Práctica 2 – Semáforos

**• 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.** 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).

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

**a.** sem detector=1

Process personas[id:1..N]

P(detector)

pasa por el detector

V(detector)

**b.** sem detector=3

Process personas[id:1..N]

P(detector)

#pasa por el detector

V(detector)

**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.

Sem recursos = 1;

Sem disponibles = 5;

Queue cola[5];

process SO{

Recurso elem;

while (true){

P(disponibles)

P(recursos)

cola.desencolar(elem) //desencolo recurso para usarlo

V(recursos)

#uso el recurso

P(recursos)

cola.encolar(elem)

V(recursos)

V(disponibles)

}

}

**3.** 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.

|  | seis acceder, alta 4, baja 5, menos situaciones que puedan darse, , 4 altas 2 bajas, 5 bajas 1 alta, eso a la vez tb despues cantidades menores, 3 y 3  Var sem: semaphoro := 6;  alta: semaphoro := 4;  baja: semaphoro := 5; |
| --- | --- |
|  | Process Usuario-Alta [I:1..L]::  { P (sem);  P (alta);  //usa la BD  V(sem);  V(alta);  }  Process Usuario-Baja [I:1..K]::  { P (sem);  P (baja);  //usa la BD  V(sem);  V(baja);  }  general y particular, c proc cuando entra decrementa un sem particular, solo 4 altas, y 2, o 5 y 1, deberia invertir P(sem) y P(alta) asi ingresan los que deban acceder  Habría que invertir P(sem) P(alta), para hacer los V no es problema el orden, lo mismo con el proceso Baja.  Cuando liberas necesitas que pasen los dos, terminan siendo lo mismo, primero el general para asegurar que lo hcie, a lo sumo entra un baja mas.  El ejercicio permite la cantidad pero no maximiza la concurrencia. |
|  |  |

**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.

Exp. dos procesos y dos tipos de personas que interactuan, una maestra y 40 alum

maestra{

//le entrega al alumno la tarea a realizar  
//avisarle al alumno  
sem espera\_alumno[40]=([40] 0); //como esta en cero el alumno espera

for i=1 to 40

alumno{

//esperar la tarea

}

la sinc entre alumno y maestra es una barrera, todos se bloquean y hace un proc qu hace un for de 1 a 40 v(alum en espera) , //genera la tarea y entrega a cada alumno (40)

for i:1 to 40{

aux = new Tarea();

Tareas[i]=aux; //dejo la tarea en un arreglo, tarea para uno.

v(espera\_alumno[i]);

}

var

sem espera\_alumno[40]=([40] 0); //como esta en cero el alumno espera

Cola entregas;

sem sem\_cola=1;

Tareas[1..40];

sem sem\_entregas=0;

boolean es\_correcta[40]=([40] false);

process maestra{

int terminaron=0;

boolean correcta;

int id\_alumno;

tarea aux;

//genera la tarea y entrega a cada alumno (40)

for i:1 to 40{

aux = new Tarea();

Tareas[i]=aux; //dejo la tarea en un arreglo

v(espera\_alumno[i]);

}

while(terminaron<40){

p(sem\_entregas); //espera a q le entregue algun alumno

p(sem\_cola);

entregas.pop(id\_alumno,aux);

v(sem\_cola);

correcta=corregir(aux);//devuelve true or false

es\_correcta[id\_alumno]=correcta;

if(correcta){

terminaron++;

}

v(espera\_alumno[id\_alumno]);

}

}

process Alumno[id= 1 .. 40 ]{

tarea aux;

p(espera\_alumno[i]); //espera a recibir la tarea

aux=Tareas[i];

while (es\_correcta[i]==false){//si está mal (o no fue corregida), corrige y vuelve a entregar

hacer(aux);//realiza la tarea

p(sem\_cola);//entrega la tarea

entregas.push(id\_alumno,aux);

v(sem\_cola);

v(sem\_entregas);

p(espera\_alumno[i]); //espera la corrección

}

}

5.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 (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.

a.

sem sem\_impresora=1;

process persona[1..N]{

var documento;

hacerDoc(documento);

//mandan a imprimir

p(sem\_impresora);

Imprimir(documento);

v(sem\_impresora);

