

## P2 – Semáforos

### CONSIDERACIONES PARA RESOLVER LOS EJERCICIOS:

- Los semáforos deben estar declarados en todos los ejercicios.
- Los semáforos deben estar inicializados en todos los ejercicios.
- No se puede utilizar ninguna sentencia para setear o ver el valor de un semáforo.
- Debe evitarse hacer busy waiting en todos los ejercicios.
- En todos los ejercicios el tiempo debe representarse con la función delay.

**1. 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.**

sem libre:=1;

Process persona [0..N-1]

```
P(libre);           //espera que este desocupado y marca como ocupado
//chequearse con el detector
V (libre);          //Marca desocupado
```

End;

**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).**

**//ES SIMILAR AL "ESCALADOR" DE LA EXPLICACION, PERO ESA NO TIENE ORDEN, QUE DEBERIA CAMBIARLE?**

Sem mutex:=1;

Bolean libre:=true;

Cola c;

Array Esperar := ([N] 0)

Process persona [0..N-1]

```
Int aux;

P (mutex);
If libre=true then    //si esta libre el detector, tomarlo, desproteger la variable
    libre = false;
    V(mutex);
End;
Else                  //sino se encola, desbloquea la variable, espera su turno
    Encolar (c, id);
    V(mutex);
    P (espera [id]);
End;
//chequearse con el detector
P (mutex);
If (c.estavacia()) then //si cuando termina de pasar la cola esta vacía, queda libre
    Libre:=true;
Else
    Desencolar (c, aux); //si hay alguien en la cola, le da el turno
    V (espera [aux]);
V (mutex);
End;
```

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

Sem libre:= 3;

Process Persona [0..N-1]

```
P (libre); //esto resta un detector, pero deja pasar si es >0
//chequearse con el detector
V (libre); //esto suma un detector (Cuando la persona paso)
```

End;

**2. 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).**

Cola Historial\_de\_fallos =(id, gravedad)

Int n:= historial\_de\_fallos.lenght()

Cant:=0;

Sem mutex=1;

Sem colita=1;

Process chequeo [id: 1..4]

P(mutex);

While (cant<n) do begin

Cant:=cant+1;

V (mutex);

P(colita);

Desencolar (Historial\_de\_fallos , id, gravedad);

V (colita);

If (gravedad==3) then

Print (id, gravedad);

End;

P(mutex);

V (mutex);

End;

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

Cola Historial\_de\_fallos =(id, gravedad)

Int n:= historial\_de\_fallos.lenght()

Cant:=0;

Sem mutex=1; //protege cantidad y cola

Sem Cont= 1;

Contador [4]=[4] 0)

Process chequeo [id: 1..4]

P(mutex);

While (cant<n) do begin

Cant:=cant+1;

Desencolar (Historial\_de\_fallos , id, gravedad);

V (mutex);

P(cont);

Contador [gravedad]:= Contador [gravedad]+1;

P(cont);

P(mutex);

V (mutex);

```

    For i:=0 to 3 do
        Print ("Fallos de gravedad", i, ":", Contador[i]);
    End;

```

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

```

Historial_de_fallos [n]=(id, gravedad)
Cant:=0;
Sem mutex=1;
Sem array=1;
Sem Cont= 1;

```

```

Contador [4]=([4] 0)

```

```

Process chequeo [id: 0..3]
    i:=0;    //para que recorra todo el array buscando los que le corresponden mas rapido
    P(mutex);
    While (cant<n) and (i<n) do begin
        If (id=historial_de_fallos[i]) then //aca compara si es el que le corresponde
            Cant:=cant+1;    //si lo es, ++ a la compartida
            V (mutex);
            i:=i+1;    //para avanzar
            Print (id, gravedad);
            P (cant);
            Contador [id]:= Contador [id]+1;
            P (cant);
        Else
            V (mutex);
            i:=i+1;
        P(mutex);
    V (mutex);
    Print ("Fallos de gravedad", id, ":", Contador[id]);
End;

```

**Cant no podría omitirse?**

**3. 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.**

```

Sem libre:= 5;    //controla cant de recursos libres
Sem scola:=1;    //controla la cola
Cola c;

