SCD - Miguel Lentisco Ballesteros

Monitor 1

Se plantea la siguiente situación: una clase donde el profesor empieza a escribir en la pizarra (de tamaño determinado) su temario, donde unos alumnos (número dado) empiezan a copiar cada uno a su ritmo. Obviamente el profesor no puede borrar cosas si TODOS los alumnos han copiado la parte que quiere borrar, y habrá alumnos que copien más rápido que otros. Además el profesor cuando acabe de escribir se irá, los alumnos también conforme hayan terminado de copiar todo.

Planteo la siguiente solución con un monitor:

```
class monitor MonitorClase(t_pizarra, n_alum : integer);
begin class
var tam_pizarra   : integer;  // Tamaño de la pizarra
    pizarra       : String[tam_pizarra]; // La pizarra (buffer)
    lec_alumnos   : integer[n_alum]; // Pos de lectura de cada alumno
    pos_esc       : integer; // Posicion donde se va a escribir
    claseAcabada  : boolean; // Si la clase ha acabado
    colaProfesor  : condition; // Para el profesor si ha llegado al tope
    colaAlumnos   : condition; // Para los alumnos si el profesor aun no escribe

export haAcabado, acabarClase, escribirPizarra, leerPizarra;

function heAcabado(id_alumno : integer) : boolean;
begin
  return claseAcabada and pos_esc == lec_alumnos[id_alumno];
end

procedure acabarClase();
begin
  claseAcabada := true;
end

procedure escribirPizarra(palabra : String);
begin
  if pos_esc - lec_alumnos.min() < tam_pizarra then
    colaProfesor.wait();
  endif
  pizarra[pos_esc % tam_pizarra] := palabra;
  pos_esc := pos_esc + 1;
  colaAlumnos.signal();
end

function leerPizarra(id_alumno : integer) : String;
```

```
begin
  if pos_esc == lec_alumnos.min() then
    colaAlumnos.wait();
    colaAlumnos.signal();
  endif
  lec_alumnos[id_alumno] := lec_alumnos[id_alumno] + 1;
  if lec_alumnos.min() == lec_alumnos[id_alumno] then
    colaProfesor.signal();
  endif
  return pizarra[lec_alumnos[id_alumno] - 1];
end

begin
  claseAcabada := false;
  tam_pizarra := t_pizarra;
  pizarra := String[tam_pizarra];
  lec_alumnos := String[n_alum];
  pos_esc := 0
end
end class

/* ********************************************************************** */

process Profesor(M : MonitorClase, apuntesProfesor : String[tam_apuntes]);
var palabra   : String;
    n_palabra : integer;
    i         : integer;
begin
  for i := 0; i < apuntesProfesor.size(); i := i + 1; do
    palabra = apuntesProfesor[i];
    // Añadir tiempo random
    M.escribirPizarra(palabra);
  done
  M.acabarClase();
end

process Alumno(M: MonitorClase, cuadernoAlumno : String[id_alumno][], id_alumno
  : integer);
var palabra : String;
    i       : integer;
begin
  i := 0;
  while !M.heAcabado() do
    palabra = leerPizarra(id_alumno);
    // Añadir tiempo random
    cuadernoAlumno[id_alumno][i] := palabra;
    i := i + 1;
  done
end
```

Monitor 2

Un sistema informático se encarga de recibir peticiones de libros por parte de un numero de clientes fijos: cuando recibe la petición, entra en la base de datos y empieza a imprimir la información para el cliente; sin embargo, si se está imprimiendo el sistema esta ocupado completamente, luego los clientes tienen que esperar una cola. Después de recibir los datos, los clientes empiezan a leer lo que desean y siguen realizando más peticiones hasta que las completen todas;

El monitor seria:

```
class monitor MonitorDatos(t_clientes : integer);
begin class
var ocupado       : boolean;
    datos         : String[];
    peticion      : integer;
    n_clientes    : integer;
    colaSistema   : condition;
    colaImprimir  : condition;
    colaSolicitud : condition;

export hayClientes, clienteAdios, solicitarDatos, imprimeInformacion,
recibirPeticion;

function hayClientes() : boolean;
begin
  return n_clientes > 0;
end

procedure clienteAdios();
begin
  n_clientes := n_clientes - 1;
end

function solicitarDatos(i : integer) : String[];
begin
  if ocupado then
    colaSolicitud.wait();
  end if
  colaSistema.signal();
  ocupado := true;
  peticion := i;
  colaImprimir.wait();
  colaSolicitud.signal();
  colaSistema.signal();
  ocupado := false;
  return datos;
end

procedure imprimeInformacion(impreso : String[]);
begin
  datos := impreso;
  colaImprimir.signal();
  colaSistema.wait();
```

```
end

function recibirPeticion() : integer;
begin
  if !ocupado then
    colaSistema.wait();
  end if
  return peticion;
end

begin
  ocupado := false;
  n_clientes := t_clientes;
end
end class

/* ********************************************************************** */

process Sistema(M : MonitorDatos);
var baseDatos : String[][];
    i         : integer;
    datos     : String[];
begin
  while M.hayClientes() do
    i := M.recibirPeticion();
    datos := baseDatos[i];
    // Tiempo random
    M.imprimirInformacion(datos);
  done
end

process Cliente(M : MonitorDatos, peticiones : integer[]);
var i       : integer;
    datos   : String[]

begin
  for i := 0; i < peticiones.size(); i := i + 1; do
    datos = M.solicitarDatos(peticiones[i]);
    // Añadir tiempo random, simular lectura de datos
  done
  M.clienteAdios();
end
```

Monitor 3

En este problema tenemos un capitán que maneja una serie de catapultas y cada catapulta tiene un número determinado de disparos. Para poder disparar, todas la catapultas que tengan disparos disponibles tienen que cargar, y cuando esten todas cargadas, el capitán da la orden para disparar. Después de cada lanzamiento, el capitán se toma un tiempo de descanso; si después

aun no están cargadas, se espera. Si una catapulta se queda sin disparos, entonces se retira sin afectar a las restantes.

La solución sería;

```
class monitor MonitorPeticion(n_catap : integer);
begin class
var n_catapultas      : integer;
    n_cap_listas      : integer;
    colaCapitan       : condition;
    colaCatapultas    : condition;

export sinDisparos, hayDisparos, dispararCatapultas, solicitarDisparo;

procedure sinDisparos();
begin
  n_catapultas := n_catapultas - 1;
  if n_catapultas == n_cap_listas then
    colaCapitan.signal();
  end if
end

function hayDisparos() : boolean;
begin
  return n_catapultas > 0;
end

procedure dispararCatapultas();
begin
  if n_catapultas != n_cap_listas then
    colaCapitan.wait();
  end if
  colaCatapultas.signal();
end

procedure solicitarDisparo();
begin
  n_cap_listas := n_cap_listas + 1;
  if n_catapultas == n_cap_listas then
    colaCapitan.signal();
  end if
  colaCatapultas.wait();
  colaCatapultas.signal();
end

begin
  n_catapultas := n_catap;
  n_cap_listas := 0;
end
end class

/* ********************************************************************** */
```

```
process Capitan(M : MonitorPeticion);
begin
  while M.quedanDisparos() do
    // Tiempo de descanso, tiempo random
    M.dispararCatapultas();
  done
end

process Catapultas(M : MonitorPeticion, n_disparos: integer);
var i : integer;
begin
  for i := 0; i < n_disparos; i := i + ; do
    // Carga de catapulta, espera tiempo random
    M.solicitarDisparo();
  done
  M.sinDisparos();
end
```