**Práctica 3 - Monitores**

#### **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;** |

##### **¿El código funciona correctamente? Justifique su respuesta.**

##### **¿Se podría simplificar el programa? En caso afirmativo, reescriba el código.**

##### **¿La solución original respeta el orden de llegada de los vehículos? Si reescribió el código en el punto b), ¿Esa solución respeta el orden de llegada?**

##### **Modifique el código de la solución original para que permita pasar el puente hasta 5 autos a la vez (sin necesidad de que se respete el orden de llegada al puente).**

##### **Modifique el inciso d) para que se respete el orden de llegada al puente.**

#### **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 terminó. Cuando la máquina está libre se le debe adjudicar a aquella persona cuyo pedido tiene mayor prioridad (valor numérico entre 0 y 100). CORREGIDO**

###### Monitor MáquinaADN {

**VAR**

boolean libre = true;

cond [1..100] personas\_esperando;

queue colaPrioridad; //cola con prioridad para las personas

**BEGIN**

**procedure entrar(int p) {** //una persona llamó a *entrar*, por ende, si la máquina estaba libre, ya no lo está. Sino, la persona se encola con prioridad y se queda esperando.

if (libre){

libre = false;

} else {

cola.encolarPrioridad(p);

wait(personas\_esperando[p])

}

}

**procedure salir() {** //una persona llamó a *salir*, por ende, si la cola está vacía, la máquina pasa a estar libre. Si no, se desencola una persona con prioridad y se la despierta para que use la máquina.

if empty(colaPrioridad) {

libre = true;

} else {

i = cola.desencolarPrioridad();

signal(personas\_esperando[i]);

}

}

**}** // MáquinaADN

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

**BEGIN**

while (true) {

Maquina.entrar(p);

// P usa la máquina

Maquina.salir(p);

}//while

} // Persona

#### **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. CORREGIDO**

###### Monitor Banco {

**VAR**

queue cola;

cond empleado;

**BEGIN**

**procedure llegue(int p) {** //una persona llamo a *llegue.* La persona se encola y avisa al empleado.

cola.push(p);

signal(empleado)

**}**

**procedure atender(var int persona) {** //un empleado llamó a *atender*. Si no hay nadie para atender ,el empleado se queda esperando. Si no, desencola una persona y se la retorna

while (empty(cola) {

wait(empleado);

}

persona = cola.desencolar();

}

**}** // Banco

###### Monitor Estado [e=1..N] {

**VAR**

string estado = ”esperando”;

cond persona;

**BEGIN**

**procedure cambiarEstado(string new\_estado)**{ //a este procedimiento lo llama el timer cuando se agotó el tiempo y le asigna “time out”, y le avisa a la persona.

if (estado == ‘esperando’) {

estado = new\_estado;

signal(persona);

}

}

**procedure esperar(var estado\_actual)**{ //a este procedimiento lo llama la persona. Si su estado es esperando, tiene que quedarse dormida.

if(estado == “esperando”)

wait(persona);

estado\_actual = estado;

}

**procedure QuiereAtenderEmpleado(var string estadoActual)**{ //a este procedimiento lo llama el empleado. Si el estado es esperando, y el empleado llamo a este procedimiento, es porque aún no se pasó el tiempo y tiene que atender a la persona.

if (estado == “esperando”){

estado = “atendiendo”

signal(persona);

}

estadoActual = estado;

}

} //Estado

###### Monitor CoordinadorTimer [c=1..N] {

**VAR**

cond timer;

boolean llegopersona = false;

**BEGIN**

**procedure iniciar ()** { // a este procedimiento lo llama la persona. Si llega primeo la persona, se setea el booleano y se despierta al timer.

llegopersona = true;

signal(timer);

}

**procedure dormir\_timer ()** { // a este procedimiento lo llama el timer. Si el ejecuta antes que la persona, tiene que esperar a que la persona llegue (llegopersona = true)

if(!llegopersona)

wait(timer);

}

} //CoordinadorTimer

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

**BEGIN**

while (true) {

CoordinadorTimer[p].iniciar();

Banco.llegue();

Estado[p].esperar(estado);

if(estado = “atendiendo”) {

// la atienden

} else {

//time out. se va

}

} //while

**}** // Persona

###### Process Empleado [e=1..2] {

// pide una persona del banco, si la persona estaba esperando, la atiende, y si era time out, no hace nada

