Programación Concurrente - Practica 2

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

Bibliografía (1)

(1)

Consigna:
Existen N personas que deben ser chequeadas por un detector de metales antes de poder ingresar al avión.

a) Analice el problema y defina qué procesos, recursos y semáforos serán necesarios/convenientes, además de las posibles sincronizaciones requeridas para resolver el problema.

b) Implemente una solución que modele el acceso de las personas a un detector (es decir, si el detector está libre la persona lo puede utilizar; en caso contrario, debe esperar).

c) Modifique su solución para el caso que haya tres detectores.

Respuesta:

a) N personas (), Detector de metales (recurso), mutex (semaforos), exclusion mutua (sincronizacion)

b) .

```
sem mutex = 1
Process Persona[id:1..N]{
    Llega al detector
    P(mutex)
    Se detecta
    V(mutex)
    Sube al avión
}
```

c) .

```
sem mutex = 3
Process Persona[id:1..N]{
    Llega al detector
    P(mutex)
    Se detecta
    V(mutex)
    Sube al avión
}
```

(2)

Consigna:
Un sistema de control cuenta con 4 procesos que realizan chequeos en forma colaborativa. Para ello, reciben el historial de fallos del día anterior (por simplicidad, de tamaño N). De cada fallo, se conoce su número de identificación (ID) y su nivel de gravedad (0=bajo, 1=intermedio, 2=alto, 3=crítico). Resuelva considerando las siguientes situaciones:

a) Se debe imprimir en pantalla los ID de todos los errores críticos (no importa el orden).

b) Se debe calcular la cantidad de fallos por nivel de gravedad, debiendo quedar los resultados en un vector global.

c) Ídem b) pero cada proceso debe ocuparse de contar los fallos de un nivel de gravedad determinado.

Respuesta:

a)

b) .

```
(int,int) log[N]
P=3
Process Proceso[id:0..P-1]{
    parte=N/P
    extra=0
    inicio=id*parte
    if id=P-1 then extra= N mod P-1
    for (i=inicio,i<inicio+parte+extra-1;i++){

    }
}
```

c) .

```
(int,int) log[N]
Process Proceso[id:0..3]{
    for (i=0 to N-1){
        (id_log,gravedad)=log[i]
        if (id = gravedad) {
            gravedades[id]+=1;
        }
    }
}
```

---

(3)

Consigna:
Un sistema operativo mantiene 5 instancias de un recurso almacenadas en una cola. Además, existen P procesos que necesitan usar una instancia del recurso. Para eso, deben sacar la instancia de la cola antes de usarla. Una vez usada, la instancia debe ser encolada nuevamente.
Respuesta:

```
sem mutex = 1, espera = 5
cola T recursos[5]
Process Proceso[id:1..P]{
    while(true){
        P(espera)
        P(mutex)
            pop(recursos,rec)
        V(mutex)
        usar(rec)
        P(mutex)
            push(recursos,rec)
        V(mutex)
        V(espera)
    }
}
```

---

(4)

Consigna:
Suponga que existe una BD que puede ser accedida por 6 usuarios como máximo al mismo tiempo. Además, los usuarios se clasifican como usuarios de prioridad alta y usuarios de prioridad baja. Por último, la BD tiene la siguiente restricción:

- no puede haber más de 4 usuarios con prioridad alta al mismo tiempo usando la BD.
- no puede haber más de 5 usuarios con prioridad baja al mismo tiempo usando la BD.

Indique si la solución presentada es la más adecuada. Justifique la respuesta.

<table>
<tr><td colspan="2">Var
sem: semaphoro := 6;
alta: semaphoro := 4;
baja: semaphoro := 5;</td></tr>
<tr><td>Process Usuario-Alta [I:1..L]::
{ P (sem);
P (alta);
//usa la BD
V(sem);
V(alta);
}</td><td>Process Usuario-Baja [I:1..K]::
{ P (sem);
P (baja);
//usa la BD
V(sem);
V(baja);
}</td></tr>
</table>

Respuesta:
No, no es la mas adecuada. P(alta) y P(baja) deberian ir antes de P(sem), ya que evita demora innecesaria al preguntar si hay primero espacio para gente de alta/baja antes de reservar el espacio para entrar. Lo mismo ocurre con V(alta) y V(baja), deberian ir antes de V(sem) ya que causo demora innecesaria si libero un usuario antes de liberar alta o baja.

