

# Control de Procesos

## Procesos

Sistemas Operativos

## Objetivos



- Aplicar el modelo de estado de procesos en el sistema
- Identificar problemas de concurrencia
- Aplicar el algoritmo de Peterson
- Analizar una solución al problema de la exclusión mutua con ayuda del hardware.

Control de Procesos

## ¿Qué es un proceso?

- Programa en ejecución
- ¿Qué caracteriza a un proceso?
  - Contexto del proceso
  - Estructura de datos

Process Control Block →

Identification	Process Id Parent PID User Id
CPU state	User registers Program counter Status info (interrupt enable ags) Stack pointer
Process scheduling	Process state Priority Wait-time Event being waited on Privileges
IPC	Flags Signals Messages
Memory	
Resources Resource Unit Owned/	
Created resources	
Children	

Control de Procesos

## Modelo de estado de procesos



## ¿Para qué compartir recursos?

- Multiprogramación:
- Colaboración:
  - compartir recursos
- El problema
  - dos o más procesos pueden entrar en conflicto cuando se tiene acceso simultáneo a recursos compartidos

Control de Procesos

## ¿Por qué es necesaria la concurrencia?

- ¿Qué es concurrencia?
  - Concurrencia = ejecución paralela o pseudo-paralela con acceso a recursos compartidos.

Control de Procesos

## problema

- Suponga que Ud. Fue contratado para programar un contador de visitas de un sitio Web
  - consiste en determinar cuántas veces una página HTML fue visitada en un servidor WWW

Control de Procesos

## El programa contador de visitas

```
.  
.   
.   
int contador = 0;  
void contar( )  
{  
    contador++;  
}
```

Control de Procesos

## Una variable compartida

- ejemplo de ejecución de dos procesos que llaman a la función `contar()` en paralelo.
- `contador` es una variable global (***variable compartida***) que tiene acceso concurrente.

Control de Procesos

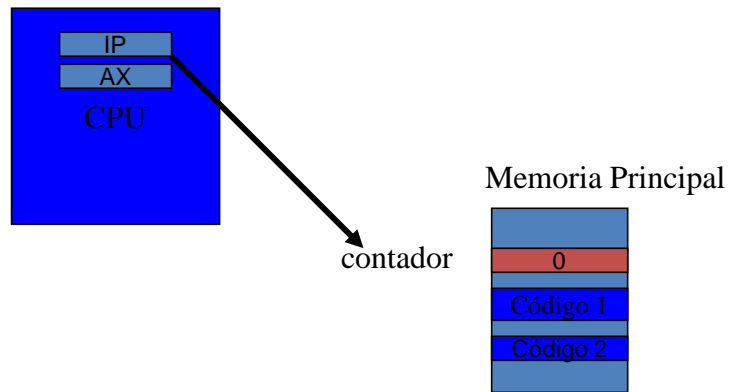
## Assembler, el micromundo de la CPU

- En la CPU una operación `contador++` se ejecuta como varias instrucciones de máquina, en assembler:

<code>MOVE AX, [CONTADOR]</code>	<i>(cargar valor en el acumulador)</i>
<code>INC AX</code>	<i>(sumarle 1 al acumulador)</i>
<code>MOVE [CONTADOR], AX</code>	<i>(escribir acumulador de vuelta en memoria)</i>

Control de Procesos

## Contexto de variables



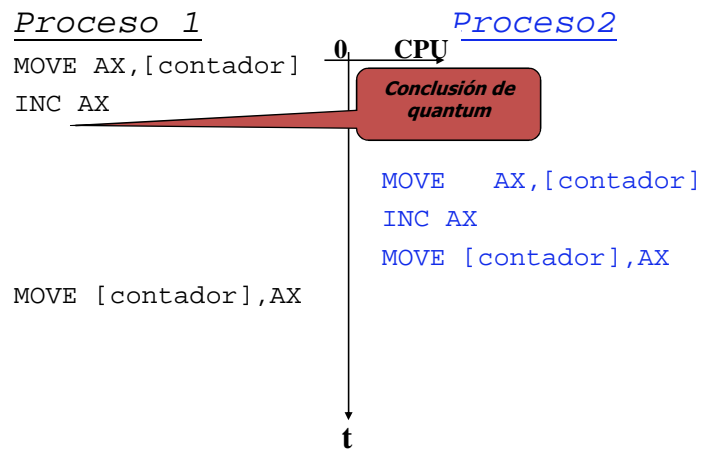
Control de Procesos

## Supongamos

- más o menos simultáneamente, los dos procesos abren un documento.
- inicialmente la variable compartida (global), `contador=0`

