4.

| Process persona [id: 1..N]{  int idE;  Banco.llegada(idE);  end  Process Empleado[id: 1..2]  while (true)  Banco.próximo(id);  end  Process Contador[id:1..N]  Timer. Iniciar();  delay (15minutos); //  Timer.Terminar();  } | Monitor Banco {  cola elibres;  cond esperaP;  int esperando = 0, cantLibres = 0;  Procedure Llegada(idE: out int)  if (cantLibres == 0){ esperando ++;  wait (esperaP);//\*}  else  cantLibres--;  pop(elibres, idE);  }  Procedure Próximo(idE: in int)  push(elibres, idE);  if (esperando > 0 ){ esperando --;  signal (esperaP);}  else  cantLibres++;  **end monitor**  //  Monitor Timer  Procedure Iniciar ()  //  end  Procedure Terminar ()  { signal(esperaP);  end |
| --- | --- |

-------------

Monitor AdminPuente{

boolean libre= true;

int cant=0;

cond cola;

Procedure entrarPuente(){

if(not libre) {cant++; wait(cola);}

else libre=false;

}

Procedure salirPuente(){

if(cant>0) { cant--; signal(cola)}

else libre=true;

}

}

Process Auto [a:1.. M]

AdminPuente.entrarPuente();

usar puente;

AdminPuente.saliPuente();

End Process;

-------------------------------------------------------------------------------------------------------

\*ejercicio para cuando el orden está dado por otra cosa que no es el de llegada \*

Monitor Puente

cond cola[N];

int libre= True;

ColaOrdenada filaAutos:

Procedure entrarPuente (int au)

if(not libre) {filaAutos.push(au)

wait (cola[au]);}

else: libre=False

end;

Procedure salirPuente (int au)

if (not empty (colaAutos) ) {

auto=filaAutos.pop()

signal(cola[auto]); }

else: libre = True

end;

End Monitor;

Process Auto [a:1..M]

Puente. entrarPuente (a);

“el auto cruza el puente”

Puente. salirPuente(a);

End Process;

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.

Monitor Estado[id: 1..N] {

String estado = "esperando";

cond vc\_empleado, vc\_cliente;

boolean listo = false;

boolean cliente\_listo = false;

Documento documento\_escritorio;

Ticket ticket\_escritorio;

Procedure cambiar\_timeout(){

if(estado == "esperando"){

estado = "timeout";

cliente\_listo = true;

signal(vc\_cliente);

}

}

Procedure cambiar\_atendiendo(gone : out boolean){

if(estado == "esperando"){

estado = "atendiendo";

cliente\_listo = true;

signal(vc\_cliente);

}else{

gone = true;

}

}

Procedure espera\_cliente(doc : out Documento){

if(!listo){

wait(vc\_empleado);

}

doc = documento\_escritorio;

}

Procedure entrega\_documentos(documento: in Documento, tkt: out Ticket){

documento\_escritorio = documento;

listo = true;

signal(vc\_empleado); //avisa que dejo el documento

//deja el documento y espera que el empleado lo agarre

wait(vc\_cliente); //espera que el empleado lo procese

tkt = ticket\_escritorio;

signal(vc\_empleado); //le aviso al empleado que agarre el ticket

}

Procedure entrega\_ticket(tkt: in Ticket){

ticket\_escritorio = tkt;

signal(vc\_cliente);

wait(vc\_empleado); //espero a que el cliente agarre el ticket

//no importa volver a setear el empleado listo a false porque es individual a cada monitor de estado

}

//este procedimiento sirve para dormir al cliente hasta ser despertado por

//el timer o por el Empleado

Procedure esperar(estado\_out : out string){

if(!cliente\_listo)

wait(vc\_cliente);

estado\_out = estado;

}

}

Monitor Timer[id:1..N]{

boolean timer\_listo = false;

Cond timer;

Procedure init\_timer(){

if(!timer\_listo){

wait(timer);