---

(5)

Consigna:
En una empresa de logística de paquetes existe una sala de contenedores donde se preparan las entregas. Cada contenedor puede almacenar un paquete y la sala cuenta con capacidad para N contenedores. Resuelva considerando las siguientes situaciones:

a) La empresa cuenta con 2 empleados: un empleado Preparador que se ocupa de preparar los paquetes y dejarlos en los contenedores; un empelado Entregador que se ocupa de tomar los paquetes de los contenedores y realizar la entregas. Tanto el Preparador como el Entregador trabajan de a un paquete por vez.

b) Modifique la solución a) para el caso en que haya P empleados Preparadores.

c) Modifique la solución a) para el caso en que haya E empleados Entregadores.

d) Modifique la solución a) para el caso en que haya P empleados Preparadores y E empleadores Entregadores.

Respuesta:

a) .

Ejercicio 5a

```
sem cantidad=N, sem hayPaquetes=0, sem mutex=1;
Paquete llenos[N]
int front=0,rear=0;
```

Preparador

```
Process Preparador{
    Paquete paquete
    while(true){
        prepara paquete
        P(cantidad)
        P(mutex)
        buffer[rear]=paquete
        V(mutex)
        rear=(rear + 1) mod N;
        V(hayPaquetes)
    }
}
```

Entregador

```
Process Entregador{
    Paquete paquete
    while(true){
        P(hayPaquetes)
        P(mutex)
        paquete=buffer[front]
        V(mutex)
        front=(front+1) mod N
        V(cantidad)
    }
}
```

b) .

Ejercicio 5b

```
sem cantidad=N, sem hayPaquetes=0, sem mutex=1;
Paquete llenos[N]
int front=0,rear=0;
```

Preparador

```
Process Preparador[id:0..P-1]{
    Paquete paquete
    while(true){
        prepara paquete
        P(cantidad)
        P(mutex)
        buffer[rear]=paquete
        rear=(rear + 1) mod N;
        V(mutex)
        V(hayPaquetes)
    }
}
```

Entregador

```
Process Entregador{
    Paquete paquete
    while(true){
        P(hayPaquetes)
        P(mutex)
        paquete=buffer[front]
        V(mutex)
        front=(front+1) mod N
        V(cantidad)
    }
}
```

c) .

Ejercicio 5c

```
sem cantidad=N, sem hayPaquetes=0, sem mutex=1;
Paquete llenos[N]
int front=0,rear=0;
```

Preparador

```
Process Preparador{
    Paquete paquete
    while(true){
        prepara paquete
        P(cantidad)
        P(mutex)
        buffer[rear]=paquete
        V(mutex)
        rear=(rear + 1) mod N;
        V(hayPaquetes)
    }
}
```

Entregador

```
Process Entregador[id:0..E-1]{
    Paquete paquete
    while(true){
        P(hayPaquetes)
        P(mutex)
        paquete=buffer[front]
        front=(front+1) mod N
        V(mutex)
        V(cantidad)
    }
}
```

d) .

Ejercicio 5d

```
sem cantidad=N, sem hayPaquetes=0, sem mutex=1;
Paquete llenos[N]
int front=0,rear=0;
```


Preparador

```
Process Preparador[id:0..P-1]{
    Paquete paquete
    while(true){
        prepara paquete
        P(cantidad)
        P(mutex)
        buffer[rear]=paquete
        rear=(rear + 1) mod N;
        V(mutex)
        V(hayPaquetes)
    }
}
```

Entregador

```
Process Entregador[id:0..E-1]{
    Paquete paquete
    while(true){
        P(hayPaquetes)
        P(mutex)
        paquete=buffer[front]
        front=(front+1) mod N
        V(mutex)
        V(cantidad)
    }
}
```

---

(6)

Consigna:
Existen N personas que deben imprimir un trabajo cada una. Resolver cada ítem usando semáforos:

a) Implemente una solución suponiendo que existe una única impresora compartida por todas las personas, y las mismas la deben usar de a una persona a la vez, sin importar el orden. Existe una función Imprimir(documento) llamada por la persona que simula el uso de la impresora. Sólo se deben usar los procesos que representan a las Personas.

