Sistemas Operativos

UNIDAD 3 – SISTEMA DE PROCESOS

PRINCIPIOS DE LA CONCURRENCIA

Índice

- Conceptos
  - Concurrencia
  - Principales Dificultades
  - Aspectos de diseño del S.O. para la concurrencia
  - Interacción de procesos
    - No cooperantes
    - Cooperantes por Compartición
    - Cooperantes por Comunicación
  - Sección Crítica y Exclusión Mutua
  - Requisitos para la exclusión mutua
- Soluciones para la Exclusión Mutua
  - Soluciones Software
  - Soluciones Hardware

Conceptos - concurrencia

Multiprogramación : Gestión de varios procesos en un sistema monoprocesador
Multiprocesamiento: Gestión de varios procesos en un sistema multiprocesador
Procesamiento distribuido : Gestión de varios procesos ejecutándose en sistemas de computadores múltiples o distribuidos. p.ej. Clusters

Contextos concurrentes

Múltiples Aplicaciones
Compartir procesador entre varias aplicaciones activas.
Aplicaciones Estructuradas
Una misma aplicación puede ser estructurada como un conjunto de procesos concurrentes
Estructura del Sistema Operativo
Algunos S.O. están estructurados como un conjunto de procesos concurrentes o hilos

Tabla 5.1. Algunos términos clave relacionados con la concurrencia.

sección crítica (*critical section*)	Sección de código dentro de un proceso que requiere acceso a recursos compartidos y que no puede ser ejecutada mientras otro proceso esté en una sección de código correspondiente.
interbloqueo (*deadlock*)	Situación en la cual dos o más procesos son incapaces de actuar porque cada uno está esperando que alguno de los otros haga algo.
círculo vicioso (*livelock*)	Situación en la cual dos o más procesos cambian continuamente su estado en respuesta a cambios en los otros procesos, sin realizar ningún trabajo útil.
exclusión mutua (*mutual exclusion*)	Requisito de que cuando un proceso esté en una sección crítica que acceda a recursos compartidos, ningún otro proceso pueda estar en una sección crítica que acceda a ninguno de esos recursos compartidos.
condición de carrera (*race condition*)	Situación en la cual múltiples hilos o procesos leen y escriben un dato compartido y el resultado final depende de la coordinación relativa de sus ejecuciones.
inanición (*starvation*)	Situación en la cual un proceso preparado para avanzar es soslayado indefinidamente por el planificador; aunque es capaz de avanzar, nunca se le escoge.

Conceptos - Concurrencia - Principales dificultades

Sistemas monoprocesador

- Compartir recursos globales esta lleno de riesgos
  El orden de las lecturas y escrituras es crítico
- Difícil la asignación óptima de recursos
  Un proceso que adquiere el control de un canal de E/S y antes de usarlo se suspende.
  ¿Qué hace el S.O. con el canal?
- Difícil la depuración
  Los resultados de distintas ejecuciones no son ni deterministas, ni reproducibles.

Sistemas multiprocesador

Además se problemas propios por la ejecución simultanea de procesos

Ejemplo: Sea un sistema multiprogramado (monoprocesador o multiprocesador):

```
global char in;
void echo(){
     in = getchar();
     printf("%c",in);
}
```

Ejecutamos 2 veces *(2 procesos)*

Cambio se produce en cualquier momento

Condición de carrera

Una condición de carrera sucede cuando múltiples procesos/hilos leen y escriben datos de manera que el resultado final depende del orden de ejecución.

P1	P2
in = getchar();	
	in = getchar();
	printf("%c",in);
printf("%c",in);	

¿Cual será la salida? ¿Cual será la solución?

Conceptos - Concurrencia – Aspectos de Diseño y Gestión

- El S.O. debe ser capaz de seguir la pista de distintos procesos
  → Gracias a los PCBs
- El S.O. debe ubicar y desubicar varios recursos para cada proceso activo:
  - Tiempo de procesador → Planificación de procesos
  - Memoria → Gestión de la memoria virtual
  - Ficheros → Gestión de ficheros
  - Dispositivos de E/S → Gestión de E/S
- El S.O. debe proteger los datos y recursos de cada proceso
- El funcionamiento y resultado de un proceso debe ser independiente de la velocidad de su ejecución en relación con otros procesos.

