

Prueba de oposición - Sistemas

Brian Bokser

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

19 de septiembre de 2017

- Materia : *Sistemas Operativos*
- Práctica : *Sincronización*

- 1 Los alumnos ya tuvieron las dos teóricas de Sincronización.
- 2 Manejan las primitivas de Semáforos, Mutex y variables atómicas.
- 3 Vieron algunos problemas y patrones básicos.

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)

$a(i)$

$b(i)$

end function

▷ El proceso i , sin sincronización

- Me parece un problema interesante porque:
 - 1 Es un patrón clásico, conocido como **Barrera**.
 - 2 Aparece como subproblema de otros ejercicios.
- En el problema vamos a trabajar:
 - 1 Mutexes.
 - 2 Semáforos.
 - 3 Maximizar el grado de concurrencia.

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)
    sem_a[i].wait()
    a(i)
    if i < n then sem_a[i+1].signal()
    else
        sem_b[1].signal()
    end if
    sem_b[i].wait()
    b(i)
    sem_b[i+1].signal()
end function
```

▷ Idea de la implementación

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)
    sem_a[i].wait()
    a(i)
    if i < n then sem_a[i+1].signal()
    else
        sem_b[1].signal()
    end if
    sem_b[i].wait()
    b(i)
    sem_b[i+1].signal()
end function
```

▷ Idea de la implementación

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

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
```

```
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!