Sincronización entre procesos

Gonzalo Guillamon

DC - FCEyN - UBA

Sistemas Operativos

Primero repasemos brevemente lo que vieron en la teórica

Primero repasemos brevemente lo que vieron en la teórica

¿Qué es una *race condition*?

Primero repasemos brevemente lo que vieron en la teórica

¿Qué es una *race condition*?

Primero repasemos brevemente lo que vieron en la teórica
¿Qué es una *race condition*?

Defecto en un proceso, donde el resultado del mismo depende inesperadamente del orden en que se ejecuten ciertos eventos.

¿Cuál es el output de los siguientes procesos A y B corriendo simultáneamente y con memoria compartida?

x comienza inicializado en 0.

A

```
x = x + 1;
print(x);
```

B

```
x = x + 1;
print(x);
```

- ¿Qué es un semáforo?

- ¿Qué es un semáforo?

  Es un tipo abstracto de datos que permite controlar el acceso de múltiples procesos a un recurso común.

- ¿Qué es un semáforo?

  Es un tipo abstracto de datos que permite controlar el acceso de múltiples procesos a un recurso común.
- Tiene un valor, al cuál no podemos acceder. La única manera de interactuar con el semáforo es mediante las primitivas wait() y signal()

- ¿Qué es un semáforo?

  Es un tipo abstracto de datos que permite controlar el acceso de múltiples procesos a un recurso común.
- Tiene un valor, al cuál no podemos acceder. La única manera de interactuar con el semáforo es mediante las primitivas wait() y signal()

  Estas primitivas son atómicas a efectos de los procesos.

Recordemos cuáles son las primitivas:

Recordemos cuáles son las primitivas:

Primitivas

- `sem(int value)`: Devuelve un nuevo semáforo inicializado en `value`.

Recordemos cuáles son las primitivas:

Primitivas

- `sem(int value)`: Devuelve un nuevo semáforo inicializado en `value`.
- `wait()`: Mientras el valor sea menor o igual a 0 se bloquea el proceso esperando un `signal`. Luego decrementa el valor de `sem`.

Recordemos cuáles son las primitivas:

Primitivas

- `sem(int value)`: Devuelve un nuevo semáforo inicializado en `value`.
- `wait()`: Mientras el valor sea menor o igual a 0 se bloquea el proceso esperando un `signal`. Luego decrementa el valor de `sem`.
- `signal(int n = 1)`: Incrementa en uno el valor del semáforo y despierta a *alguno* de los procesos que están esperando en ese semáforo.

Ejercicio

Se tienen 2 procesos A y B.
El proceso A tiene que ejecutar A1() y luego A2().
B debe ejecutar B1() y después B2().
En cualquier ejecución A1() tiene que ejecutarse antes que B2().
Construya el código con semáforos de manera tal que cualquier ejecución cumpla lo pedido.

Ejercicio

Se tienen 2 procesos A y B.
El proceso A tiene que ejecutar A1() y luego A2().
B debe ejecutar B1() y después B2().
En cualquier ejecución A1() tiene que ejecutarse antes que B2().
Construya el código con semáforos de manera tal que cualquier ejecución cumpla lo pedido.

Solución

semaforo = sem(0)

A	B
A1()	B1()
semaforo.signal	semaforo.wait
A2()	B2()

Exclusión Mutua (Mutex)

Ahora supongamos un escario similar, donde A ejecuta A1(), Crit() y A2() y B ejecuta B1(), Crit() y B2(). Donde Crit() es un subproceso crítico en el cual me quiero asegurar de que no hay mas de un proceso ejecutandolo en ningún momento.

Exclusión Mutua (Mutex)

Ahora supongamos un escario similar, donde A ejecuta A1(), Crit() y A2() y B ejecuta B1(), Crit() y B2(). Donde Crit() es un subproceso crítico en el cual me quiero asegurar de que no hay mas de un proceso ejecutandolo en ningún momento.

Solución

mutex = sem(1)

A	B
A1()	B1()
mutex.wait	mutex.wait
Crit()	Crit()
mutex.signal	mutex.signal
A2()	B2()

Exclusión Mutua (Mutex)

Ahora supongamos un escario similar, donde A ejecuta A1(), Crit() y A2() y B ejecuta B1(), Crit() y B2(). Donde Crit() es un subproceso crítico en el cual me quiero asegurar de que no hay mas de un proceso ejecutandolo en ningún momento.

Solución

mutex = sem(1)

A	B
A1()	B1()
mutex.wait	mutex.wait
Crit()	Crit()
mutex.signal	mutex.signal
A2()	B2()

Solución?

Vale la pena usar un semáforo si en la sección crítica algo una tarea sencilla (aunque crítica) como cambiar el valor de una variable?

Tipos de datos atómicos

Son tipos de datos que tienen operaciones atómicas (a nivel hardware) y se utilizan en modo usuario.

- Booleanos Atómicos
  Declaración:
  atomic< *bool* > miBooleano
  Operaciones:
  getAndSet(bool b)
  testAndSet(bool b)
- Enteros Atómicos
  Declaración:
  atomic< *int* > miEntero
  Operaciones:
  getAndInc()/getAndDec()
  getAndAdd(int v)

- Objetos Atómicos
  Declaración:
  atomic$< T >$ miObjeto
  Operaciones:
  get()
  set()
  compareAndSwap(T test, T value)
- Spinlocks
  Declaración:
  spinlock miMutex
  Operaciones:
  lock()
  unlock()

Ejercicio

Usando los procesos A y B de los ejemplos anteriores, pero quiero que A1 y B1 ejecute antes de B2 y A2.
Sirve esta Solución?

Ejercicio

Usando los procesos A y B de los ejemplos anteriores, pero quiero que A1 y B1 ejecute antes de B2 y A2.
Sirve esta Solución?

Solución:

ejecutoA = sem(0)
ejecutoB = sem(0)

A	B
A1()	B1()
ejecutoB.wait	ejecutoA.wait
ejecutoA.signal	ejecutoB.signal
A2()	B2()

Ejercicio

Usando los procesos A y B de los ejemplos anteriores, pero quiero que A1 y B1 ejecute antes de B2 y A2.
Sirve esta Solución?

Solución:

ejecutoA = sem(0)
ejecutoB = sem(0)

A	B
A1()	B1()
ejecutoB.wait	ejecutoA.wait
ejecutoA.signal	ejecutoB.signal
A2()	B2()

Pista: No