Conceptos - Interacción de procesos

Procesos No Cooperantes.

- Competidores entran en conflicto cuando compiten por el mismo recurso.
- Los procesos utilizan los recursos y los dejan como estaban.
- Los procesos pueden verse afectados por asignaciones de recursos.
- Si hay competencia se plantean 3 problemas a resolver
  1. Exclusión Mutua
     - Recursos Críticos
     - Sección Crítica
  2. Interbloqueo
  3. Inanición
- El SO debe actuar
- Los procesos deben ser proactivos.

```
/* program ExclusiónMutua */
const    int n = /* número de procesos */;

void P(int i)
{
while (cierto)
    {
       entrada_crítica(i);
       /* sección crítica */;
       salida_crítica(i);
       /* resto */;
    }
}
void main()
{
    parbegin(P(R1), P(R2), ..., P(Rn));
}
```

Figura 5.1. Exclusión mutua.

Conceptos - Interacción de procesos

Procesos Cooperantes por Compartición

- Los procesos son conscientes de que otros pueden acceder a los recursos compartidos, pero no saben quienes son (que PID tienen).
- Los procesos deben cooperar para garantizar la integridad de los datos compartidos.
- Dos tipos de acceso a los datos (lectura – escritura) con problemas:
  1. Exclusión Mutua
  2. Interbloqueo
  3. Inanición
- Sólo las operaciones de escritura deben ser mutuamente excluyentes.

- La coherencia de los datos debe mantenerse

```
P1:
      a = a + 1;
      b = b + 1;
P2:
      b = 2 * b;
      a = 2 * a;
```

```
a = a + 1;
b = 2  * b;
b = b + 1;
a = 2  * a;
```

a	3		a	3
b	5		b	5
a=a+1	4		a=a+1	4
b=b+1	6		b=2*b	10
b=2*b	12		b=b+1	11
a=2*a	8		a=2*a	8
a	8		a	8
b	12		b	11

Conceptos - Interacción de procesos

Procesos Cooperantes por Comunicación

- Los procesos son conocedores de la identidad de los otros procesos, conocen el PID.
- Están diseñados para colaborar en la consecución de un fin
- Para ello necesitan comunicarse
  - Para sincronizarse
  - Para coordinarse
- Normalmente se utilizan primitivas de paso de mensajes
  - Del lenguaje de programación (clusters, pvm, mpi)
  - Del Kernel del Sistema Operativo (send, receive)
- No se comparte nada,
  → **no es necesaria la exclusión mutua**.
- Los problemas de
  Interbloqueo e Inanición
  siguen presentes

Tabla 5.2. Interacción de procesos.

