. Los vehículos son atendidos de acuerdo al orden de llegada pero siempre dando prioridad a las ambulancias.

```
sem caja = 1
cola vehiculos
sem mutex = 1
bool libre = true
sem sig[N] = ([N] 0)
process Vehiculo[id:0..N-1]{
    boolean auto
    if (auto){
        P(caja)
        V(estoyCaja)
        //pagar
        V(Pague)
        P(Recibo)
    }
    P(mutex)
    if(!libre)
```

```
        cola.push(id,auto)
        P(sig[id]
        V(mutex)
    else
        libre = false
        V(mutex)
    //se atiende
    P(mutex)
    if (!vehiculos.vacio())
        V[vehiculos.pop()]
    else
        libre = true
    V(mutex)


process Caja{
    P(estoyCaja)
    P(pague)
    //hace recibo
    V(Recibo)
```

1. Resolver con MONITORES el siguiente problema. En un sistema operativo se ejecutan 20 procesos que

periódicamente realizan cierto cómputo mediante la función Procesar(). Los resultados de dicha función son
persistidos en un archivo, para lo que se requiere de acceso al subsistema de E/S. Sólo un proceso a la vez puede hacer
uso del subsistema de E/S, y el acceso al mismo se define por la prioridad del proceso (menor valor indica mayor
prioridad).

```
Monitor sistema{

    cola procesos
    cond meToca[N]
    esperando = 0
    bool libre = true

    procedure entrar(id: in int, prioridad: in int){
        if (!libre){
            esperando++
            procesos.push(id,prioridad)
            wait(meToca[id])
        else
            libre= false
    }

    procedure salir(){
        if (esperando > 0){
            signal(meToca[procesos.pop])
            esperando--
        else
            libre = true
    }

process proceso[id:0..19]{
```

Práctica de repaso – Memoria compartida

Semáforos

1. Resolver los problemas siguientes:
   a) En una estación de trenes, asisten P personas que deben realizar una carga de su tarjeta SUBE
   en la terminal disponible. La terminal es utilizada en forma exclusiva por cada persona de acuerdo
   con el orden de llegada. Implemente una solución utilizando únicamente procesos Persona. Nota:
   la función UsarTerminal() le permite cargar la SUBE en la terminal disponible.

```
cola personas;
sem permiso[P] = ([P] 0)
bool libre = true
sem mutex = 1

process persona [id:0..P-1]{
    P(mutex)
    if (libre){
        libre = false
        V(mutex)
    }
    else{
        personas.push(id)
        V(mutex)
        P(permiso[id])
    }
    UsarTerminal()
    P(mutex)
    if (personas.empty()){
        libre = true
```

```
    }
    else{
        V(permiso[personas.pop())
    }
    V(mutex)
{
```

b) Resuelva el mismo problema anterior pero ahora considerando que hay T terminales disponibles.
Las personas realizan una única fila y la carga la realizan en la primera terminal que se libera.
Recuerde que sólo debe emplear procesos Persona. Nota: la función UsarTerminal(t) le permite
cargar la SUBE en la terminal t.

```
sem mutex = 1
sem colaMutexTerminales = 1
int libres = T
sem permiso[P] = ([P] 0)
cola personas
cola colaTerminales // cargado con las terminales

process persona[id:0..P-1]{
    P(mutex)
    if (libres !== 0){
        libres--
        V(mutex)
    }
    else{
        personas.push(id)
```

```
        V(mutex)
        P(permiso[id])
    }
    P(colaMutexTerminales)
    terminal = colaTerminales.pop()
    V(colaMutexTerminañes)
    UsarTerminal(terminal)
    P(colaMutexTerminales)
    colaTerminales.push(terminal)
    V(colaMutexTerminales)
    P(mutex)
    if (personas.empty()){
        libres++
    }
    else{
        V(permiso[personas.pop()])
    }
    V(mutex)
```