```

```

Process proceso [id: 0..n-1]
    Recurso aux;

    P(libre);    //espera que haya un recurso disponible (sem libre>0)
    P (scola);    //protege la cola, toma un recurso, desbloquea la cola
    Aux:= desencolar (c, aux);
    V (scola);
    //usar recurso
    P (scola);    //protege la cola, devuelve el recurso
    Encolar (c, aux);
    V (scola);    //desbloquea la cola, incrementa los recursos libres
    V (libre);    //EN ESTOS DOS V() AFECTA EL ORDEN?

```

End;

4. 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.

<b>Var</b>  sem: semaphoro := 6; alta: semaphoro := 4; baja: semaphoro := 5;	
<b>Process Usuario-Alta [I:1..L]::</b>  { P (sem); <i>//sem-1 (puede pasar uno menos de sem)</i> P (alta); <i>//alta-1 (puede pasar uno menos de alta)</i> <i>//usa la BD</i> V(sem); <i>//libera uno de sem</i> V(alta); <i>//libera uno de alta</i> }	<b>Process Usuario-Baja [I:1..K]::</b>  { P (sem); P (baja); <i>//usa la BD</i> V(sem); V(baja); }

La solución propuesta no es válida puesto que, en ambos casos se empieza a bloquear la cantidad total primeramente (sem) y luego la particular (Alta o baja). Esto genera que, si un usuario logra pasar P(sem) pero se queda bloqueado en su semáforo respectivo siguiente, ya restó un acceso sobre los seis totales sin poder entrar realmente a usar la BD, pero quitándole a otro eventual proceso del tipo contrario la posibilidad de hacerlo. Se debería invertir el orden de los P() iniciales para evitar esto.

Sem sem:= 6;               //total

Sem alta:= 4;

Sem baja:= 5;

Process Usuario-Alta [I:1..L]

  P(alta);

  P(sem);

*//usa la BD;*

  V(sem);

  V(alta);

*//SI SE LIBERA PRIMERO SEM Y DESP ALTA/BAJA, NO LE DA PRIORIDAD A LOS USUARIOS DEL TIPO OPUESTO PARA ENTRAR?*

End;

Process Usuario-Baja [I:1..K]

  P(baja);

  P(sem);

*//usa la BD;*

  V(sem);

  V(baja);

End;

5. 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 empleado 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.

```

Array contenedor [1..n];
Int ocupado:=0;           //donde el entregador tiene que dejar elem
Int libre:=0;             //donde el preparador tiene que dejar elemento
Sem vacio:=n;             //inicia con el numero de contenedores
Sem lleno:=0;

Process Preparador:      //tiene que ver si hay pos libres para depositar
    While (true)
        //prepara el paquete
        P (vacio);        //espera que haya lugar y deja algo
        Contenedor[libre].agregar (paquete);
        Libre:=(libre+1) mod n; //indica el prox libre
        V (lleno);        //marca 1 más en "lleno" para el entregador
    End;
End;

```

```

Process Entregador:
    While (true)
        P(lleno);          //espera que haya algo en el contenedor
        Contenedor[ocupado].retirar(paquete);
        Ocupado:=(ocupado+1)mod n;
        P(Vacio);          //indica que está vacío
        //Entregar paquete
    End;
End;

```

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

```

Array contenedor [1..n];
Int ocupado:=0;           //donde el entregador tiene que dejar elem
Int libre:=0;             //donde el preparador tiene que dejar elemento
Sem vacio:=n;             //inicia con el numero de contenedores
Sem lleno:=0;
Sem mutexP:=1;           //Para preparar

Process Preparador [id=1..P]
    While (true)
        //prepara el paquete
        P (vacio);        //espera que haya lugar y deja algo
        P(MutexP);        //Para bloquearle a otros preparadores
        Contenedor[libre].agregar (paquete);
        Libre:=(libre+1) mod n; //indica el prox libre
        V (MutexP);       //Para desbloquearle a otros preparadores
        V (lleno);        //marca uno más "lleno" para el entregador
    End;
End;

Process Entregador:
    While (true)
        P(lleno);          //espera que haya algo en el contenedor
        Contenedor[ocupado].retirar(paquete);
        Ocupado:=(ocupado+1)mod n;
        P(Vacio);          //indica que está vacío
    End;
End;