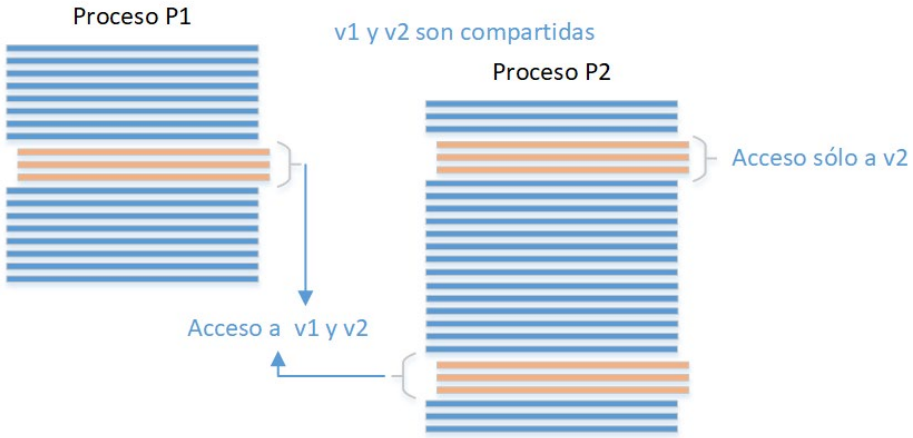
Grado de percepción	Relación	Influencia que un proceso tiene sobre el otro	Potenciales problemas de control
Procesos que no se perciben entre sí	Competencia	• Los resultados de un proceso son independientes de la acción de los otros La temporización del proceso puede verse afectada	• Exclusión mutua • Interbloqueo (recurso renovable) • Inanición
Procesos que se perciben indirectamente entre sí (por ejemplo, objeto compartido)	Cooperación por compartición	• Los resultados de un proceso pueden depender de la información obtenida de otros • La temporización del proceso puede verse afectada	• Exclusión mutua • Interbloqueo (recurso renovable) • Inanición • Coherencia de datos
Procesos que se perciben directamente entre sí (tienen primitivas de comunicación a su disposición)	Cooperación por comunicación	• Los resultados de un proceso pueden depender de la información obtenida de otros • La temporización del proceso puede verse afectada	• Interbloqueo (recurso consumible) • Inanición

Conceptos - Sección crítica y Exclusión mutua

Sección Crítica

Es la **zona de código** (de un proceso) donde se accede a los **recursos compartidos** (con otros procesos) y que **no puede ser ejecutada cuando otro proceso esté en la misma sección crítica**.

Dos zonas de código (en procesos distintos) diremos que son la misma sección crítica si acceden a los mismos recursos compartidos (a todos o a algunos)



Conceptos - Sección crítica y Exclusión mutua

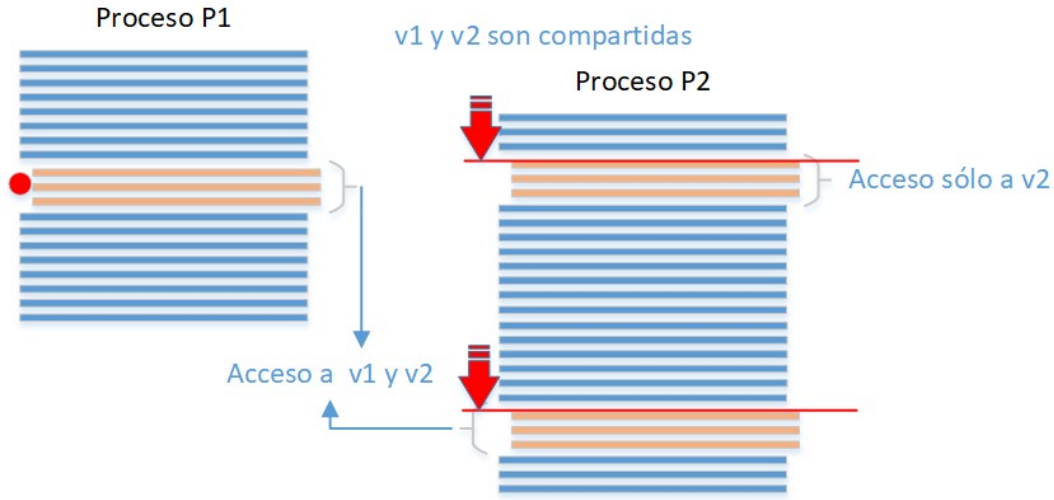
Sección Crítica

Es la **zona de código** (de un proceso) donde se accede a los **recursos compartidos** (con otros procesos) y que **no puede ser ejecutada cuando otro proceso esté en la misma sección crítica**.

Dos zonas de código (en procesos distintos) diremos que son la misma sección crítica si acceden a los mismos recursos compartidos (a todos o a algunos)

Exclusión Mutua

Es el requisito que garantiza que dos procesos, que comparten secciones críticas, no pueden ejecutar simultáneamente dentro de ellas.



