Algoritmo de Dekker

Alumnos:
Ferrusca Ortíz Jorge Luis
Hernández González Ricardo Omar

Definición:

¿Qué es?

El algoritmo de Dekker es un algoritmo de programación concurrente para exclusión mutua, que permite a dos procesos o hilos de ejecución compartir un recurso sin conflictos. Fue uno de los primeros algoritmos de exclusión mutua inventados, implementado por Edsger Dijkstra.

Definición:

¿En qué consiste?

Si ambos procesos intentan acceder a la sección crítica simultáneamente, el algoritmo elige un proceso según una variable de turno. Si el otro proceso está ejecutando en su sección crítica, deberá esperar su finalización.

¿Por qué utilizarlo?

TSL (Test and set lock)

Instrucción máquina que comprueba y modifica en contenido de una palabra de la memoria en un registro. Al tratarse de una instrucción máquina, el procesador nos garantiza que la instrucción TSL es realizada sin ningún tipo de interrupción por parte de otro hilo

Definición:

Solución:

Existen cinco versiones del algoritmo Dekker, teniendo ciertos fallos los primeros cuatro. La versión 5 es la que trabaja más eficientemente, siendo una combinación de la 1 y la 4.

- Versión 1: *Alternancia estricta*. Garantiza la exclusión mutua, pero su desventaja es que acopla los procesos fuertemente, esto significa que los procesos lentos atrasan a los procesos rápidos.
- Versión 2: *Problema interbloqueo*. No existe la alternancia, aunque ambos procesos caen a un mismo estado y nunca salen de ahí.

Definición:

Solución:

- Versión 3: *Colisión región crítica no garantiza la exclusión mutua*. Este algoritmo no evita que dos procesos puedan acceder al mismo tiempo a la región crítica.
- Versión 4: *Postergación indefinida*. Aunque los procesos no están en interbloqueo, un proceso o varios se quedan esperando a que suceda un evento que tal vez nunca suceda.

Versión 1

Alternancia Estricta.

Es llamado de esta manera ya que obliga a que cada proceso tenga un turno, o sea que hay un cambio de turno cada vez que un proceso sale de la sección crítica, por lo tanto, si un proceso es lento atrasará a otros procesos que son rápidos.

Versión 1

Alternancia Estricta.

Características:

- Garantiza la exclusión mutua.
- Su sincronización es forzada.

Versión 1

Alternancia Estricta.

- Acopla fuertemente a los procesos (procesos lentos atrasan a procesos rápidos).
- No garantiza la progresión, ya que si un proceso por alguna razón es bloqueado dentro o fuera de la sección puede bloquear a los otros procesos.

```
1  int turno_proceso; //1 proceso 1, 2 proceso 2
2  Proceso1()
3  {
4   while( true )
5   {
6    [REALIZA_TAREAS_INICIALES]
7    while( turno_proceso == 2 ){}
8    [SECCION_CRITICA]
9    turno_proceso = 2;
10   [REALIZA_TAREAS_FINALES]
11  }
12 }
13
```

```
14 Proceso2()
15 {
16  while( true )
17  {
18   [REALIZA_TAREAS_INICIALES]
19   while( turno_proceso == 1 ){}
20   [SECCION_CRITICA]
21   turno_proceso = 1;
22   [REALIZA_TAREAS_FINALES]
23  }
24 }
```

```
26 iniciar(){
27  turno_proceso = 1;
28  Proceso1();
29  Proceso2();
30 }
```

Versión 1

Alternancia Estricta.

Descripción del algoritmo:

- Cuando un proceso es ejecutado verifica si es su turno, si no es su turno se queda en espera por medio de un ciclo while.***(linea 7 y 19)***
- De lo contrario si es su turno avanza a la sección crítica.
- Cuando el proceso sale de la sección crítica cambia de turno.***(línea 9 y 21)***

Versión 2

Problema de Interbloqueo.

Su nombre se debe a que si en cada ráfaga de CPU, cada proceso queda en el mismo estado, en el estado donde se le asigna que puede entrar a la sección crítica **(línea 8 para el proceso 1 y línea 21 para el proceso 2)**. Entonces estando los dos procesos con opción a entrar, a la siguiente ráfaga de CPU ambos procesos verificarán si el proceso alterno puede entrar **(línea 9 y 22)**, viendo que el proceso alterno tiene la opción de entrar, los procesos quedan bloqueados ya que se quedaran ciclados bloqueándose mutuamente ya que no podrán entrar nunca a la sección crítica.

Versión 2

Problema de Interbloqueo.

