Algoritmos de Planificación del Procesador

Victor Tortolero, 24.569.609

Sistemas Operativos, FACYT

12 de abril de 2016

Primera version del algoritmo de Dekker

- Primer algoritmo en resolver la exclusión mutua.
- Aplica la exclusión mutua de manera correcta, y la garantiza.
- Usa variables para controlar que hilo se ejecutara.
- Revisa constantemente si la sección critica esta disponible (spinlock, busy waiting), lo que malgasta el tiempo del procesador.
- Los procesos lentos atrasan a los rapidos.

Ejemplo de la Primera version del algoritmo de Dekker

```
1  int numero_de_proceso = 1;
2  iniciar_procesos(); // inicializa y corre ambos procesos
3
4  proceso p1:
5  void main(){
6          while(!no_terminado){
7                  while(numero_de_proceso == 2); // seccion de entrada de la exclusion mutua
8                  // seccion critica
9                  numero_de_proceso = 2; // seccion de salida de la exclusion mutua
10                 // codigo restante
11         }
12 } // fin del proceso p1
13
14 proceso p2:
15 void main(){
16         while(!no_terminado){
17                 while(numero_de_proceso == 1); // seccion de entrada de la exclusion mutua
18                 // seccion critica
19                 numero_de_proceso = 1; // seccion de salida de la exclusion mutua
20                 // codigo restante
21         }
22 } // fin del proceso p2
```

Segunda version del algoritmo de Dekker

- Primer algoritmo en resolver la exclusión mutua.
- Aplica la exclusión mutua de manera correcta.
- Usa variables para controlar que hilo se ejecutara.
- Revisa constantemente si la sección critica esta disponible (spinlock, busy waiting), lo que malgasta el tiempo del procesador.

Ejemplo de la Segunda version del algoritmo de Dekker

```
1   bool dentro_de_p1 = false;
2   bool dentro_de_p2 = false;
3   iniciar_procesos(); // inicializa y corre ambos procesos
4
5   proceso p1:
6   void main(){
7           while(!no_terminado){
8                   while(dentro_de_p2); // seccion de entrada de la exclusion mutua
9                   dentro_de_p1 = true; // seccion de entrada
10                  // seccion critica
11                  dentro_de_p2 = false; // seccion de salida de la exclusion mutua
12                  // codigo restante
13          }
14  } // fin del proceso p1
15
16  proceso p2:
17  void main(){
18          while(!no_terminado){
19                  while(dentro_de_p1); // seccion de entrada de la exclusion mutua
20                  dentro_de_p2 = true; // seccion de entrada
21                  // seccion critica
22                  dentro_de_p1 = false; // seccion de salida de la exclusion mutua
23                  // codigo restante
24          }
25  } // fin del proceso p2
```

Tercera version del algoritmo de Dekker

- Garantiza la exclusión mutua.
- Es posible un deadlock (Ambos procesos encienden sus banderas simultáneamente, y ninguno saldría del ciclo).

Ejemplo de la Tercera version del algoritmo de Dekker

```
1   bool p1_quiere_entrar = false;
2   bool p2_quiere_entrar = false;
3   iniciar_procesos(); // inicializa y corre ambos procesos
4
5   proceso p1:
6   void main(){
7           while(!no_terminado){
8                   p1_quiere_entrar = true; // seccion de entrada de la exclusion mutua
9                   while(p2_quiere_entrar); // seccion de entrada
10                  // seccion critica
11                  p1_quiere_entrar = false; // seccion de salida de la exclusion mutua
12                  // codigo restante
13          }
14  } // fin del proceso p1
15
16  proceso p2:
17  void main(){
18          while(!no_terminado){
19                  p2_quiere_entrar = true; // seccion de entrada de la exclusion mutua
20                  while(p1_quiere_entrar); // seccion de entrada
21                  // seccion critica
22                  p2_quiere_entrar = false; // seccion de salida de la exclusion mutua
23                  // codigo restante
24          }
25  } // fin del proceso p2
```

Cuarta version del algoritmo de Dekker

- Es posible posponer un proceso de manera indefinida.
- Apaga las banderas por cortos periodos de tiempo para tomar control.