```

```

        //Entregar paquete
    End;
End

```

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

```

Array contenedor [1..n];
Int ocupado:=0;      //donde el entregador tiene que dejar elem
Int libre:=0;        //donde el preparador tiene que dejar elemento
Sem vacio:=n;        //inicia con el numero de contenedores
Sem lleno:=0;
Sem mutexE:=1;       //Este es para entregar

Process Preparador:  //tiene que ver si hay pos libres para depositar
    While (true)
        //prepara el paquete
        P (vacio);    //espera que haya lugar y deja algo
        Contendor[libre].agregar (paquete);
        Libre:=(libre+1) mod n; //indica el prox libre
        V (lleno);    //marca 1 más en "lleno" para el entregador
    End;
End;

Process Entregador [id=1..E]
    While (true)
        P(lleno);    //espera que haya algo en el contenedor
        P(MutexE);
        Contenedor[ocupado].retirar(paquete);
        Ocupado:=(ocupado+1)mod n;
        V(MutexE);
        P(Vacio);    //indica que está vacío
        //realizar entrega
    End;
End;

```

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

```

Array contenedor [1..n];
Int ocupado:=0;      //donde el entregador tiene que dejar elem
Int libre:=0;        //donde el preparador tiene que dejar elemento
Sem vacio:=n;        //inicia con el numero de contenedores
Sem lleno:=0;
Sem mutexP:=1;       //Estos son para preparar y entregar
Sem mutexE:=1;

Process Preparador [id=1..P]
    While (true)
        //prepara el paquete
        P (vacio);    //espera que haya lugar y deja algo
        P(MutexP);    //Para bloquearle a otros preparadores
        Contendor[libre].agregar (paquete);
        Libre:=(libre+1) mod n; //indica el prox libre
        V (MutexP);    //Para desbloquearle a otros preparadores
        V (lleno);    //marca uno más "lleno" para el entregador
    End;

```

```

End;
End;

Process Entregador [id=1..E]
While (true)
    P(lleno);                //espera que haya algo en el contenedor
    P(MutexE);
    Contenedor[ocupado].retirar(paquete);
    Ocupado:=(ocupado+1)mod n;
    V(MutexE);
    P(Vacio);                //indica que está vacío
    //realizar entrega
End;
End;

```

**6. 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.**

sem mutex = 1;

```

Process Persona [id: 0..N-1]
    P(mutex);
    //imprimirdocumento()
    V (mutex);
End;

```

**//ES SIMILAR AL “ESCALADOR” DE LA EXPLICACION, PERO ESA NO TIENE ORDEN, QUE DEBERIA CAMBIARLE?**

```

cola c;
sem mutex = 1;
sem Espera [N] = ([N] 0);
boolean libre:= true;

```

```

Process Persona [id: 0..N-1]
int sig;

    P(mutex);
    If (libre) then
        Libre:=false;
        V(mutex);
    End;
    Else
        encolar(c, id);
        V(mutex);
        P(espera[id]);
    End;
//Imprimir documento
    P(mutex);
    if (cola.isempty ()) then
        libre:=true;
    else
        desencolar(c, aux);
        V(espera[id]);
    End;

```

```
V(mutex);
End;
```

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

```
ColaOrdenada c;
sem mutex = 1;
sem Espera [N] = ([N] 0);
boolean libre:= true;
```

```
Process Persona [id: 0..N-1]
int sig;
    P(mutex);
    If (libre) then
        Libre:=false;
        V(mutex);
    End;
    Else
        encolar(c, id);
        V(mutex);
        P(espera[id]);
    End;
    //Imprimir documento
    P(mutex);
    if (cola.isempty ()) then
        libre:=true;
    else
        desencolar(c, id);
        V(espera[id]);
    V(mutex);
End;
```

