Posibles soluciones a los ejercicios del parcial práctico del 4-11-24

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

1) 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. Los clientes son atendidos por orden de llegada por el primer empleado que se encuentre libre. Al solicitar atención, el cliente le muestra un código QR al empleado que lo atiende, el cual le entrega el paquete como respuesta. Implemente un programa que permita modelar el funcionamiento de la UP 16 usando **PMA**, asumiendo que hay N clientes que retiran un único paquete y 3 empleados para atenderlos. Además, asuma que no habrá códigos QR erróneos ni paquetes faltantes. **Notas:** la función *obtenerQR(idCliente)* retorna el código QR asociado al identificador de cliente recibido como entrada. De forma similar*, obtenerPaquete(qr)* retorna el paquete asociado al QR recibido como entrada.

```
chan solicitud_atencion(int,string);
chan entrega_paquete[N](paquete);

Process Clientes [i=1..N] {
      string qr;
      Paquete paq;
      // obtener código QR
      qr = obtenerQR(i);
      // solicitar atención
      send solicitud_atencion(i,qr);
      // recibe paquete
      receive entrega_paquete[i] (paq);
}

Process Empleado [i=1..3] {
      string qr;
      Paquete     paq;
      int idCliente;
      while (true) {
            // recibe solicitud de atencion
            receive solicitud_atencion (idCliente,qr);
            // obtiene paquete para QR recibido
            paq = obtenerPaquete(qr)
            // entrega paquete a cliente
            send entrega_paquete[idCliente] (paq);
      }
}
```

2) Resuelva el mismo problema anterior pero ahora usando **PMS**.

```
Process Cliente [i=1..N] {
      string qr;
      Paquete paq;
      // obtiene QR
      qr = obtenerQR(i);
      // solicita atención
      Admin!solicitudAtencion(i,qr);
      // recibe paquete
      Empleado[*]?entregaPaquete(paq);
}

Process Admin {
      queue cola;
      paquete paq;
      int idCliente, idEmpleado;
      while (true) {
            // evalúa no determinísticamente
            if Cliente[*] ? solicitudAtencion(idCliente,qr) ->
                  push(cola,(idCliente,qr)); // si hay solicitud de atención de un
cliente, entonces lo encola
            [] (not empty(cola)); Empleado [*] ? solicitudTrabajo(idEmpleado) ->
                  (idCliente,qr) = pop(cola); // si hay solicitudes sin atender y
empleados libres, asigna trabajo
                  Empleado[idEmpleado] ! asignacionTrabajo (idCliente,qr);
        fi
      }
}

Process Empleado [i=1..3] {
      string qr;
      paquete paq;
      int idCliente;
      while (true){
            // avisa que puede trabajar al admin
            Admin!solicitudTrabajo(i);
            // recibe asignación de trabajo
            Admin?asignacionTrabajo(idCliente,qr);
            // obtiene paquete
            paq = obtenerPaquete(qr);
            // entrega paquete al cliente
            Cliente[idCliente]!entregaPaquete(paq);
      }
}
```

3) El problema *WordCount* consiste en calcular la cantidad de veces que aparece una palabra determinada en una porción, documento o archivo de texto. Implemente una solución a *WordCount* en ADA, considerando que se cuenta con una base de datos de texto distribuida en N partes. Además, existe un administrador que determina la palabra que se desea buscar y se la envía a las N tareas contadoras para que la busquen en la parte de la base de texto que poseen. Finalmente, el administrador calcula la cantidad total de apariciones. **Notas:** las tareas contadoras cuentan con la función *contarPalabras(pal)*, la cual retorna la cantidad de veces que aparece la palabra *pal* en su parte de la base de datos. El administrador cuenta con la función *obtenerPalabra()* que retorna la palabra a buscar. El problema se resuelve una única vez y su solución debe maximizar la concurrencia.

Alternativa 1:

```
Procedure WordCount is

Task Admin is
      entry Palabra (pal: out string);
      entry Cantidad (cant: in integer);
End admin;

Task type Contador;
ArrC: array (1..N) of Contador;

Task body Contador is
      pal: string;
      cant: integer;
Begin
      -- recibir palabra a buscar
      Accept palabra (palabra: in integer) do
            pal := palabra;
      end palabra;
      -- contar apariciones en base local
      cant := contarPalabras(pal);
      -- enviar cantidad parcial a Admin
      Admin.Resultado(cant);
End contador;

Task body Admin is
      palabra: string := obtenerPalabra;
      total: integer:= 0;
Begin
      -- enviar palabra a buscar a contadoras
      for i in N loop
            ArrC(i).palabra(palabra);
      end loop;
      -- recibir de cada tarea contadora y acumular
      for i in N loop
            accept Cantidad (res: in integer) do
                  total:= total + res;
            end Cantidad;
      end loop;
```

```
End Admin;
Begin
      null;
End WordCount;
```

Alternativa 2: La solución anterior es aceptable pero podrían producirse demoras si hay retardo en alguna de las tareas contadoras. La siguiente solución mejora lo anterior al combinar ambos entry's en el mismo select. De esa manera, se evita demora innecesaria por el potencial retardo de algún contador.

```
Procedure WordCount is

Task Admin is
      entry Palabra (pal: out string);
      entry Cantidad (cant: in integer);
End admin;

Task type Contador;
ArrC: array (1..N) of Contador;

Task body Contador is
      pal: string;
      cant: integer;
Begin
      -- solicitar palabra a buscar
      Admin.palabra(pal);
      -- contar apariciones en base local
      cant := contarPalabras(pal);
      -- enviar cantidad parcial a Admin
      Admin.Resultado(cant);
End contador;

Task body Admin is
      palabra: string := obtenerPalabra;
      total: integer:= 0;
Begin
      for i in 1..2*N loop
            select
                  -- enviar palabra a buscar a contadora
                  accept Palabra (pal: out integer) do
                        pal := palabra;
                  end Palabra;
            or
                  -- recibir cantidad parcial de tarea contadora y acumular
                  accept Cantidad (res: in integer) do
                        total:= total + res;
                  end Cantidad;
            end select;
      end loop;
End Admin;
Begin
      null;
```

```
End WordCount;
```