1. Implemente una solución para el siguiente problema. Un sistema debe validar un conjunto de 10000
transacciones que se encuentran disponibles en una estructura de datos. Para ello, el sistema dispone
de 7 workers, los cuales trabajan colaborativamente validando de a 1 transacción por vez cada uno.
Cada validación puede tomar un tiempo diferente y para realizarla los workers disponen de la
función Validar(t), la cual retorna como resultado un número entero entre 0 al 9. Al finalizar el
procesamiento, el último worker en terminar debe informar la cantidad de transacciones por cada
resultado de la función de validación. Nota: maximizar la concurrencia.

```
transacciones colaint vectorContador[9] = ([9] 0);
Transacciones transacciones;
sem mutexTransacciones = 1; sem mutexContador[9] = ([9] 1); sem
int workersTerminados = 0;

Process Worker[a:1..7] {
    Transaccion transaccion;
    int resultado;
    P(mutexTransacciones);
    while (!transacciones.isEmpty()) {  // si hay transacciones,
        transaccion = transacciones.pop();
        V(mutexTransacciones);
        resultado = Validar(transaccion);
        P(mutexContador[resultado]);
        vectorContador[resultado]++;  // sumo 1 al contador del
        V(mutexContador[resultado]);
        P(mutexTransacciones);
    }
    V(mutexTransacciones);
    P(mutexWorker);
    workersTerminados++;  // aumento la cantidad de workers ter
    if (workersTerminados == 7) {  // si todos terminaron, impri
        for (int i = 0; i < 9; i++) {
            print(vectorContador[i]);
        }
    }
    V(mutexWorker);
}
```

1. Implemente una solución para el siguiente problema. Se debe simular el uso de una máquina
   expendedora de gaseosas con capacidad para 100 latas por parte de U usuarios. Además, existe un

repositor encargado de reponer las latas de la máquina. Los usuarios usan la máquina según el orden
de llegada. Cuando les toca usarla, sacan una lata y luego se retiran. En el caso de que la máquina se
quede sin latas, entonces le debe avisar al repositor para que cargue nuevamente la máquina en forma
completa. Luego de la recarga, saca una botella y se retira. Nota: maximizar la concurrencia; mientras
se reponen las latas se debe permitir que otros usuarios puedan agregarse a la fila.

Monitores

1. Resolver el siguiente problema. En una elección estudiantil, se utiliza una máquina para voto
electrónico. Existen N Personas que votan y una Autoridad de Mesa que les da acceso a la máquina
de acuerdo con el orden de llegada, aunque ancianos y embarazadas tienen prioridad sobre el resto.
La máquina de voto sólo puede ser usada por una persona a la vez. Nota: la función Votar() permite
usar la máquina.

```
Monitor mesa {
    cond colas[N], autoridad, fin
    cola personas

    procedure llegada (int id, int prioridad){
        personas.push(id,prioridad)
        signal(autoridad)
```

```
        wait(colas[id])
    }

    procedure salida(){
        signal (fin)
    }

    procedure siguiente(){
        if (personas.vacio()){
            wait(autoridad)
        }
        int id = personas.pop()
        signal(colas[id])
        wait (fin)
    }

}

process Persona [id:0.N-1]{
    int prioridad = obtenerPrioridad()
    mesa.llegada(id,prioridad)
    Votar()
    Mesa.salida()
}

process autoridad{
    for i := 1 to N{
        Mesa.siguiente()
    }
}
```

1. Resolver el siguiente problema. En una empresa trabajan 20 vendedores ambulantes que forman 5

equipos de 4 personas cada uno (cada vendedor conoce previamente a qué equipo pertenece). Cada
equipo se encarga de vender un producto diferente. Las personas de un equipo se deben juntar antesde comenzar a trabajar. Luego cada integrante del equipo trabaja independientemente del resto
vendiendo ejemplares del producto correspondiente. Al terminar cada integrante del grupo debe
conocer la cantidad de ejemplares vendidos por el grupo. Nota: maximizar la concurrencia.

```
Monitor equipo[id:0..3]{
    cond barrera
    int cantidad = 0
    int cantidadTotal = 0

    procedure iniciar(){
        cantidad++
        if (cantidad == 4)
            signal_all(barrera)
        else
            wait(barrera)
    }

    procedure finalizar(cantidadPorVendedor: in int; cantidadTot
        cantidadTotal+= cantidadPorVendedor
        cantidad--
        if (cantidad == 0)
            signal_all(barrera)
        else
            wait(barrera)
        cantidadTotalConocer = cantidadTotal
    }

}
```

```
process vendedor[id:0..19]{
    int idEquipo //conocido ya por vendedor
    int cantidadVendida
    int cantidadVendidaEquipo
    equipo[idEquipo].iniciar()
    //vender(cantidadVendida)
    equipo[idEquipo].finalizar(cantidadVendida,cantidadVendidaE
}
```

1. Resolver el siguiente problema. En una montaña hay 30 escaladores que en una parte de la subida
   deben utilizar un único paso de a uno a la vez y de acuerdo con el orden de llegada al mismo. Nota:
   sólo se pueden utilizar procesos que representen a los escaladores; cada escalador usa sólo una vez
   el paso

