비동기식 병행 실행

6th Week

Kim, Eui-Jik

Contents

- 소개
- 상호 배제
- 상호 배제 프리미티브 구현
- 상호 배제 문제에 대한 소프트웨어 해결책
- 세마포어

소개

- 스레드와 프로세스
  - 스레드의 병행(Concurrent)
    - 한 시스템에서 동시에 여러 스레드 존재
    - 독립적 실행 또는 협력 실행
    - 비동기적 실행
      - 독립적으로 실행

상호 배제

- 상호 배제(Mutual exclusion)
  - 공유 변수에 대한 접근 제어 부재에 의한 오류 발생
    - 많은 병행 스레드에서 공유 데이터 접근, 데이터의 갱신 시 오류 발생
  - 각 스레드의 공유 변수에 대한 배타적인 접근(Exclusive access) 필요
  - 공유 변수에 대한 순차화(Serializing) 접근
    - 한 스레드가 공유 변수의 값을 증가 시키는 동안 다른 모든 스레드는 우선 대기
    - 실행 중인 스레드가 공유 변수의 작업을 마친 후 대기 중인 프로세스 중 하나가 공유 변수 접근
  - 어느 한 스레드가 공유 변수를 갱신하는 동안에는 다른 스레드들이 동시에 접근하는 일을 방지

상호 배제

- 자바 멀티 스레딩 사례 연구 : 생산자/소비자 관계
  - 생산자/소비자 관계
    - 여러 스레드(또는 프로세스)가 공통 작업 수행을 위해 서로 협동하고, 병행 처리되는 대표적인 예
    - 생산자
      - 데이터 생산, 공유 객체에 저장
    - 소비자
      - 공유 객체로부터 데이터 읽음

예제 5-1 생산자/소비자 예제에서 사용하는 Buffer 인터페이스

```
01  // [예제 5-1] Buffer.java
02  // Buffer 인터페이스는 버퍼 데이터에 접근하는 메소드들을 명시함
03
04  public interface Buffer
05  {
06      public void set(int value); // Buffer에 값을 넣음
07      public int get();           // Buffer에서 값을 반환함
08  }
```

예제 5-2 생산자/소비자 관계에서 생산자 스레드를 나타내는 Producer 클래스

```
01  // [예제 5-2] Producer.java
02  // Producer의 run 메소드는 생산자 스레드가
03  // 1~4 값을 Buffer 타입 sharedLocation에 넣도록 동작함
04
05  public class Producer extends Thread
06  {
07      private Buffer sharedLocation; // 공유 객체 참조
08
09      // Producer 생성자
10      public Producer(Buffer shared)
11      {
12          super("Producer");        // "Producer"라는 스레드를 생성함
13          sharedLocation = shared; // sharedLocation을 shared로 초기화함
14      } // Producer 생성자 끝
15
16      // Producer의 run 메소드
17      // 1~4 값을 sharedLocation 버퍼에 넣음
18      public void run()
19      {
20          for (int count = 1; count <= 4; count++)
21          {
22              // 0~3초 사이의 시간 동안 휴면한 후 버퍼에 값을 넣음
23              try
24              {
25                  Thread.sleep((int) (Math.random() * 3001));
26                  sharedLocation.set(count); // 버퍼에 쓰기
27              } // try 끝
28
29              // 휴면 스레드가 인터럽트되면 스택 트레이스를 출력함
30              catch (InterruptedException exception)
31              {
32                  exception.printStackTrace();
33              } // catch 끝
34
35          } // for 문끝
36
37          System.err.println(getName() + " done producing." +
38              "\nTerminating " + getName() + ".");
39
40      } // run 메소드 끝
41
42  } // Producer 클래스 끝
```

예제 5-3 생산자/소비자 관계에서 소비자 스레드를 나타내는 Consumer 클래스

```
01  // [예제 5-3] Consumer.java
02  // Consumer의 run 메소드는 스레드가 루프를 네 번 돌면서
03  // 매번 sharedLocation의 값을 읽어오게 함
04
05  public class Consumer extends Thread
06  {
07       private Buffer sharedLocation; // 공유 객체 참조
08
09       // Consumer 생성자
10       public Consumer(Buffer shared)
11       {
12            super("Consumer"); // "Consumer"라는 스레드를 생성함
13            sharedLocation = shared; // sharedLocation을 shared로 초기화함
14       } // Consumer 생성자 끝
15
16       // sharedLocation의 값을 네 번 읽고 sum에 합산함
17       public void run()
18       {
19            int sum = 0 ;
20
21            // 휴면 상태에서 Buffer 값을 읽어오는 것으로 바꿈
22            for (int count = 1; count <= 4; ++count)
23            {
24                 // 0~3초 휴면 후 Buffer 값을 읽고 sum에 합산함
25                 try
26                 {
27                      Thread.sleep((int) (Math.random() * 3001));
28                      sum += sharedLocation.get();
29                 }
30
31                 // 스레드가 휴면 시간이 만료되지 않은 채 인터럽트되면 스택 트레이스를 출력함
32                 catch (InterruptedException exception)
33                 {
34                      exception.printStackTrace();
35                 }
36            } // for 문 끝
37
38            System.err.println(getName() + " read values totaling: "
39                 + sum + ".\nTerminating " + getName() + ".");
40
41       } // run 메소드 끝
42
43  } // Consumer 클래스 끝
```

