Practica 2 - SEMÁFOROS

* **Si tengo un semáforo que vale 0, y le hago P, se queda bloqueado hasta que el semáforo valga > que 0.**

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

## a. 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 caso contrario debe esperar). CORREGIDO

sem mutex\_detector = 1;

#### process Personas [i=1..N] {

begin

//llega una persona, pide el uso del detector. Si el semáforo está en 1, al hacer el P, la deja pasar. Si el semáforo está en 0, al hacer el P, no la deja pasar, hasta que se le haga el V.

P(mutex\_detector);

//usa el detector y lo libera, haciendo el V y dejando el semáforo en 1 nuevamente.

V(mutex\_detector);

}

## b. Modifique su solución para el caso que haya tres detectores. CORREGIDO

Se inicializa el semáforo en 3 → **CONTADOR DE RECURSOS**

Si fueran 3 detectores, el semáforo *mutex\_detector*, debería inicializarse en 3. Así, se permitiría que 3 personas hagan P, cada uno de ellos incrementaría en 1 el semáforo hasta que éste sea 0 y no deje pasar a ninguna persona más.

# 2. Un sistema operativo mantiene 5 instancias de un recurso almacenadas en una cola, cuando un proceso necesita usar una instancia del recurso la saca de la cola, la usa y cuando termina de usarla la vuelve a depositar. CORREGIDO

queue cola;

sem mutex\_cola = 1; // Debe inicializarse en 1, para que sólo UN proceso pueda utilizarla, es decir, se le podrá hacer un solo P, hasta que ese proceso la libere, haciendo el V.

sem mutex\_instancias = 5; //contador de recursos. Hay 5 instancias de un recurso en una cola.

#### process Proceso [i=1..N] {

**begin**

//El semáforo inicialmente vale 5. Por ende, como máximo 5 procesos van a poder pasar. Cuando el semáforo quede en 0, es porque ya no hay más instancias del recurso disponibles. En ese caso, no dejará pasar a ningun proceso más.

P(mutex\_instancias);

//Se pide el acceso a la cola para tomar una instancia del recurso. Si no hay otro proceso usando la cola, el P lo dejará pasar al proceso.

P(mutex\_cola):

elemento = cola.pop();

V(mutex\_cola);

//usa el elemento

//devuelve el elemento

P(mutex\_cola)

push.cola(elemento);

V(mutex\_cola);

Incrementa en 1 el semáforo de cantidad de instancias disponibles, ya que terminó de utilizar el recurso.

V(mutex\_instancias);

}

# 3. 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 actué como coordinador, resolverlo entre los camiones. CORREGIDO

sem mutex\_trigo= 5; // contador de recursos. Como máximo, puede haber 5 camiones de trigo

sem mutex\_maiz=5; // contador de recursos. Como máximo, puede haber 5 camiones de maíz.

sem mutex\_cantidad =7; // contador de recursos. Como máximo, puede haber 7 camiones en total.

#### Process Trigo [i=1..T]{

#### begin

//entra un camión de trigo, se debe decrementar la cantidad de camiones de trigo que pueden entrar.

P(mutex\_trigo);

//también se debe decrementar la cantidad total de camiones que pueden entrar.

P(mutex\_cantidad);

//se va el camión de trigo.

//incremento la cantidad de camiones de trigo.

V(mutex\_trigo);

//incremento la cantidad total de camiones.

V(mutex\_antidad);

}

#### Process Maíz[j=1..M]{

**begin**

//entra un camión de maiz, se debe decrementar la cantidad de camiones de maiz que pueden entrar.

P(mutex\_maiz);

//también se debe decrementar la cantidad total de camiones que pueden entrar.

P(mutex\_cantidad)

//se va el camión de maiz.

//incremento la cantidad de camiones de maiz.

V(mutex\_maíz);

//incremento la cantidad total de camiones.

V(mutex\_cantidad);

}

# 4. 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. CORREGIDO

array string[1..40] tareas; //tareas

array string[1..40] notas; //notas

sem [1..40] mutex\_alumnos =[(0)]; //semáforo para c/ alumno. Empiezan dormidos.

sem mutex\_maestra= 0; // exclusión mutua de la maestra.

queue cola;

sem mutex\_cola = 1;

#### Process Alumno [i=1..40] {

**begin**

//el alumno se duerme hasta que la maestra le de la tarea.