Conceptos - Sección crítica y Exclusión mutua

Sección Crítica

Es la **zona de código** (de un proceso) donde se accede a los **recursos compartidos** (con otros procesos) y que **no puede ser ejecutada cuando otro proceso esté en la misma sección crítica**.

Dos zonas de código (en procesos distintos) diremos que son la misma sección crítica si acceden a los mismos recursos compartidos (a todos o a algunos)

Exclusión Mutua

Es el requisito que garantiza que dos procesos, que comparten secciones críticas, no pueden ejecutar simultáneamente dentro de ellas.

Soluciones para garantizar la Exclusión Mutua

Soluciones Software (Espera Activa)

Algoritmos de Deckker y Peterson

Soportadas por el Hardware

- Instrucciones hardware atómicas (Test & Set, Exchange)
- Inhabilitación de Interrupciones

Soportadas por el Sistema Operativo y lenguajes de programación

- Semáforos
- Monitores
- Paso de Mensajes

Conceptos - Sección crítica y Exclusión mutua

Requisitos para la Exclusión Mutua

- Sólo un proceso debe tener permiso para entrar en la sección crítica por un recurso en un instante dado.
- Un proceso que se interrumpe en una sección crítica debe hacerlo sin interferir con los otros procesos.
- No puede permitirse el interbloqueo o la inanición.
- Cuando ningún proceso está en su sección crítica, cualquier proceso que solicite entrar en la suya debe poder hacerlo sin dilación.
- No se deben hacer suposiciones sobre la velocidad relativa de los procesos o el número de procesadores.
- Un proceso permanece en su sección crítica sólo por un tiempo finito.

Exclusión Mutua

SOLUCIONES SOFTWARE

Soluciones Software – Algoritmo de Dekker

Espera activa:

- Un proceso siempre está esperando a entrar en su sección crítica.
- Un proceso no puede hacer nada productivo hasta que obtiene permiso para entrar en su sección crítica.

/* PROCESO 0 */	/* PROCESO 1 */
...	...
while (turno != 0)	**while** (turno != 1)
/* no hacer nada*/;	/* no hacer nada*/;
/*sección crítica*/;	/*sección crítica*/;
turno = 1 ;	turno = 0 ;
...	...

Garantiza la exlusión mutua.

Problemas

1. Alternancia estricta. El ritmo de ejecucíón lo marca el más lento
2. Un fallo en un proceso provoca un Interbloqueo. Tanto si el fallo es en la Sección Crítica o no.

Corrutinas

- Diseñadas para poder pasar el control de la ejecución entre ellas.
- No es una técnica apropiada para dar soporte al procesamiento concurrente.

Soluciones Software – Algoritmo de Dekker

/* PROCESO 0 */	/* PROCESO 1 */
...	...
while (señal[1])	**while** (señal[0])
/* no hacer nada*/;	/* no hacer nada*/;
señal[0] := true;	señal[1] := true;
/*sección crítica*/	/*sección crítica*/
señal[0] := false;	señal[1] := false;
...	...

Si el proceso falla fuera de su sección crítica, el otro puede continuar. (fallo excluye las asignaciones)
Si el proceso falla en su sección crítica, el otro se bloquea.
No garantiza la exclusión mútua

P0 while y encuentra señal[1]=false
P1 while y encuentra señal[0]=false
P0 señal[0]=true y entra
P0 señal[1]=true y entra

Cada proceso debe tener la llave de su sección crítica. Si el otro falla podrá continuar.

Cada proceso puede examinar el estado del otro pero no lo puede alterar.

Cuando un proceso desea entrar en su sección crítica comprueba en primer lugar el otro proceso.

Si no hay otro proceso en su sección crítica fija su estado para la sección crítica.

Este método no garantiza la exclusión mutua.

Cada proceso puede comprobar las señales y luego entra en la sección crítica al mismo tiempo.

