




# Posibles soluciones a los ejercicios del parcial

Estas son posibles soluciones, no significa que sean la única forma, traté de hacerlas simples y siguiendo las ideas que hemos visto en las teorías y/o explicaciones prácticas para no confundir.



Resolver con **PMA (Pasaje de Mensajes ASINCRÓNICOS)** el siguiente problema. Simular la atención en un locutorio con 10 cabinas telefónicas, que tiene *un empleado* que se encarga de atender a los clientes. Hay *N clientes* que al llegar esperan hasta que el empleado les indica a que cabina ir, la usan y luego se dirigen al empleado para pagarle. El empleado atiende a los clientes en el orden en que hacen los pedidos, pero siempre dando prioridad a los que terminaron de usar la cabina. **Nota:** maximizar la concurrencia; suponga que hay una función *Cobrar()* llamada por el empleado que simula que el empleado le cobra al cliente.

Solución válida, pero puede generar Busy waiting.

```
chan solicitar (int);  
chan usarCabina[N] (int);  
chan liberar (int, int);  
chan irse[N] ();
```

```
Process Cliente[id: 0..N-1] {  
    int numC;  
  
    send solicitar (id);  
    receive usarCabina[id](numC);  
    // Usa la cabina Telefónica numC  
    send liberar (id, numC);  
    receive irse[id] ();  
};
```


```
Process Empleado {  
    int idC, numC, i;  
    queue libres;  
  
    for (i=1; i<11; i++) push (libres, i);  
  
    while (true) {  
        if (not empty (liberar)) →  
            receive liberar (idC, numC);  
            Cobrar(numC, idC);  
            send irse[idC] ();  
            push (libres, numC);  
        □ (not empty (libres)) and (empty (liberar)) and (not empty (solicitar)) →  
            receive solicitar (idC);  
            pop (libres, numC);  
            send usarCabina[idC] (numC);  
        fi;  
    };  
};
```

## Solución que no genera Busy waiting.

```
chan solicitar (int, int, text);  
chan usarCabina[N] (int);  
chan irse[N] ();
```

```
Process Cliente[id: 0..N-1] {  
    int numC;  
  
    send solicitar (id, NULL, "pedir");  
    receive usarCabina[id](numC);  
    // Usa la cabina Telefónica numC  
    send solicitar (id, numC, "liberar");  
    receive irse[id] ();  
};
```

```
Process Empleado {  
    int idC, numC, i, idAux;  
    text op;  
    queue libres, colaPed;  
  
    for (i=1; i<11; i++) push (libres, i);  
  
    while (true) {  
        receive solicitar (idC, numC, op);  
        if (op == "liberar") {  
            Cobrar(numC, idC);  
            send irse[idC] ();  
            if (empty(colaPed)) push (libres, numC)  
            else { pop (colaPed, idAux);  
                    send usarCabina[idAux] (numC);  
                };  
        }  
        else if (empty(libres) or not empty(colaPed)) push (colaPed, idC)  
        else { pop (libres, numC);  
                send usarCabina[idC] (numC);  
            };  
        };  
    };  
};
```




Resolver con **PMS (Pasaje de Mensajes SINCRÓNICOS)** el siguiente problema. Simular la atención de una estación de servicio con un único surtidor que tiene ***un empleado*** que atiende a los ***N clientes*** de acuerdo al orden de llegada. Cada cliente espera hasta que el empleado termina de cargarle combustible y se retira. ***Nota:*** cada cliente carga combustible sólo una vez; todos los procesos deben terminar.

```
Process Cliente[id: 0..N-1] {  
  AdminOrden ! cargar (id);  
  Empleado ? finCarga ();  
};
```

```
Process Empleado {  
  int idC;  
  
  while (true) {  
    AdminOrden ! siguiente ();  
    AdminOrden ? datoCliente (idC);  
    CargarCombustible();  
    Cliente[idC] ! finCarga();  
  };  
};
```

```
Process AdminOrden{  
  queue cola;  
  int idC;  
  
  do Cliente[*] ? cargar(idC) → push (cola, idC);  
    □ (not empty (cola)); Empleado ? siguiente () →  
      pop(col, idC);  
      Empleado ! datoCliente (idC);  
  od  
};
```





Resolver con **ADA** la siguiente situación. En una oficina hay *un empleado* y *P personas* que van para ser atendidas para realizar un trámite. Cuando una persona llega espera a lo sumo 20 minutos a que comience a atenderla el empleado para resolver el trámite que va a hacer, y luego se va; si pasó el tiempo se retira sin realizar el trámite. El empleado atienden las solicitudes en orden de llegada. Cuando las P personas se han retirado el empleado también se retira. **Nota:** cada persona hace sólo un pedido y termina; suponga que existe una función *Atender()* llamada por el empleado que simula que el empleado está resolviendo el trámite del cliente; todas las tareas deben terminar.

### *Procedure ParcialADA is*

```
task type Persona;  
task Empleado is  
  entry Resolver (t: IN text; res: OUT text);  
  entry Terminar;  
end Empleado;  
task Admin is  
  entry MeVoy;  
end Admin;  
  
task body Empleado is  
  fin: boolean := false;  
begin  
  while (not fin) loop  
    select  
      accept Resolver(t: IN text; res: OUT text) do  
        res := Atender(t);  
      end Resolver;  
    or  
      accept Terminar;  
      fin := true;  
    end select;  
  end loop;  
end Empleado;
```

arrPersonas: array (0..P-1) of *Persona*;

```
task body Persona is  
  Resultado: text;  
  Tramite: text := .... ;  
begin  
  Select  
    Empleado.Resolver (Tramite, Resultado);  
  or delay 1200.0  
    null;  
  end loop;  
  Admin.MeVoy;  
end Persona;
```

```
task body Admin is  
begin  
  for i in 1..P loop  
    accept MeVoy;  
  end loop;  
  Empleado.Terminar;  
end Admin;
```

```
Begin  
  null;  
End ParcialADA;
```