}

}

Procedure comenzar\_timer(){

timer\_listo = true;

signal(timer);

}

}

Monitor Banco {

Queue clientes;

Cond empleados;

Procedure llegada(idC : in int){

clientes.enqueue(idC); //se encola

signal(empleados); //le avisa al empleado que hay gente en la cola

}

Procedure proximo(id\_cliente : out int){

//tengo más de un cajero, es importante que vuelvan a chequear que la cola tenga gente antes de poder pasar

//a desencolar

while(clientes.empty()) wait(empleados); //si no hay gente para atender el empleado espera aca

id\_cliente = clientes.dequeue();

}

}

Process Cliente[id:1..N]{

int id\_empleado; Documento documento; Ticket ticket;string estado\_out;

Timer[id].comenzar\_timer();

Banco.llegada();

Estados[id].esperar(estado\_out);

if(estado\_out == "atendiendo"){

Estados[id].entrega\_documentos(documento, ticket); //

}

}

Process Timer[id: 1.. N]{

Timer[id].init\_timer();

delay(15);

Estados[id].cambiar\_timeout();

}

Le

Process Empleado[id: 1..2]{

Documento documento;

boolean gone;

int id\_cliente;

while(true){

gone = false; //esta variable cambia si el cliente ya se fue

Banco.proximo(id\_cliente);

Estados[id\_cliente].cambiar\_atendiendo(gone);

if(!gone){ //si el cliente no se fue, lo atendemos

Estado[id].espera\_cliente(documento);

Ticket t = procesar(documento);

Estado[id].entrega\_ticket(t);

}

}

}

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[id:1..100]{  maquinaDeAdn.permisoParaUsarla(id)  //analizar adn  maquinaDeAdn.termineDeUsarla()  } | **monitor maquinaDeAdn**  bool libre = true; cond espera[N];  int idAux, esperando = 0; colaOrdenada cola;  Procedure PermirsoParaUsarla (idP)  { if (not libre) { cola.encolar(idP);  esperando ++;  wait (espera[idP]);}  else  libre = false;  }  Procedure termineDeUsarla ()  { if (esperando > 0 ) { esperando --;  cola.desencolar(idAux);  signal (espera[idAux]);}  else  libre = true;}  end  **end monitor** |
| --- | --- |

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

Monitor BD

cond cola;

int cant= 0;

Procedure entrarBD (int id)

while ( cant == 5) wait (cola); # si hay 5 en cant tiene que esperar

cant = cant + 1;

end;

Procedure salirBD (int id)

cant = cant – 1;

signal(cola);

end;

End Monitor;

Process Lector [id:1..M]

Puente. entrarBD (id);

“lee BD”

Puente. salirBD(id);

End Process;

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.

| **Process Cliente[id: 1..N]**  text comprobante,planilla;    oficinaDeRegistros.encolarse(id);  completar(planilla);  oficinaDeRegistros.esperar(id);  caja.atención(planilla,comprobante);  **end**  **Process Empleado::**  text planilla;  while (true){  oficinaDeRegistros.próximo();  caja.esperarDatos(planilla);  comprobante = resolver solicitud en base a datos  caja.enviarComprobante(comprobante);  **end** | **Monitor oficinaDeRegistros {**  cond empleado, clientes[1..N];  queue cola();  bool llamados[1..N] = ([1..N], false);  **Procedure encolarse(idP)**  **{**  cola.encolar(idP);  signal(empleado);  **}**  **Procedure esperar(idP)**  **{**  if (! llamados[idP]) wait(clientes[idP]);  **}**  **Procedure próximo()**  {  if (empty(cola)) → wait(empleado)  nro=cola.desencolar();  llamados[nro] = True  signal(clientes[nro])  }  **end monitor**    **Monitor caja**  cond vcCliente, vcEmpleado;  text planilla, comprobante;  boolean listo = false;  **Procedure Atención(P: in text; C: out text)**  {  planilla = P;  listo = true;  signal (vcEmpleado);  wait (vcCliente);  C = comprobante;  signal (vcEmpleado);  **}**  **Procedure Esperardatos(P: out text)**  { if (not listo)-> wait (vcEmpleado);  P = planilla;  **}**  **Procedure EnviarComprobante(C: in text)**  comprobante = C;  signal (vcCliente);  wait (vcEmpleado);  listo = false;  **}**  **endMonitor;** |
| --- | --- |