**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.**

```
Cola c;
Sem mutex:= 1;
Sem espera_persona = ([N] 0)
Sem Listo=0;
```

```
Process Persona [id 1..n]
    P(mutex);           //se encola
    Encolar (c, id);
    V (mutex);
    V (Espera_coordinador); //le avisa que hay alguien en la cola
    P (espera_personas [id]); //espera su turno
    //imprimir documento
    V (listo);           //avisa al coordinador que termino
End;
```

```
Process Coordinador
    Int sig
    For i:=1 to n do begin
```

```

P(espera_coordinador);      //espera que haya alguien en la cola
P(mutex);                  //desencola uno y lo habilita
sig:= [desencolar(c, id)];
V (mutex);
V (espera_persona[sig]);
P (listo);                  //espera que el proceso que mando a imprimir termine para mandar otro
End;
End;

```

**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.**

```

Cola c;
Sem mutex:= 1;
Sem espera_persona = ([N] 0)
Sem impre:=5;
Sem Listo=0;
Impresora [N]
Cola colaimpresoras=1, 2, 3, 4, 5;

Process Persona [id 1..n]
    P(mutex);                //se encola
    Encolar (c, id);
    V (mutex);
    V (Espera_coordinador);  //le avisa que hay alguien en la cola
    P (espera_personas [id]); //espera su turno
    //imprimirdocumentoen (impresora [id]); //toma la que le dio
    V (listo);               //avisa al coordinador que termino
End;

Process Coordinador
    Int sig
    For i:=1 to n do begin
        P(espera_coordinador); //espera que haya alguien en la cola
        P(mutex);              //desencola uno y lo habilita
        sig:= [desencolar(c, id)];
        V (mutex);
        P(colaimpre);
        impresora [id]=colaimpresora.desencolar()
        P(colaimpre);
        V (espera_persona[sig]);
        P (listo);              //espera que el proceso que mando a imprimir termine para mandar otro
        P(colaimpre);
        //encolar.colaimpresoras(impresora[id]); //la vuelve a dejar
        P(colaimpre);
    End;
End;

```

**7. 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).**

```

array puntajeGrupo [10]; //Se guarda el puntaje de cada grupo
array tareaAlumno [50]; //Para saber qué tarea (grupo) le tocó a cada alumno
sem eligiotarea=0;
sem terminoTarea = 0;
sem esperarGrupo [10] = ([10] 0);
sem mutex = 1;
cola de int colaAlumnos; //Cola de ids de alumnos que fueron entregando la tarea

```

```

Process Alumno [id: 0..49]
    Int tarea = elegir ();                //Recibe su tarea asignada
    tareaAlumno [id]:= tarea;
    V(eligiotarea);                        //Le avisa al coordinador que tiene una tarea elegida
    P(barrera);
    //hace su tarea
    P(mutex);                             //Se encola
    encolar(colaAlumnos, id);
    V(mutex);
    V(terminoTarea);                      //Avisa que termino la tarea
    P(esperarGrupo[tarea]);              //Espera que todo el grupo haya terminado
    //Ve su nota;
end;

```

```

Process Profesor
    int puntaje = 1;                      //1 para el grupo que termino 1º, 2 para el 2º...
    int eligentarea=0;
    array tareasTerminadas [10] = ([10] 0);

    while(eligenTarea < 50)do begin
        P(eligioTarea);                  //Un alumno lo habilita cuando elige tarea
        eligenTarea = eligenTarea + 1;
    end
    for i= 1..50 do begin
        V(barrera); //Ya todos eligieron tarea. El profesor los despierta
    End;
    while(puntaje > 0) do begin           //Cuando ya todas se hayan corregido
        P(terminoTarea);                 //espera a que un alumno termine la tarea
        P(mutex);                        //Saca a un alumno de la cola
        int alumno = desencolar(colaAlumnos, id);
        V(mutex);
        Int idTarea = tareaAlumno [alumno];
        tareasTerminadas[idTarea] = tareasTerminadas[idTarea] + 1;
        if(tareasTerminadas[idTarea] == 5) then    //Si todos terminaron
            puntajeGrupo[idTarea] = puntaje;      //Le asigna el puntaje
            for i= 1 to 5:                         //Para que vean su nota
                V(esperarGrupo[idTarea]);
            End;
            puntaje = puntaje + 1;                 //Incrementa el puntaje para el siguiente
        end;
    end;
end;