P(alumnos[i]);

//mientras no haya aprobado la tarea

while(notas[i] < 4) {

//realiza tarea

tareas[i] = realizarTarea();

P(mutex\_cola);

cola.push(i); // me encolo porque ya termine la tarea

V(mutex\_cola);

V(mutex\_maestra); // despierto a la maestra para que me corrija

//me duermo hasta que la maestra me corrija la tarea

P(alumnos[i]);

} **// while**

} **// ALUMNO**

#### Process Maestra:: {

**var**

int cantAprobados = 0;// cantidad de alumnos que ya aprobaron

**begin**

//entrega las tareas y despierta a los alumnos

for i=1 to 40 do {

tareas[i] = entregarTarea();

V(alumnos[i]);

}

P(mutex\_maestra); FALTABA ESTE P TAMBIEN. LA MAESTRA TIENE QUE ESPERAR QUE ALGUN ALUMNO LA DESPIERTE PARA CORREGIR ALGUNA TAREA, NO TIENE QUE IR AUTOMATICAMENTE A CORREGIR, SINO QUE DEBE ESPERAR A QUE ALGUNO ALUMNO QUE HAYA FINALIZADO SU TAREA, LA DESPIERTE.

//corrige las tareas, mientras no hayan aprobado todos

while (cantAprobados < 40) {

P(mutex\_cola);

pop(cola,a);

V(mutex\_cola);

notas[a] = corregir(tareas[a]); // pone la nota.

V(alumnos[a])**;** //despierta al alumno que le corrigió

if (notas[a] >= 4) then cantAprobados ++;

P(mutex\_maestra);

} //while

} // MAESTRA

# 5. Suponga que se tiene un curso con 50 alumnos. Cada alumno elige una de las 10 tareas para realizar entre todos. 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. Cuando todos los alumnos que tenían la misma tarea terminaron el profesor les otorga un puntaje que representa el orden en que se terminó esa tarea.

##### 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). CORREGIDO

sem mutex\_profesor = 0; //exclusión mutua del profesor

sem [1..50] mutex\_alumnos = ([0]);//semáforo para cada uno de los alumnos

queue colaTareas; //estructura para que los alumnos entreguen sus tareas

sem mutex\_colaTareas = 1;

sem [1..10] mutex\_tareaAlumno = [(0)]; // semáforo para despertar a los 5 para darles la nota

array int [1..10] notas = 0;//estructura donde el profesor pone las notas.

array int [1..10] alumno\_tarea = 0; //estructura donde se cuenta la cantidad de alumnos que llegaron de cada tarea, ya que se necesita que hayan llegado los 5 para poder entregarles la nota.

int cant = 0; //cantidad de alumnos que ya eligió su tarea.

sem mutex\_cant = 1; //se necesita un semáforo para la cantidad porque muchos procesos pueden acceder a ella.

sem mutex\_tarea = 1; //semáforo para controlar el acceso a “elegirTarea”

#### Process Alumno [i=1..50] {

**VAR**

string tarea;

int nota;

**BEGIN**

//elegir una tarea

P(mutex\_tarea)

tarea = elegir();

V(mutex\_tarea);

P(mutex\_cant);

cant ++; //un alumno más que eligió su tarea

if (cant == 50) { //los 50 alumnos ya eligieron su tarea

for i=1 to 50 do

V(alumnos[i]); // despertar a todos los alumnos

} else { //no todos los alumnos eligieron su tarea

V(mutex\_cant);

P(alumnos[i]); //me duermo hasta que llegan todos los alumnos

} //else

//realizar la tarea

P(mutex\_colaTareas)

cola.push(tarea);

V(mutex\_colaTareas);

//avisar al profesor y esperar el puntaje del grupo

V(profesor);

P(mutex\_tareaAlumno[tarea]);  
nota = notas[tarea];

//se retira el alumno

**}** //Alumno

#### Process Profesor :: {

**var**

int cantidad = 1;

puntajeActual = 10

**begin**

while (cantidad < 50) {

P(profesor); //el profesor se duerme hasta que un alumno lo despierte

P(mutex\_cola);  
 cola.pop(tarea)**;** //saca una tarea de la cola  
 V(mutex\_cola);  
 alumno\_tarea[tarea] ++;//espera hasta que hayan llegado los 5 alumnos de esa tarea

if (alumno\_tarea[tarea] == 5) {//ya llegaron los 5 de esa tarea, corregir.

