Prueba de oposición - Algoritmos 2016

Brian Bokser

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

23 de octubre de 2016

Contenidos

Introducción

- Materia : *Sistemas Operativos*
- Práctica : *Tercera práctica, Sincronización*

Contexto

Los alumnos ya tuvieron las dos teóricas de Sincronización

Conocen las ideas clásicas, ahora tienen que ejercitarlas.

Enunciado

Suponga que se tienen N procesos P_i, cada uno de los cuales ejecuta un conjunto de sentencias a_i y b_i . Se los quiere sincronizar de manera tal que los b_i se ejecuten ***después de que se hayan ejecutado todos los*** a_i.

Enunciado

Suponga que se tienen N procesos P_i, cada uno de los cuales ejecuta un conjunto de sentencias a_i y b_i . Se los quiere sincronizar de manera tal que los b_i se ejecuten ***después de que se hayan ejecutado todos los*** *a_i.*

function P(i) ▷ El proceso i, sin sincronización

a(i)

b(i)

end function

Motivación

- Es conocido como Barrera o Rendezvous, y es fundamental.
- Aparece como subproblema de otros ejercicios.
- Es un buen ejemplo para entender los conceptos de sincronización.

Aproximándonos

Primer idea:

Aproximándonos

Primer idea: Turnos!

Aproximándonos

Primer idea: Turnos!

a_1 hasta a_n, luego b_1 hasta b_n. Aplicación directa del “problema de los turnos”

Aproximándonos

Primer idea: Turnos!
a_1 hasta a_n, luego b_1 hasta b_n. Aplicación directa del “problema de los turnos”

function P(i) ▷ Idea de la implementación
    sem_a[i].wait()

    a(i)

    sem_a[i+1].signal()

    sem_b[i].wait()

    b(i)

    sem_b[i+1].signal()
end function

Aproximándonos

Primer idea: Turnos!
a_1 hasta a_n, luego b_1 hasta b_n. Aplicación directa del “problema de los turnos”

function P(i) ▷ Idea de la implementación
    sem_a[i].wait()

    a(i)

    sem_a[i+1].signal()

    sem_b[i].wait()

    b(i)

    sem_b[i+1].signal()
end function

Problemas

Problema: Queremos la mayor **Concurrencia** posible. No queremos garantizar un orden de los a_i ni de los b_i

Problemas

Problema: Queremos la mayor **Concurrencia** posible. No queremos garantizar un orden de los a_i ni de los b_i

Segunda idea: Contar cuantos ejecutaron a(i)

¿Qué vamos a necesitar?

1. La cantidad de procesos N
2. contador = 0
3. mutex_contador = Mutex(1)
4. sem_barrera = Semaforo(0)

Implementando

```
function P(i)
    a(i)

    sem_barrera.wait()

    b(i)
end function
```

Implementando

```
function P(i)
    a(i)

    mutex_contador .wait()
        contador++
        if contador == N then

        end if
    mutex_contador .signal()

    sem_barrera.wait()

    b(i)
end function
```

Implementando

```
function P(i)
    a(i)

    mutex_contador .wait()
        contador++
        if contador == N then
            sem_barrera.signal()
        end if
    mutex_contador .signal()

    sem_barrera.wait()

    b(i)
end function
```

Implementando

```
function P(i)
    a(i)

    mutex_contador .wait()
        contador++
        if contador == N then
            sem_barrera.signal()
        end if
    mutex_contador .signal()

    sem_barrera.wait()

    b(i)
end function
```

Problema: Muchos procesos esperando, pero envíamos un solo signal

Terminando la solución

```
function P(i)
    a(i)

    mutex_contador .wait()
        contador++
        if contador == N then



        end if
    mutex_contador .signal()

    sem_barrera.wait()

    b(i)
end function
```

Terminando la solución

```
function P(i)
    a(i)

    mutex_contador .wait()
        contador++
        if contador == N then
            for i = 1 to N do
                sem_barrera.signal()
            end for
        end if
    mutex_contador .signal()

    sem_barrera.wait()

    b(i)
end function
```

¿Preguntas?

¡Muchas gracias!