상호 배제

- [예제5-1]의 Buffer 인터페이스 구현

예제 5-4 생산자와 소비자 스레드가 set, get 메소드를 통해 공유하는 정수 값을 관리하는 UnsynchronizedBuffer 클래스

```
01  // [예제 5-4] UnsynchronizedBuffer.java
02  // UnsynchronizedBuffer는 공유되는 정수 값 하나를 나타냄
03
04  public class UnsynchronizedBuffer implements Buffer
05  {
06      private int buffer = -1; // 생산자, 소비자가 공유함
07
08      // buffer 값을 설정하는 루틴
09      public void set(int value)
10      {
11          System.err.println(Thread.currentThread().getName() +
12                  " writes " + value);
13
14          buffer = value;
15      } // set 메소드 끝
16
18      public int get()
19      {
20          System.err.println(Thread.currentThread().getName() +
21                  " reads " + buffer);
22
23          return buffer;
24      } // get 메소드 끝
25
26  } // UnsynchronizedBuffer 클래스 끝
```

공유변수 buffer 선언, '-1'로 초기화

파라미터를 buffer 값으로 대입

buffer 값 반환

상호 배제

예제 5-5 비동기식으로 스레드들이 공유 객체의 데이터를 수정하게 해주는 SharedBuffer 클래스

```
01  // [예제 5-5] SharedBufferTest.java
02  // SharedBufferTest는 생산자와 소비자 스레드를 생성함
03
04  public class SharedBufferTest
05  {
06      public static void main(String[] args)
07      {
08          // 스레드들이 사용할 공유 객체를 생성함
09          Buffer sharedLocation = new UnsynchronizedBuffer();
10
11          // 생산자와 소비자 객체를 생성함
12          Producer producer = new Producer(sharedLocation);
13          Consumer consumer = new Consumer(sharedLocation);
14
15          producer.start(); // producer 스레드 시작
16          consumer.start(); // consumer 스레드 시작
17
18      } // main 메소드 끝
19
20  } // SharedBufferTest 클래스 끝
```

결과 1

```
Consumer reads -1
Producer writes 1
Consumer reads 1
Consumer reads 1
Consumer reads 1
Consumer read values totaling: 2.
Terminating Consumer.
Producer writes 2
Producer writes 3
Producer writes 4
Producer done producing.
Terminating Producer.
```

결과 2

```
Producer writes 1
Producer writes 2
Consumer reads 2
Producer writes 3
Consumer reads 3
Producer writes 4
Producer done producing.
Terminating Producer.
Consumer reads 4
Consumer reads 4
Consumer read values totaling: 13.
Terminating Consumer.
```

결과 3

```
Producer writes 1
Consumer reads 1
Producer writes 2
Consumer reads 2
Producer writes 3
Consumer reads 3
Producer writes 4
Producer done producing.
Terminating Producer.
Consumer reads 4
Consumer read values totaling: 10.
Terminating Consumer.
```

∴스레드들이 "상호배제"를 통해 공유데이터에 동시에 접근하는 것을 조절해야 함!!

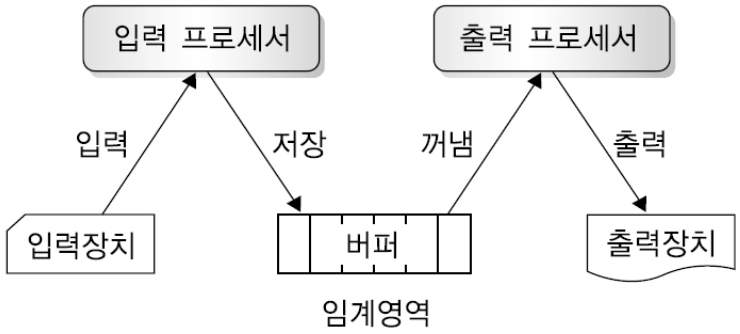
상호 배제

- 임계 영역 (Critical Section)
  - 스레드(또는 프로세스)가 수정 가능한 공유 데이터에 접근할 때 임계영역에 있다고 함
  - 한번에 한 스레드만 특정 자원에 접근 실행 가능
    - 임계영역에 있는 스레드는 수정 가능한 공유데이터에 대해 배타적 접근 권한을 가짐
    - 현재 접근을 시도하는 다른 모든 스레드는 대기

