Posibles soluciones a los ejercicios del parcial práctico de memoria distribuida del 25-11-24

Importante: las soluciones que se muestran a continuación no son las únicas que se pueden considerar correctas para los ejercicios planteados.

1) La UNLP está organizando su maratón anual y P personas que deben pasar a retirar su remera y chip antes de la carrera. Para ello, los corredores deben dirigirse al edificio de Rectorado, donde habrá un organizador que los atenderá. Los corredores son atendidos de acuerdo con el orden de llegada, teniendo prioridad los corredores ancianos sobre los jóvenes y adultos. Adicionalmente, los corredores con discapacidad tienen prioridad sobre todos los anteriores. Para ser atendidos, los corredores entregan su DNI al organizador, quien como respuesta les entrega la remera oficial y el chip asociado. Implemente un programa que permita resolver el problema anterior usando **PMA**. **Notas**: para conocer su prioridad, cada corredor puede llamar a la función *obtenerPrioridad()*, la cual retorna 0 si la persona tiene alguna discapacidad, 1 si es anciana, o 2 en otro caso; la función *obtenerRyC(dni)* retorna la remera y el chip asociado para el DNI recibido como entrada; la función *obtenerDNI()* retorna el DNI para el corredor que la invoca.

```
chan atencionGeneral (int,int);
chan atencionConDiscapacidad (int,int);
chan atencionAnciano (int,int);
chan respuesta [P] (remera, chip);

Process Corredor [i=0..P-1] {

      Remera r; Chip c;
      int prioridad = obtenerPrioridad();
      int dni = obtenerDNI();
      // según la prioridad, mando mensaje en canal correspondiente
      if (prioridad == 0)
            send atencionConDiscapacidad(i,dni);
      elif (prioridad == 1)
            send atencionAnciano(i,dni);
      else
            send atencionGeneral(i,dni);
      fi
      // espero respuesta en canal propio
      receive atencion[i] (r,c);
}
```

```
Process Organizador {
      Remera r; Chip c;
      int dni,id;
      while (true) {
            if (not empty(atencionConDiscapacidad))    ->
                  receive atencionConDiscapacidad(id, dni);
                  (remera,chip) = obtenerRyC(dni);
                  send respuesta[id] (remera,chip);
            // sólo atiendo si no hay mensajes de prioridad superior
            [] (not empty(atencionAnciano)) && (empty(atencionConDiscapacidad)) ->
                  receive atencionAnciano(id, dni);
                  (remera,chip) = obtenerRyC(dni);
                  send respuesta[id] (remera,chip);
            // sólo atiendo si no hay mensajes de prioridad superior
            [] (not empty(atencionGeneral) && (empty(atencionAnciano)) &&
(empty(atencionConDiscapacidad))  ->
                  receive atencionGeneral(id, dni);
                  (remera,chip) = obtenerRyC(dni);
                  send respuesta[id] (remera,chip);
            fi
      }
 }
```

Alternativa: la solución anterior implica hacer busy waiting en el Organizador, en el caso de que no hubiera mensajes pendientes. Si bien la solución es Aceptable (en este caso particular), es posible evitar el busy waiting mediante el uso de un canal "señalizador". Los corredores deben enviar primero en el canal correspondiente y luego en el señalizador. El Organizador se bloquea en el señalizador primero y luego consulta disponibilidades según prioridad. Si bien se elimina el busy waiting, la consecuencia es un mayor número de mensajes intercambiados.

```
chan atencionGeneral (int,int);
chan atencionConDiscapacidad (int,int);
chan atencionAnciano (int,int);
chan respuesta [P] (remera, chip);
chan señalizador ();

Process Corredor [i=0..P-1] {
      Remera r; Chip c;
      int prioridad = obtenerPrioridad();
      int dni = obtenerDNI();
      // según la prioridad, mando mensaje en canal correspondiente
      if (prioridad == 0)
            send atencionConDiscapacidad(i,dni);
      elif (prioridad == 1)
            send atencionAnciano(i,dni);
      else
            send atencionGeneral(i,dni);
      fi
      // envío señal para avisar que hay mensaje pendiente de atencion
      send señalizador();
      // espero respuesta en canal propio
      receive atencion[i] (r,c);
}
```

```
Process Organizador {

      Remera r; Chip c;
      int dni, id;

      while (true) {
            receive señalizador();
            if (not empty(atencionConDiscapacidad))   ->
                  receive atencionConDiscapacidad(id,dni);
            // sólo atiendo si no hay mensajes de prioridad superior
            [] (not empty(atencionAnciano)) && (empty(atencionConDiscapacidad)) ->
                  receive atencionAnciano(id,dni);
            // sólo atiendo si no hay mensajes de prioridad superior
            [] (not empty(atencionGeneral) && (empty(atencionAnciano)) &&
(empty(atencionConDiscapacidad))  ->
                  receive atencionGeneral(id,dni);
            fi
            (remera,chip) = obtenerRyC(dni);
            send respuesta[id] (remera,chip);

      }
}
```