1. Resolver con *SEMÁFOROS* el siguiente problema. En una fábrica de muebles trabajan *50 empleados*. A llegar, los empleados forman 10 grupos de 5 personas cada uno, de acuerdo al orden de llegada (los 5 primeros en llegar forman el primer grupo, los 5 siguientes el segundo grupo, y así sucesivamente). Cuando un grupo se ha terminado de formar, todos sus integrantes se ponen a trabajar. Cada grupo debe armar M muebles (cada mueble es armado por un solo empleado); mientras haya muebles por armar en el grupo los empleados los irán resolviendo (cada mueble es armado por un solo empleado). *Nota:* Cada empleado puede tardar distinto tiempo en armar un mueble. Sólo se pueden usar los procesos *"Empleado"*, y todos deben terminar su ejecución. Maximizar la concurrencia.

```
int empleados = 0
int grupo = 1
sem sem_empleado = 1
sem grupos[10] = ([10] 0)
sem trabajar[10] = ([10] 1)
int cantidadMueblesGrupo[10] = ([10] 0)

process Empleado [id:0..49]{
    int miGrupo
    P(sem_empleado)
```

```
    empleados++
    miGrupo = grupo
    if (empleados = 5){
        for i:= 1 to 4
            V(grupos[miGrupo])
        empleado = 0
        grupo++
        V(sem_empleado)
    }
    else{
        V(sem_empleado)
        P(grupos[miGrupo])
    }
    P(trabajar[miGrupo])
    while (cantidadMueblesGrupo[miGrupo] < M){
        //trabajar()
        cantidadMueblesGrupo[miGrupo]++
        V(trabajar[miGrupo])
        P(trabajar[miGrupo])
    }
    V(trabajar[miGrupo])
}
```

3- Resolver el siguiente problema. En una montaña hay 30 escaladores que en una parte de la subida
deben utilizar un único paso de a uno a la vez y de acuerdo con el orden de llegada al mismo. Nota:
sólo se pueden utilizar procesos que representen a los escaladores; cada escalador usa sólo una vez
el paso

```
Monitor paso{
    int esperando = 0
```

```
    bool libre = true
    cond cola

    procedure pedir(){
        if (libre)
            libre = false
        else{
            esperando++
            wait(cola)
        }
    }

    procedure salir(){
        if (esperando > 0){
            esperando--
            signal(cola)
        }
        else
            libre = true
    }
}

process escalador [id:0..29]{
    paso.pedir()
    //usarPaso()
    paso.salir()
}
```

1. Resolver con *SEMÁFOROS* el siguiente problema. En una planta verificadora de vehículos, existen *7 estaciones* donde se dirigen 150 vehículos para ser verificados. Cuando un vehículo llega a la planta, el *coordinador* de la planta le indica a qué estación debe dirigirse. El coordinador selecciona la estación que tenga menos vehículos asignados en ese momento. Una vez que el vehículo sabe qué estación le fue asignada, se dirige a la misma y espera a que lo llamen para verificar. Luego de la revisión, la estación le entrega un comprobante que indica si pasó la revisión o no. Más allá del resultado, el vehículo se retira de la planta. *Nota:* maximizar la concurrencia.

```
cola vehiculos
sem mutex_cola = 1
sem autos_espera[150] = ([150] 0)
sem hayVehiculo = 0
sem estaciones_espera[7] = ([7] 0)
sem estaciones_colas[7] = ([7] 1)
int estaciones_asignadas[150]
text resultados_revisiones[150]
int cantidada_autos_estacion[7] = ([7] 0)
cola autos_estacion[7]
sem recalcularAutos[7] = ([7] 1)


process vehiculo [id:0..149]{
    int miEstacion
    text resultado
    P(mutex_cola]
    vehiculos.push(id)
    V(mutex_cola)
    V(hayVehiculo)
    P(autos_espera[id])
    miEstacion = estaciones_asignadas[id]
    P(estaciones_colas[miEstacion])
    autos_estacion[miEstacion].push(id)
    V(estaciones_colas[miEstacion])
    V(estaciones_espera[miEstacion])
    //se dirige a la estacion y espera revision
    P(autos_espera[id])
    resultado = resultados_revisiones[id]
    //se va


process cordinador{
    while (true]{
        int idVehiculo
```

```
        int idEstacionMinima
        P(hayvehiculo)
        P(mutex_cola)
        idVehiculo = vehiculos.pop()
        V(mutex_cola)
        idEstacionMinima = Min(cantidad_autos_estacion)
        estaciones_asignadas[idVehiculo] = idEstacionMinima
        P(recalcularAutos[idEstacionMinima])
        cantidad_autos_estacion[idEstacionMinima]++
        V(recalcularAutos[idEstacionMinima])

}


process estacion[id:0..6]{
    while (true){
        int idvVehiculo
        text resultado
        P(estaciones_espera[id])
        P(estaciones_colas[id])
        idVehiculo = autos_estacion[id].pop()
        V(estaciones_colas[id])
        resultado = //revisarauto(idVehiculo)
        resultados_revisiones[idVehiculo] = resultado
        V(autos_espera[idVehiculo])
        P(recalcularAutos[id])
        cantidad_autos_estacion[id]--
        V(recalcularAutos[id])
    }
```