**VAR**

string estado;

**BEGIN**

while true {

Banco.atender(p);

Estado[p].QuiereAtenderEmpleado(estado);

if (estado == ’atendiendo’)

// lo atiende

} //while

} // empleado

###### Process Timer [t=1..N] {

**BEGIN**

CoordinadorTimer[t].dormir\_timer();

delay(15);

Estado[t].cambiarEstado(“timeout”)

**}** // timer

#### **En una casa viven una abuela y sus N nietos. Además la abuela compró caramelos que quiere convidar entre sus nietos. Inicialmente la abuela deposita en una fuente X caramelos, luego cada nieto intenta comer caramelos de la siguiente manera: si la fuente tiene caramelos el nieto agarra uno de ellos, en el caso de que la fuente esté vacía entonces se le avisa a la abuela quien repone nuevamente X caramelos. Luego se debe permitir que el nieto que no pudo comer sea el primero en hacerlo, es decir, el primer nieto que puede comer nuevamente es el primero que encontró la fuente vacía. NOTA: siempre existen caramelos para reponer. Cada nieto tarda t minutos en comer un caramelo (t no es igual para cada nieto). Puede haber varios nietos comiendo al mismo tiempo. CORREGIDO**

###### Process Nieto [id=1..N] {

**BEGIN**

while true {

Fuente.agarrarCaramelo();

//come caramelo

}

} //Nieto

###### Process Abuela {

**BEGIN**

while true {

Fuente.rellenar(30);

}

} // Abuela

###### Monitor Fuente {

**VAR**

int cantCaramelos;

boolean primero = true;

boolean primeroEsperando = false;

cond abuela, nietos, nietoAviso;

**BEGIN**

**procedure agarrarCaramelo() {**

boolean ok = false; //todavía no agarré un caramelo

while (not ok) {

if (cantCaramelos > 0) && (not primeroEsperando){

cantCaramelos --;

ok = true; //ya agarré un caramelo

} else {

if (primero) { //soy el primero en ver que no había más caramelos

primero = false; //el próximo ya no es el primero

signal(abuela);

primeroEsperando = true; // el primero en avisar ya esta esperando para comer

wait(nietoAviso); // duermo al nieto que avisó que la fuente estaba vacía

cantCaramelos --;

primeroEsperando = false; // ya agarre caramelos asique ya no estoy esperando

ok = true;

primero = true;

signalAll(nietos); //despierta a todos los nietos

} else { // no soy el primero

wait(nietos);

}

}

**procedure rellenar(int cant){**

//rellenar la fuente

cantCaramelos = cant;

signal(nietoAviso);

wait(abuela);

}

} // Fuente

#### **En un entrenamiento de fútbol hay 20 jugadores que forman 4 equipos (cada jugador conoce el equipo al cual pertenece llamando a la función *DarEquipo*()). Cuando un equipo está listo (han llegado los 5 jugadores que lo componen), debe enfrentarse a otro equipo que también esté listo (los dos primeros equipos en juntarse juegan en la cancha 1, y los otros dos equipos juegan en la cancha 2). Una vez que el equipo conoce la cancha en la que juega, sus jugadores se dirigen a ella. Cuando los 10 jugadores del partido llegaron a la cancha comienza el partido, juegan durante 50 minutos, y al terminar todos los jugadores del partido se retiran (no es necesario que se esperen para salir). CORREGIDO**

###### Monitor Equipo [e=1..4] {

**VAR**

int cantJugadores = 0;

cond equipo; // para despertar a los jugadores de un equipo

int cancha;

**BEGIN**

**procedure formarEquipo (var int cancha)** {

cantJugadores ++;

if (cantJugadores == 5) {

AsignarCancha.obtenerCancha(cancha);

signalAll(equipo);

} else {

wait(equipo);

}

}

} //EQUIPO

###### Process Jugador [j=1..20] {

**VAR**

int equipo;

int c;

**BEGIN**

equipo = *DarEquipo*();

Equipo[equipo].formarEquipo(c);

Cancha[c].jugar();

} // JUGADOR

###### Monitor Cancha [c=1..2]{

**VAR**

int cantJugadores = 0;

cond dormirJugadores;