Características:

- Garantiza la exclusión mutua.
- No garantiza espera limitada.

```
1  boolean p1_puede_entrar, p2_puede_entrar;
2
3  Proceso1()
4  {
5   while( true )
6   {
7    [REALIZA_TAREAS_INICIALES]
8    p1_puede_entrar = true;
9    while( p2_puede_entrar ){}
10   [SECCION_CRITICA]
11   p1_puede_entrar = false;
12   [REALIZA_TAREAS_FINALES]
13  }
14 }
```

```
16 Proceso2()
17 {
18  while( true )
19  {
20   [REALIZA_TAREAS_INICIALES]
21   p2_puede_entrar = true;
22   while( p1_puede_entrar ){}
23   [SECCION_CRITICA]
24   p2_puede_entrar = false;
25   [REALIZA_TAREAS_FINALES]
26  }
27 }
```

```
29 iniciar()
30 {
31  p1_puede_entrar = false;
32  p2_puede_entrar = false;
33  Proceso1();
34  Proceso2();
35 }
```

Versión 2

Problema de Interbloqueo.

Descripción del algoritmo:

- El proceso que es ejecutado después de realizar sus tareas iniciales, a este proceso se le permite entrar ***(línea 8 y 21)***.

Versión 2

Problema de Interbloqueo.

- Cuando ya puede entrar verifica si otro proceso tiene la opción de poder entrar, si otro proceso también tiene la opción de poder entrar se da un interbloqueo. De lo contrario el proceso avanza a la sección crítica ***(línea 9 y 22).***
- Al salir de la sección crítica el proceso cambia su opción ***(línea 11 y 24)***. Y permite al otro proceso avanzar a la sección crítica.

Versión 3

Colisión región crítica, no garantiza la exclusión mutua.

Se da una colisión en la región crítica por la forma en que son colocados, por así decirlo, ***los permisos***, ya que primero se comprueba si otro proceso está dentro y luego se indica que el proceso en el que se está actualmente cambia diciendo que está dentro. El problema se da cuando los procesos después de haber tenido sus ráfagas de CPU pasan de la fase de comprobación **(línea 8 y 21)** y se tiene libre el camino para entrar a la región crítica, generando esto una colisión.

Versión 3

Colisión región crítica, no garantiza la exclusión mutua.

Características:

- No garantiza la exclusión mutua.
- Colisión en la región crítica.

```
1  boolean p1_esta_dentro, p2_esta_dentro;
2
3  Proceso1()
4  {
5   while( true )
6   {
7    [REALIZA_TAREAS_INICIALES]
8    while( p2_esta_dentro ){}
9    p1_esta_dentro = true;
10   [SECCIÓN_CRITICA]
11   p1_esta_dentro = false;
12   [REALIZA_TAREAS_FINALES]
13  }
14 }
```

```
16 Proceso2()
17 {
18  while( true )
19  {
20   [REALIZA_TAREAS_INICIALES]
21   while( p1_esta_dentro ){}
22   p2_esta_dentro = true;
23   [SECCIÓN_CRITICA]
24   p2_esta_dentro = false;
25   [REALIZA_TAREAS_FINALES]
26  }
27 }
```

```
29 iniciar()
30 {
31  p1_esta_dentro = false;
32  p2_esta_dentro = false;
33  Proceso1();
34  Proceso2();
35 }
```

Versión 3

Colisión región crítica, no garantiza la exclusión mutua.

Descripción del algoritmo:

- Al ejecutarse el proceso y después de realizar sus tareas iniciales, verifica si otro proceso está dentro de la sección crítica ***(línea 8 y 21)***.

Versión 3

Colisión región crítica, no garantiza la exclusión mutua.

- Si el otro proceso está dentro entonces espera a que salga de la sección crítica. De lo contrario pasa la fase de comprobación y cambia su estado a que está dentro ***(línea 9 y 22)***.
- Luego de pasar la sección crítica cambia su estado ***(línea 11 y 24)***, termina sus tareas finales.

Versión 4

Postergación indefinida.

Su nombre se debe a que en una parte del codigo **(línea 12 y 30)** es colocado un retardo con un tiempo aleatorio, y el retardo puede ser muy grande que no se sabe hasta cuando entrará a la sección crítica.

Versión 4

Postergación indefinida.

