ALGORITMOS DE EXCLUSIÓN MUTUA CON MEMORIA COMPARTIDA
ALGORITMOS NO EFICIENTES

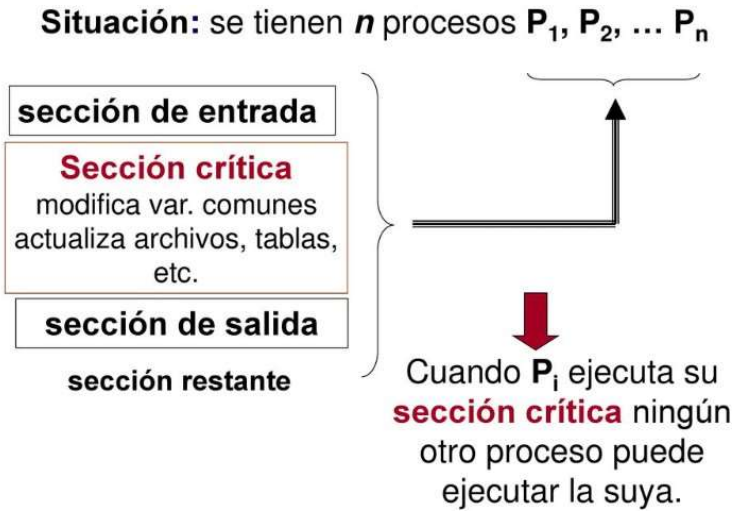
TIPOS DE SINCRONIZACIÓN Y SU SOLUCIÓN

Se definirá los dos tipos de sincronización necesarios entre procesos concurrentes: exclusión mutua y condición de sincronización. Además, se presentan los distintos mecanismos con los que se pueden implementar ambos tipos de sincronización analizando el primer mecanismo propuesto que denominamos inhibición de interrupciones.

EXCLUSIÓN MUTUA

Se denomina exclusión mutua cuando los procesos desean utilizar un recurso no compartible, la sincronización necesaria entre ellos. Un proceso que está accediendo a un recurso no compartible se dice que se encuentra en una sección critica. Es decir, la sección critica es la parte del código es la parte del código que utiliza un proceso en el acceso a un recurso no compartible y por tanto debe ejecutarse en exclusión mutua.

Garantizar la exclusión mutua en la ejecución de las secciones criticas consiste en diseñar un protocolo de entrada y salida mediante el cual sincronice la entrada de os procesos a su sección critica.



La ejecución concurrente de los procesos en Pascal FC la indicaremos mediante la estructura cobegin/coend.

Algoritmos no eficientes

Primer intento

La propuesta más simple al problema de la exclusión mutua consiste en usar una sola variable global (compartida) que informe del estado de la sección critica. Por ejemplo, si esta variable sale sclibre significa que la sección critica está libre y, por tanto, se puede ejecutar. Si la variable scocupada, la sección critica esta siendo

ejecutada por otro proceso y, por lo tanto, ningún otro proceso podrá ejecutarla.

```
      Process P0                          Process P1
repeat                              repeat
     /*protocolo de entrada*/            /*protocolo de entrada*/
     a) while v = socupada do;           a) while v = socupada do;
     b) v := socupada;                   b) v := socupada;
     /*ejecuta la sección crítica*/      /*ejecuta la sección crítica*/
     c)Sección Crítica0;                 c)Sección Crítica1;
     /*protocolo de salida*/             /*protocolo de salida*/
     d) v:=sclibre;                      d)v:=sclibre;
      Resto0                             Resto1
forever                             forever
```

```
program Intento1;
var
  estado:integer;

{estado=0=ocupado  y  estado=1=libre}

process P1;
begin
   repeat
     while estado=0 do;
     estado:=0;
     writeln('Proceso 1 está en su sección crítica');
     estado:=1;
   forever
end;

process P2;
begin
   repeat
     while estado=0 do;
     estado:=0;
     writeln('Proceso 2 está en su sección crítica');
     estado:=1;
   forever
end;

begin
estado:=1;
  cobegin
     P1;
     P2;
  coend;
end.
```

Esta solución no es correcta, ya que puede ocurrir que ambos procesos ejecuten el while antes de ejecutar **v:=scocupada** y que los dos entren en la sección critica a la vez. Por tanto, este algoritmo **no garantiza la condición de exclusión mutua**

Segundo intento (Alternancia)

El siguiente intento también usa una variable global turno, que contendrá el número (identificador) del proceso que puede entrar en la sección critica. La implementación de los protocolos es la siguiente:

Process P_0	Process P_1
repeat	repeat
while turno = 1 do;	**while turno = 0 do;**
Sección Crítica$_0$;	Sección Crítica$_1$;
turno:= 1;	**turno:= 0;**
Resto$_0$	Resto$_1$
forever	forever

```
program intento2;
var turno:integer;

process p1;
  begin
    repeat
      while turno=2 do;
      (*Seccion Critica proceso 1*)
      writeln('Proceso 1 en su Seccion Critica');
      turno:=2;
    forever
  end;

process p2;
  begin
    repeat
      while turno=1 do;
      (*Seccion Critica proceso 2*)
      writeln('Proceso 2 en su Seccion Critica');
      turno:=1;
    forever
  end;

begin
  cobegin
    p1;
    p2;
  coend
end.
```

Inconvenientes:

La solución planteada provoca que el derecho de usar la sección critica sea alternativo entre los procesos. Esta situación hace que no se satisfaga la condición de progreso de la ejecución, pues puede que, P1 llegue antes que P0, que tiene el turno, o puede que un proceso quiera entrar dos veces seguidas.

Los procesos están exclusivamente acoplados, por ende, el algoritmo es poco tolerante a fallos. **Si un proceso falla el otro quedará detenido**

Tercer intento

El problema de la alternancia se produce porque no se conserva suficiente información acerca del estado de cada proceso; únicamente se recuerda cual es el proceso al que se le permite entrar en su sección crítica. Para evitarlo vamos a usar dos variables compartidas, cada una perteneciente a un proceso. Ambos procesos pueden acceder a las dos variables, pero cada proceso sólo puede modificar su propia variable. Cuando una de las variables contenga el valor en la sección critica significa que el proceso en cuestión está ejecutando su sección critica, y cuando tenga el valor **restoproceso** indicará que el proceso en cuestión no está ejecutando su sección critica. La implementación es la siguiente.

```
        Process P0                          Process P1
repeat                              repeat
        a ) while C1 = enSC do;             a) while C0 = enSC do;
        b ) C0:= enSC;                      b) C1:= enSC;
        c) Sección Crítica0;                c) Sección Crítica1;
        d) C0:= restoproceso;               d)C1:= restoproceso;
        Resto0                              Resto1
forever                             forever
```

```
program inteto3;
var S1,S2:boolean;

process p1;
  begin
    repeat
      while S2=true do;
      S1:=true;
      (*Seccion Critica del proceso 1*)
      writeln('Proceso 1 en su seccion critica');
      S1:=false;
    forever
  end;

process p2;
  begin
    repeat
      while S1=true do;
      S2:=true;
      (*Seccion Critica del proceso 2*)
      writeln('Proceso 2 en su seccion critica');
      S2:=false;
    forever
  end;

begin
  cobegin
    p1;
    p2;
  coend
end.
```

Las variables C0 y c1 (Flag1 y Flag2) deben inicializarse a restoproceso, indicando que ningún proceso este ejecutando su sección crítica al comienzo de la ejecución de ambos. El protocolo de entrada consiste en un bucle donde se comprueba el valor de la variable correspondiente al otro proceso. Sin embargo, este algoritmo no garantiza la exclusión mutua, es decir, varios procesos pueden ejecutar su sección critica al mismo tiempo.

1. P0 ejecuta la instrucción a) y encuentra que C1 vale restroproceso;
2. P1 ejecuta a instrucción a) y encuentra que C0 vale restoproceso;
3. P0 asigna en sección crítica a la variable C0 y entra en su SC;
4. P1 asigna en sección crítica a la variable C1 entra en su SC;

Se llega a la situación en la que P0 y P1 se encuentran ambos ejecutando sus secciones criticas y por lo tanto violando la exclusión mutua.

Cuarto intento (Espera Infinita)

En el algoritmo anterior, la falta de exclusión mutua surge porque un proceso, P0, toma una decisión relacionada con el estado del otro proceso, P1, antes de que éste tenga la oportunidad de cambiar el estado de su variable, C1.