```

**8. 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.**

```

int cantLlegaron = 0;
int cantTareas = T;
int tareasRealizadas [E] = ([E] 0); //lleva el conteo de tareas realizadas por empleado
sem mutexContador = 1;
sem mutexTareas = 1;
sem mutex_cont2 = 1;
sem barrera = 0;
sem empleadoTerminó = 0;
Cola colaTareas;           //cola de tareas(se asume que ya estan cargadas)
Int ganador;

Process Empleado [id: 0..E-1]::
    P(mutexContador);
    cantLlegaron = cantLlegaron + 1;
    if(cantLlegados == E) then           //si llegaron todos, el ultimo los despierta
        for i:= 1 to E do
            V(barrera);
        V(mutexContador);

    P(barrera);           //espera a que lleguen los demás empleados
    P(mutexTareas);
    while(cantTareas > 0) do begin
        tarea = desencolar(colaTareas);           //agarra una tarea y decrementa
        cantTareas = cantTareas - 1;
        V(mutexTareas);
        //realizarTarea;
        tareasRealizadas[id] = tareasRealizadas[id] + 1;           //incrementa su contador de tareas
        P(mutexTareas);           //vuelve a bloquear tareas

    End;
    V(mutexTareas);           //para que los demás puedan entrar al while y ver que no hay tareas(?)

    P(Mutex_cont2);
    Cant_terminaron:=cant_terminaron+1;
    If (cant_terminaron=E) then           //si todos terminaron
        Ganador:=tareasrealizadas.max().pos();           //posición (id) del que hizo mas tareas
        For i:=1 to E do
            V(barrera);
        Entregarpremioa(ganador);
    V(mutex_cont2);
    P(barrera);
    If (id==ganador) then
        //recibirpremio
End;

*_*_*_*_*_*_*_*_
int cantLlegaron = 0;
int cantTareas = T;
int tareasRealizadas [E] = ([E] 0); //lleva el conteo de tareas realizadas por empleado
sem mutexContador = 1;
sem mutexTareas = 1;
sem barrera = 0;
sem empleadoTerminó = 0;

```

```

Cola colaTareas;          //cola de tareas(se asume que ya estan cargadas)

Process Empleado [id: 0..E-1]::
    P(mutexContador);
    cantLlegaron = cantLlegaron + 1;
    if(cantLlegados == E) then          //si llegaron todos, el ultimo los despierta
        for i:= 1 to E do
            V(barrera);
        V(mutexContador);

        P(barrera);          //espera a que lleguen los demás empleados
        P(mutexTareas);
        while(cantTareas > 0) do begin
            tarea = desencolar(colaTareas);          //agarra una tarea y decrementa
            cantTareas = cantTareas - 1;
            V(mutexTareas);
            //realizarTarea;
            tareasRealizadas[id] = tareasRealizadas[id] + 1;          //incrementa su contador de tareas
            P(mutexTareas);          //vuelve a bloquear tareas

        End;
        V(mutexTareas);          //para que los demás puedan entrar al while y ver que no hay tareas(?)
        V(empleadoTermino);          //le avisa al jefe que termino
        //SE PODRÍA HACER COMO AL PRINCIPIO QUE REvisa CON UN IF SI
        //TERMINARON=E Y AHÍ HACER V(COORDINADOR)?
    }

Process Jefe
    int i;
    int max = -1;
    int empleadosFinalizaron = 0;
    int idEmpleadoGanador;

    while(empleadosFinalizaron < E) do begin
        P(empleadoTerminó);          //por cada uno que pasa se desbloquea una vez e incrementa
        empleadosFinalizaron = empleadosFinalizaron + 1;
    end;
    for i:=1 to E do begin          //recorre todos los id de empleados
        if (tareasRealizadas[i] > max) then
            max = tareasRealizadas[i];
            idEmpleadoGanador = i;
        end;
    end;
    //entregarPremioAlGanador

End;