b) Modifique la solución de (a) para el caso en que se deba respetar el orden de llegada.

c) Modifique la solución de (a) para el caso en que se deba respetar estrictamente el orden dado por el identificador del proceso (la persona X no puede usar la impresora hasta que no haya terminado de usarla la persona X-1).

d) Modifique la solución de (b) para el caso en que además hay un proceso Coordinador que le indica a cada persona que es su turno de usar la impresora.

e) Modificar la solución (d) para el caso en que sean 5 impresoras. El coordinador le indica a la persona cuando puede usar una impresora, y cual debe usar.

Respuesta:

Ejercicio a

```
sem mutex=1;
Process Persona[id:0..P-1]{
    P(mutex)
    Imprimir(documento)
    V(mutex)
}
```

Ejercicio b

```
sem mutex=1;
sem T espera[P]=([P] 0)
cola c;
boolean libre=true;
Process Persona[id:0..P-1]{
    int aux;
    P(mutex)
    if(libre) {libre=false; V(mutex)}
    else{
        push(c,id)
        V(mutex)
        P(espera[id])
    }
    Imprimir(documento)
    P(mutex)
    if(empty(c)) libre=true
    else { pop(c,aux)); V(espera[aux]) }
    V(mutex)
}
```

Ejercicio c

```
sem mutex=1;
sem T espera[P]=([P] 0)
boolean libre=true;
Process Persona[id:0..P-1]{
    if id != 0 {
        P(espera[id])
    }
    imprimir(documento)
    if id != P-1{
        V(espera[id+1]) // el 0 no entra de vuelta igual Preguntar
    }
}
```

Ejercicio d

```
sem coord=0, mutex=1, espera=([P], 0), termine = 1
cola T c[N];
```

Persona

```
Process Persona[id:0..P-1]{
    int aux;
    P(mutex)
    push(c,id)
    V(mutex)
    V(coord)
    P(espera[id])
    Imprimir(documento)
    V(termine)
}
```

Coordinador

```
int aux;
Process Coordinador{
    while(true){
        P(coord);
        P(mutex);
        pop(c,aux);
        V(mutex);
        P(termine)
        V(espera[aux]);
    }
}
```

Ejercicio e

```
sem coord=0, mutex=1, espera=([P], 0), termine = 5
cola T c[P]; cola Impresoras impresoras[5]; sem impMutex=1;
```

Persona

```
Process Persona[id:0..P-1]{
    Impresora imp;
    int aux;
    P(mutex)
    push(c,id)
    V(mutex)
    V(coord)
    P(espera[id])
    P(impMutex)
    pop(impresoras,imp)
    V(impMutex)
    Imprimir(documento,imp)
    P(impMutex)
    push(impresoras,imp)
    V(impMutex)
    V(termine)
}
```

Coordinador

```
int aux;
Process Coordinador{
    while(true){
        P(coord);
        P(mutex);
        pop(c,aux);
        V(mutex);
        P(termine)
        V(espera[aux]);
    }
}
```

(7)

Consigna:
Suponga que se tiene un curso con 50 alumnos. Cada alumno debe realizar una tarea y existen 10 enunciados posibles. Una vez que todos los alumnos eligieron su tarea, comienzan a realizarla. Cada vez que un alumno termina su tarea, le avisa al profesor y se queda esperando el puntaje del grupo, el cual está dado por todos aquellos que comparten el mismo enunciado. Cuando un grupo terminó, el profesor les otorga un puntaje que representa el orden en que se terminó esa tarea de las 10 posibles.

> **Nota:** Para elegir la tarea suponga que existe una función elegir que le asigna una tarea a un alumno (esta función asignará 10 tareas diferentes entre 50 alumnos, es decir, que 5 alumnos tendrán la tarea 1, otros 5 la tarea 2 y así sucesivamente para las 10 tareas).

