CONCURRENCIA

Karim Guevara Puente de la Vega

2017

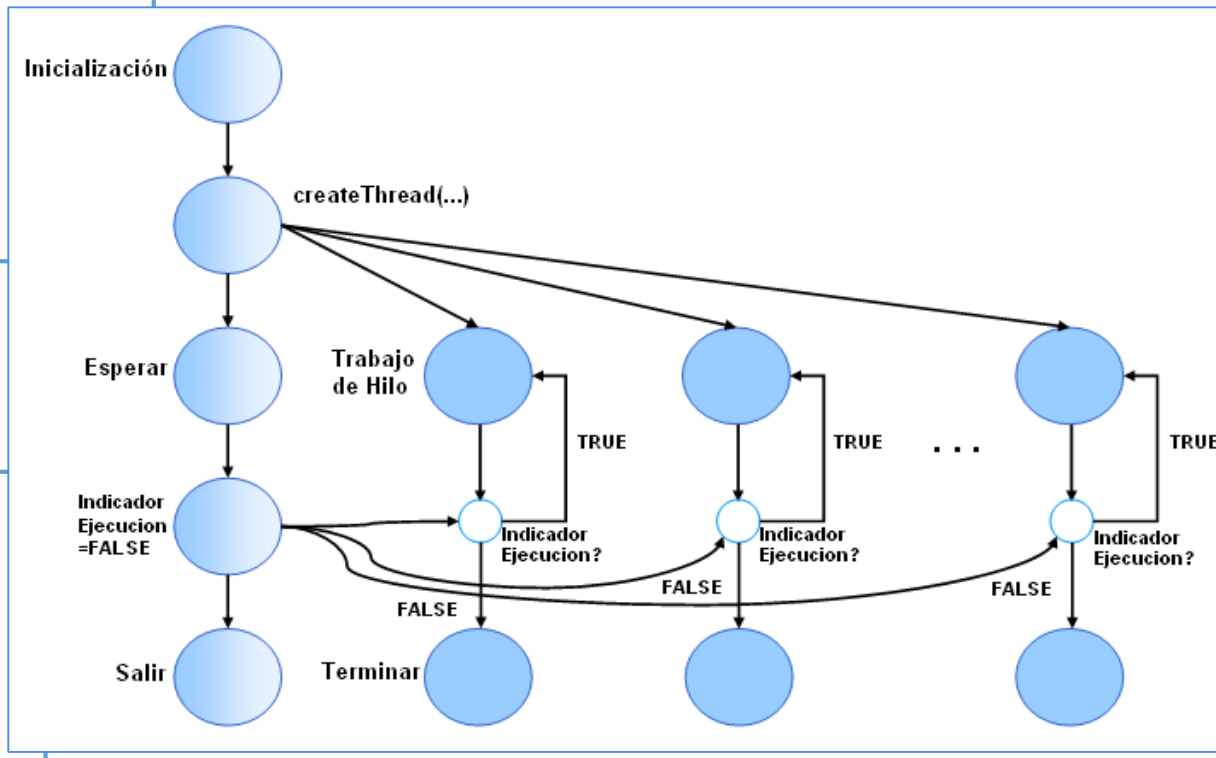
Introducción

- Procesos cooperan en sus tareas, durante su ciclo de vida:
  - Entre ellos o con el proceso padre que los haya creado
  - Acceso controlado a estructuras de datos compartidas
    - P.e. ls –l /etc/* | more

- **Sincronización**
  - Acto de asegurarse que los procesos/hilos independientes comienzan a ejecutar un bloque de código concreto en el mismo tiempo lógico.