notas[tarea] = puntajeActual;

puntajeActual --;

for (i = 1; i < 5; i++)//despierta a los 5 alumnos   
 V(mutex\_tareaAlumno[i]);  
 }// if

cantidad ++; //suma una tarea más que corrigió

}//while

} //Profesor

# 6. A una empresa llegan E empleados y por día hay T tareas para hacer (T>E), una vez que todos los empleados llegaron empezaran a trabajar. Mientras haya tareas para hacer los empleados tomaran una y la realizarán. Cada empleado puede tardar distinto tiempo en realizar cada tarea. Al finalizar el día se le da un premio al empleado que más tareas realizó. CORREGIDO

sem esperar\_empleados = 1; // barrera para esperar a que hayan llegado todos los empleados

sem [1..E] mutex\_empleados = ([0]);

queue colaTareas;

sem mutex\_cola = 1;

sem mutex\_cantTareas = 1;

array [1..E] tareas\_empleado = 0;//arreglo para saber cuantas tareas realiza cada empleado

int cantEmpleados = 0;

int cantTareas = 0;//para saber cuantas tareas se realizaron

sem sem\_fin = 0; //semáforo para saber cuando todos los empleados ya hicieron su trabajo

int idEmpleadoMax;

#### Process Empleado [i:1..E]{

**VAR**

**BEGIN**

//barrera para esperar a que lleguen todos los empleados

P(esperar\_empleados);

cantEmpleados ++; //llegó un empleado más

if(cantEmpleados == E) {//llegaron todos los empleados

for =1 to E do

V(mutex\_empleados [i]; //despierto a todos los empleados

} else {

V(esperar\_empleados);//desbloquea la variable para que pueda entrar otro proceso empleado

P(mutex\_empleados[i]); //me duermo hasta que llegan todos los empleados

}

//llegaron todos los empleados. Empiezan a trabajar

P(mutex\_cantTareas);

while (cantTareas < T) {

//tomar una tarea

cantTareas ++;

V(mutex\_cantTareas);

//realizar tarea

tareas\_empleado[i] ++; //el empleado i hizo una tarea más

P(mutex\_cantTareas);

}// while

V(mutex\_cantTareas);

V(sem\_fin);// le aviso al coordinador que ya se terminó todo el trabajo

P(mutex\_empleados[i]);//me quedo dormido esperando el cálculo del premio

if (idEmpleadoMax = i) { // el ganador soy yo

// retirar premio

}

} //Empleado

#### Process Coordinador :: {

**VAR**

int max = -1;

**BEGIN**

for i=1 to E do {

P(sem\_fin);

//calculo el empleado máximo

if (tareas\_empleado[i] > max) {

max := tareas\_empleado[i];

idEmpleadoMax = i;

}

}

//ya se sabe quien es el empleado que hizo más tareas.

for i=1 to E do {

V(mutex\_empleados[i]);

}

} //Coordinador

# 7. Existe una casa de comida rápida que es atendida por 1 empleado. Cuando una persona llega se pone en la cola y espera a lo sumo 10 minutos a que el empleado lo atienda. Pasado ese tiempo se retira sin realizar la compra. CORREGIDO

array string [1..N] estado = [(‘esperando’)];

queue cola;

sem mutex\_cola = 1;

sem mutex\_empleado = 0;

sem [1..N] mutex\_personas =([0]);

sem [1..N] mutex\_timer =[(0)];

sem [1..N] mutex\_estado = [(1)];

#### Process Persona [i=1..N] {

**BEGIN**

P(mutex\_cola);

push(cola,i); //la persona se pone en la cola

V(mutex\_cola);

V(mutex\_empleado)**;** //despierta al empleado

V(mutex\_timer[i]);//despierta al timer

P(mutex\_personas[i]);//se duerme hasta que el timer le avisa hasta que terminó el tiempo, o el empleado la atiende.

//NO VA EL P(MUTEX\_ESTADO[I]) PORQUE SI ME DESPIERTA EL TIMER ES XQ YA ME TENGO QUE IR Y YA ME MODIFICO EL ESTADO, Y SI ES EL EMPLEADO TAMBIÉN YA LO ESCRIBIÓ, ES INNECESARIO EL P

Cuando la persona acceda a su estado, el empleado o el timer ya se lo modificaron, y la persona estaba dormida. Por ende hasta que el timer o el empleado la despierten, no va a poder acceder a su estado.

if(estado[i] == “timeOut”) {

//pasaron los 10 minutos, la persona se va.

}

}//Persona

#### Process Empleado :: {

**BEGIN**

while(true) {

//el empleado se queda durmiendo hasta que una persona le solicita que la atienda.

P(mutex\_empleado);

P(mutex\_cola);

pop(cola,persona); //desencola una persona

V(mutex\_cola);

//mira el estado de la persona

P(mutex\_estado[persona]);

if (estado[persona] == “esperando”) { //significa que todavía no pasaron los 10 minutos.