Respuesta:

Ejercicio 7

```
int eligiocount = 50
A=50
sem eligio = 0, mutex = 1, barrera=0, terminoGrupo = 0, alus = 0, grupoMutex[10] = 1, corrigio[10]
    = 0, barreraGrupo[10] = 0, mutexColas = 1
int cantGrupo[10] = 5
cola colaGrupos,
```

Alumno

```
Process Alumno[id:0..A-1]{
    int idGrupo = elegir()
    P(mutex)
    eligiocount = eligiocount-1
    if eligiocount == 0 {
        V(barrera);
    }
    V(mutex)
    P(alus)
    hacer tarea
    P(grupoMutex[idGrupo])
    cantGrupo[idGrupo]=cantGrupo[idGrupo]-1;
    if (cantGrupo[idGrupo]== 0){
        V(grupoMutex[idGrupo])
        P(mutexColas)
        push(colaGrupos,idGrupo)
        V(mutexColas)
        V(terminoGrupo)
    }else{
        V(grupoMutex[idGrupo])
    }
    P(corrigio[idGrupo])
}
```

Profesor

```
Process Profesor{
    P(barrera)
    int orden=1;
    for i=1 to A do{
        V(alus)
    }
    for j=1 to 10 do{
        P(terminoGrupo)
        P(mutexColas)
        pop(colaGrupos,idGrupo)
        V(mutexColas)

        nota[idGrupo]=orden;
        orden++;

        for j=1 to 5 do {
            V(corrigio[idGrupo])
        }
    }
}
```

(7 alternativa barrera)

Ejercicio 7 alternativa

```
A=50
sem eligio = 0, mutex = 1, barrera=0, terminoGrupo = 0, alus = 0, grupoMutex[10] = 1, corrigio[10]
    = 0, barreraGrupo[10] = 0, mutexColas = 1
int cantGrupo[10] = 5
cola colaGrupos,
```

Alumno

```
Process Alumno[id:0..A-1]{
    int idGrupo = elegir()
    V(barrera)
    P(alus)
    hacer tarea
    P(grupoMutex[idGrupo])
    cantGrupo[idGrupo]=cantGrupo[idGrupo]-1;
    if (cantGrupo[idGrupo]== 0){
        P(mutexColas)
        push(colaGrupos,idGrupo)
        V(mutexColas)
        V(terminoGrupo)
    }
    V(grupoMutex[idGrupo]) // preguntar
    P(corrigio[idGrupo])
}
```

Profesor

```
Process Profesor{
    for i=1 to A do{
        P(barrera)
    }
    for i=1 to A do{
        V(alus)
    }
    int orden=1;
    for j=1 to 10 do{
        P(terminoGrupo)
        P(mutexColas)
        pop(colaGrupos,idGrupo)
        V(mutexColas)
        nota[idGrupo]=orden;
        orden++;
        for j=1 to 5 do {
            V(corrigio[idGrupo])
        }
    }
}
```

(8)

Consigna:
Una fábrica de piezas metálicas debe producir T piezas por día. Para eso, cuenta con E empleados que se ocupan de producir las piezas de a una por vez (se asume T>E). La fábrica empieza a producir una vez que todos los empleados llegaron. Mientras haya piezas por fabricar, los empleados tomarán una y la realizarán. Cada empleado puede tardar distinto tiempo en fabricar una pieza. Al finalizar el día, se le da un premio al empleado que más piezas fabricó.
Respuesta:

Ejercicio 8

```
int llegaron = 0, piezas[E] = ([E]0)
sem esperando = 0, mutex = 1
Process Empleado[id:0..E-1]{
    // llega a la fabrica
    bool ultimo;
    P(mutex)
    llegaron++;
    if (llegaron==E){
        for i=1 to E do { V(esperando) }
    }
    V(mutex)
    P(esperando)
    P(mutex)
    while(T>0)
    {
        T--;
        if(T==0) {ultimo==true}
        V(mutex)
        fabricar_pieza(materiales,mano_de_obra)
        piezas[id]++;
        P(mutex)
    }
    V(mutex);
    if (ultimo){
        dar(premio,max(piezas).pos) -- max devuelve pos y cantidad
        for i = 0 to E-1{ -- Sino el premiado puede ya haber finalizado
            V(salida)
        }
    }
    P(salida)
}
```

---

(9)

Consigna:
Resolver el funcionamiento en una fábrica de ventanas con 7 empleados (4 carpinteros, 1 vidriero y 2 armadores) que trabajan de la siguiente manera:

- Los carpinteros continuamente hacen marcos (cada marco es armando por un único carpintero) y los deja en un depósito con capacidad de almacenar 30 marcos.
- El vidriero continuamente hace vidrios y los deja en otro depósito con capacidad para 50 vidrios.
- Los armadores continuamente toman un marco y un vidrio (en ese orden) de los depósitos correspondientes y arman la ventana (cada ventana es armada por un único armador).