Sincronización y Comunicación

```
static int indicadorEjecuciion = TRUE
void main() {
    for (i=0; i<N; i++)
            createThread(. . .);
    sleep(1000);
    indicadorEjecucion=FALSE;
    . . .
}
```



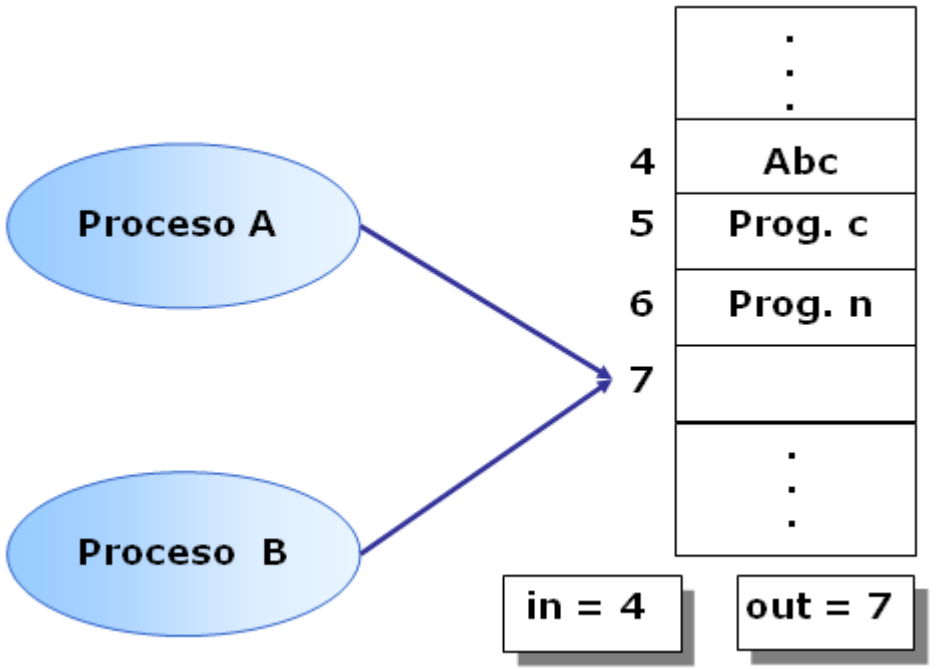

```
TrabajoHilo() {
    . . .
    while (indicadorEjecucion){
            //iteracion de trabajo
    }
    return resultado;
}
```

Sincronización y Comunicación

- Tres problemas:
  - ¿Cómo puede un proceso pasar información a otro proceso?
  - ¿Cómo se asegura que dos o más procesos no se estorben mutuamente al efectuar actividades críticas?
  - ¿Cómo se puede sincronizar la ejecución de los procesos dependientes?

Condición de competencia

- Procesos colaboradores pueden compartir el almacenamiento común
  - Estos leen y escriben en la memoria principal, en un archivo compartido.
  - P.e. spooler de impresió

