Algoritmo de Dekker

Lara Sala Kevin Arturo
Loidi Gutiérrez Javier
Sistemas Operativos
Grupo 5

Inicios de la concurrencia.

- Surgió desde finales del siglo XIX y principios del siglo XX.
- Vías ferroviarias y telégrafos.
- La aplicación en el cómputo: 1960’s.

Ejemplo de concurrencia

- Contexto de un banco.
  - ¿Manera más óptima de realizar un retiro de una cuenta?
- Solución
  - Sincronización de procesos.

Contexto histórico

- Surgió a finales del siglo XIX y principios del siglo XX
- Vías ferroviarias
  - ¿Cómo evitar colisiones entre trenes?
  - Uso de semáforos
- Telegrafía
  - ¿Cómo manejar múltiples transmisiones con un número limitado de cables?
  - Solución: Multiplexor.

Concurrencia en el cómputo

- Década de los 60’s.

- E.W.Dijkstra. 1965, Technological University, Eindhoven, Netherlands. Theodorus Jozef Dekker.

- Fundamento: ¿Cómo solucionar la problemática de limitar el acceso a una sola computadora a la vez, a un recurso compartido?
- Solution of a problem in Concurrent programming control.

Algoritmo de Dekker, planteamiento.

- Tomó tres años aproximadamente llegar a una primera solución.
- Se tienen N computadoras, cada una corriendo un proceso cíclico infinito, con una sección crítica.
- Se deben programar las computadoras, de manera que solamente una puede acceder a su sección crítica a la vez.
- Se “comunicarán” a través de un medio de almacenamiento compartido (memoria).
- Las operaciones atómicas en esta problemática, serán las de escritura y lectura.

Algoritmo de Dekker, condiciones.

- No hay prioridades entre computadoras.
- La velocidad de cada una de las computadoras será irrelevante.
- Si alguna computadora se atora o detiene fuera de la sección crítica., no será impedimento para que las demás continúen ejecutándose.
- Si hay bloqueo mutuo, no se considerará como solución.

Algoritmo de Dekker, estructura.

- El algoritmo tiene la siguiente estructura:

```
do{
    //Sección de entrada.
    //Sección crítica.
    //Sección de salida.
    //Código restante.
}while(true);
```

- Objetivo: Proveer el acceso controlado a dos procesos a un solo recurso.

Algoritmo de Dekker, primera versión.

```
main(){
   int thread_no = 1;
   startThreads();
}
Thread1(){
   do {
      // Sección de entrada
      while (thread_no == 2)
            //Espera mientras hilo 2 está en la S.C.


      // Sección crítica
      //Fin sección crítica
      thread_no = 2; //Le otorga el  pase al proceso 2.


   } while (completed == false)
}
```

```
Thread2(){
   do {
      //Sección de entrada
      while (thread_no == 1)
            //Espera mientras el hilo 1 está en la S.C.

      // Sección crítica
      // Fin sección crítica


      threadno = 1; //Le otorga el pase al hilo 1


   } while (completed == false)
}
```

Algoritmo de Dekker, primera versión.

- Ventajas
  - Otorga exclusión mutua.
- Desventajas
  - Bloqueo infinito.
  - Espera activa.

Algoritmo de Dekker, segunda versión.

```
main(){
   // Banderas que indican si están en la S.C
   boolean th1 = true;
   boolean th2 = false;
   startThreads();
}

Thread1(){
   do {
      //Sección de entrada
      while (th2 == true); //Espera hasta que Hilo 2 salga de la RC.

      th1 = true;//Indica que Hilo 1 acaba de entrar en la RC.
      // Sección crítica
      // Fin sección crítica

      th1 = false; //Hilo 1 sale de la S.C.

   } while (completed == false)
}
```

```
Thread2(){
   do {
      // Sección de entrada
      while (th1 == true); //Espera hasta que Hilo 1 salga de la RC.
      th2 = true; //Indica que Hilo 2 acaba de entrar en la RC.
      // Sección crítica
      // Fin sección crítica


      th2 = false; //Hilo 2 sale de la S.C.


   } while (completed == false)
}
```

Algoritmo de Dekker, segunda versión.

- Ventajas.
  - Corrige el bloqueo infinito
- Desventajas.
  - Ya no hay exclusión mutua.
  - Espera activa.

