UNIVERSIDAD FRANCISCO DE VITORIA
Computación de Alto Rendimiento

COMPUTACION DE ALTO RENDIMIMIENTO

Secciones Críticas y Exclusión Mutua

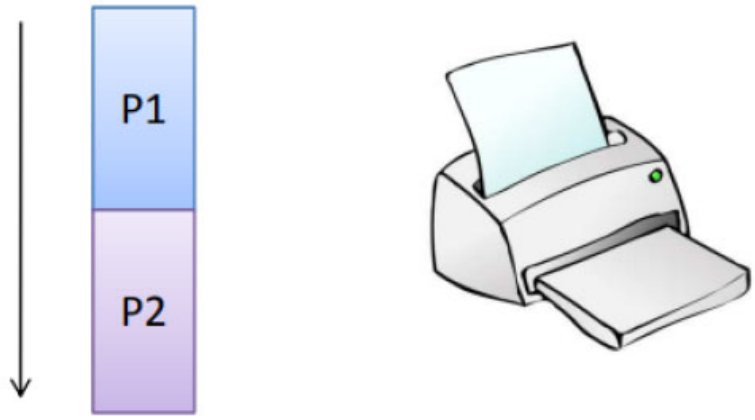
Grado en Ingeniería Informática

Curso 2022/2023

Problemas asociados a la concurrencia

El problema de la sección Crítica

- Cuando un proceso accede a un dato (o recurso) compartido decimos que éste es un recurso crítico y la parte de programa que lo usa se llama sección crítica SC
- La ejecución de secciones críticas deben ser *mutuamente exclusivas: en cualquier instante sólo un proceso puede ejecutar la SC.*
- Todos los procesos necesitan de un "permiso" antes de entrar en la SC.



Conceptos asociados al problema de la sección Crítica

- La parte de código antes de la SC se denomina ***sección de entrada***.
- La parte código que sigue a la SC se denomina ***sección de salida .***
- El problema de la SC consiste en establecer un *protocolo* de utilización del recurso independiente del orden y la velocidad de ejecución de los procesos.

Soluciones ?

- Asumimos que la velocidad relativa de ejecución de un proceso es != 0

- Estructura general de un proceso

```
repeat
   sección de entrada
    section crítica SC
   sección de salida
    sección restante SR
forever
```

- Soluciones: debemos describir las secciones de entrada y de salida.

Requisitos para la Exclusión Mutua

- Sólo un proceso debe tener permiso para entrar en la sección crítica por un recurso en un instante dado.
- No puede permitirse el interbloqueo o la inanición.
- Cuando ningún proceso esta 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 numero de procesadores.
- Un proceso permanece en su sección crítica solo por un tiempo finito.

Tipos de solución

- Soluciones software puras
  - Responsabilidad de cada proceso para llevar la coordinación con los otros.
- Soluciones hardware
  - se asume la existencia de instrucciones especiales a nivel del procesador
- Soluciones a nivel del SO (semáforos)
  - Se disponen de *funciones y estructuras* de datos a disposición del programador

.

Soluciones software puras (Lenguajes que implementan concurrencia Ej. ADA, JAVA, etc

- Consideraremos 2 procesos
  - Algoritmos 1 (Decker)
  - Algoritmo 2 no válido
  - Algoritmo 3 válido (algoritmo de Peterson)
- Generalización a n procesos
  - Algoritmo de Lamport (“panadero”)
- Notación
  - Dos procesos: P0 y P1
  - Mas generalmente tendremos dos procesos Pi Pj (i != j)

Algoritmo 1 (Decker)

- La variable compartida *turno* se inicializa a i o a j antes de jecutar Pi
- La SC de Pi se ejecuta si turno = i
- Pi está en espera "activa" si Pj está dentro de su SC. Exclusión mutua ok!
- Existe una alternancia estricta. El ritmo de ejecución depende del mas lento.
- Si un proceso falla dentro o fuera (while) de su SC el otro nunca podrá entrar a su SC. !!!!!!!

```
Process Pi:
repeat
   while(turno!=i){};
      SC
   turno:=j;
      SR
forever
```

Algoritmo 2 (Ej. de solución no válida)

- Une variable booleana(0,1) por Proceso: flag[0] y flag[1] inicializadas a FALSE
- Pi indica que desea entrar en su zona crítica haciendo : flag[i]:=true
- Exclusión mutua ok ?
- Si uno falla fuera de la SC el otro no está bloqueado BIEN !!:
- Consideremos la secuencia:
  - T0: flag[0]:=true
  - T1: flag[1]:=true
- Ambos procesos están en su SC NO SIRVE !!!!

```
Process Pi:
repeat
   flag[i]:=true;
   while(flag[j]){};
      SC
   flag[i]:=false;
      SR
forever
```

Algoritmo 3 (Peterson)

- Inicio: flag[0]:=flag[1]:=fal se turno:= i o j
- El deseo de entrar en la zona crítica se indica con flag[i]:=true
- flag[i]:= false en la sección de salida.
- Si P0 y P1 intentan simultáneamente entrar en su SC se lo impide la variable *turno*

```
Process Pi:
repeat
   flag[i]:=true;
   turno:=j;
   do {} while
   (flag[j]and turno=j);
      SC
   flag[i]:=false;
      SR
forever
```

Algoritmo 3: prueba

- Exclusion mutua :
  - Pi y Pj estarían ambos en su SC si flag[0] = flag[1] = false y solamente si turno != j en ambos (imposible)
  - Pi no puede entrar en su SC si se mantiene en el bucle while() con la condición: flag[j] = true and turn = j. (imposible)
  - Pj no está interesado en entrar en su SC. Este caso es imposible porque implica flag[j] = false

Solución a n procesos: algoritmo de Lamport (panadería)

- Antes de ejecutar la SC cada proceso recoge un número. Aquel que tenga el número mas pequeño entra en la SC.
- Si Pi y Pj tienen el mismo número:
  - si i<j Pi entrará primero, si no Pj
- Pi pone su número a 0 en la Sec. de Salida
- Notación:
  - (a,b) < (c,d) si a < c o si a = c y b < d
  - max(a0,...ak) es un número b tal que:
    - b >= ai para i=0,..k

algoritmo de la panadería (cont.)