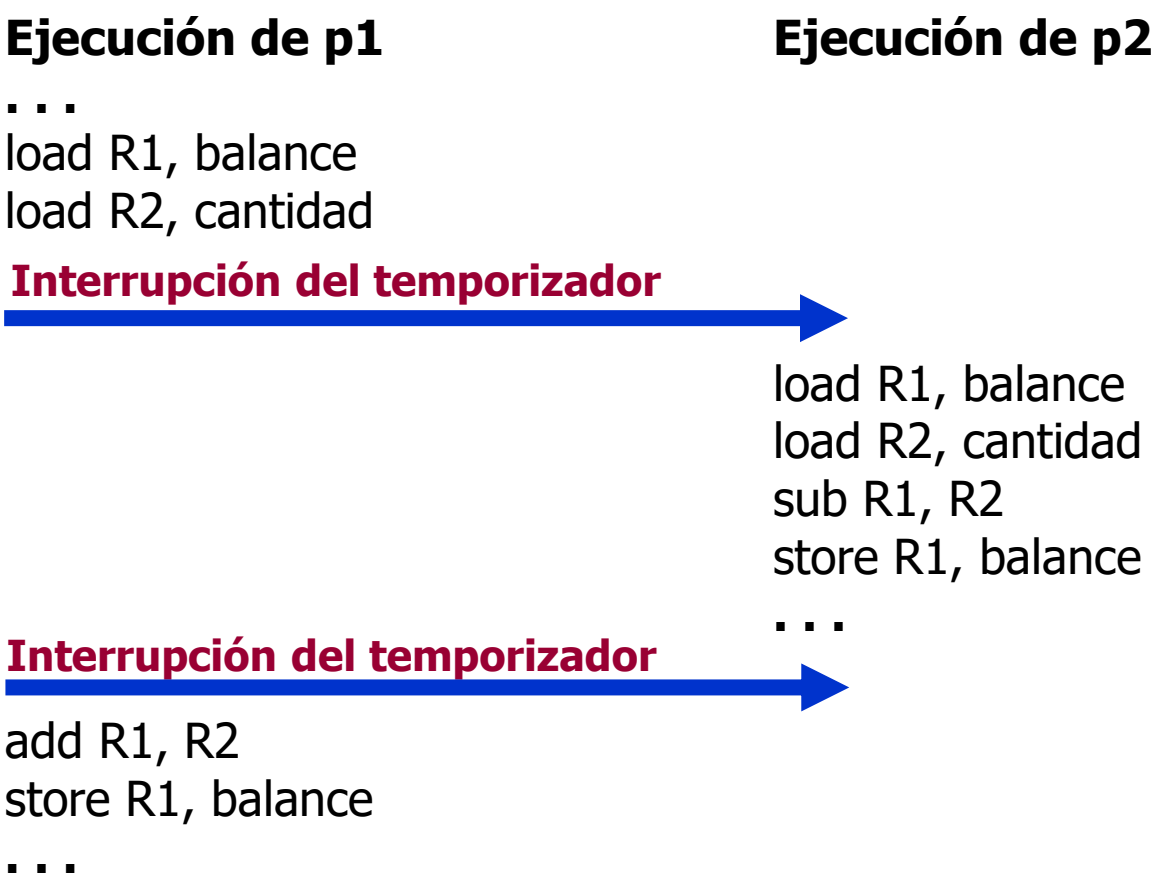


Condición de competencia

- Procesos o hilos cooperativos concurrentes
  PROGRAMACIÓN CONCURRENTE

shared double balance; **Esquema de código para p1** . . . balance=balance+cantidad; . . .	shared double balance; **Esquema de código para p2** . . . balance=balance-cantidad; . . .
load R1, balance load R2, cantidad add R1, R2 store R1, balance	load R1, balance load R2, cantidad sub R1, R2 store R1, balance

Condición de competencia



Condición de competencia

- Data race, race condition
- Cuando el resultado de una operación depende de:
  - Las velocidades relativas de los procesos
  - Del orden particular en que se intercalan las operaciones de los procesos concurrentes
- Lo produce:
  - Procesos paralelos (concurrentes) que comparten datos o recursos
    - P.e. directorio de spooler, in, out, balance

Sección o región crítica

- Se producen en el sw concurrente, siempre que dos procesos/hilos acceden a una variable compartida
  - No deberían ejecutar esa parte del código de forma concurrente

Sección o región crítica

shared double balance;

Esquema de código para p1

. . .

balance=balance+cantidad;

. . .

shared double balance;

Esquema de código para p2

. . .

balance=balance-cantidad;

. . .

```
load R1, balance
load R2, cantidad
add R1, R2
store R1, balance
```

```
load R1, balance
load R2, cantidad
sub R1, R2
store R1, balance
```

Sección o región crítica

- **Problema**: acceso a la variable compartida antes que el proceso que la este modificando la deje en estado consistente
  - Operaciones de acceso a la variable compartida no son atómicas

- ¿cómo evitar las condiciones de competencia?

EXCLUSIÓN MUTUA

  - Si un proceso está haciendo uso de un dato compartido, los otros no pueden hacerlo hasta que el otro termine.

Sección o región crítica

- **¿Cómo implementar la Exclusión mutua?**
  - Porteros: PROTOCOLO ENTRADA/SALIDA

```
Proceso Pi

. . .
while (TRUE) {
    . . .
    entrar_RC    //protocolo de entrada
    Sección crítica
    salir_RC     //protocolo de salida
    . . .
}
. . .
```

Condiciones para el sincronismo / comunicación

- **Exclusión mutua-** dos procesos nunca pueden estar simultáneamente dentro de sus regiones críticas.
- **Ausencia de postergación innecesaria-** un proceso puede ingresar a su SC, si los demás están en sus secciones no críticas
- **Entrada garantizada-** (ausencia de inanición) ningún proceso tendrá que esperar indefinidamente para ingresar a su SC.
- **No hacer suposiciones sobre el hardware**