Ejemplo de la Cuarta version del algoritmo de Dekker

```
1   bool p1_quiere_entrar = false;
2   bool p2_quiere_entrar = false;
3   iniciar_procesos(); // inicializa y corre ambos procesos
4
5   proceso p1:
6   void main(){
7           while(!no_terminado){
8                   p1_quiere_entrar = true; // seccion de entrada de la exclusion mutua
9                   while(p2_quiere_entrar){ // seccion de entrada
10                          p1_quiere_entrar = false; // seccion de entrada
11                          // esperar por una cantidad de tiempo pequeña aleatoria
12                          p1_quiere_entrar = true;
13                  }
14                  // seccion critica
15                  p1_quiere_entrar = false; // seccion de entrada
16                  // codigo restante
17          }
18  } // fin del proceso p1
19
20  proceso p2:
21  void main(){
22          while(!no_terminado){
23                  p2_quiere_entrar = true; // seccion de entrada de la exclusion mutua
24                  while(p1_quiere_entrar){ // seccion de entrada
25                          p2_quiere_entrar = false; // seccion de entrada
26                          // esperar por una cantidad de tiempo pequeña aleatoria
27                          p2_quiere_entrar = true;
28                  }
29                  // seccion critica
```

Quinta version del algoritmo de Dekker

- Marca procesos como preferidos para determinar el uso de las secciones criticas.
- El estatus de "Preferido" se turna entre los procesos.
- Garantiza la exclusión mutua.
- Evita deadlock's, y el posponer un proceso de manera indefinida.

Ejemplo de la Quinta version del algoritmo de Dekker I

```
1   int proceso_preferido = 1;
2   bool p1_quiere_entrar = false;
3   bool p2_quiere_entrar = false;
4   iniciar_procesos(); // inicializa y corre ambos procesos
5
6   proceso p1:
7   void main(){
8           while(!no_terminado){
9                   p1_quiere_entrar = true;
10                  while(p2_quiere_entrar){
11                          if(proceso_preferido == 2){
12                                  p1_quiere_entrar = false;
13                                  while(proceso_preferido == 2); // busy wait
14                                  p1_quiere_entrar = true;
15                          }
16                  }
17                  // seccion critica
18                  proceso_preferido = 2;
19                  p1_quiere_entrar = false; // seccion de entrada
20                  // codigo restante
21          }
22  } // fin del proceso p1
```

Ejemplo de la Quinta version del algoritmo de Dekker II

```
23
24  proceso p2:
25  void main(){
26          while(!no_terminado){
27                  p2_quiere_entrar = true;
28                  while(p1_quiere_entrar){
29                          if(proceso_preferido == 1){
30                                  p2_quiere_entrar = false;
31                                  while(proceso_preferido == 1); // busy wait
32                                  p2_quiere_entrar = true;
33                          }
34                  }
35                  // seccion critica
36                  proceso_preferido = 1;
37                  p2_quiere_entrar = false;
38                  // codigo restante
39          }
40  } // fin del proceso p2
```

Test and Set

- Operación Atómica.
- Retorna el valor del lock, y lo cambia a verdadero.
- Si el valor retornado es **falso**, obtenemos el lock. Si es **verdadero**, esta ocupado por otro proceso.

```
1  boolean test_and_set(boolean *target){
2          boolean rv = *target;
3          *target = true;
4          return rv;
5  }
```

Compare and Swap

- Operación Atómica.
- Retorna el valor del lock, y lo cambia a verdadero.
- Si el valor retornado es **falso**, obtenemos el lock. Si es **verdadero**, esta ocupado por otro proceso.

```
1  int compare_and_swap(int *value, int expected, int new_value){
2          int temp = *value;
3
4          if(*value == expected)
5                  *value = new_value;
6
7          return temp;
8  }
```

References I

Presentacion de la university of limerick, Disponible en `http://garryowen.csisdmz.ul.ie/~cs4023/resources/oth6.pdf`.