2) Las unidades postales (UP) son sucursales de correo que se ocupan habitualmente de recibir y entregar paquetes. En particular, la UP 16 sólo se ocupa de entregar paquetes a las personas que se presentan a retirarlos y cuenta con un único empleado. Como los clientes pueden tener que esperar para su atención, muchos aprovechan la plaza que está frente a la UP16 para hacer ejercicio. Los clientes son atendidos por orden de llegada y se dividen en dos clases: los clientes "pacientes" que esperan a lo sumo 10 minutos a que el empleado los atienda y, pasado ese plazo, dan una vuelta a la plaza; y los clientes "ansiosos" que exigen atención inmediata y sino se van a dar una vuelta a la plaza. Más allá del tipo de cliente, luego de 3 intentos sin lograr que lo atiendan, el cliente se retira sin el paquete. El proceso de atención consiste en que el empleado le entrega un paquete al cliente, dado el código QR provisto por éste. Implemente un programa que permita modelar el funcionamiento de la UP 16 usando **ADA**, asumiendo que hay N1 clientes "pacientes" y N2 clientes "ansiosos", y que todos retiran un único paquete. Además, asuma que no habrá códigos QRs erróneos ni paquetes faltantes. **Notas**: la función *obtenerQR()* retorna el código QR asociado al cliente que la invoca; *obtenerPaquete(qr)* retorna el paquete asociado al QR recibido como entrada; la función *caminata()* modela la vuelta a la plaza.

```
Procedure UP16 IS

TASK Type ClienteAnsioso;
TASK Type ClientePaciente;
TASK Empleado IS
      entry atencion (in qr: CodigoQR, out paq: Paquete);
end Empleado;
emp: Empleado;
clientesP: array (1..N1) of ClientePaciente;
clientesA: array (1..N2) of ClienteAnsioso;

TASK Body ClientePaciente IS
      qr: CodigoQR;
      atendido: bool := false;
      intentos: integer := 0;
Begin
      qr = obtenerQR()
      while (intentos < 3) AND (atendido == false) loop
            SELECT
                  -- Pedir atención
                  Emp.atencion(qr,paq);
                  atendido := true;
            OR DELAY (10*60)
                  -- dar vuelta a la plaza;
                  caminata();
                  -- cuenta el intento
                  intentos++;
            End SELECT;
      end loop
End ClientePaciente;
```

```
TASK Body ClienteAnsioso IS
      qr: CodigoQR;
      atendido: bool := false;
      intentos: integer := 0;
Begin
      qr = obtenerQR()
      while (intentos < 3) AND (atendido == false) loop
            SELECT
                  -- Pedir atención
                  Emp.atencion(qr,paq);
                  atendido := true;
            ELSE
                  -- dar vuelta a la plaza;
                  caminata();
                  -- cuenta el intento
                  intentos++;
            End SELECT;
      end loop
End ClienteAnsioso;

TASK Body Empleado IS
Begin
      loop
            Accept atención (qr: in CodigoQR, paq: out Paquete) do
                  -- atender
                  paq = obtenerPaquete(qr);
            End atención
      End loop;
End Empleado;
begin
      null;
end UP16;
```