Soluciones Software – Algoritmo de Dekker

/* PROCESO 0 */	/* PROCESO 1 */
...	...
señal[0] := true;	señal[1] := true;
while (señal[1])	**while** (señal[0])
/* no hacer nada*/;	/* no hacer nada*/;
/*sección crítica*/	/*sección crítica*/
señal[0] := false;	señal[1] := false;
...	...

Si el proceso falla fuera de su sección crítica, el otro puede continuar. (fallo excluye las asignaciones)
Si el proceso falla en su sección crítica, el otro se bloquea.
Garantiza la exclusión mutua
Problema
Se provocará un interbloqueo cuando dos procesos den valor a sus señales antes de entrar en la sección crítica.

El anterior falla porque se permite cambiar el estado después de la consulta del otro proceso

Dar valor a la señal para entrar en la sección crítica antes de comprobar otros procesos.

Si hay otro proceso en la sección crítica cuando se ha dado valor a la señal, el proceso queda bloqueado hasta que el otro proceso abandona la sección crítica.

Soluciones Software – Algoritmo de Dekker

```
/* PROCESO 0 */
...
señal[0] := true;
while (señal[1] ) {
   señal[0]:=false;
   /*esperar*/
   señal[0]:=true;
}
/*sección crítica*/
señal[0]:=false;
...
```

```
/* PROCESO 1 */
...
señal[1] := true;
while (señal[0] ) {
   señal[1]:=false;
   /*esperar*/
   señal[1]:=true;
}
/*sección crítica*/
señal[1]:=false;
...
```

Casi correcta. Se produce un **livelock (bloqueo vital)**

P0 pone `señal[0]` a cierto.
P1 pone `señal[1]` a cierto.
P0 comprueba `señal[1]`.
P1 comprueba `señal[0]`.
P0 pone `señal[0]` a falso.
P1 pone `señal[1]` a falso.
P0 pone `señal[0]` a cierto.
P1 pone `señal[1]` a cierto.

Un cambio en la velocidad relativa resuelve la situación.

Un proceso activa su señal para indicar que desea entrar en la sección crítica, pero debe estar listo para desactivar la variable señal.

Se comprueban los otros procesos. Si están en la sección crítica, la señal se desactiva y luego se vuelve a activar para indicar que desea entrar en la sección crítica. Esta operación se repite hasta que el proceso puede entrar en la sección crítica.

Soluciones Software – Algoritmo de Dekker

```
boolean  señal[2];
int turno;
void P0() {
  while (true){
     señal[0]:=true;
     while(señal[1])
          if turno==1{
               señal[0]:=false;
               while (turno==1)
                    /*no hacer nada*/
               señal[0]:=true;
               }
          /*sección crítica*/
          turno:=1;
          señal[0]:=false;
          /*resto*/
     }
}
```

```
void P1() {
  while (true){
     señal[1]:=true;
     while(señal[0])
          if turno==0{
               señal[1]:=false;
               while (turno==0)
                    /*no hacer nada*/
               señal[1]:=true;
               }
          /*sección crítica*/
          turno:=0;
          señal[1]:=false;
          /*resto*/
     }
}

void main(){
    señal[0]:=false;
    señal[1]:=false;
    turno:=1
    parbegin(P0, P1)
}
```

Solución Correcta

- Cada proceso tiene un turno para entrar en la sección crítica.
- Si un proceso desea entrar en la sección crítica, debe activar su señal y puede que tenga que esperar a que llegue su turno.

Algoritmo de Dekker

Espera activa:

- Un proceso siempre está esperando a entrar en su sección crítica.
- Un proceso no puede hacer nada productivo hasta que obtiene permiso para entrar en su sección crítica.

/* PROCESO 0 */	/* PROCESO 1 */
...	...
while (turno != 0)	**while** (turno != 1)
/* no hacer nada*/;	/* no hacer nada*/;
/*sección crítica*/;	/*sección crítica*/;
turno = 1 ;	turno = 0 ;
...	...

Garantiza la exlusión mutua.

Problemas

1. Alternancia estricta. El ritmo de ejecucíón lo marca el más lento
2. Un fallo en un proceso provoca un Interbloqueo. Tanto si el fallo es en la Sección Crítica o no.