//atiende a la persona

V(mutex\_personas[persona]);

estado[persona] = “atendiendo”;

}

V(mutex\_estado[persona]);

} //while

} //Empleado

#### PROCESS Timer [t:1..N] { BEGIN P(mutex\_timer[t]); delay(10); //pasan 10 minutos P(mutex\_estado[t]); //se bloquea el acceso al estado de esa persona

if (estado[t] == 'esperando') {

//el estado inicial de las personas es ESPERANDO. Si después de que pasaron los 10 minutos el estado sigue siendo esperando, significa que la persona ya se tiene que ir, ya que el empleado no la atendió en ese tiempo, sino le hubiera seteado el estado y le hubiera puesto “atendiendo”.

estado[t] = 'timeOut';  
V(mutex\_personas[t]);//despierta a la persona para que se vaya

}

V(mutex\_estado[t]);//se desbloquea el acceso al estado de esa persona  
}

# 8. 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 armado 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). CORREGIDO

queue marcos;

sem mutex\_marcos = 1;

sem mutex\_cantMarcos = 30;

queue vidrios;

sem mutex\_vidrios = 1;

sem mutex\_cantVidrios = 50;

sem mutex\_esperoMarco = 0;

sem mutex\_esperoVidrio = 0;

#### Process Carpintero [c=1..4]{

**begin**

while (true) {

P(mutex\_cantMarcos);

P(mutex\_marcos);

push(cola,marco); // guarda el marco

V(mutex\_marcos);

V(mutex\_esperoMarco); //señalizo que hay un marco

}

**}** // carpintero

#### Process Vidriero :: {

**BEGIN**

while (true) {

P(mutex\_cantVidrios);

P(mutex\_vidrios);

push(cola,vidrio); // guarda el vidrio

V(mutex\_vidrios);

V(mutex\_esperoVidrio); //señalizo que hay un vidrio

}

}// vidriero

#### Process Armador [a=1..2] {

**var**

**begin**

while (true) {

P(mutex\_esperoMarco);

P(mutex\_esperoVidrio);

// sólo cuando se le haya hecho V a ambos semáforos (mutex\_esperoMarco y mutex\_esperoVidrio) se dejará continuar al proceso armador.

P(mutex\_marcos);

marco = marcos.pop();

V(mutex\_marcos);

P(mutex\_vidrios);

vidrio = vidrios.pop();

V(mutex\_vidrios);

//arma la ventana

}

}// armador

# 9. Resolver el funcionamiento en una empresa de genética. Hay N clientes que sucesivamente envían secuencias de ADN a la empresa para que sean analizadas y esperan los resultados para poder enviar otra secuencia a analizar. 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 el 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; si al terminar ya han pasado las 5 horas despierta al otro servidor y él descansa, sino continúa con el siguiente pedido. CORREGIDO

sem [1..N] mutex\_clientes = [(0)];

~~sem [1..2] sem\_servidores =[(0)];~~

queue cola;

sem mutex\_cola = 1;

sem mutex\_timer= 0;

int idServidor= 1;

sem mutex\_idServidor =1;

sem [1..2] mutex\_continuar = (1,0); //servidor 1 = 1,servidor 2 = 0

int [1..N] resultados;

sem mutex\_empezar = 1;

sem mutex\_hayADN = 0;

#### Process Cliente [c= 1..N] {

**VAR**

**BEGIN**

while (true) {

//enviar ADN

P(mutex\_cola);

cola.push(i);

V(mutex\_cola);

V(mutex\_hayADN); //despierta al servidor

P(mutex\_clientes[i]);//se duerme y espera los resultados

}// while

**}** // Cliente

#### Process Servidor [i= 1..2] {

**VAR**

int elemento;

**BEGIN**

while true {

P(mutex\_continuar[i]);

P(mutex\_empezar);

V(mutex\_timer);

P(mutex\_idServidor);

while (idServidor == i) { //mientras me toque a mi

V(mutex\_idServidor);

P(mutex\_hayADN); //espera a que haya ADN

P(mutex\_cola);

cola.pop(elemento,idP);

V(mutex\_cola);

// procesamiento

resultados[idP]= resultado;

V(mutex\_clientes[idP]); //avisa al cliente

P(mutex\_idServidor);

} //while id servidor

V(mutex\_idServidor);

V(empezar); //cuando el servidor que estaba procesando termina, hace el V para que el otro pueda empezar.

} //while true

**} // Servidor**

#### Process Timer {

**BEGIN**

while true {

P(mutex\_timer);

delay(5);//pasan 5 horas

P(mutex\_idServidor);

idServidor ++;

if (idServidor == 3); // es porque ya le toca al 1

idServidor = 1;

V(mutex\_continuar[idServidor]); // le aviso a ese servidor que le toca seguir a él

V(mutex\_idServidor);

}

**}** // Timer