2. Resolver con MONITORES el siguiente problema. En un sistema operativo se ejecutan *20 procesos* que periódicamente realizan cierto cómputo mediante la función *Procesar()*. Los resultados de dicha función son persistidos en un archivo, para lo que se requiere de acceso al subsistema de E/S. Sólo un proceso a la vez puede hacer uso del subsistema de E/S, y el acceso al mismo se define por la prioridad del proceso (menor valor indica mayor prioridad).

```
Monitor admin{
    cola procesos
    cond espera[20]
    bool libre = true

    procedure pedir(id: in int){
        if (libre){
            libre = false
        }
        else{
            procesos.insertarOrdenado(id) // inserta segun prio
            wait(espera[id])
        }
    }

    procedure salir(){
        if (!procesos.vacio()){
            int prox = procesos.pop()
            signal(espera[prox])
        }
        else
            libre = true
    }

}

process proceso[id:0..19]{
    admin.pedir(id)
    //procesar()
    admin.salir()
```

2. Resolver con MONITORES el siguiente problema: En una elección estudiantil, se utiliza una máquina para voto electrónico. Existen N Personas que votan y una Autoridad de Mesa que les da acceso a la máquina de acuerdo con el orden de llegada, aunque ancianos y embarazadas tienen prioridad sobre el resto. La máquina de voto sólo puede ser usada por una persona a la vez. Nota: la función Votar() permite usar la máquina.

```
Monitor maquina{
    cola personas
    cond espera[N]
    cond avisoAutoridad
    cond finUso
    boolean hayPersona=false

    procedure pedir(id: in int, edad: in int, embarazada: in boo
        personas.insertarOrdenadoPorPrioridad(id,edad,embarazada
        hayPersona=true
        signal(avisoAutoridad)
        wait(espera[id])
        hayPersona=false
    }

    procedure salir(){
        signal(finUso)
    }

    procedure siguiente(){
        if (!hayPersona)
            wait(avisoAutoridad)
        int idPersona = personas.pop()
        signal(espera[idPersona])
        wait(finUso)
    }
}
process persona[id:0..N-1]{
    int edad = //obtenerEdad()
```

```
    bool embarazada = //obtenerPrioridad()
    maquina.pedir(id,edad,prioridad)
    //usarMaquina()
    maquina.salir()
}

process autoridad{
    for i: 1 to N
        maquina.siguiente()
}
```

(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.

```
sem mutex_cola = 1
sem hayEmpleado = 0
int empleado_grupo[30]
int cantidadXGrupo[3] = ([3] 0)
int barreraXGrupo[3] = ([3] 0)
sem barreraGrupo[3] = ([3] 1)
sem irse[3] = ([3] 0)
sem grupos[3] = ([3] 1)
int producidos[30] = ([30] 0)
sem TerminoGrupo = 0
cola empleados

process empleado[id:0..29]{
    int miGrupo
    P(mutex_cola)
```

```
    empleados.push(id)
    V(mutex_cola)
    V(hayEmpleado)
    miGrupo = empleado_grupo[id]
    P(grupos[id])
    while (cantidadXGrupo[miGrupo] < 345){
        //Trabajar()
        cantidadXGrupo[miGrupo]++
        producidos[id]++
        V(grupos[id])
        P(grupos[id])
    }
    V(grupos[id])
    P(barreraGrupo[miGrupo]
    barreraXGrupo[miGrupo]++
    if (barreraXGrupo == 10){
        for i:= 1 to 9
            V(irse[miGrupo])
        V(Terminogrupo)
        V(barreraGrupo[miGrupo]
    }
    else{
        V(barreraGrupo[miGrupo]
        P(irse[miGrupo]
    }
}

process cordinador{
    int cant = 0
    int grupo = 1
    int max
    int idEmpleado
    while (cant < 30){
        P(hayEmpleado)
        P(mutex_cola)
        idEmpleado = empleados.pop()
```

```
        V(mutex_cola)
        if (cant == 10) or (cant == 20)
            grupo++
        empleado_grupo[idEmpleado] = grupo
        cant++
    }

    for i:= 1 to 3
        P(Terminogrupo)

    max = devolverCantidadMaxima(producidos)

    for i := 0 to 29
         if (producidos[i] == max){
             print(i)
}
```