Este problema se puede solucionar si en lugar de asignar el valor en Sección crítica a la variable relacionada con el proceso le asignamos el valor **quiereentra**r, indicando con ello que el proceso quiere entrar en su sección crítica.

```
      Process P0                          Process P1
repeat                              repeat
    C0:= quiereentrar;                  C1:= quiereentrar;
    while C1 = quiereentrar do;         while C0 = quiereentrar do;
    Sección Crítica0;                   Sección Crítica1;
    C0:= restoproceso;                  C1:= restoproceso;
    Resto0                              Resto1
forever                             forever
```

```
program Intento4;
var
  s1,s2:integer;

process P1;
begin
   repeat
     s1:=1;
     while s2=1 do;
     writeln('Proceso 1 está en su sección crítica');
     s1:=0;
   forever
end;

process P2;
begin
   repeat
   s2:=1;
   while s1=1 do;
   writeln('Proceso 2 está en su sección crítica');
   s2:=0;
   forever
end;

begin
s1:=0;
s2:=0;
  cobegin
     P1;
     P2;
  coend;
end.
```

1. De la estructura del programa se deduce que cuando P0 entra en sección crítica entonces C1 no vale **quiereentrar.**
2. Si C1 no vale **quiereentrar**, entonces P1 no está en su SC, puesto que esta sección critica está entre las sentencias de asignación a la variable C1 (flag2).
3. Como consecuencia de las dos deducciones anteriores cuando P0 entra a su SC, P1 no está en la suya.
4. Si P0 está en su sección crítica, entonces C0 vale **quiereentrar**, puesto que la sección critica está entre las sentencias de asignación a la variable C0(flag 1).
5. Si C0 vale **quiereentrar**, entonces P1 no entra en Sección critica, ya que la testea previamente.

6. Se deduce que, si P0 está en sección critica, P1 no está en la suya.

Sin embargo, este intento de solución produce un problema de progreso en la ejecución. Ejemplo:

a) P0 realiza la primera asignación
b) P1 realiza la primera asignación

Indicando ambos que desean entrar a su sección critica, y encontrándose ambos procesos en un bucle infinito en sus instrucciones while. Se llega por tanto a una situación no deseable. Cada proceso espera que el otro cambie el valor de su variable respectiva y ambos quedan en una espera infinita.

Quinto Intento

En el algoritmo anterior, la falta de exclusión mutua surge porque un proceso, P0, toma una decisión relacionada con el estado del otro proceso, P1, antes de que éste tenga la oportunidad de cambiar el estado de su variable, C1.

Este problema se puede solucionar si en lugar de asignar el valor en sección critica a la variable relacionada con el proceso le asignamos el valor **quiereentra**r, indicando con ello que el proceso quiere entrar en su sección critica. En definitiva, que las asignaciones del protocolo de entrada se hagan antes del bucle de espera ocupada

```
      Process P0
repeat
    C0:= quiereentrar;
    while C1 = quiereentrar do
        begin
            C0:= restoproceso;
            (*hacer algo durante
              unos momentos*)
            C0:= quiereentrar
        end;
    Sección Crítica0;
    C0:= restoproceso;
    Resto0
forever
```

```
      Process P1
repeat
    C1:= quiereentrar;
    while C0 = quiereentrar do
        begin
            C1:= restoproceso;
            (*hacer algo durante
              unos momentos*)
            C1:= quiereentrar
        end;
    Sección Crítica1;
    C1:= restoproceso;
    Resto1
forever
```

```
program intento5;
var
  s1,s2:integer;

(*s1=true=1  y  s2=false=0*)

process P1;
begin
   repeat
     s1:=1;
     while s2=1 do
       begin
         s1:=0;
         s1:=1;
       end;
     writeln('***Seccion critica P1***');
     s1:=0;
   forever
end;

process P2;
begin
   repeat
     s2:=1;
     while s1=1 do
       begin
         s2:=0;
         s2:=1;
       end;
     writeln('....Seccion critica P2....');
     s2:=0;
   forever
end;
```

Aunque se garantiza la exclusión mutua, este tratamiento de cortesía puede llevar a que los procesos se queden de manera indefinida cediéndose mutuamente el paso con sólo suponer que esos “pocos momentos” duran lo mismo para ambos procesos.

No se produce espera ilimitada porque existe una esperanza de que se salga de esta situación, pero no se asegura que se acceda a la sección crítica en un tiempo finito.

