[concu3](https://docs.google.com/document/d/1TsKUcs0hv9IVrV0oyPN7i_QZy8vlVUNRvzYFh7ikzHE/edit)

//Enlace al doc compartido de la pipol de los sábados!

1. Se dispone de un puente por el cual puede pasar un solo auto a la vez. Un auto pide permiso para pasar por el puente, cruza por el mismo y luego sigue su camino.

| **Monitor Puente**  cond cola;  int cant= 0;  Procedure entrarPuente (int au)  while ( cant > 0) wait (cola);  cant = cant + 1;  end;  Procedure salirPuente (int au)  cant = cant – 1;  signal(cola);  end;  **End Monitor;**  **Process Auto [a:1..M]**  Puente. entrarPuente (a);  “el auto cruza el puente”  Puente. salirPuente(a);  **End Process;** | a. ¿El código funciona correctamente?  Justifique su respuesta.  b. ¿Se podría simplificar el programa? En  caso afirmativo, rescriba el código.  c. ¿La solución original respeta el orden de  llegada de los vehículos? Si rescribió el código  en el punto b), ¿esa solución respeta el orden  de llegada? |
| --- | --- |

a)

No sé si está bien, porque se le pasa el id del proceso y no se usa para nada, además no sé si es necesario poner un while, con preguntar en un if creo que alcanzaría. Además está todo el tiempo preguntando por la condición (busy waiting?). Creo que una solución más simple, dado que no se tiene que respetar un orden estricto ni nada parecido es algo así:

b)

**Monitor Puente{**

Procedure SolicitarPasoPorPuente()

{

PasarPorPuente();

}

**}**

**Process Auto[id: 1..M]{**

Puente.SolicitarPasoPorPuente();

**}**

c)

La solución planteada, no necesariamente respeta el orden de llegada, y que el primer auto si pasa, pero luego podría acceder al puente cualquier otro auto que no sea el segundo precisamente. Y en cuanto a la solución más simple no respeta el orden de llegada tampoco.

2.Implementar el acceso a una base de datos de solo lectura que puede atender a lo sumo 5 consultas simultáneas.

process lectura[0..N-1]{

BasedeDatos.acceder();

leerenBasededatos();

BasedeDatos.fin();

}

monitor BasedeDatos{

int cantLeyendo=0;

int cantEsperando=0;

cond esperar;

bool hayLugar=true;

procedure acceder(){

if(not hayLugar){ //si hay menos de 5 leyendo

cantEsperando++; //contador de espera en lectura

wait(esperar); //me duermo a la espera de lugar

}else{

cantLeyendo++; //cant de lecturas

if(cantLeyendo==5){

hayLugar=false;//si hay 5 leyendo, no hay

//mas lugar

}

}

}

procedure fin(){

if(cantEsperando>0){//si hay gente esperando

cantEsperando--;//resto la cantidad en espera porq

signal(esperar); // despierto al primero dormido

}else{

cantLeyendo--;//no habia gente esperando, deje de

//leer y no fui reemplazado por otro

hayLugar=true; //marco q hay lugar porq

//hay menos de 5 leyendo

}

}

}

3.En un laboratorio de genética se debe administrar el uso de una máquina secuenciadora de ADN. Esta máquina se puede utilizar por una única persona a la vez. Existen 100 personas en el laboratorio que utilizan repetidamente esta máquina para sus estudios, para esto cada persona pide permiso para usarla, y cuando termina el análisis avisa que termino. Cuando la máquina está libre se le debe adjudicar a aquella persona cuyo pedido tiene mayor prioridad (valor numérico entre 0 y 100)

process persona[1..100]{

int prioridad=obtenerPrioridad();

while(true){

secuenciadora.solicitarAcceso(id,prioridad);

utilizarSecuenciadora();

secuenciadora.finalizar();

}

}

monitor secuenciadora{

cond vcPersona[100];

colaOrdenada fila;

int esperando=0;

bool libre=true;

procedure solicitarAcceso(id,prioridad){

if(not libre){

esperando++;

fila.insertarOrdenado(id,prioridad);

wait(vcPersona[id]);

}else{

libre=false;

}

}

procedure finalizar(){

if(esperando>0){

esperando--;

signal(vcPersona[fila.primero()]);

}else{

libre=true;