9. Hay N vehículos (cada uno tiene su propio peso) que deben pasar por un puente de acuerdo al orden de llegada, pero el puente no soporta más de 50000kg. Nota: maximizar concurrencia. Suponga que ningún vehículo supera el peso soportado por el puente.

Process vehiculo[id:1..N]{

double peso;

Puente.entrar(peso)

#pasa el puente

Puente.Salir(peso)

}

Monitor Puente {

double pesoActual = 0;

//N var condicion? Si

cond[N] dormidosEncolados

cola colaEsperando;

procedure entrar(in pesoAuto){

if (cantEncolados > 0) or (pesoAuto + pesoActual > 50000){

//ah cierto, “en orden” (dbates)

colaEsperando.push(idA, pesoAuto);

cantEncolados++;

wait(dormidosEncolados[idA]);

cantEncolados--;

}

pesoActual = pesoActual + pesoAuto;

}

procedure salir(in pesoAuto){

double proximoPeso;

pesoActual -= pesoAuto;

proximoPeso=pesoActual;

while 50000 > proximoPeso{

if colaEsperando.Empty{

break }

idA, pesoA=colaEsperando.top();

o directamente sumar proxpeso aca?

if proximoPeso + pesoA < 50000{

signal(dormidosEncolados[idA]);

colaEsperando.pop();

proximoPeso=proximoPeso + pesoA;

}

}

}

}

------ Versión Bigu -----! El + Crack <3 :V😎🙉

Process vehiculo[id:1..N]{

double peso;

Puente.entrar(peso)

#pasa el puente

Puente.Salir(peso)

}

Monitor Puente {

double pesoActual = 0;

//N var condicion? Si

cola colaEsperando;

procedure entrar(in pesoAuto){

if (cantEncolados > 0) and (pesoAuto + pesoActual > 50000){

colaEsperando.push(idA, pesoAuto);

cantEncolados++;

wait(dormidosEncolados);

pesoActual = pesoActual + pesoAuto;

cantEncolados--;

if (cantEncolados > 0) {

idA,pesoA=colaEsperando.top();

if(pesoActual + pesoA < 50000){

signal(dormidosEncolados);

colaEsperando.pop();

}

}

}else{

pesoActual = pesoActual + pesoAuto;

}

}

procedure salir(in pesoAuto){

pesoActual -= pesoAuto;

iif (cantEncolados > 0) {

idA,pesoA=colaEsperando.top();

if(pesoActual + pesoA < 50000){

signal(dormidosEncolados);

colaEsperando.pop();

}

}

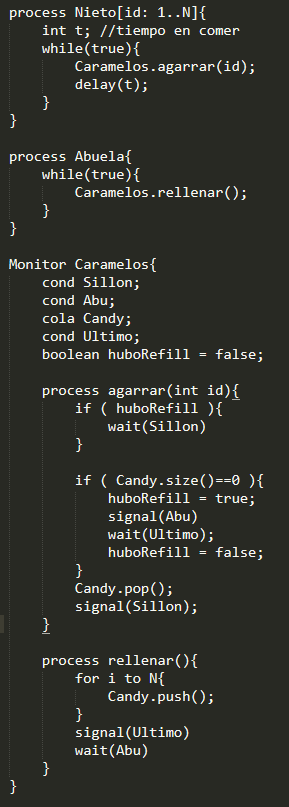
}

}

6. 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.

