

**Posibles soluciones a los ejercicios del parcial práctico del 15-12-25**

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

1. Implemente una solución para el siguiente problema usando **SEMÁFOROS**. Se debe simular el procesamiento de un sistema clasificador de 365 datasets meteorológicos. Todos los datasets se encuentran ya cargados en una estructura de datos. Para procesarla, el sistema dispone de 5 workers que trabajan colaborativamente procesando los datasets de a uno por vez. Cada procesamiento toma exactamente el mismo tiempo y para realizarlo los workers disponen de la función *Procesar(d)*, la cual retorna como resultado un booleano que indica si los datos corresponden a un clima tropical (true) o no (false). De acuerdo con el resultado, cada worker inserta el dataset en una cola de "tropicales" o de "no-tropicales". **Nota:** maximizar concurrencia.

```
sem sem_d = 1;
sem sem_trop = 1;
sem sem_notrop = 1;
Datasets ds [365] = ...; // Ya cargada
Queue tropicales, notropicales;

Process worker[id=0..4] {
    // determinar sección de datos para el worker
    int strip = 365 / 5;
    int ini = id*strip;
    int fin = (id+1)*strip-1;
    // procesar datos de worker
    for i := ini to fin do
        bool res = Procesar(ds[i]);
        // de acuerdo al resultado lo encolo. cada cola es independiente.
        if (res == true) then
            P(sem_trop);
            push(tropicales,d);
            V(sem_trop);
        else
            P(sem_notrop);
            push(notropicales,d);
            V(sem_notrop);
        end
    end
end
}
```

Comentario: otra solución adecuada consiste en que cada worker emplee colas locales para clasificar los datasets que procesa y, antes de finalizar, sí encolar en las colas globales usando semáforos para garantizar la exclusión mutua.

2. En el comedor de un establecimiento laboral asisten  $C$  comensales. Cada comensal lleva su propio almuerzo y lo calienta usando el único microondas disponible para todos los comensales. El microondas es usado en forma exclusiva por cada comensal de acuerdo con el orden de llegada. Implemente una solución utilizando **SEMÁFOROS** que sólo emplee procesos Comensales. **Nota:** la función *UsarMicroondas()* le permite calentar el almuerzo al comensal.

```
Sem s=1;
Sem esperar[C]={ [C]0 }
Boolean libre = true;
Queue cola;

Process Comensal[id=0..C-1] {
  // solicita acceso al microondas
  P(s)
  if (libre==false) {
    push(cola,id);
    V(s)
    P(esperar[id]);
  else
    libre=false;
    V(s);
  end
  // usa microondas con exclusión mutua
  UsarMicroondas();
  // libera microondas para que otro pueda usarlo
  P(s)
  if (not empty(cola)) then
    V(esperar[pop(cola)]);
  else
    libre=true;
  V(s);
}
```

3. En un cine se utiliza una terminal para el canje de entradas vendidas en forma anticipada por la web. Existen N asistentes que deben usar la terminal y un Empleado del cine que administra el acceso de acuerdo con el orden dado por la edad (cuando la terminal está libre, deja pasar al de mayor edad de entre los que están esperando por usarla). La terminal sólo puede ser usada por una persona a la vez. Implemente una solución al problema con **MONITORES** utilizando procesos para representar a los Asistentes y al Empleado. **Notas:** existe la función *CanjearEntrada()* que representa el uso de la terminal; cada asistente conoce su edad mediante la función *obtenerEdad()*.

```
Monitor AdminTerminal {
    cond colas[N], autoridad, fin;
    Queue(int,int) esperando;

    procedure llegada (int id, int edad) {
        push(esperando, (edad, id)); // inserta ordenado por edad
        signal(authoridad);
        wait(colas[id]);
    }

    procedure salida () {
        signal(fin);
    }

    procedure siguiente () {
        if (empty(esperando))
            wait (autoridad);
        int id = pop(esperando);
        signal(colas[id]);
        wait (fin);
    }
}

Process Cliente [i: 1..N]{
    int edad = obtenerEdad();
    // solicitar acceso
    AdminTerminal.llegada(i,edad);
    // usar terminal
    CanjearEntrada();
    // liberar
    AdminTerminal.salida();
}

Process Empleado {
    while (true) {
        // dar acceso para usar terminal a la siguiente persona
        AdminTerminal.siguiente();
    }
}
```

**Posibles soluciones a los ejercicios del parcial práctico del 15-12-25**

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

1. En un estadio de fútbol se dará un recital al que asistirán P personas. Para ello, cada asistente debe retirar su entrada en la boletería del estadio presentando su DNI (cada persona retira una única entrada). Los asistentes son atendidos por orden de llegada por un empleado del club que atiende en la boletería. Implemente una solución al problema usando **PMA**, considerando que el empleado realiza tareas administrativas por 15 minutos cuando no hay asistentes para atender. **Notas:** cada persona conoce su DNI mediante la función *obtenerDNI()*; existe la función *obtenerEntrada(dni)* que le retorna al empleado la entrada para el DNI provisto por argumento; existe la función *realizarTareasAdmin()* que representa las tareas administrativas que el empleado realiza por 15 minutos.

```
chan pedirEntrada(int,int);
chan recibirEntrada[P](Entrada);

Process Asistente [i:0..N-1] {
    Entrada ent;
    int dni = obtenerDNI();
    send pedirEntrada (i,dni);
    receive recibirEntrada [i] (ent);
}

Process Empleado {
    int idA, dni;
    Entrada ent;
    while (true) do
        if not(empty(pedirEntrada)) then
            receive pedirEntrada (idA,dni);
            ent = obtenerEntrada(dni);
            send recibirEntrada[idA] (ent);
        else
            realizarTareasAdmin();
    end
}
```