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.

```
Monitor enfermera{
    int medicosLibres = 5
    cola medicos
    cond hayMedico

    procedure atender(idMedico:out int, gravedad:out int){
        if (medicosLibres = 0){
            wait(hayMedico)
        }
```

```
        idMedico = medicos.pop()
        gravedad = //obtenerGravedad()
        medicosLibres--
    }

    procedure darId(id:in int){
        medicos.push(id)
        medicosLibres++
        signal(hayMedico)
    }
}

Monitor escritorio[id:0..4]{
    cola pacientes
    cond esperaPaciente, finRevision
    cond espera[P]
    boolean hayPaciente = false

    procedure revision(id: in int, gravedad: in int){
        pacientes.push(id,gravedad) //inserta ordenado segun la
        hayPaciente = true
        signal(esperaPaciente)
        wait(espera[id])
    }

    procedure atenderPaciente(){
        if (!hayPaciente)
            wait(esperaPaciente)
        int prox = pacientes.pop()
        signal(espera[prox])
        wait(finRevision)
    }

    procedure salir(){
        signal(finRevision)
    }
```

```
}

process paciente[id:0..P-1]{
    int idMedico
    int gravedad
    enfermera.atender(idMedico,gravedad)
    escritorio[idMedico].revision(id,gravedad)
    //aplicaRevision()
    escritorio[idMedico].salir()

}

process medico [id:0..4]{
    while (true){
        enfermera.darId(id)
        escritorio[id].atenderPaciente()
    }
}
```

1. Resolver con *SEMÁFOROS* el siguiente problema. En un restorán trabajan *C cocineros* y *M mozos*. De forma repetida, los cocineros preparan un plato y lo dejan listo en la bandeja de platos terminados, mientras que los mozos toman los platos de esta bandeja para repartirlos entre los comensales. Tanto los cocineros como los mozos trabajan de a un plato por vez. Modele el funcionamiento del restorán considerando que la bandeja de platos listos puede almacenar hasta P platos. No es necesario modelar a los comensales ni que los procesos terminen.

```
cola bandeja[P]
sem hayPlato = 0
sem mutex_bandeja = 1
sem vacios = P

process cocinero[id:0..C-1]{
    while (true){
        P(vacios)
        plato unPlato= //prepararPlato
```

```
        P(mutex_bandeja)
        bandeja.push(unPlato)
        V(mutex_bandeja)
        V(hayPlato)
    }
}


}

process mozo[id:0..M-1]{
    while
    P(hayPlato)
    P(mutex_bandeja)
    plato = bandeja.pop()
    V(mutex_bandeja)
    V(vacios)
}
```

2. Resolver con MONITORES el siguiente problema. En una planta verificadora de vehículos existen *5 estaciones de verificación*. Hay *75 vehículos* que van para ser verificados, cada uno conoce el número de estación a la cual debe ir. Cada vehículo se dirige a la estación correspondiente y espera a que lo atiendan. Una vez que le entregan el comprobante de verificación, el vehículo se retira. Considere que en cada estación se atienden a los vehículos de acuerdo con el orden de llegada. ***Nota:*** maximizar la concurrencia.

```
Monitor puesto[id:0..4]{
    cond esperaAuto, hayComp, finComp
    text comprobante
    cola vehiculos
    int esperando = 0


    procedure pedir(id:in int, comp: out text){
        vehiculos.push(id)
```

```
        signal(esperaAuto)
        esperando++
        wait(hayComp)
        comp = comprobante
        signal(finComp)
    {

    procedure recibirId(idAuto: out int){
        if (esperando == 0){
            wait(esperaAuto)
        }
        idAuto = vehiculos.pop()
        esperando--
    }

    procedure darComprobante(c: in text){
        comprobante = c
        signal(hayComp)
        wait(finComp)
    }



process vehiculo[id:0..74]{
    int idPuesto = //obtenerEstacion()
    text comprobante
    puesto[idPuesto].pedir(id,comprobante)

}

process estacion[id:0..4]{
    int idAuto
    text comprobante
    while (true){
        puesto[id].recibirId(idAuto)
        comprobante = //generarComprobante(idAuto)
```

```
            puesto[id].darComprobante(comprobante)
        }
    }
```