Respuesta:

Ejercicio 9

```
sem marcos = 30, vidrios = 50,
hayMarcos = 0, hayVidrios = 0
depoMarco = 1, depoVidrio = 1
cola<Marco> colaMarcos; cola<Vidrio> colaVidrios;
```

Carpintero

```
Process Carpintero[id:0..3]{
    while(true) {
        hace marco // preguntar adentro o afuera
        P(marcos)
        P(depoMarco)
        push(colaMarcos,marco);
        V(depoMarco)
        V(hayMarcos)
    }
}
```

Vidriero

```
Process Vidriero{
    while (true){
        hace vidrio // preguntar adentro o afuera
        P(vidrios)
        P(depoVidrio)
        push(colaVidrios,vidrio);
        V(depoVidrio)
        V(hayVidrios)
    }
}
```

Armador

```
Process Armador[id:0..1]{
    while (true){
        P(hayMarcos)
        P(depoMarco)
        pop(colaMarcos,marco)
        V(depoMarco)
        V(marcos)
        P(hayVidrios)
        P(depoVidrio)
        pop(colaVidrios,vidrio)
        V(depoVidrio)
        V(vidrios)
        arma ventana
    }
}
```

(10)

Consigna:
A una cerealera van T camiones a descargar trigo y M camiones a descargar maíz. Sólo hay lugar para que 7 camiones a la vez descarguen, pero no pueden ser más de 5 del mismo tipo de cereal. Nota: no usar un proceso extra que actué como coordinador, resolverlo entre los camiones.
Respuesta:

Ejercicio 10

```
sem trigo=5,maiz=5,lugares=7;
```

Camiones trigo

```
process CamionTrigo [id:0..T-1]{
    while(true){
        P(trigo)
        P(lugares)
        descargar(carga)
        V(trigo)
        V(lugares)
    }
}
```

Camiones maiz

```
process CamionMaiz [id:0..M-1]{
    while (true){
        P(maiz)
        P(lugares)
        descargar(carga)
        V(maiz)
        V(lugares)
    }
}
```

(11)

Consigna:
En un vacunatorio hay un empleado de salud para vacunar a 50 personas. El empleado de salud atiende a las personas de acuerdo con el orden de llegada y de a 5 personas a la vez. Es decir, que cuando está libre debe esperar a que haya al menos 5 personas esperando, luego vacuna a las 5 primeras personas, y al terminar las deja ir para esperar por otras 5. Cuando ha atendido a las 50 personas el empleado de salud se retira. Nota: todos los procesos deben terminar su ejecución; asegurarse de no realizar Busy Waiting; suponga que el empleado tienen una función VacunarPersona() que simula que el empleado está vacunando a UNA persona.
Respuesta:

Ejercicio 11

```
sem haycinco = 0;
int cant = 5, P = 5, idGrupo=0
cola grupo[10]
```

Persona

```
Process Persona[id:0..P-1]{
    P(mutex)
    cant--; // protege esto
    push(grupo[idGrupo],id)
    if(cant==0){
        idGrupo++;
        cant=5;
        V(haycinco)
    }
    V(mutex)
    P(vacuno[id])
    se va
}
```

Empleado

```
int listo[5];
int i,atendidos;
int idGrupoRevisado=0;
Process Empleado{
    for (atendidos = 0; atendidos < 10; atendidos++){
        P(haycinco)
        for i:=0 to 4 {
            listo[i]=pop(grupo[idGrupoRevisado])
            Vacunar(persona)
        }
        for i:=0 to 4{
            V(vacuno[listo[i]])
        }
        idGrupoRevisado++;
    }
    se va
}
```