2. En un estadio de fútbol se dará un recital al que asistirán  $P$  personas. Para ello, cada asistente debe retirar su entrada en alguna de las 5 boleterías del estadio presentando su DNI (cada persona retira una única entrada). Los asistentes son atendidos por orden de llegada por alguno de los 5 empleados del club (cada uno atiende una boletería). Implemente una solución usando **PMS**. **Notas:** maximizar la concurrencia. Cada persona conoce su DNI mediante la función *obtenerDNI()*; la función *obtenerEntrada(dni)* le retorna al empleado la entrada para el DNI provisto por argumento.

```

Process Asistente [id=0..P-1] {
    int idE, dni = obtenerDNI();
    Entrada ent;
    //Solicita empleado para su atención
    Buffer!pedirEmpleado(id);
    Buffer?asignarEmpleado(idE);
    // enviar DNI y espera entrada
    Empleado[idE]!entregarDNI(dni);
    Empleado[idE]?entregarEntrada(ent);
}

Process Empleado[id=0..4] {
    int idA, dni;
    while (true) do
        Buffer!pedirAsistente(id);
        Buffer?asignarAsistente(idA);
        Asistente[idA]?entregarDNI(dni);
        ent = obtenerEntrada(dni);
        Asistente[idA]!entregarEntrada(ent);
    end
}

Process Buffer {
    Queue asistentes;
    int idA, idE;
    for i in 1..2*P {
        if (Asistente[*] ? pedirEmpleado (idA)) ->
            push(asistentes,idA);
        [] (not(empty(asistentes)); Empleado[*] ? pedirAsistente(idE)) ->
            idA = pop(asistentes);
            Empleado[idE]!asignarAsistente (idA);
        end
    }
}

```

3. Se requiere modelar el acceso a un servidor de procesamiento de alto rendimiento que cuenta con 128 núcleos computacionales. Las tareas se dividen en livianas, medias y pesadas, requiriendo 1, 2, y 3 núcleos computacionales, respectivamente. Suponga que hay una cantidad conocida de tareas (T1 livianas, T2 medias y T3 pesadas) que se ejecutan una única vez. El servidor sólo puede ejecutar una tarea si tiene núcleos computacionales disponibles. Implemente una solución en **ADA** considerando que las tareas medias tienen prioridad sobre las pesadas, y las livianas sobre todas las demás.

```
Task Type TareaLiviana;  
Task Type TareaMedia;  
Task Type TareaPesada;  
  
Task AdministradorDeServidor IS  
    entry entrada_liviana();  
    entry salida_liviana();  
    entry entrada_media();  
    entry salida_media();  
    entry entrada_pesada();  
    entry salida_pesada();  
end AdministradorDeServidor;  
  
admin: AdministradorDeServidor;  
livianas: array (1..T1) of TareaLiviana;  
medias: array (1..T2) of TareaMedia;  
pesadas: array (1..T3) of TareaPesada;  
  
TASK Body TareaLiviana IS  
Begin  
    admin.entrada_liviana ();  
    -- Ejecutar  
    admin.salida_liviana ();  
End  
  
TASK Body TareaMedia IS  
Begin  
    admin.entrada_media ();  
    -- Ejecutar  
    admin.salida_media ();  
End  
  
TASK Body TareaPesada IS  
Begin  
    admin.entrada_pesada ();  
    -- Ejecutar  
    admin.salida_pesada ();  
End
```

```
TASK Body AdministradorDeServidor IS
    nucleos_ocupados: int = 0;
Begin
    loop
        SELECT
            Accept salida_liviana ();
            nucleos_ocupados--;
        OR
            Accept salida_media ();
            nucleos_ocupados-=2;
        OR
            Accept salida_pesada ();
            nucleos_ocupados-=3;
        OR
            WHEN ((nucleos_ocupados+3 <= 128) AND (entrada_liviana'count ==0)
AND (entrada_media'count ==0))
                => Accept entrada_pesada ();
                nucleos_ocupados+=3;
            OR
                WHEN ((nucleos_ocupados+2 <= 128) AND (entrada_liviana'count ==0))
                => Accept entrada_media ();
                nucleos_ocupados+=2;
            OR
                WHEN (nucleos_ocupados+1 <= 128)
                => Accept entrada_liviana ();
                nucleos_ocupados++;
        End SELECT;
    End loop;
End AdministradorDeServidor;
```