**BEGIN**

**procedure jugar(){**

cantJugadores++;

if (cantJugadores < 10){

wait(dormirJugadores);

} else {

delay(50); //juegan

signalAll(dormirJugadores);

}

}

} // CANCHA

###### Monitor AsignarCancha {

**VAR**

int cantJugadores = 0;

**BEGIN**

**procedure obtenerCancha(var int cancha)**{

cantJugadores++;

if (cantJugadores <= 2) {

cancha = 1;

} else {

cancha = 2;

}

}

} //AsignarCancha

#### **Suponga una comisión con 50 alumnos. Cuando los alumnos llegan forman una fila, una vez que están los 50 en la fila el jefe de trabajos prácticos les entrega el número de grupo (número aleatorio del 1 al 25) de tal manera que dos alumnos tendrán el mismo número de grupo (suponga que el jefe posee una función *DarNumero*() que devuelve en forma aleatoria un número del 1 al 25, el jefe de trabajos prácticos no guarda el número que le asigna a cada alumno). Cuando un alumno ha recibido su número de grupo comienza a realizar la práctica. Al terminar de trabajar, el alumno le avisa al jefe de trabajos prácticos y espera la nota. El jefe de trabajos prácticos, cuando han llegado los dos alumnos de un grupo les devuelve a ambos la nota del GRUPO (el primer grupo en terminar tendrá como nota 25, el segundo 24, y así sucesivamente hasta el último que tendrá nota 1). CORREGIDO**

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

**VAR**

int grupo;

int nota;

**BEGIN**

Jefe.llegue(grupo);

// comienza a hacer la práctica

Jefe.termine(grupo);

//espera la nota

Jefe.pedirNota(grupo,nota);

} // ALUMNO

###### Monitor Jefe {

**VAR**

cond alumno, alumnos[25];

array [1..25] grupos = [0];

int notaActual = 25;

int cant = 0;

array [1..25] notas;

**VAR**

**procedure llegue(var int grupo){**

cant++;

if (cant < 50) {

wait(alumno);

} else {

signalAll(alumno);

}

grupo = *darGrupo*();

}

**procedure termine (int grupo)**{

grupos[grupo]++;

if (grupos[grupo]) == 1){

wait (alumnos[grupo]);

} else { // llegaron los dos alumnos de ese grupo.

notas[grupo] = notaActual;

notaActual--;

signal(alumnos[grupo]); //despierta a los dos alumnos de ese grupo para darles la nota

}

}

**procedure pedirNota(int grupo, var int nota)** {

nota = notas[grupo];

}

}// jefe

#### **Resolver el uso de un equipo de videoconferencia que puede ser usado por una única persona a la vez. Hay P Personas que utilizan este equipo (una única vez cada uno) para su trabajo de acuerdo a su prioridad. La prioridad de cada persona está dada por un número entero positivo. Además existe un Administrador que cada 3 hs. incrementa en 1 la prioridad de todas las personas que están esperando por usar el equipo. CORREGIDO**

###### Monitor VideoConferencia {

**VAR**

boolean libre = true;

array cond [1..P] personas;

queue cola;

**BEGIN**

**procedure pedir(int prioridad, int id) {**

if (libre){

libre=false;

} else {

cola.encolarPrioridad(id, prioridad);

wait(personas[id])

**}**

**procedure liberar() {**

if empty(colaPrioridad) {

libre = true;

} else {

i = cola.desencolarPrioridad(id, prioridad);

*//p=cola.pop();*

*//signal (personas[p.id])*

signal(personas[id]);

}

}

**procedure actualizarPrioridad(){**

//se incrementa en uno la prioridad de todos los que están en la cola

for j=1 to cola.size {

cola[j].prioridad ++;

}

}

} // VIDEOCONFERENCIA

###### Process Persona [id=1..P] {

**VAR**

int prioridad;

**BEGIN**

prioridad = *dameLaPrioridad*();

Videoconferencia.pedir(prioridad, id);

//usa el equipo de videoconferencia

Videoconferencia.liberar()

} // PERSONA

###### Process Administrador {

**BEGIN**

while true {

delay(180);

VideoConferencia.actualizarPrioridad()

}

} // ADMINISTRADOR