Práctica 2 – Semáforos

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/sincronizaciones serán necesarios/convenientes para resolverlo.
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.
d. Modifique la solución anterior para el caso en que cada persona pueda pasar más de una vez, siendo aleatoria esa cantidad de veces.

b)

```
Sem s = 1;
Process Persona[id:1..N]::{
   P(s);
   // Persona se escanea;
   V(s);
}
```

c)

```
Sem s = 3;
Process Persona[id:1..N]::{
   P(s);
   // Persona se escanea;
   V(s);
}
```

d)

```
Sem s = 1; int pases = m;
Process Persona[id:1..N]::{
   For (i = 1 to pases){
      P(s);
      // Persona se escanea;
      V(s);
   }
}
```

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).
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.

a)

```
Sem s = 1; fallo[n] fallos; int restantes = n;
Process Chequeo[id:0..3]::{
   fallo f;
   P(s);
   While(n > 0){
      f = fallos[restantes];
      restantes--;
      V(s);
      If(f.getNivel().equals(3)){
         Println(f.getId())
      }
      P(s);
   }
   V(s);
{
```

b)

```
Sem s=1; fallo[n] fallos; int restantes = n; int[0..3] contadorGlobal = 0; sem[0..3] mutexContador = 1;
Process Chequeo[id:0..3]::{
   fallo f;
   P(s);
   While(n > 0){
      f = fallos[restantes];
      restantes--;
      V(s);
      P(mutexContador[f.getNivelGravedad()]);
      contadorGlobal[f.getNivelGravedad()]++;
      V(mutexContador[f.getNivelGravedad()]);
      P(s);
   }
   V(s);
{
```

c)

```
Sem s=1; fallo[n] fallos; int restantes = n; int[0..3] contadorGlobal = 0;
Process Chequeo[id:0..3]::{
   fallo f;
   P(s);
   While(n > 0){
      f = fallos[restantes];
      restantes--;
      V(s);
      If(f.getNivelGravedad.equals(id)){
         contadorGlobal[id]++;
      }
      Else{
         P(s);
         restantes++;
         V(s);
      }
      P(s);
   }
```

```
   V(s);
{
```

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 para su reúso.

```
ColaRecursos[5] c; sem activos = 5; sem acceso = 1;
Process Proceso[id:1..P]::{
   recurso r;
   While(True){
      P(activos);
      P(acceso);
      r = c.pop();
      V(acceso);
      r.usar();
      P(acceso);
      c.push(recurso);
      V(acceso);
      V(activos);
   }
}
```

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.

```
Var
   total: sem=6;
   alta: sem=4;
   baja: sem=5;
```

```
Process Usuario-Alta [I:1..L]::{
   P(total);
   P(alta);
   //Usa la BD
   V(total);
   V(alta);
}
```

```
Process Usuario-Baja [I:1..K]::{
   P(total);
   P(baja);
   //Usa la BD
   V(total);
   V(baja);
}
```

La solución anterior presenta un error el cual se evidencia en la liberación de los semáforos. Puede ocurrir que tal cual este implementado haya 6 usuarios de baja prioridad queriendo utilizar la BD pero solo 5 de ellos la podrían utilizar, mismo caso pero con 4 en usuarios de alta prioridad. Esto se da porque si ocupamos primero el semáforo **total** puede que un usuario (usando el primer caso ejemplificado) de baja prioridad tome el control pero se quede esperando a que se libere un semáforo **baja**. Para corregir esto hay que realizar un V(alta/baja (respectivamente)) para que se evalue correctamente el límite de accesos de cada usuario. Quedaría tal que asi:

```
Var
   total: sem=6;
```

alta: sem=4; baja: sem=5;	

Process Usuario-Alta [I:1..L]::{
P(alta);
P(total);
//Usa la BD
V(total);
V(alta);
}
	Process Usuario-Baja [I:1..K]::{
	P(baja);
	P(total);
	//Usa la BD
	V(total);
	V(baja);
	}

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 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.

a)

Sem x=1; sem y = 0; Paquete [n] contenedor;	

Process Preparador::{
Paquete pack;
while (true){
//Preparar paquete;
P(x);
contenedor.push(p);
V(y);
}
}
	Process Entregador::
	while (true){
	P(y);
	Paquete pack = contenedor.pop();
	V(x);
	//Entregar paquete;
	}
	}

b)

sem x = 1; sem y = 0; Paquete [n] contenedor; sem vacio = 1;	

Process Preparador [id:1..P]::{
Paquete pack;
while (true){
//Preparar paquete;
P(vacio);
P(x);
contenedor.push(pack);
V(x);
V(y);
}
}
	Process Entregador::
	while (true){
	P(y);
	Paquete pack = contenedor.pop();
	V(vacio);
	//Entregar paquete;
	}
	}

c)

Sem x=1; sem y = 0; Paquete [n] contenedor; sem vacio = 1;