- Variables globales:
  - choosing: array[0..n-1] of boolean;
    - inicializado a false
  - number: array[0..n-1] of integer;
    - inicializados a 0
- Validez:
  - Si Pi está dentro SC es que Pk ha escogido un number[k]!= 0, y (number[i],i) < (number[k],k)

El algoritmo de la panadería (cont.)

```
Process Pi:
repeat
   choosing[i]:=true;
   number[i]:=max(number[0]..number[n-1])+1;
   choosing[i]:=false;
   for j:=0 to n-1 do {
     while (choosing[j]) {};
     while (number[j]!=0
        and (number[j],j)<(number[i],i)){};
   }
   SC
   number[i]:=0;
   SR
forever
```

Desventajas de las soluciones soft.

- Los Pi que desean entrar en la SC están en ***Espera Activa, consumen CPU***

- Es preferible bloquear (suspender) los procesos que están esperando a entrar en la SC.

Soluciones hardware : desactivación de interrupciones

- Monoprocesador: la exclusión mutua está garantizada mas la eficacia se deteriora: dentro de la SC es imposible entrelazar la ejecución de varios procesos
- Multiprocesador: la exclusión mutua no está garantizada (los mecanismos de interrupción son locales)
- No es una solución aceptable

```
Process Pi:
repeat
   disable interrupts
    critical section
   enable interrupts
    remainder section
forever
```

Soluciones hardware: instrucciones particulares

- Normal: El acceso a una misma dirección de memoria no se puede hacer mediante varios procesos.
- Extensiones: Disponibilidad de instrucciones máquina que ejecutan 2 acciones (ej: test & set) sobre la misma variable de manera atómica (indivisible)
- La ejecución de de esta instrucción es mutuamente exclusiva incluso si tenemos varios procesadores
- Necesitamos algoritmos mas complejos para satisfacer las 3 exigencias principales del problema de la sección crítica

Deshabilitar interrupciones

- Monoprocesador. No hay mecanismos de interrupción entre procesadores en el que pueda basarse la exclusión mutua.
- Se añade al kernel la posibilidad de deshabilitar las interrupciones por un proceso.
- Un proceso continuara 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.
- Se limita la capacidad del procesador para intercalar programas.

```
while (true){
      …
      /*deshabilitar interrupciones*/
      /*sección crítica*/
      /*habilitar interrupciones*/
      …
}
```

Instrucciones especiales de máquina

- Multiprocesador compartiendo memoria principal.
- Al acceder a una posición de memoria, se excluye cualquier otro acceso a la misma posición (exclusión mutua).
- Con esta instrucción se facilita la creación de semáforos.

Instrucción TEST&SET

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

Instrucción intercambiar

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

Instrucción test and set

- Ejemplo

```
/* program mutualexclusion */
const int nproc = //number of processes;

int cerrojo;

void main(){
    cerrojo = 0;
    parallel_begin(P(1), P(2), …, P(n))
}

void P(int i){
    while (true) {
          while (!testset(cerrojo))
              /* do nothing */;

          /* critical section */;
           cerrojo = 0;
          /* remainder */;
    }
}
```

Ejecución de varios procesos en paralelo

El único proceso que puede entrar en su sección crítica es aquel que encuentre la variable cerrojo igual a 0

Al salir de la sección crítica vuelve a establecer el valor de cerrojo a 0

Instrucción Exchange

- Utilizada en procesadores Intel

```
void exchange (int register, int memory)
{
    int temp;
    temp = memory;
    memory = register;
    register = temp;
}
```

- Se realiza un intercambio entre un registro del procesador y una posición de memoria

Instrucción Exchange

- Ejemplo

```
/* program mutualexclusion */
int const n = /* number of processes*/;

int cerrojo;

void main()
{
    cerrojo = 0;
    parbegin (P(1), P(2), ..., P(n));
}

void P(int i){
    int llavei = 1;
    while (true) {
        do exchange (llavei, cerrojo)
        while (llavei != 0);
        /* critical section */;
        cerrojo = 0;
        /* remainder */;
    }
}
```

Se inicializa *cerrojo* a 0

El único proceso que puede entrar en su sección crítica es aquel que encuentre la variable *cerrojo* igual a 0

Al salir de la sección crítica vuelve a establecer el valor de *cerrojo* a 0

Ventajas de las Instrucciones Máquina

- Aplicable a uno o varios procesadores y a cualquier número de procesos.
- Algoritmo simple.
- Puede utilizarse con múltiples secciones críticas cada una de ellas con una variable distinta.

Inconvenientes de las Instrucciones Máquina

- **Existe espera activa**: consumo de procesador durante la espera.

- **Posible inanición**: no existe un método de elección del proceso a entrar.