Sincronización entre procesos

Sistemas Operativos

Primer Cuatrimestre 2023

Repaso: Qué es una **race condition**?

Defecto de un programa concurrente por el cual su correctitud depende del orden de ejecución de ciertos eventos.

Ejercicio

¿Cuál es la salida esperada de los procesos **A** y **B** corriendo concurrentemente y asumiendo que la variable **x** reside en memoria compartida?

$x = 0$

A

```
x = x + 1;

print(x);
```

B

```
x = x + 1;

print(x);
```

Ojo con la atomicidad

La operación sobre memoria compartida involucra:

1. Leer valor de x
2. Operar (sumarle 1 a este valor)
3. Escribir nuevo valor en x

Ejemplo de ejecución:

- Proceso 1: lee x, el valor es 0
- Proceso 1: suma 1 a x, el valor es ahora 1
- Proceso 2: lee x, el valor es 0
- Proceso 1: almacena 1 en x
- Proceso 2: suma 1 a x, el valor es ahora 1
- Proceso 2: almacena 1 en x

Pregunta: ¿Cómo podemos solucionar el problema de race conditions en este caso?

¿Qué es una variable atómica?

- Es un objeto que nos permite realizar operaciones de escritura y lectura de forma atomica.
- Tiene un valor entero, se puede interactuar con la variable mediante algunas primitivas como getAndInc() y getAndAdd().
- Estas primitivas son atómicas a efectos de los procesos.

¿Cómo lo uso?

Primitivas

- `getAndInc()`: Devuelve el entero atomico sumado uno.
- `getAndAdd(unsigned int value)`: Devuelve el entero atomico sumado la cantidad especificada.
- `set(unsigned int value)`: Asigna un valor pasado por parametro el objeto.

¿Cómo solucionamos la pérdida de sumas del ejemplo?

Esquema de la solución

atomic<int> x

x.set(0)

A	B
x.getAndInc()	x.getAndInc()

¿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 entero, 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.

¿Cómo lo uso?

Primitivas

- `sem(unsigned 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()`: Incrementa en uno el valor del semáforo y despierta a **alguno** de los procesos que están esperando en ese semáforo.

Importante antes de arrancar

- Puedo saber cual es el proceso que se despierta en un signal?

  No! Es determinístico pero depende de demasiadas variables.
- Puedo asumir que el proceso que se despierta es el próximo en correr?

  No! Es determinístico pero depende de demasiadas variables.
- Puedo consultar el valor de un semáforo?

  No! Revisar la interfaz, no hay observador.

¿Cómo solucionamos la pérdida de sumas del ejemplo?

Esquema de la solución

mutex = sem(1)

A	B
NoCrit()	NoCrit()
mutex.wait()	mutex.wait()
Crit()	Crit()
mutex.signal()	mutex.signal()
NoCrit()	NoCrit()

Nota: Se puede generalizar este esquema para resolver el problema de la sección crítica.

Ejercicio 1

Enunciado

Se tienen un proceso productor P que hace producir() y dos consumidores C_1, C_2 que hacen consumir1() y consumir2() respectivamente.
Se desea sincronizarlos de manera tal que las secuencias de ejecución sean: producir, producir, consumir1, consumir2, producir, producir, consumir1, consumir2,...

Solución:

permisoC2 = sem(0)
permisoC1 = sem(0)
permisoP = sem(1)

P	C1	C2
permisoP.wait()	permisoC1.wait()	permisoC2.wait()
producir()	consumir1()	consumir2()
producir()	permisoC2.signal()	permisoP.signal()
permisoC1.signal()		

Ejercicio 2

Enunciado

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.

Esquema de la solución

permisoB = sem(0)

A	B
...	...
A1()	B1()
...	...
A2()	B2()
...	...

Ejercicio 3

Enunciado

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

permisoB = sem(0)

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

Ejercicio 3

Enunciado

Usando los procesos A y B de los ejemplos anteriores, ahora quiero que A1 y B1 ejecuten antes de B2 y A2.

Solución?:

permisoB = sem(0)

permisoA = sem(0)

A	B
A1()	B1()
permisoA.wait()	permisoB.wait()
permisoB.signal()	permisoA.signal()
A2()	B2()

Pista: No

Ejercicio 3

Enunciado

Usando los procesos A y B de los ejemplos anteriores, pero quiero que A1 y B1 ejecuten antes de B2 y A2.

Solución:

permisoB = sem(0)

permisoA = sem(0)

A	B
A1()	B1()
permisoB.signal()	permisoA.signal()
permisoA.wait()	permisoB.wait()
A2()	B2()

Nota: Este patrón se suele llamar rendezvous (punto de encuentro) o barrera

Ejercicio 4

Enunciado

Un grupo de N estudiantes se dispone a hacer un tp de su materia favorita. Para ello acordaron dividirse el trabajo y cada quien conoce a la perfección como implementarTp() y luego como experimentar(). Curiosamente, cada etapa puede ser llevada acabo de manera independiente por cada estudiante.
Sin embargo, acordaron que para que alguien pudiera experimentar() todos deberían haber terminado de implementarTp().
Se pide diseñar un programa concurrente que utilice procesos y que modele esta situación utilizando semáforos.

Solución con turnstile:

barrera = sem(0)
mutex = sem(1)
int counter = 0

Estudiantes

---

implementarTp()

mutex.wait()
counter++
mutex.signal()

if (counter == n): barrera.signal()

barrera.wait()
barrera.signal()

experimentar()

Solución con multiple signals:

Definición

barrera.signal(unsigned int n):
    for($i = 0$; $i < n$; i++):
        barrera.signal()

Solución con multiple signals:

barrera = sem(0)
mutex = sem(1)
int counter = 0

Estudiantes

---

implementarTp()

mutex.wait()
counter++
mutex.signal()

if (counter == n): barrera.signal(n)

barrera.wait()

experimentar()

Nota: Este patrón se suele llamar barrera

Ejercicio 4 bis

Enunciado

Supongamos que el grupo se da cuenta que es una pésima idea diagramar el esquema de todo un tp de esta forma tan rígida. Propone entonces un esquema iterativo con etapas ahora más cortas de implementacion y experimentación. ¿Podemos reutilizar nuestra solución?

Solución

Coming soon...