- 상호 배제 프리미티브
  - 상호 배제와 관련 가장 기본적인 연산 호출
  - 스레드의 임계영역에서 일어나는 일을 캡슐화
    - enterMutualExclusion() : 임계 영역에 들어가려고 할 때
    - exitMutualExclusion() : 임계 영역을 나올 때

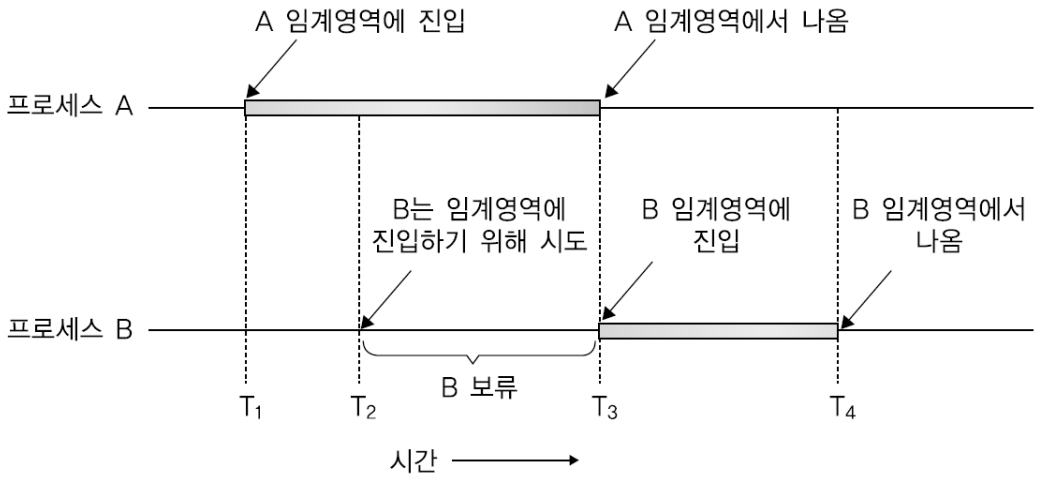
[참고] 임계영역

- 임계 영역(Critical Section)
  - 둘 이상의 프로세스가 공유할 수 없는 자원을 임계자원이라 하며, 프로그램에서 임계자원을 이용하는 부분을 임계영역이라 함
    - 공유 메모리가 참조되는 프로그램의 부분(데이터나 데이터 구조)으로 다수의 프로세스가 접근 가능한 영역이면서 한 순간에 하나의 프로세스만 사용할 수 있는 영역(공유 자원의 독점을 보장하는 코드 영역)을 의미
    - 프로세스들이 공유 데이터를 통해 협력 시, 한 프로세스가 임계영역에 들어가면 다른 모든 프로세스는 임계영역으로의 진입 금지
    - 다중 처리 시스템과 단일 처리 시스템(시분할)환경에 적용되는 하나의 실행단위, 실행 구간을 의미
      - 임계영역 내에서 빠른 속도로 작업을 수행, 한 프로세스가 오랫동안 머무르면 안됨
      - 프로세스가 무한 루프 등에 빠지지 않도록 관리