No se si esta bien jeje

Opción 1



-----------------------------------------------------O-----------------------------------------------------

**Opcion 2 - la mejor pa 💣💥**

**Monitor organizador{**

int fuenteCaramelos=0;

cond inicio;

bool iniciar,reponer,alguienLlamoALaAbuela = false;

procedure **iniciar**(in caramelos){

fuenteCaramelos = caramelos;

iniciar = true;

signal\_all(inicio);

}

procedure **esperarIniciar**(){

if not iniciar{

wait(inicio)

}

}

procedure **depositar**(in caramelos) {

if not reponer

wait(vcAbuela)

reponer=false

fuenteCaramelos = caramelos;

signal(primero)

}

procedure **tomarCaramelo**(out caram){

while alguienLlamoALaAbuela{

wait(esperandoFuente)

}

if fuenteCaramelos == 0{

alguienLlamoALaAbuela = true;

reponer=true;

signal(vcAbuela);

wait(primero);

signal\_all(esperandoFuente)

alguienLlamoALaAbuela=false

}

fuenteCaramelo--;

caram = fuenteCaramelos.sacarUno();

}

**process Abuela {**

int caramelos

organizador.iniciar()

while true{

caramelos = #Compro Caramelos sugu

organizador.depositar(caramelos);

**}**

**process Nieto[id= 1..N]{**

int t;

t=random()

organizador.esperarIniciar()

while true {

organizador.tomarCaramelo()

delay(t) #comer caramelo

}

**}**

-----------------------------------------------------O-----------------------------------------------------

Opcion 3: el mio quedo muy sencillo asi que debo haber caido en una trampa :c

Monitor Fuente{

cola caramelos;

cond nietos;

cond ultimo;

cond abuela;

int cant\_n = 0;

Procedure tomar\_caramelo(caramelo: out elem){

while (cant\_n > 0) wait(nietos);

cant\_n = cant\_n + 1;

if (caramelos.empty()){

signal(abuela)

wait(ultimo);

};

caramelo = caramelos.pop();

cant\_n = cant\_n - 1;

signal\_all(nietos);

}

Procedure poner\_caramelos(caramelos\_in: in cola){

for (i:1..caramelos\_in.length){

caramelos.push(caramelos\_in.pop());

};

signal(ultimo);

}

//no sabía si vale la pena hacer un procedure solo para esto

Procedure esperar\_vacio(ultimo\_o: out int){

wait(abuela);

}

}

Process Abuela{

cola caramelos;

while(true){

caramelos = generar\_caramelos();

Fuente.poner\_caramelos(caramelos);

Fuente.esperar\_vacio();

}

}

Process Nieto[id:1..N]{

elem caramelo;

while(true){

fuente.tomar\_caramelo(caramelo);

comer(caramelo);

}

}

opcion 4 victoria (yo entendi que comian todo el tiempo)(esta re mal ni lo miren)

NO, HICE CUALQUIERA

ESTA MAL NO ME FIJE DEL PRIMER NIETO QUE NO COMIO

PROCESS NIETO [ID: 1..N]{

While (true){

Fuente.agarrarCaramelo();

//comer caramelo???

delay(tiempo?); ??? ver!!!

}

}

PROCESS ABUELA {

While (true){

Fuente.recargarCaramelos();

}

}

MONITOR Fuente {

int caramelos= x;

cond esperaCaramelos;

cond avisaAbuela;

PROCEDURE agarrarCaramelo(){

if(caramelos>0){

caramelos--;

signal(esperaCaramelos);

}else{

signal(avisaAbuela);

wait(esperaCaramelos);

caramelos--; ???? esta bien o pregunto otra vez si hay caramelos?

signal(esperaCaramelos);

}

}

PROCEDURE recargarCaramelos(){

wait(avisaAbuela);

caramelos=X;???? esta bien? **(más dudas que caso nisman)**

}

}

**8. 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).**

**#respeta enunciado, 0 concurrencia (más abajo otra solucion)**

process alumno()[idA:1..50]{

int idG;

int t;

organizarInicio.iniciarAlumno(idA, idG);

random(t);

