Posibles soluciones a los ejercicios del parcial práctico del 4-12-23

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

1. Un sistema debe validar un conjunto de 10000 transacciones que se encuentran disponibles en una estructura de datos. Para ello, el sistema dispone de 7 workers, los cuales trabajan colaborativamente validando de a 1 transacción por vez cada uno. Cada validación puede tomar un tiempo diferente y para realizarla los workers disponen de la función Validar(t), la cual retorna como resultado un número entero entre 0 al 9. Al finalizar el procesamiento, el último worker en terminar debe informar la cantidad de transacciones por cada resultado de la función de validación. Nota: maximizar la concurrencia.

```
sem sem_t = 1;
sem sem_cont [10] =  [10] {1};
sem sem_bar = 1;
sem s_dormir = 0;
Queue transacciones; // Ya cargada
int contadores [10] = [10] {0};
int cant = 0;

Process worker[i=1..7] {

    P(sem_t);
    while (not empty(transacciones)) do
        t = pop(transacciones);
        V(sem_t);
        res_val = Validar(t);
        P(sem_cont[res_val]);
        contadores[res_val]++;
        V(sem_cont[res_val]);
        P(sem_t);
    end;
    V(sem_t);

    P(s_bar);
    cant = cant + 1;
    if (cant < 7) {
        V(s_bar)
        P(s_dormir)
    } else {
        for i := 0 to 9 do
            print(i,contadores[i]);
        for i := 1 to 6 do
            V(s_dormir);
        V(s_bar);
    }
}
```

2) En una empresa trabajan 20 vendedores ambulantes que forman 5 equipos de 4 personas cada uno (cada vendedor conoce previamente a que equipo pertenece). Cada equipo se encarga de vender un producto diferente. Las personas de un equipo se deben juntar antes de comenzar a trabajar. Luego cada integrante del equipo trabaja independientemente del resto vendiendo ejemplares del producto correspondiente. Al terminar cada integrante del grupo debe conocer la cantidad de ejemplares vendidos por el grupo. Nota: maximizar la concurrencia.

```
Process Vendedor [id: 0..19]
{   int idE = //valor entre 0…4, ya conocido por el vendedor;
        int cant = 0, total;

    MEquipo[idE].Inicio ();
    Cant = //VenderPtroductos();
    MEquipo[idE].Fin (cant, total);
}

Monitor MEquipo [id= 0..4]
{
    cond barrera;
    int cant = 0, total = 0;

    Procedure Inicio ()
    {   cant++;
        If (cant == 4)   {  signal_all (barrera);
                cant = 0;
                         }
        else   wait (barrera);
    }

    Procedure Fin (parcial: int; resultado: out int )
    {   total = total + parcial;
cant++;
        If (cant == 4)   signal_all (barrera);
        else   wait (barrera);
        resultado = total;
    }

}
```

1. Resolver con PMS. En la estación de trenes hay una terminal de SUBE que debe ser usada por P personas de acuerdo con el orden de llegada. Cuando la persona accede a la terminal, la usa y luego se retira para dejar al siguiente. Nota: cada Persona una sólo una vez la terminal.

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

    Admin ! pedido (i);
    Admin ? usar ();
    // usar la terminal
    Admin ! liberar ();
}


Process Admin {

    bool libre = true;
    queue personas;
    int idPers;

    while (true) {
        if (libre = true); Persona [*] ? pedido (idPers) ->
            libre = false;
            Persona[idPers] ! usar ();
        [] (libre = false); Persona [*] ? pedido (idPers) ->
            q_push(estudiantes,idPers);
        [] Persona [*] ? liberar (idPers) ->
            if (empty(personas))
                libre = true;
            else
                Persona [pop(personas)] ! usar ();
        }
    }
}
```

2) 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 y los ancianos 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. Implemente un programa que permita resolver el problema anterior usando ADA.

```
Procedure Negocio is

task type Personas;

task Cajero is

    entry cobrarGeneral (boletas: IN String, pago: IN Dinero, recibos: OUT String,
vuelto: OUT dinero);
    entry cobrarMenosDe5 (boletas: IN String, pago: IN Dinero, recibos: OUT String,
vuelto: OUT dinero);
    entry cobrarEmbAnc (boletas: IN String, pago: IN Dinero, recibos: OUT String, vuelto:
OUT dinero);
end Cajero;


arrPersonas: array(1..P) of Persona;
cajero: Cajero;

task body Cajero is
    int c1, c2;
begin
    for i in 1..P loop
        select
            when (cobrarEmbAnc'count == 0) && (cobrarMenosDe5'count == 0) =>
                accept cobrarGeneral (boletas: IN String, pago: IN Dinero, recibos: OUT
String, vuelto: OUT dinero) do
                    (recibos,vuelto) := Cobrar(boletas, pago);
                end cobrarGeneral;
            OR
            when (cobrarEmbAnc'count == 0)  =>
                accept cobrarMenosDe5 (boletas: IN String, pago: IN Dinero, recibos: OUT
String, vuelto: OUT dinero) do
                    (recibos,vuelto) := Cobrar(boletas, pago);
                end cobrarGeneral;
            OR
            accept cobrarEmbAnc (boletas: IN String, pago: IN Dinero, recibos: OUT
String, vuelto: OUT dinero) do
                (recibos,vuelto) := Cobrar(boletas, pago);
            end cobrarGeneral;
        end select;
    end for;
end Cajero;

task body Persona is
    pago, vuelto: dinero;
    recibos, boletas: string;
begin
    if (soyEmbAnc()) then
        cobrarEmbAnc(boletas,pago,recibos,vuelto);
    else
        if (len(boletas) < 5) then
            cobrarMenosDe5(boletas,pago,recibos,vuelto);
```

```
        else
            cobrarGeneral(boletas,pago,recibos,vuelto);
        end if;
    end if;
end Persona;

end Negocio;
```