Corrutinas

- Diseñadas para poder pasar el control de la ejecución entre ellas.
- No es una técnica apropiada para dar soporte al procesamiento concurrente.

Algoritmo de Dekker

/* PROCESO 0 */	/* PROCESO 1 */
...	...
while (señal[1])	**while** (señal[0])
/* no hacer nada*/;	/* no hacer nada*/;
señal[0] := true;	señal[1] := true;
/*sección crítica*/	/*sección crítica*/
señal[0] := false;	señal[1] := false;
...	...

Si el proceso falla fuera de su sección crítica, el otro puede continuar. (fallo excluye las asignaciones)
Si el proceso falla en su sección crítica, el otro se bloquea.
No garantiza la exclusión mútua

P0 while y encuentra señal[1]=false
P1 while y encuentra señal[0]=false
P0 señal[0]=true y entra
P0 señal[1]=true y entra

Cada proceso debe tener la llave de su sección crítica. Si el otro falla podrá continuar.

Cada proceso puede examinar el estado del otro pero no lo puede alterar.

Cuando un proceso desea entrar en su sección crítica comprueba en primer lugar el otro proceso.

Si no hay otro proceso en su sección crítica fija su estado para la sección crítica.

Este método no garantiza la exclusión mutua.

Cada proceso puede comprobar las señales y luego entra en la sección crítica al mismo tiempo.

Algoritmo de Dekker

/* PROCESO 0 */	/* PROCESO 1 */
...	...
señal[0] := true;	señal[1] := true;
while (señal[1])	**while** (señal[0])
/* no hacer nada*/;	/* no hacer nada*/;
/*sección crítica*/	/*sección crítica*/
señal[0] := false;	señal[1] := false;
...	...

Si el proceso falla fuera de su sección crítica, el otro puede continuar. (fallo excluye las asignaciones)
Si el proceso falla en su sección crítica, el otro se bloquea.
Garantiza la exclusión mutua
Problema
Se provocará un interbloqueo cuando dos procesos den valor a sus señales antes de entrar en la sección crítica.

El anterior falla porque se permite cambiar el estado después de la consulta del otro proceso

Dar valor a la señal para entrar en la sección crítica antes de comprobar otros procesos.

Si hay otro proceso en la sección crítica cuando se ha dado valor a la señal, el proceso queda bloqueado hasta que el otro proceso abandona la sección crítica.

Algoritmo de Dekker

```
/* PROCESO 0 */                 /* PROCESO 1 */
...                             ...
señal[0] := true;               señal[1] := true;
while (señal[1] ) {             while (señal[0] ) {
   señal[0]:=false;                señal[1]:=false;
   /*esperar*/                     /*esperar*/
   señal[0]:=true;                 señal[1]:=true;
}                               }
/*sección crítica*/             /*sección crítica*/
señal[0]:=false;                señal[1]:=false;
...                             ...
```

Casi correcta. Se produce un **livelock (bloqueo vital)**

P0 pone `señal[0]` a cierto.
P1 pone `señal[1]` a cierto.
P0 comprueba `señal[1]`.
P1 comprueba `señal[0]`.
P0 pone `señal[0]` a falso.
P1 pone `señal[1]` a falso.
P0 pone `señal[0]` a cierto.
P1 pone `señal[1]` a cierto.

Un cambio en la velocidad relativa resuelve la situación.

Un proceso activa su señal para indicar que desea entrar en la sección crítica, pero debe estar listo para desactivar la variable señal.

Se comprueban los otros procesos. Si están en la sección crítica, la señal se desactiva y luego se vuelve a activar para indicar que desea entrar en la sección crítica. Esta operación se repite hasta que el proceso puede entrar en la sección crítica.