#Delay(t) #Hace la tarea

grupo[idG].entregar()

}

process JTP(){

organizarInicio.iniciarJTP();

entregas.esperandoGrupos()

}

Monitor organizarInicio

int cantAlumnos;

cond esperandoATodos[1..50]

cond JTPespera;

cola 50alumnos;

int numG;

procedure iniciarAlumno(in idA; out idG){

cantAlumnos++;

50alumnos.push(idA)

if cantAlumnos < 50{

wait(esperandoATodos[idA]);}

else{

signal(JTPespera);

idG = numG;

signal(JTPespera);

}

procedure iniciarJTP(){

if cantAlumnos < 50{

wait(JTPespera);

}

for i: 1..50{

idA= cola.pop();

numG= DarNumero();

signal(esperandoATodos[idA]);

wait(JTPespera);

}

}

}

Monitor entregas{

int vectorEntregas[1..25] = 0

int nota=25;

int notas[1..25];

queue colaEntregas;

cond esperandoNota[1..25];

procedure darNota(out notaEntregar){

notaEntregar = nota;

nota--;

}

produre avisarEntregaGrupal(in idG, out notaFinal){

colaEntregas.push(idG)

signal(esperaJTP);

wait(esperandoNota[idG]);

notaFinal=notas[idG];

}

prodecure esperandoGrupos(){

while cantEquiposTerminados < 25{

if colaEntregas.empty()

wait(esperaJTP);

idG=colaEntregas.pop()

vectorEntregas[idG]++; #representa cant de alumnos x grupo que entregaron

if vectorEntregas[idG] == 2{

cantEquiposTerminados ++;

notas[idG]=darNota()

signal\_all(esperandoNota[idG]);

}

}

}

}

**# Opcion2 :No respeta el enunciado pero maximiza la concurrencia**

process alumno()[idA:1..50]{

int idG;

int t;

JTP.iniciarAlumno(idG);

random(t);

#Delay(t) #Hace la tarea

grupo[idG].entregar()

}

Monitor JTP{

int vectorEntregas[1..25] = 0

int nota=25;

procedure DarNumero(out grupo)

#viene implementado

procedure iniciarAlumno(in idA; out idG){

cantAlumnos++;

if cantAlumnos < 50{

wait(esperandoATodos);

else{

signal\_all(esperandoATodos);

idG=DarNumero(idG)

}

procedure darNota(out notaEntregar){

notaEntregar = nota

nota--

}

}

Monitor grupo[id=1..25]{

int cantAlumnos=0; #sorry

cond esperarNota;

int notaGrupo;

procedure entregar(out nota)

cantAlumnos++;

if cantAlumos == 2 { # no cumple el enunciadoo (el jtp deberia regular que lleguen los 2)

JTP.darNota(id,notaGrupo)

signal(esperarNota)

}

wait(esperarNota)