Algoritmo de Dekker, tercera versión.

```
main(){
   // Banderas que indican intención de entrar a la S.C
   boolean th1wantstoenter = false;
   boolean th2wantstoenter = false;
   startThreads();
}

Thread1(){
   do {
      th1wantstoenter = true;
      // Sección de entrada.

      while (th2wantstoenter == true)
        //Espera mientras hilo 2 quiera entrar a
    la S.C

      // Sección crítica
      // Fin sección crítica

      th1wantstoenter = false; //Hilo 1 salió de la S.C

   } while (completed == false)
}
```

```
Thread2(){
   do {
      th2wantstoenter = true;
      // Sección de entrada

      while (th1wantstoenter == true)
          //Espera mientras hilo 1 quiera entrar a
          la S.C

      // Sección crítica
      // Fin sección crítica

      th2wantstoenter = false; //Hilo 2 salo de la S.C.

   } while (completed == false)
}
```

Algoritmo de Dekker, tercera versión.

- Ventajas
  - Provee una nueva forma de resolver el problema
- Desventajas
  - No soluciona ninguna problemática de la versión anterior

Algoritmo de Dekker, cuarta versión.

```
main(){
  // Banderas que indican la intención de entrar a la S.C.
  boolean th1wantstoenter = false;
  boolean th2wantstoenter = false;
  startThreads();
}
Thread1(){
  do {
    th1wantstoenter = true;
    while (th2wantstoenter == true) {
    // Provee acceso al otro hilo por un tiempo aleatorio
   de tiempo

    th1wantstoenter = false;
    th1wantstoenter = true;
  }
  // Sección de entrada.

  // Sección crítica
  // Fin sección crítica

  th1wantstoenter = false; //Hilo 1 terminó su ejecución.

  } while (completed == false)
}
```

```
Thread2(){
  do {
    th2wantstoenter = true;
    while (th1wantstoenter == true) {
    // Provee acceso al otro hilo por un tiempo aleatorio de tiempo
    th2wantstoenter = false;
    th2wantstoenter = true;
  }
  // Sección de entrada

  // Sección crítica
  // Fin sección crítica

  th2wantstoenter = false; //Hilo 2 terminó su ejecución.

  } while (completed == false)
}
```

Algoritmo de Dekker, cuarta versión.

- Ventajas.
  - Soluciona los dos problemas anteriores: Provee exlcusión mutua y evita bloqueos infinitos
- Desventajas.
  - Surge un nuevo problema: tiempo de espera aleatorio.

Algoritmo de Dekker, quinta versión.

```
main(){
  int favouredthread = 1; //Indica el turno de los hilos.
  // Banderas que indican la intención de entrar a la S.C.
  boolean th1wantstoenter = false;
  boolean th2wantstoenter = false;
  startThreads();
}
Thread1(){
  do {
    thread1wantstoenter = true;
    // Sección de entrada

    while (th2wantstoenter == true) {
      if (favaouredthread == 2) {
      // Si el segundo hilo quiere entrar y es su turno
        // obtiene acceso a la S.C.

        th1wantstoenter = false; //Hilo 1 espera su turno.
        while (favouredthread == 2);
        th1wantstoenter = true;
      }
    }
    // Sección crítica para hilo 1

    Favouredthread = 2; //Ahora es el turno del hilo 2.
    // Fin sección crítica

    th1wantstoenter = false; //Indica que Hilo 1 ya terminó su S.C.

  } while (completed == false)
}
```

```
Thread2(){
  do {
    th2wantstoenter = true;
    // Sección de entrada
    while (th1wantstoenter == true) {
      if (favaouredthread == 1) {
        // Si el primer hilo quiere entrar y es su turno.
        // obtiene acceso a la S.C.
        th2wantstoenter = false;

        while (favouredthread == 1); //Hilo 2 espera su turno
        th2wantstoenter = true;
      }
    }
    // Sección crítica para hilo 1

    favouredthread = 1; //Ahora el turno es del hilo 1
    // Fin sección crítica

    th2wantstoenter = false; //Indica que hilo 2 ya terminó su S.C.

  } while (completed == false)
}
```

Ventajas

- Garantiza exclusión mutua.
- Garantiza la libertad de bloqueos mutuos.
- Garantiza la libertad de inanición.
- Por su simpleza, se considera portable.

Desventajas.

- Solo admite dos procesos a la vez.
- Hace uso de espera activa.
- No suspende a los procesos.
- Puede presentar algunos fallos: Compiladores o ciclos infinitos.

Conclusiones

- Es la implementación básica de un Mutex.
- No es lo mismo que un multiplex.
- ¿Factible u obsoleto?
  - Depende.
  - A criterio de cada usuario.