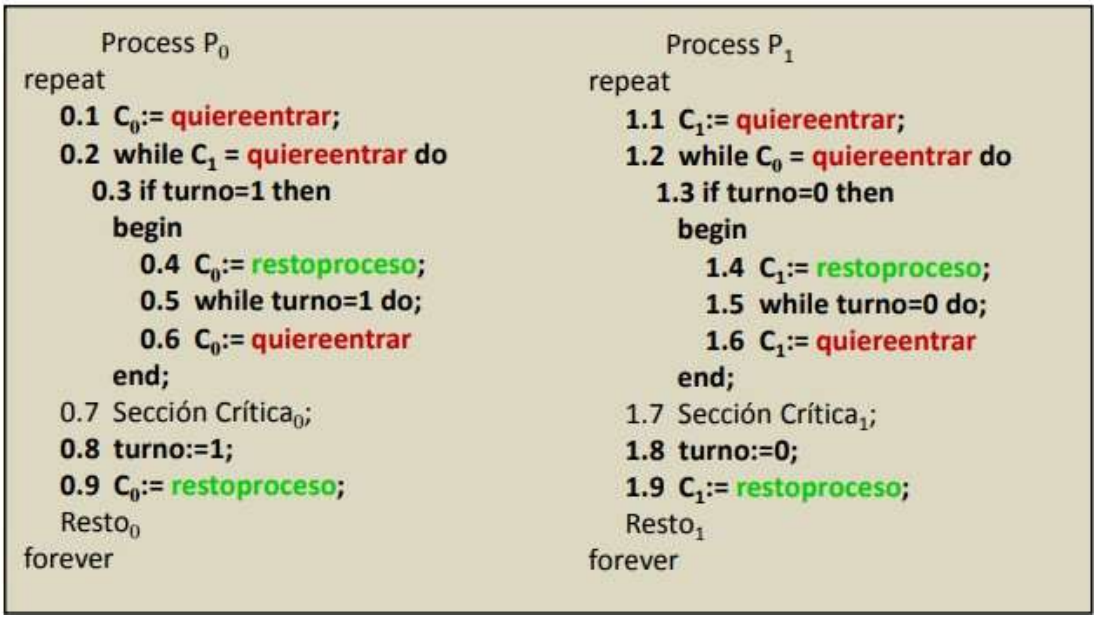
Por consiguiente, esta solución no permite estudiar su eficiencia ya que no se conoce el número de veces que se repite el ciclo del protocolo de espera.

Algoritmo de Dekker

Dekker, combinó la idea de los turnos del segundo algoritmo y la del tratamiento de cortesía del quinto. Como resultado diseño una solución software que consiste en que cada proceso comparta dos variables.

Ahora, con la variable turno no se transfiere el derecho a entrar a la SC de un proceso a otro, sino el derecho de comprobar si se puede entrar a la SC, por lo tanto, no existe alternancia.

La inicialización de las variables será la siguiente: C0=restoproceso; C1=restoproceso; turno= 0 ó 1 (es indiferente).



```
      Process P0                          Process P1
repeat                              repeat
   0.1 C0:= quiereentrar;              1.1 C1:= quiereentrar;
   0.2 while C1 = quiereentrar do      1.2 while C0 = quiereentrar do
     0.3 if turno=1 then                 1.3 if turno=0 then
       begin                               begin
         0.4 C0:= restoproceso;              1.4 C1:= restoproceso;
         0.5 while turno=1 do;               1.5 while turno=0 do;
         0.6 C0:= quiereentrar               1.6 C1:= quiereentrar
       end;                                end;
   0.7 Sección Crítica0;               1.7 Sección Crítica1;
   0.8 turno:=1;                       1.8 turno:=0;
   0.9 C0:= restoproceso;              1.9 C1:= restoproceso;
   Resto0                              Resto1
forever                             forever
```

```
program prog1;
var
   turno,suma:integer;
   s1,s2:boolean;

(*suma es variable compartida*)

process P1;
  begin
    repeat
      s1:=true;
      while s2=true do
        begin
          if turno=2 then
            begin
               s1:=false;
               while turno=2 do;
               s1:=true;
            end;
        end;
      writeln('El proceso 1 se encuentra en su SC');
      suma:=suma+2;
      writeln('La suma total es:  ',suma);
      turno:=2;
      s1:=false;
    forever
  end;
```

```
process P2;
  begin
    repeat
      s2:=true;
      while s1=true do
        begin
          if turno=1 then
            begin
              s2:=false;
              while turno=1 do;
              s2:=true;
            end;
        end;
      writeln('El proceso 2 se encuentra en su SC');
      suma:=suma-1;
      writeln('La suma total es:  ',suma);
      turno:=1;
      s2:=false;
    forever
  end;

begin
 suma:=0;
 turno:=2;
 s1:=false;
 s2:=false;
 cobegin
   P1;
   P2;
 coend;
end.
```

Inconvenientes:

- Solo puede manejar un máximo de dos procesos.
- Hace uso de espera activa.
- No suspende a los procesos que están esperando acceso.
- Puede llegar a entrar en ciclos infinitos

Ejercicio propuesto

- Realizar los algoritmos de Pascal FC a C++ con el IDE Codeblocks