Características:

- Garantiza la exclusión mutua.
- Un proceso o varios se quedan esperando a que suceda un evento que tal vez nunca suceda.

```
1  boolean p1_puede_entrar, p2_puede_entrar;
2
3  Proceso1()
4  {
5   while( true )
6   {
7    [REALIZA_TAREAS_INICIALES]
8    p1_puede_entrar = true;
9    while( p2_puede_entrar )
10   {
11    p1_puede_entrar = false;
12    retardo( tiempo_x ); //tiempo_x es un ti
13    p1_puede_entrar = true;
14   }
15   [SECCION_CRITICA]
16   p1_puede_entrar = false;
17   [REALIZA_TAREAS_FINALES]
18  }
19 }
20
```

```
21 Proceso2()
22 {
23  while( true )
24  {
25   [REALIZA_TAREAS_INICIALES]
26   p2_puede_entrar = true;
27   while( p1_puede_entrar )
28   {
29    p2_puede_entrar = false;
30    retardo( tiempo_x ); //tiempo_x es
31    p2_puede_entrar = true;
32   }
33   [SECCION_CRITICA]
34   p2_puede_entrar = false;
35   [REALIZA_TAREAS_FINALES]
36  }
37 }
38
```

```
39 iniciar()
40 {
41  p1_puede_entrar = false;
42  p2_puede_entrar = false;
43  Proceso1();
44  Proceso2();
45 }
```

Versión 4

Postergación indefinida.

Descripción del algoritmo:

- Luego de realizar sus tareas iniciales el procesos solicita poder entrar en la sección crítica, si el otro proceso no puede entrar ***(línea 9 y 27)*** ya que su estado es falso entonces el proceso entra sin problema a la sección crítica.

Versión 4

Postergación indefinida.

- De lo contrario si el otro proceso también puede entrar entonces se entra al ciclo donde el proceso actual se niega el paso así mismo y con un retardo de x tiempo siendo este aleatorio ***(línea 12 y 30)*** se pausa el proceso, para darle vía libre a los otros procesos.

Versión 4

Postergación indefinida.

- Luego de terminar su pausa entonces el proceso actual nuevamente puede entrar y nuevamente si el otro proceso puede entrar se repite el ciclo y si no hay otro proceso, entonces el proceso puede entrar en la sección crítica.
- Cambia su estado ***(línea 16 y 34)*** y luego realiza sus tareas finales.

Versión 5

Algoritmo Óptimo.

Este algoritmo es una combinación del algoritmo 1 y 4.

Características

- Garantiza la exclusión mutua.
- Progreso
- Espera limitada

```
1  boolean p1_puede_entrar, p2_puede_entrar;
2  int turno;
3
4  Proceso1()
5  {
6   while( true )
7   {
8    [REALIZA_TAREAS_INICIALES]
9    p1_puede_entrar = true;
10   while( p2_puede_entrar )
11   {
12    if( turno == 2 )
13    {
14     p1_puede_entrar = false;
15     while( turno == 2 ){}
16     p1_puede_entrar = true;
17    }
18   }
19   [REGION_CRITICA]
20   turno = 2;
21   p1_puede_entrar = false;
22   [REALIZA_TAREAS_FINALES]
23  }
24 }
```

```
26 Proceso2()
27 {
28  while( true )
29  {
30   [REALIZA_TAREAS_INICIALES]
31   p2_puede_entrar = true;
32   while( p1_puede_entrar )
33   {
34    if( turno == 1 )
35    {
36     p2_puede_entrar = false;
37     while( turno == 1 ){}
38     p2_puede_entrar = true;
39    }
40   }
41   [REGION_CRITICA]
42   turno = 1;
43   p2_puede_entrar = false;
44   [REALIZA_TAREAS_FINALES]
45  }
46 }
```

```
48 iniciar()
49 {
50  p1_puede_entrar = false;
51  p2_puede_entrar = false;
52  turno = 1;
53  Proceso1();
54  Proceso2();
55 }
```

Versión 5

Algoritmo Óptimo.

Descripción del algoritmo:

- Se realiza las tareas iniciales, luego se verifica si hay otro procesos que puede entrar, si lo hay se entra al ciclo y si es el turno de algún otro proceso ***(línea 12 y 34)*** cambia su estado a ya no poder entrar a la sección crítica y nuevamente verifica si es el turno de algún otro proceso ***(línea 15 y 37).***

Versión 5

Algoritmo Óptimo.

Si lo es, se queda ciclado hasta que se da un cambio de turno, luego nuevamente retoma su estado de poder entrar a la sección crítica, regresa al ciclo y verifica si hay otro proceso que puede entrar entonces nuevamente se encicla, de lo contrario entra a la sección crítica.

Versión 5

Algoritmo Óptimo.

- Al salir de la sección crítica el proceso cambia su turno, cambia su estado y realiza sus tareas finales. ***(línea 20 y 42)***

Bibliografía:

- http://aprendiendo-software.blogspot.mx/2011/12/algoritmo-de-dekker-version-5.html
- https://es.slideshare.net/mastermind87/algoritmos-de-dekker
- Silva, M, (2015). *Sistemas Operativos.* 1a Edición. Grupo Editor Argentino. Ciudad Autónoma de Buenos Aires