}

}

}

4.Suponga que N personas llegan a la cola de un banco. Una vez que la persona se agrega en la cola no espera más de 15 minutos para su atención, si pasado ese tiempo no fue atendida se retira. Para atender a las personas existen 2 empleados que van atendiendo de a una y por orden de llegada a las personas.

process Persona[1..N]{

bool esperar; int id\_empleado;

string estado; text Documento;

TiempoEspera[id].iniciar();

Banco.llegar(id);

Estado[id].esperar(estado,id\_empleado);

if(estado==”atendiendo”){

Escritorio[id\_empleado].EntregarSolicitud(Documento);

}else{

//aca llegas por timeout

}

}

monitor Escritorio[id=1..2]{

text doc; text res;

cond client, empl;

boolean listo=false;

procedure EntregarSolicitud(documento: in text, resultado; out text){//llega cliente

doc=documento;

listo=true;

signal(empl);

wait(client);//despierta el empleado

resultado=res;

signal(empl);

}

procedure recibirSolicitud(documento: out text){//a este

//procedure lo llama empleado

if(not listo){//si no llego el cliente

wait(empl);

}

}

procedure responderSolicitud(respuesta: in text){

res=respuesta;

signal(client);

}

}

monitor Banco{

cond empleado;

cola fila;

process llegar(id: in int){

fila.push(id);

signal(empleado);

}

process atender(idCliente: out int){

if(empty(fila){

wait(empleado);

}

idCliente = fila.pop();

}

}

monitor Estado[1..N]{

cond persona;

string estado\_actual=”esperando”;

int id\_emp;

process atender(estado:in out string, id\_empleado: in int){

if(estado\_actual==”esperando”){

estado\_actual=”atendiendo”;

id\_emp=id\_empleado;

signal(persona);

}

estado=estado\_actual;

}

process cambiarEstado(estado: in string){

if(estado\_actual==”esperando”){

estado\_actual=estado;//estado se recibe como

//timeout

signal(persona);

}

}

process esperar(estado: in out string, id\_empleado:out int){

if(estado\_actual == “esperando”){

wait(persona);

}

estado=estado\_actual;

id\_empleado=id\_emp;

}

}

monitor TiempoEspera[1..N]{

cond timer;

bool llegopersona=false;

procedure iniciar(){

llegopersona=true;

signal(timer);

}

procedure dormir\_timer(){

if(not llegapersona){

wait(timer);

}

}

}

process empleado[1..2]{

string estado; text doc; text res;

int idCliente;

while (true){

Banco.atender(idCliente);

Estado[idCliente].atender(estado,id);

if(estado==”atendiendo”){

Escritorio[id].recibirSolicitud(doc);

res=resolverSolicitud(doc);

Escritorio[id].responderSolicitud(res);

}

}

}

process timer[id = 1..N]{

TiempoEspera[id].dormir\_timer();

delay(15);

Estado[id].cambiarEstado(“timeout”);

}

//**Este ejercicio era un rompe cerebros-- Fin del comunicado//**

[Práctica 3](https://docs.google.com/document/d/1VvaS3v0S2-1Dm55WB8YlKfHYdIsfzIKdpxRe4doEk44/edit#heading=h.21byif4yj667)

[2014 - Parcial Coloquio - 002 (1).pdf](https://drive.google.com/file/d/1BXSRj3ROzptBDFsNOOz2EebTrNqg5h8y/view?usp=sharing)

5. Se tiene una oficina de registros que atiende un único empleado, existen C clientes que cuando llegan se encolan para ser atendidos por el empleado, mientras esperan para ser atendidos

toman una planilla y la llenan para ganar tiempo, luego que completaron la planilla esperan a que los llame el empleado, al momento de la atención el cliente le da la planilla al empleado,

este carga los datos y le entrega un comprobante de registro.

| Monitor Oficina[]::  {  procedure llamar\_prox\_cliente(){  }  }  Monitor MesaDeAyuda[]::  {  procedure solicitar\_planilla(text in out doc){  }  } | Process Empleado[]::  {  while(true){  Oficina.llamar\_prox\_cliente();    MesaDeAyuda.solicitar\_planilla(planilla);  //procesar Planilla  MesaDeAyuda.emitir\_comprobante(doc);  }  }  Process Cliente[id: 1..C]::  {      } |
| --- | --- |