Process Preparador::{	**Process Entregador [id:1..E]::**

Paquete pack;
while (true){
//Preparar paquete;
P(vacio);
contenedor.push(p);
V(y);
}
}
	while (true){
	P(y);
	P(x);
	Paquete pack = contenedor.pop();
	V(x);
	V(vacio);
	//Entregar paquete;
	}
	}

d)

Sem x=1; sem y = 0; Paquete [n] contenedor; sem vacio = 1; sem mutex = 1;	
Process Preparador [id:1..P]::{	**Process Entregador [id:1..E]::**{

Paquete pack;
while (true){
//Preparar paquete;
P(vacio);
P(mutex);
contenedor.push(p);
V(mutex);
V(y);
}
}
	while (true){
	P(y);
	P(mutex);
	Paquete pack = contenedor.pop();
	V(mutex);
	V(vacio);
	//Entregar paquete;
	}
	}

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.
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.

a)

sem s = 1;

Process Persona [id:1..N]::{
Documento documento;
P(s);
Imprimir(Documento);
V(s);
}

b)

sem s = 1; Cola[N] c; int siguiente = 0; sem personas [N] ([N] 0); bool libre = true;

```
Process Persona [id:1..N]::{
   Documento documento;
   P(s);
   if (libre){
      libre = false;
      V(s);
   }
   else{
      c.push(id);
      V(s);
      P(personas[siguiente]);
   }
   Imprimir(documento);
   P(s);
   if (empty(c)){
      libre = true;
   }
   else{
      siguiente = c.pop();
      V(personas[siguiente]);
   }
   V(s);
}
```

c)

```
sem s[N] ([N] 0); int siguiente = 1;
Process Persona [id:1..N]::{
   Documento documento;
   if (id != siguiente){
      P(s[siguiente]);
   }
   Imprimir(Documento);
   siguiente++;
   V(s[siguiente])
}
```

d)

```
sem s = 1; Cola[N] c; sem personas [N] ([N] 0); int siguiente = 0; sem hayElementos = 0; sem imprimiendo = 0;
Process Persona [id:1..N]::{
   Documento documento;
   P(s);
   c.push(id);
   V(s);
   V(hayElementos);
   P(personas[siguiente]);
   Imprimir(documento);
   V(imprimiendo);
}

Process Coordinador::{
   for (i = 1 to N){
```

```
      P(hayElementos);
      P(s);
      siguiente = c.pop();
      V(s);
      V(personas[siguiente]);
      P(imprimiendo);
   }
}
```

e)

```
sem s = 1; Cola[N] c; sem personas [N] ([N] 0); int siguiente = 0; sem hayElementos = 0; sem imprimiendo
[5] ([5] 0);
Process Persona [id:1..N]::{
   Documento documento;
   P(s);
   c.push(id);
   V(s);
   V(hayElementos);
   P(personas[siguiente]);
   P(imprimiendo);
   Imprimir(documento);
   V(imprimiendo);
}

Process Coordinador::{
   for (i = 1 to N){
      P(hayElementos);
      P(s);
      siguiente = c.pop();
      V(personas[siguiente]);
      V(imprimiendo[randm(1..5)]);
      V(s);
   }
}
```

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 (depende de 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).

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. 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 debe conocer cual es el empleado que más piezas fabricó.

a) Implemente una solución asumiendo que T > E.
b) Implemente una solución que contemple cualquier valor de T y E.

a)

```
int piezasRestantes = T;  int [1..E] contadorPiezasEmpleado ([1..E] 0); sem empleados ([1..E] 0); int presente = 0; sem s = 1; sem max = 0; sem finish = 1; int ultimoEmpleado = E;
```

```
Process Empleado [id:1..E]::{
  //Llega empleado;
  P(s);
  presente ++;
  V(s);
  if (presente.equals(E)){
    for (i=1 to E){
      V(empleados[i]);
    }
  }
  P(empleados[id]);
  P(s);
  while (piezasRestantes > 0){
    //Empleado toma pieza;
    piezasRestantes--;
    V(s);
    //Empleado fabrica pieza;
    contadorPiezasEmpleado[id]++;
    P(s);
  }
  V(s);
  P(finish);
  if (id.equals(ultimoEmpleado)){
    V(max);
  }
  V(finish);
}
```

```
Procces Maximo::{
  P(max);
  print(max(contadorPiezasEmpleado));
}
```

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).

```
sem depositoMarcos = 30; sem depositoVidrios = 50; sem marco = 1; sem vidrio = 1; sem hayMarco = 0; sem hayVidrio = 0;
```

```
Process Carpintero [id:1..4]::{
  while (true){
    P(depositoMarcos);
    //Carpintero hace marco;
    P(marco);
    //Carpintero deposita marco;
    V(marco);
    V(hayMarco);
  }
```

```
Process Vidriero::{
  while (true){
    P(depositoVidrios);
    //Vidriero hace vidrio;
    P(vidrio);
    //Vidriero deposita vidrio;
    V(vidro);
    V(hayVidrio);
  }
```

}	}

```
Process Armador [id:1..2]::{
  while (true){
    P(hayMarco);
    P(marco);
    //Armador toma marco;
    V(depositoMarco);
    V(marco);
    P(hayVidrio);
    P(vidrio);
    //Armador toma vidrio;
    V(depositoVidrio);
    //Armador construye ventana;
  }
}
```

