Posibles soluciones a los ejercicios del primer recuperatorio del parcial práctico

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. En un restorán trabajan *C cocineros* y *M mozos*. De forma repetida, los cocineros preparan un plato y lo dejan listo en la bandeja de platos terminados, mientras que los mozos toman los platos de esta bandeja para repartirlos entre los comensales. Tanto los cocineros como los mozos trabajan de a un plato por vez. Modele el funcionamiento del restorán considerando que la bandeja de platos listos puede almacenar hasta P platos. No es necesario modelar a los comensales ni que los procesos terminen.

   NOTA: es el problema de productores/consumidores con tamaño de buffer limitado visto en la página 14 de la teoría 4.

```
Platos bandeja[P];
int ocupado = 0, libre = 0;
sem vacio = P, lleno = 0;
sem mutexC = 1, mutexM = 1;

process Cocinero [i = 0..C-1] {
    while(true) {
        // Cocina el plato
        Plato plat = HacerPlato();
        // Espera hasta que haya espacio en la bandeja
        P(vacio);
        // Pone el plato en la bandeja
        P(mutexC);
        bandeja[libre] = plat;
        libre = (libre+1) mod P;
        V(mutexC);
        // Marca que hay un plato más en la bandeja
        V(lleno);
    }
}
process Mozo [i = 0..M-1] {
    Plato plat;
    while(true) {
        // Espera hasta que haya algún plato en la bandeja
        P(lleno);
        // Saca un plato de la bandeja
        P(mutexM);
        plat = buf[ocupado];
        ocupado = (ocupado+1) mod P;
        V(mutexM);
        // Marca que hay un lugar libre más en la bandeja
        V(vacio);
        repartir(plat)
    }
}
```

2. Resolver con MONITORES el siguiente problema. En una planta verificadora de vehículos existen *5 estaciones de verificación*. Hay *75 vehículos* que van para ser verificados, cada uno conoce el número de estación a la cual debe ir. Cada vehículo se dirige a la estación correspondiente y espera a que lo atiendan. Una vez que le entregan el comprobante de verificación, el vehículo se retira. Considere que en cada estación se atienden a los vehículos de acuerdo con el orden de llegada. ***Nota:*** maximizar la concurrencia.

```
Process Vehículo [id = 0..74] {
    Comprobante comp;
    int est = .......;
    Admin[est].EsperarAtencion(id, comp);
}

Process Estación [id = 0..4] {
    Comprobante comp;
    int idV;
    while (true) {
        Admin[id].siguiente(idV);
        // Verificar vehículo idV y generar comprobante
        comp = generarComprobante(idV)
        Admin[id].entregarComprobante(comp);
    }
}

Monitor Admin [id = 0..4] {
    cola autos;
    Comprobante comp;
    cond esperaV, esperaE, fin;

    Procedure EsperarAtencion (id: IN int; C: OUT Comprobante) {
        push (autos, id);
        signal (esperaE);
        // Espera a que esté su comproibante
        wait (esperaV);
        C := comp;
        // Avisa que ya tomo el comprobante y que se va.
        signal (fin);
    }

    Procedure siguiente (idV: OUT int) {
        if (empty(autos)) wait (esperaE);
        pop (autos, idV);
    }

    Procedure entregarComprobante (C: IN Comprobante) {
        comp = C;
        // Despierta al vehículo para que tome el comprobante
        signal (esperaV);
        // Espera a que lo haya tomado para atender al siguiente
        wait (fin);
    }

}
```

1. Resolver con Pasaje de Mensajes Sincrónicos (PMS) el siguiente problema. En un torneo de programación hay *1 organizador*, *N competidores* y *S supervisores*. El organizador comunica el desafío a resolver a cada competidor. Cuando un competidor cuenta con el desafío a resolver, lo hace y lo entrega para ser evaluado. A continuación, espera a que alguno de los supervisores lo corrija y le indique si está bien. En caso de tener errores, el competidor debe corregirlo y volver a entregar, repitiendo la misma metodología hasta que llegue a la solución esperada. Los supervisores corrigen las entregas respetando el orden en que los competidores van entregando. ***Nota:*** maximizar la concurrencia y no generar demora innecesaria.

```
Process Organizador {
    int idC;
    for i in 0..N-1 do
        string des = new Desafio();
        Competidor[*] ? pedido(idC);
        Competidor[idC] ! recibirDesafio(des);
    }
}

Process Competidor [id = 0..C-1] {
    string des, res;
    bool ok = false;
    Organizador ! pedio(id);
    Organizador ? recibirDesafio(des);
    while not(ok) {
        res = resolverDesafio(des);
        Coordinador ! entregarDesafio(id,des);
        Supervisor[*] ? recibirRespuesta (ok);
    }
}

Process Coordinador {
    Cola pedidos;
    string des;
    int idComp, idSup;
    while (true) {
        if (Competidor[*] ? entregarDesafio (idComp,des)) ->
            push(pedidos,(idComp,des));
        [] not(empty(pedidos)); Supervisor[*] ? pedirTrabajo(idSup) ->
            (idComp,des) = pop(pedidos);
            Supervisor[idSup] ! procesarTrabajo (idComp,des);
        end if;
    }
}

Process Supervidor [id = 0..S-1] {
    string des;
    bool ok = false;
    int idComp;
    while (true) {
        Coordinador ! pedirTrabajo (id);
        Coordinador ? procesarTrabajo (idComp,des);
        ok = corregirDesafio(des);
        Competidor[idComp] ! recibirRespuesta(ok);
    }
}
```

2. Resolver con ADA el siguiente problema. Una empresa de venta de calzado cuenta con *S sedes*. En *la oficina central* de la empresa se utiliza un sistema que permite controlar el stock de los diferentes modelos, ya que cada sede tiene una base de datos propia. El sistema de control de stock funciona de la siguiente manera: dado un modelo determinado, lo envía a las sedes para que cada una le devuelva la cantidad disponible en ellas; al final del procesamiento, el sistema informa el total de calzados disponibles de dicho modelo. Una vez que se completó el procesamiento de un modelo, se procede a realizar lo mismo con el siguiente modelo. **Nota:** suponga que existe una función *DevolverStock(modelo,cantidad)* que utiliza cada sede donde recibe como parámetro de entrada el modelo de calzado y retorna como parámetro de salida la cantidad de pares disponibles. Maximizar la concurrencia y no generar demora innecesaria.

```
Procedure ADA is

    task type Sede;

    task OficinaCentral is
        entry pedirModelo(m: OUT Modelo);
        entry darResultado (c: IN integer);
    end OficinaCentral;

    arrSedes: array (1..S) of Sede;

    task body Sede is
        mode: Modelo;
        cant: integer;
    begin
        loop
            OficinaCentral.pedirModelo(mode);
            DevolverStock(mode,cant);
            OficinaCentral.darResultado(cant);
        end loop;
    end Sede;

    task body OficinaCentral is
        mode: Modelo;
        cant, total: integer;
    begin
        loop
            mode := siguienteModelo();
            total := 0;
            -- Envía los pedidos
            for i in 1..S loop
                accept pedirModelo(m: OUT Modelo) do
                    m := mode;
                end pedirModelo;
            end loop;
            -- Recibe los resultados
            for i in 1..S loop
                accept darResultado(c: IN integer) do
                    total := total + c;
                end darResultado;
            end loop;
            -- Imprimir modelo y cantidad
        end loop;
    end OficinaCentral ;
Begin
    null;
End ADA;
```