}

b.

sem sem\_impresora=1;

sem sem\_en\_uso=1;

boolean en\_uso=false;

cola cola\_espera(id); sem sem\_cola=1;

sem esperar\_turno[N]=([N] 0);

sem sem\_cant\_espera=1;

int cant\_en\_espera=0;

process persona[1..N]{

var documento;

hacerDoc(documento);

p(sem\_en\_uso);

if(en\_uso){

v(sem\_en\_uso);

p(sem\_cola);

cola\_espera.pop(id);

cant\_en\_espera++;

v(sem\_cola);

p(esperar\_turno[id]);

p(sem\_en\_uso);

en\_uso=true;

v(sem\_en\_uso);

p(sem\_cola);

cant\_en\_espera--;//no espero maaaaaaaaaaaaaaaas

v(sem\_cola);

imprimir(documento);

p(sem\_cola);

if(cant\_en\_espera>0){

int aux;

cola\_espera.pop(aux);

v(sem\_cola);

p(sem\_en\_uso);

en\_uso=false;

v(sem\_en\_uso);

v(esperar\_turno[aux]);

}

}else{

//preguntar si hay gente esperando

p(sem\_cola);

if(cant\_en\_espera==0){

v(sem\_cola);

p(sem\_en\_uso);

en\_uso=true;

v(sem\_en\_uso);

Imprimir(documento);

p(sem\_cola);

if(cant\_en\_espera>0){

int aux;

cola\_espera.pop(aux);

v(sem\_cola);

p(sem\_en\_uso);

en\_uso=false;

v(sem\_en\_uso);

v(esperar\_turno[aux]);

}

}else{

p(sem\_cola);

cola\_espera.pop(id);

v(sem\_cola);

p(esperar\_turno[id]);

p(sem\_en\_uso);

en\_uso=true;

v(sem\_en\_uso);

p(sem\_cola);

cant\_en\_espera--;//no espero maaaaaaaaaaaaaaaas

v(sem\_cola);

imprimir(documento);

p(sem\_cola);

if(cant\_en\_espera>0){

int aux;

cola\_espera.pop(aux);

v(sem\_cola);

p(sem\_en\_uso);

en\_uso=false;

v(sem\_en\_uso);

v(esperar\_turno[aux]);

}

}

}

}

c.

sem semaforo=1; cola personas\_espera; sem hayEsperando=0; sem personas[N]=([N] 0);

process coordinador{

int id;

while(true){

p(hayEsperando);

p(semaforo); personas\_espera.pop(id);v(semaforo);

v(personas[id]);

}

}

process persona[1..N]{

Documento doc;

hacer(doc);

p(semaforo);

personas\_espera.push(id);

v(semaforo);

v(hayEsperando);

p(personas[id]);

Imprimir(doc);

}

**6.**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

sem sem\_cola=1;

Cola cola\_entregas;

int entregas\_grupo[10]=([10] 0);

sem correcciones=1;

sem espera\_grupo[10]=([10]0);

int puntaje[10]=([10]0);

sem entregas=0;

sem barrera=0;

int cant\_tareas\_asignadas=0;

process alumnos[1..50]{

int grupo; int puntaje;

Tarea tarea;

elegir(tarea,grupo); //elige una tarea del pool de tareas

p(sem\_cant);cant\_tareas\_asignadas++;

if (cant\_tareas\_asignadas==50){

//despertar a todos los alumnos

for(i=1 to 50){

v(barrera);

}

}

v(sem\_cant);

p(barrera);//aca iria barrera

realizar(tarea);//realiza la tarea

p(sem\_cola);

cola\_entregas.push((tarea,grupo));//entrega la tarea

v(sem\_cola);

v(entregas);//aviso q entregue

p(espera\_grupo[grupo]);//espera q terminen sus compañeros

p(correcciones);

//este semaforo podria no estar porq el dato q esta en //puntaje[grupo] no se modifica desde que se despiertan los //alumnos

puntaje=puntaje[grupo];//recibe el puntaje

v(correcciones);

}

process profesor{

int cant\_entregas[10]=([10]0);

int total\_entregas=0;

int puntos=10;

Tarea tarea;

int grupo;

while(total\_entregas<50){

p(entregas);//espero a que entregue algun alumno.

p(sem\_cola);

cola\_entregas.pop((tarea,grupo));//entrega la tarea

v(sem\_cola);

total\_entregas++;

cant\_entregas[grupo]++;

if(cant\_entregas[grupo]==5){//terminaron todos los del grupo

p(correcciones);

puntaje[grupo]=puntos;//guardo cuanto sacaron

v(correcciones);

puntos--;//resto la cant de puntos disponibles

for(int i=1 to 5){

v(espera\_grupo[grupo]); //despierto a los 5

//alumnos del grupo.