Control de Procesos

## Diagrama de ejecución(Trace)



Control de Procesos

## Condición de competencia

- Problema:
  - El resultado es que el registro en memoria [contador] queda en 1, pero lo correcto sería 2.
- ¿Qué causa?
- El problema se produce porque los dos procesos tratan de actualizar una variable compartida al mismo tiempo
- ¿Cómo se produce?
- Un proceso comienza a actualizar cuando el otro no ha terminado.
  - Esto se conoce como *race condition* (condición de carrera)
    - **Definición:** cuando el resultado depende del orden particular en que se intercalan las operaciones de procesos concurrentes.

Control de Procesos

## La sección crítica

- **Definición:** si un proceso está accediendo a variables compartidas se dice que está en la *sección crítica*
- *Solución al problema de acceso a la sección crítica*
  - Se resuelve si se garantiza que sólo un proceso a la vez puede estar actualizando variables compartidas. A esto se conoce como la exclusión mutua.
- **Definición:**
  - *exclusión mutua*, nunca hay más de un proceso ejecutando en su sección crítica.

Control de Procesos

## Condiciones de la solución

- Ausencia de postergación indefinida.
  - No debe existir discriminaciones
- Entrada eventual.
  - Si un proceso entra a la sección crítica, entonces después de trabajar debe salir de la sección crítica, dando así la posibilidad de que otro proceso ingrese a la sección crítica .

Control de Procesos



# Implementación de la exclusión mutua

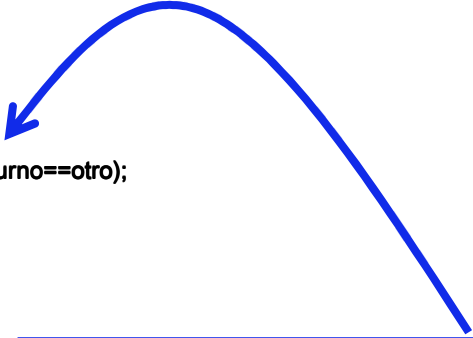
- **Modelo:**

```
Entrada_exclusión_mutua;  
Sección_Critica;  
Salida_exclusión_mutua;  
sección no crítica;
```

Control de Procesos

## El algoritmo de Peterson

```
void proceso(i)  
{  
    int otro=1-i;  
    interesado[i] = 1;  
    turno = otro;  
    while (interesado[otro] && turno==otro);  
    sección crítica  
    interesado[i]= 0;  
    sección no crítica  
}  
void main(){  
    interesado[0]=interesado[1]=0  
    turno=0;
```



i	turno	otro	interesado[0]	interesado[1]
0	1	1	1	1
1	0	0	1	1

Control de Procesos

## La instrucción Test-and-Set-Lock (TSL)

- TSL (a,b) lee el contenido de una dirección b de memoria en un registro a y pone un 1 en la dirección b
  - todo en una sola acción atómica o indivisible, ningún otro proceso puede acceder a esa dirección de memoria hasta que la instrucción se complete.

TSL(a,b)

$$\left. \begin{array}{l} a \leftarrow b \\ b \leftarrow 1 \end{array} \right\} \text{ Se ejecutan sin interrupciones}$$

Control de Procesos

## Implementación de TSL

- En Intel
  - XCH A, B
  - Intercambia los contenidos de A y B

Control de Procesos

## Ejemplo de aplicación de TSL

```
void proceso() {  
    desealngresarPrimero=1;  
    While(desealngresarPrimero)  
        TLS(desealngresarPrimero,salir); //espera activa  
        Sección crítica  
        salir=0;  
        Sección no crítica  
}  
void main(){  
    int salir=0
```

Control de Procesos

- Fin

Control de Procesos