nota=notaGrupo

}

```java

Monitor Jefe{

int notas[25] = [25](0);

int notas\_cant[25] = [25](0);

cond vc\_espera\_nota[25];

int nota\_actual = 25;

int cant = 0;

cond vc\_espera;

Procedure recibir\_numero(nro : out int){

cant++;

if(cant == 50){

signal\_all(vc\_espera);

}else{

wait(vc\_espera);

}

nro = DarNumero();

}

Procedure corregir(nro\_grupo : in int, nota : out int){

notas\_cant[nro\_grupo]++;

if(notas\_cant[nro\_grupo] == 2){

notas[nro\_grupo] = nota--;

signal(vc\_espera\_nota[nro\_grupo]);

}else{

wait(vc\_espera\_nota[nro\_grupo])

}

nota = notas[nro\_grupo];

}

}

Process Alumno[id: 1..50]{

int nro\_grupo;

Jefe.recibir\_numero(nro\_grupo);

Jefe.corregir(nro\_grupo, nota)

}

**7. En un entrenamiento de futbol 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).**

process jugador[20]::{

int equipo; int cancha;

equipo=DarEquipo()

organizador[equipo].anunciar\_llegada(cancha)

#Voy a la cancha🇵🇪

organizarPartido[cancha].anunciar\_llegada()

}

process partido[2]{

organizarPartido[id].iniciar\_partido()

delay(50min)

organizarPartido[id].terminar\_partido()

}

Monitor organizadorEquipo[idE:1..4]{

int jugadoresTotal= 0;

cond lleguenTodos;

int cancha

procedure anunciar\_llegada(out cancha\_devolver){

jugadoresTotal++;

if jugadoresTotal == 5{

definirCancha.llego\_el\_equipo(cancha)

cancha\_devolver=cancha

signal\_all(lleguenTodos)

}

else {wait(lleguenTodos);}

}

}

Monitor organizarPartido[idP:1..2]{

int jugadoresTotales,prePitido=0

cond lleguenTodos,partido,jugar,esperandoPartido;

procedure anunciar\_llegada(){

jugadoresTotales++;

if jugadoresTotales== 10{

signal\_all(esperandoPartido)

signal(partido)

}

else {wait(esperandoPartido)}

wait(jugar)

/\*prePitido++;

if prePitido== 10{

signal(partido)}

wait (jugar) \*/

}

procedure iniciar\_partido()

if jugadoresTotales !=10{

wait(partido)

}

procedure terminar\_partido()

signal\_all(jugar)

}

}

Monitor definirCanchas{

int equipos\_que\_llegaron=0

int nro\_de\_cancha ←? esta de más esta no?

procedure llego\_el\_equipo(out nro\_de\_cancha){

equipos\_que\_llegaron ++;

if equipos\_que\_llegaron <=2

nro\_de\_cancha=1;

else

nro\_de\_cancha=2;

}

}

procesos A1 A2 A3 A4 A5

procesos B1 B2 B3 B4 B5

procesos C1 C2 C3 C4 C5

procesos D1 D2 D3 D4 D5

Opcion: que quilombo matenme

Monitor Entrenamiento{

cond equipos[4];

int contador[4] = 0;

Procedure Llegar(equipo: in int){

contador[equipo-1] = contador[equiupo-1] + 1;

if (contador = 5){

signal\_all(equipos[equipo-1]);

} else {

wait(equipos[equipo-1]);

};

};

}

Monitor Canchas[id:0..1]{

bool ocupada = false;

cond esperando;

int equipos[2] = 0;

int contador[2] = 0;

cond iniciar;

Procedure LlegarCancha(equipo: in int; jugo: out bool){

if ((equipos[0] = 0) or (equipos[1] = 0)){

if (equipos[0] = 0){

equipos[0]=equipo;

contador[0] = 1;

jugo = true;

wait(esperando);

} else {

equipos[1] = equipo;

contador[1] = 1;

jugo = true;

wait(esperando);

};

} else if ((equipos[0]=equipo) or (equipos[1]=equipo)){

if (equipos[0] = equipo){

contador[0] = contador[0] + 1;

jugo = true;

if ((contador[0]=5) and (contador[1]=5)){

signal(inicio);

} else {

wait(esperando);

}

} else {

contador[1] = contador[1] + 1;

jugo = true;

if ((contador[0]=5) and (contador[1]=5)){

signal(inicio);

} else {

wait(esperando);

};

};

};

};

Procedure Iniciar(){

if ((contador[0]<5) and (contador[1]<5)){

wait(iniciar);

};

};

Procedure Terminar(){

signal\_all(esperando);

}

}

Process Partido[id:0..N-1]{

Cancha[id].Iniciar();

delay(50);

Cancha[id].Terminar();

}

Process Jugador[id:0..N-1]{

bool jugo = false;

int equipo = darEquipo();

Entrenamiento.Llegar(equipo);

Canchas[1].LlegarChancha(equipo, jugo);

if (!jugo) {

Canchas[2].LlegarChancha(equipo, jugo);

}

}