}

}

}

}

### Ejercicio 7:

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 tomarán 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.

sem sem\_todos=0; sem sem\_cola=1; Cola cola\_tareas;

int cant\_empleados=0; sem cont\_emp=1; int tareas\_empleado[E]=([E]0);

sem nohaymastareas=0;

process Empleado[id= 1.. E]{

Tarea tarea;

//llega el empleado al lugar

P(cont\_emp);

cant\_empleados++;

V(cont\_emp);

if(cant\_empleados == E){

for 1 to E {

V(sem\_todos);

}

}

P(sem\_todos);

P(sem\_cola);

while(cola\_tareas.isEmpty == false){

tarea= cola\_tareas.pop();

V(sem\_cola);

hacer\_tarea(tarea);

tareas\_empleados[id]= tarea\_empleados[id]++;

P(sem\_cola);

}

V(sem\_cola);

v(nohaymastareas);

}

process premio{

int max=0; int id\_max;

p(nohaymastareas);

for i=1 to E{

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

max=tareas\_empleados[i];

id\_max=i;

}

}

doy\_premio(id\_max);

}

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 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).

Marco marcos[30];

sem sem\_marco=1;

int marco\_ocupado=0;

int marco\_libre=0;

sem hayMarcos=0;

sem hacer\_marco=30;

sem hacer\_vidrios=50;

Marco marco;

Vidrio vidrios[50];

sem sem\_vidrio=1;

int vidrio\_ocupado=0;

int vidrio\_libre=0;

sem hayVidrios=0;

| process carpintero[1..4]{  Marco marquito;  //crea el marco  while(true){  p(hacer\_marco);  crearMarco(marquito);  //lo guarda en el lugar libre  p(sem\_marco);  marcos[marco\_libre]= marquito;  marco\_libre=(marco\_libre+1 mod 30);  v(sem\_marco);  v(hayMarcos);  //avisar q hizo un marco  }  } | process vidriero[1]{  Vidrio vidrio;  while(true){  p(hacer\_vidrios);  crearVidrio(vidrio);  p(sem\_vidrio);  vidrios[vidrio\_libre]=vidrio;  vidrio\_libre=(vidrio\_libre+1 mod 50);  v(sem\_vidrio);  p(hayVidrios);  }  } | process armadores[1..2]{  Marco marco;  Vidrio vidrio;  //esperando q haya marco  while(true){  P(hayMarcos);  P(sem\_marco);  marco= marcos[marco\_ocupado];  marco\_ocupado=( marco\_ocupado + 1) mod 30;  V(sem\_marco);  V(hacer\_marco);  //esperar vidrio  p(hayVidrios);  P(sem\_vidrio);  vidrio= vidrios[vidrio\_ocupado];  vidrio\_ocupado= (vidrio\_ocupado +1) mod 50;  v(sem\_vidrio);  v(hacer\_vidrio);  hacer\_ventana(marco,vidrio);  }  } |
| --- | --- | --- |

### Ejercicio 9:

9. 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.

sem maiz=5;

sem trigo=5;

sem camiones=7;

process Camion\_Trigo[1..T]{

//llega al lugar

chuchu();

p(sem\_trigo); //ver si hay lugar para trigo

p(sem\_camiones); //ver si hay lugar para el camion

descarga(); //descargar

v(sem\_camiones);

v(sem\_trigo);

}

process Camion\_Maiz[1..M]{

chuchu(); //llega al lugar

p(sem\_maiz); //ver si hay lugar para maiz

p(sem\_camiones); //ver si hay lugar para el camion

descarga(); //descargar

v(sem\_camiones);

v(sem\_maiz);

}