Solucion alternativa

Persona

```
Process Persona[id:0..P-1]{
    P(mutexCantidad)
    idGrupo=cant/5
    cant++
    V(mutexCantidad)
    P(mutexCola[idGrupo])
    push(colaGrupos[idGrupo],id)
    V(mutexCola[idGrupo])
    V(barrera)
    P(vacuno[id])
    se va
}
```

Empleado

```
int listo[5];
int i;
Process Empleado{
    for (idGrupoRevisado = 0; idGrupoRevisado < 10;
    ↪ idGrupoRevisado++){
        for i:=0 to 4{
            P(barrera)
        }
        for i:=0 to 4 {
            listo[i]=pop(grupo[idGrupoRevisado])
            Vacunar(persona)
        }
        for i:=0 to 4{
            V(vacuno[listo[i]])
        }
    }
    se va
}
```

(12)

Consigna:
Simular la atención en una Terminal de Micros que posee 3 puestos para hisopar a 150 pasajeros. En cada puesto hay una Enfermera que atiende a los pasajeros de acuerdo con el orden de llegada al mismo. Cuando llega un pasajero se dirige al puesto que tenga menos gente esperando. Espera a que la enfermera correspondiente lo llame para hisoparlo, y luego se retira. Nota: sólo deben usar procesos Pasajero y Enfermera. Además, suponer que existe una función Hisopar() que simula la atención del pasajero por parte de la enfermera correspondiente.
Respuesta:

Ejercicio 12

```
int P = 150, hisopados = 0
cola<Persona> puestos[3]
sem espera[P] = ([P]0), hayPaciente[3] = 0
```

Pasajero

```
Process Pasajero[id:0..P-1]{
    int min;
    P(mutex)
    min = puestos.min(x -> x.size())
    for     P(mutexCola[min])
    push(id, puestos[min])
    V(mutexCola[min]) // consultar
    V(mutex)
    V(haypaciente[min])
    P(espera[id])
    se va
}
```

Enfermera

```
Process Enfermera[id:0..2]{
    P(hayPaciente[id])
    P(mutexHisopados)
    while(hisopados < 150){
        hisopados++
        V(mutexHisopados)
        P(mutexCola[id])
        pop(idpaciente,puestos[id])
        V(mutexCola[id])
        Hisopar(idpaciente)
        V(espera[idpaciente])
        if (hisopados==150) {
            primeroEnSalir=id;
        } else{
            P(hayPaciente[id])
        }
        P(mutexHisopados)
    }
    V(mutexhisopados)
    int siguiente=(id+1) mod 2
    if (siguiente != primeroEnSalir){
        V(hayPaciente[siguiente])
    }
}
```

Respuesta:

Ejercicio 12

```
int P = 150, hisopados = 0
cola<Persona> puestos[3]
int cant_puestos[3] = ([3]0)
sem espera[P] = ([P]0), hayPaciente[3] = 0
```

Pasajero

```
Process Pasajero[id:0..P-1]{
    int min, min_cant = -1;
    P(mutex)
    for i=0;i<3;i++{
        if cant_puestos[i]<min_cant{
            min_cant=cant_puestos[i]
            min=i
        }
    }
    P(mutexCola[min])
    push(id, puestos[min])
    V(mutexCola[min])
    cant_puestos[min]++
    V(mutex)
    V(haypaciente[min])
    P(espera[id])
    se va
}
```

Enfermera

```
Process Enfermera[id:0..2]{
    P(hayPaciente[id])
    P(mutexHisopados)
    while(hisopados < 150){
        hisopados++
        V(mutexHisopados)
        if (hisopados==150) {
            for(i=0 to 2){
                V(hayPaciente[i])
            }
        }
        P(mutexCola[id])
        pop(idpaciente,puestos[id])
        V(mutexCola[id])
        Hisopar(idpaciente)
        V(espera[idpaciente])
        P(hayPaciente[id])
        P(mutexHisopados)
    }
    V(mutexHisopados)
}
```

(Practica adicional)

(Ejercicio 3)