Algoritmo de Dekker: Solución correcta

```
boolean  señal[2];
int turno;
void P0() {
  while (true){
     señal[0]:=true;
     while(señal[1])
          if turno==1{
               señal[0]:=false;
               while (turno==1)
                    /*no hacer nada*/
               señal[0]:=true;
               }
          /*sección crítica*/
          turno:=1;
          señal[0]:=false;
          /*resto*/
     }
}
```

```
void P1() {
  while (true){
     señal[1]:=true;
     while(señal[0])
          if turno==0{
               señal[1]:=false;
               while (turno==0)
                    /*no hacer nada*/
               señal[1]:=true;
               }
          /*sección crítica*/
          turno:=0;
          señal[1]:=false;
          /*resto*/
     }
}

void main(){
    señal[0]:=false;
    señal[1]:=false;
    turno:=1
    parbegin(P0, P1)
}
```

- Cada proceso tiene un turno para entrar en la sección crítica.
- Si un proceso desea entrar en la sección crítica, debe activar su señal y puede que tenga que esperar a que llegue su turno.

Exclusión Mutua

SOLUCIONES HARDWARE

Exclusión Mutua: Soluciones Hardware

Inhabilitación de interrupciones:

- Un proceso continuará ejecutándose hasta que solicite un servicio del sistema operativo o hasta que sea interrumpido.
- Para garantizar la exclusión mutua es suficiente con impedir que un proceso sea interrumpido.

```
while true{
    /* inhabilitar interrupciones */
    /* seccion critica */
    /* habilitar interrupciones /*
    /* resto; }
```

- Problemas:
  - Se limita la capacidad del procesador para intercalar programas.
  - Multiprocesador:

    Inhabilitar las interrupciones de un procesador no garantiza la exclusión mutua.

Exclusión Mutua: Soluciones Hardware

Instrucciones especiales de máquina

- Se realizan en un único ciclo de instrucción.
- No están sujetas a injerencias por parte de otras instrucciones.
  - Leer y escribir.
  - Leer y examinar.

Exclusión Mutua: Soluciones Hardware

La instrucción Comparar y Fijar (Test and Set)

```
booleano TS (int i){
if (i == 0) {
   i = 1;
   return cierto;
   }
else return falso;
}
```

```
/* program exclusionmutua */
const int n= /*numero de procesos*/
int cerrojo;
void P(int i) {
   while (true) {
        while(! TS(cerrojo))
             /*no hacer nada*/
        /*Seccion crítica*/
        cerrojo := 0;
        /*resto*/
    }
}
void main (){
   cerrojo := 0;
   parbegin(P(1), P(2), ..... , P(n));
}
```

Exclusión Mutua: Soluciones Hardware

La instrucción Intercambiar

```
void intercambiar(int registro, int memoria){
     int temp;
     temp = memoria;
     memoria = registro;
     registro = temp;
}
```

```
/* program exclusionmutua */
const int n= /*numero de procesos*/
int cerrojo;
void P(int i) {
   int clave
   while (true) {
        clave :=1;
        while( clave != 0 )
             intercambiar(clave, cerrojo)
        /*Seccion crítica*/
        intercambiar(clave,cerrojo);
        /*resto*/
    }
}
void main (){
   cerrojo := 0;
   parbegin(P(1), P(2), ..... , P(n));
}
```

Exclusión Mutua: Soluciones Hardware

Ventajas:

- Es aplicable a cualquier número de procesos en sistemas con memoria compartida, tanto de monoprocesador como de multiprocesador.
- Es simple y fácil de verificar.
- Puede usarse para disponer de varias secciones críticas.

Exclusión Mutua: Soluciones Hardware

Desventajas:

- La espera activa consume tiempo del procesador.
- Puede producirse inanición cuando un proceso abandona la sección crítica y hay más de un proceso esperando.
- Interbloqueo:

  Si un proceso con baja prioridad entra en su sección crítica y existe otro proceso con mayor prioridad, entonces el proceso cuya prioridad es mayor obtendrá el procesador para esperar a poder entrar en la sección crítica.

Fin

UNIDAD 3 – SISTEMA DE PROCEOSS

PRINCIPIOS DE LA CONCURRENCIA