```

### 9. 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).

```
array depositoMarcos [30];    array depositoVidrios [50];
```

```
sem depMarcosVacio = 30;          //30 por que puede poner esa cantidad
```

```

sem depMarcosLleno = 0;
sem depVidriosVacio = 50;           //50 por que puede poner esa cantidad
sem depVidriosLleno = 0;
sem mutexV = 1;                     sem mutexM = 1;
int ocupadoM = 0;                   int libreM = 0;
int ocupadoV = 0;                   int libreV = 0;

Process Carpintero [id: 1..4]
    while (true)
        //hacer marco
        P(depMarcosVacio);           //si hay al menos un vacio
        P(mutexM);
        depositarMarco(depositomarcos[libreM]) := marco;    //deposita marco
        libreM = (libreM + 1) mod 30;
        V(mutexM);
        V(depMarcosLleno);           //avisa que hay un marco mas depositado
    End;
End;

Process Vidriero
    while(true)
        //hacer vidrio
        P(depVidriosVacio);           //si hay al menos un lugar vacio
        P(mutexV);
        depositarVidrio(depositovidrios[libreV]) := vidrio;    //depositar vidrio
        libreV = (libreV + 1) mod 50;
        V(mutexV);
        V(depVidriosLleno);           //avisa que hay un vidrio mas depositado
    End;
End;

Process Armador [id: 1..2]
    while(true)
        P(depMarcosLleno);           //espera a que haya al menos un marco
        P(mutexM);                   //bloquea la variable y toma uno
        Marco := AgarrarMarco(depositomarcos[ocupadoM]);
        ocupadoM = (ocupadoM + 1) mod 30;
        V(mutexM);
        V(depMarcosVacio);           //avisa que hay un lugar mas para marcos

        P(depVidriosLleno);
        P(mutexV);
        Vidrio := AgarrarVidrio (depositovidrios[ocupadoV]);
        ocupadoV = (ocupadoV + 1) mod 50;
        V(mutexV);
        V(depVidriosVacio);

        //armarVentana(marco, vidrio);
    end;
end;

```

**10.A una cerealera van T camiones a descargarse 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 actúe como coordinador, resolverlo entre los camiones.**

```
sem total= 7;
sem maiz = 5;
sem trigo = 5;
```

```
Process camiondetrigo [id: 1..T]
```

```
    P(trigo);           //revisa que haya lugar entre los de trigo
    P(total);           //revisa que haya lugar en el total
    //descarga trigo;
    V(total);
    V(trigo);
```

```
End;
```

```
Process camiondemaiz [id: 1..M]
```

```
    P(maiz);
    P(total);
    descargar();
    V(total);
    V(maiz);
```

```
End;
```

```
=====OTRA OPCION DUDOSA=====
```

```
Int ncm:=0;           //cant camiones de maíz adentro
Int nct:=0;           //cant camiones de trigo adentro
Int total:=0;
Int dcm:=0;           //cant esperando en el semaforo
Int dct:=0;           //cant esperando
```

```
Sem baton:=1;
Sem trigo:=5;
Sem maiz:=5;
```

```
Process trigo [id: 1..T]
```

```
    P(e);
    If (total>6 and nct>4) then      //si no puede entrar
        dct:=dct+1;                 //hay uno mas esperando
        V(e);                       //desbloquea las variables
        P(trigo);                   //espera que haya lugar
    End;
    Nct:=nct+1;                     //aca ya pudo entrar, uno mas adentro
    Total:= total+1
    //LIBERAR BATON:
    If (dct>0) and (nct<5) and (total>6) then      //si hay camiones de trigo esperando...
        Dct:=dct-1;      //decrementa la cantidad y deja pasar uno
        V (trigo);
    End;
    elif (dcm>0) and (ncm<5) and (total>6) then      //si hay camiones de maiz esperando...
        Dcm:=dct-1;      //decrementa la cantidad y deja pasar uno
        V (maiz);
    End;
    Else
        V (e);           //si no hay nadie esperando, libera el baton
    //TERMINO DE LIBERAR BATON
    //descarga
    P(e);                 //tenes el baton para acceder a la variable compartida
```

```

Nct:=nct-1;           //uno menos descargando
//LIBERAR BATON:
If (dct>0) and (nct<5) and (total>6) then      //si hay camiones de trigo esperando...
    Dct:=dct-1;      //decrementa la cantidad y deja pasar uno
    V (trigo);
End;
If (dcm>0) and (ncm<5) and (total>6) then      //si hay camiones de maiz esperando...
    Dcm:=dct-1;      //decrementa la cantidad y deja pasar uno
    V (maiz);
End;
Else
    V (e);           //si no hay nadie esperando, libera el baton
//TERMINO DE LIBERAR BATON
End;

