Tipos de sincronización y su solución:
Los dos tipos de sincronización necesaria entre procesos concurrentes son: exclusión mutua y condición de sincronización.
Exclusión mutua:
Cuando los procesos deben utilizar un recurso no compartible, la sincronización necesaria entre ellos se denomina **Exclusión mutua**.
Un proceso que accede a un recurso no compartible se encuentra en su sección crítica (es parte del código que utiliza un proceso cuando accede a un recurso no compartible y se ejecuta en exclusión mutua).
Cuando un proceso está en su sección crítica el resto de los procesos no deben estar en sus secciones críticas, y deben esperar hasta que la sección crítica se libere.
Garantizar la exclusión mutua en la ejecución de las secciones críticas consiste en diseñar un ´protocolo de entrada y salida que sincronice la entrada de los procesos a sus secciones críticas. Un esquema sería:

Process P1	Process P2	Process Pi
….	….	….
Protocolo de entrada	Protocolo de entrada	Protocolo de entrada
Sección crítica	Sección crítica	Sección crítica
Protocolo de salida	Protocolo de salida	Protocolo de salida
….	….	….

Los protocolos de entrada y salida son porciones de código que permiten cumplir con las siguientes condiciones para que resuelvan el problema de exclusión mutua:

- **Exclusión mutua:** no pueden acceder dos procesos a la vez a la sección crítica. (Exclusión mutua hace referencia a la ejecución mutuamente exclusiva de las secciones críticas)
- **Limitación en la espera:** de cada proceso para acceder a la sección crítica; es decir, que ningún proceso espere indefinidamente para entrar en su sección crítica.
- **Progreso en la ejecución:** es decir que cuando un proceso quiera ejecutar su sección crítica pueda hacerlo si ésta está libre.

Condición de sincronización:
La podemos definir como la propiedad requerida de que un proceso no realice un evento hasta que otro proceso haya realizado una acción determinada.

Soluciones a los dos tipos de sincronización:
Los mecanismos que disponemos para implementar los distintos tipos de sincronización son:

1. Inhibición de las interrupciones.
2. Espera ocupada (busy waiting).
3. Semáforos.
4. Regiones críticas.
5. Regiones críticas condicionales
6. Monitores.
7. Operaciones de paso de mensajes send/ receive.
8. Llamadas a procedimientos remotos.
9. Invocaciones remotas

Estos mecanismos (exceptuando Inhibición de las interrupciones) están englobados en:

1. **Soluciones basadas en variables compartidas** (Espera ocupada, Semáforos, Regiones críticas, Regiones críticas condicionales, Monitores)

2. **Soluciones basadas en el paso de mensajes** (Operaciones de paso de mensajes send/ receive, Llamadas a procedimientos remotos, Invocaciones remotas)

En el primer grupo se encuentran los mecanismos de espera ocupada, semáforos, regiones críticas, regiones críticas condicionadas y monitores. (Estos mecanismo aparecen ordenados de menor a mayor nivel, es decir los 1° son mas complejos de usar. Las soluciones que se pueden alcanzar son distintas dependiendo del nivel por el cual aproximemos el problema.)

La característica común en estos mecanismos es que independientemente del nivel del nivel, es que existe un problema elemental de exclusión mutua en el acceso concurrente a la misma posición de memoria. Este problema está resuelto a nivel hardware que serializa los accesos concurrentes.

En el segundo grupo, se encuentran incluidas las operaciones de paso de mensajes send/receives, las llamadas a procesamientos remotos y las invocaciones remotas. Las acciones atómicas de los procesos no son las operaciones de acceso a memoria, sino las sentencias de paso de mensajes.

Vamos a abordar la primera solución propuesta para implementar la sincronización denominado Inhibición de interrupciones.

La espera ocupada para la Exclusion mutua

El termino espera ocupada hace referencia que esta implementación de la sincronización se basa en que el proceso espera mediante la comprobación continua del valor de una variable manteniendo ocupada la CPU del sistema.

Según el tipo de operaciones que se utilicen para desarrollar una solución basada en espera ocupada podemos distinguir:

- Soluciones software: las únicas operaciones atómicas que se consideran son las instrucciones de bajo nivel para leer y almacenar (Load-store) de/en direcciones de memoria. Si dos instrucciones de este tipo se produjeran simultáneamente el resultado sería la ejecución secuencias en un orden desconocido.
- Soluciones hardware: se utilizan instrucciones especializadas que llevan una serie de acciones de forma indivisible como leer y escribir , intercambiar el contenido de dos posiciones de memoria, etc.

Soluciones software: Algoritmos no eficientes:

1. **Primer intento**

Process P_0	Process P_1
repeat	repeat
/*protocolo de entrada*/	/*protocolo de entrada*/
a) while v=scocupada do;	a) while v=scocupada do;
b) v:=scocupada;	v:=scocupada;
/* Ejecuta la sección crítica*/	/* ejecuta la sección crítica */
c) Sección crítica$_0$	c) Sección crítica$_1$
/*protocolo de salida*/	/*protocolo de salida*/
d) v:=sclibre;	d) v:=sclibre;
Resto$_0$	Resto$_1$
forever	forever

Puede ocurrir que ambos procesos ejecuten el while antes de ejecutar v:=scocuada y que y que los dos entren a la vez en la sección crítica a la vez, **NO GARANTIZA LA CONDICIÓN DE EXCLUSIÓN MUTUA**

2. Segundo Intento (Alternancia)

El siguiente intento también usa una variable global turno, pero en este caso contendrá el número (identificador) del proceso que puede entrar en la sección crítica. 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$_1$
/*protocolo de salida*/	/*protocolo de salida*/
turno:=1;	turno:=0;
Resto$_0$	Resto$_1$
forever	forever

Inconvenientes del segundo intento:

- 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.
- Los procesos están altamente acoplados por ende el algoritmo es poco tolerante a fallos, si un proceso falla el otro quedará detenido (en espera indefinida).

3. Tercer intento (falta de exclusión mutua):

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 puede modificar solo su propia variable. Cuando una de las dos variables contenga el valor enSC significa que el proceso en cuestión está ejecutando su sección crítica, y cuando tenga el valor restoproceso indicará que el proceso en cuestión no está ejecutando su sección crítica. La implementación es la siguiente-.

Process P_0	Process P_1
repeat	repeat
a) while c_1=enSC do;	a) while turno = 0 do;
b) c_0 := enSC;	b) c_1 := enSC;
c) Sección Crítica$_0$;	c) Sección Crítica$_1$;
d) C_0 :=restoproceso;	d) C_1:=restoproceso;
Resto$_0$	Resto$_1$
forever	forever

Las variables c_0 y c_1 deben inicializarse a restoproceso, indicando que ningún proceso está 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 crítica al mismo tiempo.

Consideremos la siguiente secuencia de ejecución:

1. P_0 ejecuta la instrucción a) y encuentra que c_1 vale restoproceso;
2. P_1 ejecuta a instrucción a) y encuentra que c_0 vale restoproceso;
3. P_0 asigna enSC a la variable c_0 y entra en su sección crítica;
4. P_1 asigna enSC a la variable c_1 entra en su sección crítica;