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.

a) Implemente una solución que use un proceso extra que actúe como coordinador entre los camiones. El coordinador debe retirarse cuando todos los camiones han descargado.
b) Implemente una solución que no use procesos adicionales (sólo camiones).

a)

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

Process CamionTrigo [id: 1..T]::{
P(trigo);
P(total);
//Camion descarga trigo;
V(total);
V(trigo);
}
	Process CamionMaiz [id:1..M]::{
	P(maiz);
	P(total);
	//Camion descarga maíz;
	V(total);
	V(maiz);
	}

```
Process Coordinador::{

}
```

b)

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

Process CamionTrigo [id: 1..T]::{
P(trigo);
P(total);
//Camion descarga trigo;
V(total);
V(trigo);
}
	Process CamionMaiz [id:1..M]::{
	P(maiz);
	P(total);
	//Camion descarga maíz;
	V(total);
	V(maiz);
	}

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; suponga que el empleado tienen una función VacunarPersona() que simula que el empleado está vacunando a UNA persona.

sem s = 1; **int Cola**[50] c; **bool** libre = true; **int** contador = 0; **sem** despierto_empleado = 0; **sem** vacunado([50] 0);

Process Empleado::{
```
Cola[5] vacunados;
for (int i=1 to 10){
   P(despierto_empleado);
   for (int j=1 to 5){
      P(s);
      persona = c.pop();
      V(s);
      VacunarPersona(persona);
      vacunados.push(persona);
   }
   for (int j=1 to 5){
      personaVacunada = vacunados.pop();
      V(vacunado[personaVacunada]);
   }
}
```
}

Process Persona [id:1..50]::{
```
P(s);
c.push(id);
contador++;
if (contador == 5){
   V(despierto_empleado);
   contador = 0;
}
V(s);
P(vacunado[id]);
```
}

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 Recepcionista, quien le indica qué puesto es el que tiene menos gente esperando. Luego se dirige al puesto y espera a que la enfermera correspondiente lo llame para hisoparlo. Finalmente, se retira.

a) Implemente una solución considerando los procesos Pasajeros, Enfermera y Recepcionista.
b) Modifique la solución anterior para que sólo haya procesos Pasajeros y Enfermera, siendo los pasajeros quienes determinan por su cuenta qué puesto tiene menos personas esperando.

Nota: suponga que existe una función Hisopar() que simula la atención del pasajero por parte de la enfermera correspondiente.

a)

int Cola[150] colaPasajeros; **sem** s; **sem** esperaRetiro ([150] 0); **sem** despiertaRecepcionista = 0; **Cola**[N] **Cola**[3] puestos; **sem** mutexColas ([3] 1); **bool** seguir = true; **sem** despiertaEnfermera([3] 0);

Process Pasajeros [id:1..150]::{
```
P(s);
colaPasajeros.push(id);
V(s);
V(despiertaRecepcionista);
P(esperaRetiro[id]);
```
}

Process Enfermera [id:1..3]::{
```
while(seguir){
   P(despiertaEnfermera[id]);
   if (not empty(puestos[id])){
      P(mutexColas[id]);
      persona = puestos[id].pop();
      V(mutexColas[id]);
      Hisopar(persona);
      V(esperaRetiro[persona]);
   }
}
```
}

Process Recepcionista::{
```
for (i= 1 to 150){
   P(despiertaRecepcionista);
   P(s);
```

```
      pasajero = c.pop();
      V(s);
      colaAux = obtenerColaMenosLlena();
      P(mutexColas[colaAux.id]);
      puestos[colaAux.id].push(pasajero);
      V(despiertaEnfermera[colaAux.id]);
      V(mutexColas[colaAux.id]);
   }
   seguir = false;
   for (i= 1 to 3){
      V(despiertaEnfermera[i]);
   }
}
```

b)

sem s; **sem** esperaRetiro ([150] 0); **Cola**[N] **Cola**[3] puestos; **sem** mutexColas ([3] 1); **int** atendidos 0; **sem** despiertaEnfermera([3] 0); **sem** contador 1;

```
Process Pasajeros [id:1..150]::{
   P(s);
   colaAux = obtenerColaMenosLlena();
   V(s);
   P(mutexColas[colaAux.id]);
   puestos[colaAux.id].push(id);
   V(mutexColas[colaAux.id]);
   V(despiertaEnfermera[colaAux.id]);
   P(esperaRetiro[id]);
}
```

```
Process Enfermera [id:1..3]::{
   while (atendidos < 150){
      P(despiertaEnfermera[id]);
      if (not empty(puestos[id])){
         P(mutexColas[id]);
         persona = puestos[id].pop();
         V(mutexColas[id]);
         Hisopar(persona);
         V(esperaRetiro[persona]);
         P(contador);
         atendidos++;
         if (atendidos == 150){
            for (i=1 to 3) {V(despiertaEnfermera[i])}
         }
         V(contador);
   }
}
```