```

**\*Para el proceso de maíz seria cambiando invirtiendo dct por dcm y nct por ncm**

11.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.

**//REVISAR SI ESTO TIENE SENTIDO, SOBRETODOS EL REINICIO DE CONTADOR**

```

sem personasEsperando [5]=([5] 0);
Contador:=0;
Máximo:=0;
Sem mutex:=1;
Sem mutexMax:=1;
Sem sig_grupo:=1;      //deja pasar un grupo de 5 a vacunarse
Sem enfermero:=0;

Process persona [1..n]
    P(mutexMax);
    If (total < 50) do begin      //si no van mas de 50, se queda esperando a que la vacunen, si no se va
        V (mutexMax);
        P(mutex);
        Contador:=contador+1;
        If (contador==5) then
            For i:=1 to 5 do begin
                V(barrera);      //el quinto despierta a todos
                V (enfermero); //le avisa al enfermero que hay un grupo esperando
                Contador:=0;      //reinicia contador
            V(mutex);
            P(barrera);           //espera que se completen 5
            //pasa a vacunarse
            P (sig_grupo);        //espera que el enfermero termine
        Else
            //si ya se atendieron las 50 del dia, se va
            V (mutexMax);
    End;

```

```

Process Enfermero
    While (total<50) do begin
        P(enfermero);
        For i:=1 to 5 do
            //vacunarpersona()
            V (sig_grupo);
        End;
    //se retira
End;

```

**12. 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.**

```

Cola p1;
Cola p2;
Cola p3;
Array vectorcolas[3]= ([3] 0)
sem vector_colas= 1;
sem cadaCola[3]= ([3] 1)
sem enfermera[3]= ([3] 0)
sem siguienteEn [3]= ([3] 0)
sem irse [150] = ([150] 0)
int c_hisopados:=0;
sem canthisopados:=1;

```

```

Process Pasajero [id 1..150]
    P(vector_colas);           //elige la cola con menos pasajeros
    Co:=vectorcolas.min().pos(); //pasa la posicion donde hay menos elem (personas)
    P (cadaCola[Co]);          //protege la cola elegida
    Encolar ([Co], id);
    Vectorcolas[Co]:= Vectorcolas[Co]+1;           //incrementa en 1 el vector de colas
    V(vector_colas);
    V (cadaCola[Co]);
    V (enfermera [Co]);        //le avisa a la enfermera que hay alguien esperando
    P (siguiente_en [Co]);      //espera que su enfermera lo llame
    //avanza a hisoparse
    P (irse [id]);              //espera que la enfermera le diga que puede irse
    //avanza
End;

```

```

Process Enfermera [id 1..3]
    P (canthisopados);
    While (c_hisopados <150) then
        V (canthisopados);           //libera la variable canthisop
        P(enfermera [id]);           //espera que haya alguien
        P(mutex_colas);
        Desencolar (cola[id], pasajero); //desencola un pasajero y lo hisopa
        V(mutex_colas);
        V (siguiente_en [id]);        //le avisa que pase
        //hisopar pasajero
        V (irse [pasajero]);          //le avisa al pasajero que ya puede irse
    End;

```

```
        P (canthisopados);  
        c_hisopados:=c_hisopados+ 1;           //actualiza contador  
    end;  
    V (canthisopados);                          //libera la variable canthisop  
    //se retira  
End;
```