Consigna:
Se tiene un curso con 40 alumnos, la maestra entrega una tarea distinta a cada alumno, luego cada alumno realiza su tarea y se la entrega a la maestra para que la corrija, esta revisa la tarea y si está bien le avisa al alumno que puede irse, si la tarea está mal le indica los errores, el alumno corregirá esos errores y volverá a entregarle la tarea a la maestra para que realice la corrección nuevamente, esto se repite hasta que la tarea no tenga errores.
Respuesta:

Ejercicio 3

```
A=40
sem tarea[A]=([0] A), mutexCola=1,entrego=0,correcion ([0] A)
boolean estaBien[A]=([false] A)

```

Alumno

```
proccess Alumno [id: 0..A]{
    P(tarea[id])
    while(not estaBien[id]){
        hace tarea
        P(mutexCola)
        push(C,[tarea,id])
        V(mutexCola)
        V(entrego)
        P(correcion[id])
    }
}
```

Maestra

```
proccess Maestra {
    int correctos=0,id;
    boolean aprobado;
    Tarea tarea;
    for i to A{
        V(tarea[i])
    }
    while (correctos<A){
        P(entrego)
        P(mutexCola)
        tarea,id = pop(C)
        V(mutexCola)
        aprobado = corrijoTarea(tarea)
        if(aprobado){
            estaBien[id]=true
            correctos++
        }
        V(correcion[id])
    }
}
```

(Ejercicio 5)

Se debe simular una maratón con C corredores donde en la llegada hay UNA máquina expendedoras de agua con capacidad para 20 botellas. Además, existe un repositor encargado de reponer las botellas de la máquina. Cuando los C corredores han llegado al inicio comienza la carrera. Cuando un corredor termina la carrera se dirigen a la máquina expendedora, espera su turno (respetando el orden de llegada), saca una botella y se retira. Si encuentra la máquina sin botellas, le avisa al repositor para que cargue nuevamente la máquina con 20 botellas; espera a que se haga la recarga; saca una botella y se retira. Nota: maximizar la concurrencia; mientras se reponen las botellas se debe permitir que otros corredores se encolen.

Ejercicio 5

```
sem mutex=1,barrera[C]=([0] C), mutexCola=1,fill=20,botellita=20,avisarMenos=0, esperar[C]=([0] C)
Cola c;
```

Corredor

```
proccess Corredor [id: 0..C]{
    P(mutex)
    cantidad++;
    V(mutex)
    if cantidad==C{
        for i=0 to C-1{
            V(barrera[id]
        }
    }
    P(barrera[id])
    corre
    P(mutex)
    if(libre){
        libre=false;
        V(mutex);
    }else{
        P(mutexCola)
        push(cola,id)
        V(mutexCola)
        V(mutex)
        P(esperar[id])
    }
    P(fill)
    cantBotellitas--;
    V(avisarMenos)
    P(mutex)
    if(empty(c)) libre=true;
    else { aux=pop(c); V(esperar[aux]) }
    V(mutex)
}
```

Repositor

```
proccess Repositor {
    int cantCorredores=20
    int extra=-1
    int resto=C mod 20
    if (resto !=0){
        extra=0
    }
    int cantRefills=C/20+extra -- division de enteros
    for j=1 to cantRefills{
        if j=cantRefills and extra=0{
            cantCorredores=resto
        }
        for k=1 to cantCorredores{
            P(avisarMenos)
        }
        cantBotellitas=20;
        for m=1 to 20{
            V(fill)
        }

    }

}
```

(Ejercicio 6)

Consigna:
Una empresa de turismo posee 4 combis con capacidad para 25 personas cada una y UN vendedor que vende los pasajes a los clientes de acuerdo al orden de llegada. Hay C clientes que al llegar intentan comprar un pasaje para una combi en particular (el cliente conoce este dato); si aún hay lugar en la combi seleccionada se le da el pasaje y se dirige hacia la combi; en caso contrario se retira. Cada combi espera a que suban los 25 pasajeros, luego realiza el viaje, y cuando llega al destino deja bajar a todos los pasajeros. Nota: maximizar la concurrencia; suponga que para cada combi al menos 25 clientes intentarán comprar pasaje.
Respuesta:

Ejercicio 6

```
sem mutexCola=1,hayGenteEncolada=0,
sem esperar[C]=([0] C)
sem cantidadEnCombi[4]=([0] 4), cantidadEnCombi[4]=([0] 4)
Cola colaCompra;
boolean hayLugar[C]=([false] C)
```

Combi

```
proccess Combi[id:0..3]{
    for i=1 to 25{
        P(cantidadEnCombi[id])
    }
    viaja
    for i=1 to 25{
        V(esperarLlegada[id])
    }
}
```

Vendedor

```
proccess Vendedor{
    int cantidadPasajes[4]=([25] 4)
    for i=1 to C{
        P(hayGenteEncolada)
        P(mutexCola)
        id,idCombi = pop(colaCompra)
        V(mutexCola)
        if cantidadPasajes[idCombi] > 0{
            cantidadPasajes[idCombi]--
            hayLugar[id]=true;
        }
        V(esperar[id])
    }
}
```

Cliente

```
proccess Cliente[id:0..C-1]{
    int idCombi=elegirCombi()
    P(mutexCola)
    push(colaCompra,(id,idCombi))
    V(mutexCola)
    V(hayGenteEncolada)
    P(esperar[id])
    if (hayLugar[id]){
        V(cantidadEnCombi[idcombi])
        P(esperarLlegada[idcombi])
    }
}
```

(Ejercicio 7)

En una herrería hay 15 empleados que forman 5 grupos de 3 personas; los grupos se forman de acuerdo al orden de llegada (los 3 primeros pertenecen al grupo 1, los 3 siguientes al grupo 2, y así sucesivamente). Ni bien conoce el grupo al que pertenece el empleado comienza a trabajar (no debe esperar al resto de grupo para comenzar). Cada grupo debe hacer exactamente P unidades de un producto (cada unidad es hecha por un único empleado). Al terminar de hacer las P unidades de un grupo, sus 3 empleados se retiran. Nota: maximizar la concurrencia; ningún grupo puede hacer unidades de más.

Proceso Empleados

```
process Empleados[id:0..14]{
    P(mutexCantidad)
    miGrupo=cant/3
    cant++
    V(mutexCantidad)
    P(mutexUnidades[miGrupo])
    while(cantUnidades[miGrupo]<P){
        cantUnidades[miGrupo]++
        V(mutexUnidades[miGrupo])
        -- realiza unidad
        P(mutexUnidades[miGrupo])
    }
    V(mutexUnidades[miGrupo])
}
```

Proceso Empleados

```
process Empleados[id:0..14]{
    Entrada.llegar(miGrupo)
    estaLleno=false
    while(not estaLleno){
        Produccion[miGrupo].producir(estaLleno)
        if (not estaLleno){
            -- producir
        }
    }
}
```

Monitor Produccion

```
Monitor Produccion[id:0..4]{
    int cantProducidos=0
    Procedure producir(out bool estaLleno){
        cantProducidos[miGrupo]++
        estaLleno=cantProducidos==P
    }
}
```

(Ejercicio 12)

En una empresa de software hay 3 programadores que deben arreglar errores informados por N clientes. Los clientes continuamente están trabajando, y cuando encuentran un error envían un reporte a la empresa para que lo corrija (no tienen que esperar a que se resuelva). Los programadores resuelven los reclamos de acuerdo al orden de llegada, y si no hay reclamos pendientes trabajan durante una hora en otros programas. Nota: los procesos no deben terminar (trabajan en un loop infinito); suponga que hay una función ResolverError que simula que un programador está resolviendo un reporte de un cliente, y otra Programar que simula que está trabajando en otro programa.

Proceso Cliente

```
process Cliente[id:0..N-1]{
    while(true){
        P(mutexProblemas)
        push(colaProblemas,problema)
        V(mutexProblemas)
    }
}
```

Proceso Programador

```
process Programador[id:0..2]{
    while(true){
        P(mutexProblemas)
        if(empty(colaProblemas)){
            V(mutexProblemas)
            Programa()
        }else{
            problema=pop(colaProblemas)
            V(mutexProblemas)
            ResolverError(problema)
        }
    }
}
```

Bibliografía

[1] G. R. Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming*. 1900 E Lake Ave Glenview, IL 60025 United States: Addison-Wesley, 1999.