Posibles soluciones a los ejercicios del parcial práctico del 18-12-23

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

1. Resolver con **SEMÁFOROS** el siguiente problema. Se debe simular el uso de una máquina expendedora de gaseosas con capacidad para 100 latas por parte de U usuarios. Además existe un repositor encargado de reponer las latas de la máquina. Los usuarios usan la máquina según el orden de llegada. Cuando les toca usarla, sacan una lata y luego se retiran. En el caso de que la máquina se quede sin latas, entonces le debe avisar al repositor para que cargue nuevamente la máquina en forma completa. Luego de la recarga, saca una lata y se retira. **Nota**: maximizar la concurrencia; mientras se reponen las latas se debe permitir que otros usuarios puedan agregarse a la fila.

```
sem mutex = 1;
sem espera[C] = ([C] 0);
sem reponer = 0;
sem finReponer = 0;

bool libre = true;
queue colaMaq;
int cantLatas = 100;

Process Usuario[id: 0..U-1] {
  int i, auxU;

  P(mutex);
  if (not libre) {
        push(colaMaq, id);
        V(mutex);
        P(espera[id]);
  }
  else {
        libre = false;
        V(mutex);
  }
  if (cantLatas == 0) {
        V(reponer);
        P(finReponer);
  }
  //saca una lata
  cantLatas --;
  P(mutex);
  if (empty(colaMaq))
        libre = true
  else {
        pop(colaMaq, auxU);
        V(espera[auxU]);
  }
  V(mutex);
```

```
}

Process Repositor{
  while (true) {
        P(reponer);
        //reponer la máquina
        cantLatas = 100;
        V(finReponer);
  }
}
```

2) Resolver el siguiente problema con **MONITORES**. En una montaña hay 30 escaladores que en una parte de la subida deben utilizar un único paso de a uno a la vez y de acuerdo al orden de llegada al mismo. **Nota**: sólo se pueden utilizar procesos que representen a los escaladores; cada escalador usa sólo una vez el paso.

```
Process Escalador [id: 0..29]
{  Admin.SolicitarAcceso();
   //usa el paso
   Admin.LiberarAcceso();
}


Monitor Admin
{  cond espera;
   int cant = 0;
   bool libre = true;

   Procedure SolicitarAcceso()
   { if (not libre) { cant++;
                      wait (espera);
                    }
     else libre = false;
   }

   Procedure LiberarAcceso()
   { if (cant > 0) { signal (espera);
                     cant--;
                   }
     else libre = true;
   }

 }
```

1. Resolver con **PASAJE DE MENSAJES ASINCRÓNICO (PMA)** el siguiente problema. En un negocio de cobros digitales hay P personas que deben pasar por la única caja de cobros para realizar el pago de sus boletas. Las personas son atendidas de acuerdo con el orden de llegada, teniendo prioridad aquellos que deben pagar menos de 5 boletas de los que pagan más. Adicionalmente, las personas embarazadas tienen prioridad sobre los dos casos anteriores. Las personas entregan sus boletas al cajero y el dinero de pago; el cajero les devuelve el vuelto y los recibos de pago.

```
chan señal();
chan atencion[P] (Recibos,Dinero);
chan pagarEmb (int,Boletas,Dinero);
chan pagarMenosDe5 (int,Boletas,Dinero);
chan pagarGeneral (int,Boletas,Dinero);

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

   Boletas bol;
   Recibos rec;
   Dinero pago, vuelto;

   if (soyEmbAnc())
         send pagarEmb(i,boletas,pago);
   else
         if (len(bol) < 5) then
               send pagarMenosDe5(i,bol,pago);
         else
               send pagarGeneral(i,bol,pago);
   send señal();
   receive atencion [i] (rec,vuelto);
}


Process Caja {
   Boletas bol;
   Recibos rec;
   Dinero pago, vuelto;
   while (true) {
         receive señal();
         if (not empty(pagarEmb)) ->
               receive pagarEmb (idP,bol,pago);
         [] (empty(pagarEmb)) and (not empty(pagarMenosDe5)) ->
               receive pagarMenosDe5 (idP,boletas,pago);
         []    (empty(pagarEmb)) and (empty(pagarMenosDe5)) and (not empty (pagar
General)) ->
               receive pagarGeneral (idP,bol,pago);
         end if
         // atender
         (rec,vuelto) = cobrar(bol,pago);
         send atencion[idP] (rec,vuelto);
```

```
    }
  }
```

2) Resolver con **ADA** el siguiente problema. La oficina central de una empresa de venta de indumentaria debe calcular cuántas veces fue vendido cada uno de los artículos de su catálogo. La empresa se compone de 100 sucursales y cada una de ellas maneja su propia base de datos de ventas. La oficina central cuenta con una herramienta que funciona de la siguiente manera: ante la consulta realizada para un artículo determinado, la herramienta envía el identificador del artículo a cada una de las sucursales, para que cada uno de éstas calcule cuántas veces fue vendido en ella. Al final del procesamiento, la herramienta debe conocer cuántas veces fue vendido en total, considerando todas las sucursales. Cuando ha terminado de procesar un artículo comienza con el siguiente (suponga que la herramienta tiene una función *generarArtículo* que retorna el siguiente ID a consultar). **Nota**: maximizar la concurrencia. Supongo que existe una función *ObtenerVentas(ID)* que retorna la cantidad de veces que fue vendido el artículo con identificador ID en la base de datos de la sucursal que la llama.

```
Procedure ParcialADA is

      task type sucursal;
      task herramienta is
            entry PedirArticulo(id: OUT integer);
            entry DarResultado (cant: IN integer);
            entry SiguienteBusqueda;
      end herramienta;

      arrsucursales: array (1..100) of sucursal;

      task body sucursal is
            codigo, valor: integer;
      begin
            loop
                  Herramienta.PedirArticulo (codigo);
                  valor := ObtenerVentas (codigo);
                  Herramienta.darResultado(valor);
                  Herramienta.SiguienteBusqueda;
            end loop;
      end sucursal;

  task body Herramienta is
            cod, valor: integer;
      begin
            loop
                  cod := GenerarArtículo();
                  valor := 0;
                  for i in 1..200 loop
                        select
                              accept PedirArticulo (id: OUT integer) do
                                    id := cod;
                              end PedirArticulo;
```

```
                              or
                                    when (PedirArticulo'count = 0) =>
                                          accept darResultado(cant: IN integer) do
                                                valor := valor + cant;
                                          end darResultado;
                              end select;
                        end loop;
                        for i in 1..100 loop
                              accept siguienteBusqueda;
                        end loop;
                  end loop;
            end Herramienta;

Begin
      null;
End ParcialADA;
```