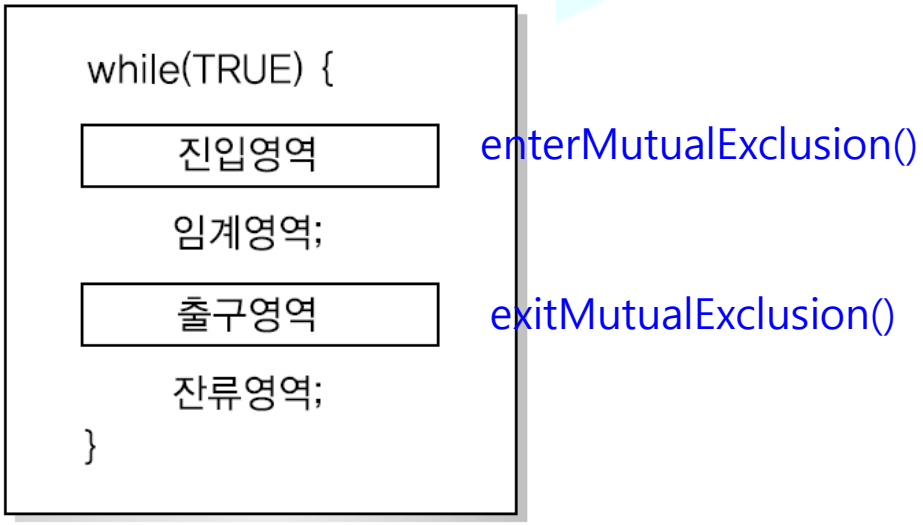
[참고] 임계영역

- 진입 상호 배제
  - 프로세스 하나가 임계 영역에 있으면 다른 프로세스가 임계 영역에 들어가지 못하게 하는 것
  - 임계 영역에 들어가기를 원하는 프로세스는 진입 상호배제를 수행해야 함
    - 프로세스가 접근하지 않은 임계 영역은 잠금 상태
    - 프로세스는 임계 영역에서 작업을 수행하기 전에 키를 얻어 임계 영역의 잠금 상태를 해제해야 함
    - 프로세스가 키를 반환할 때까지 다른 모든 프로세스에 대해 잠김 상태 유지
  - 임계 영역을 떠나는 프로세스는 출구 상호배제를 수행함으로써 다른 프로세스가 임계 영역에 들어갈 수 있도록 함



[참고] 임계영역

- 프로세스들이 서로 협력하여 자원을 사용할 수 있도록 프로토콜을 설계하여 임계영역 문제를 해결 가능
  - 진입영역(진입코드)
    - 각 프로세스는 접근하려는 자원의 임계영역에 들어갈 수 있는지 여부를 미리 요청해야 하며, 이를 코드로 구현한 부분
  - 출구영역
    - 임계영역에서 수행을 마치고 나갈 프로세스를 선택
  - 잔류영역
    - 진입영역과 출구영역을 제외한 나머지 영역으로, 임계영역을 마치고 나와 수행함



병행 프로세스에서 영역 구분

상호 배제 문제에 대한 소프트웨어 해결책

- 데커의 알고리즘
  - 첫 번째 버전 (고정된 동기화와 바쁜 대기 문제 야기)
    - 제어를 위해 변수 사용
    - 상호 배제 구현
    - [문제점1 – 바쁜대기(busy waiting)] 임계 영역이 사용가능한지 계속적으로 확인
      - 스레드가 반복해서 변수(threadNumber)값 확인
      - 중요한 프로세서 시간 낭비 (별일 아닌데 프로세서를 사용해야 함)
    - [문제점2 - 고정된 동기화 문제]
      - 한 스레드가 다른 스레드보다 빈번히 임계영역 사용한다면
      - 더 빠른 스레드가 더 느린 스레드의 속도에 맞춰 제한
    - 각 스레드는 엄격한 교대에 의해 실행

상호 배제 문제에 대한 소프트웨어 해결책

예제 5-6 상호 배제 구현-버전 1

```
01  시스템:
02
03  int threadNumber = 1;
04
05  startThreads(); // 두 스레드를 초기화하고 시작함
06
07  스레드 T1:
08
09  void main() {
10
11      while (!done)
12      {
13          while (threadNumber == 2); // enterMutualExclusion
14
15          // 임계 영역 코드
16
17          threadNumber = 2; // exitMutualExclusion
18
19          // 임계 영역 외부 코드
20
21      } // 바깥쪽 while 문끝
22
23  } // 스레드 T1 끝
24
25  스레드 T2:
26
27  void main() {
28
29      while (!done)
30      {
31          while (threadNumber == 1); // enterMutualExclusion
32
33          // 임계 영역 코드
34
35          threadNumber = 1; // exitMutualExclusion
36
37          // 임계 영역 외부 코드
38
39      } // 바깥쪽 while 문끝
40
41  } // 스레드 T2 끝
```

상호배제 구현을 위해 'threadNumber" 변수 사용

스레드가 반복해서 threadNumber 값 확인
-> 바쁜대기(busy waiting)
-> 프로세서 활용도 저하

각 스레드들은 자신들의 임계영역에 드나들 때 엄격하게 교대
-> 고정된 동기화(lockstep synchronization)

상호 배제 문제에 대한 소프트웨어 해결책

- 두 번째 버전(상호 배제 위반)
  - 고정된 동기화 문제 해결
    - 두 개의 변수 사용

      t1Inside

      t2Inside
  - 상호 배제 보장하지 않음
    - 두 스레드가 동시에 임계 영역 사용 가능
  - 적절하지 않은 방법

예제 5-7 상호 배제 구현 – 버전 2

```
01  시스템:
02
03  boolean t1Inside = false;
04  boolean t2Inside = false;
05
06  startThreads(); // 두 스레드를 초기화하고 시작함
07
08  스레드 T1:
09
10  void main() {
11
12      while (!done)
13      {
14          while (t2Inside); // enterMutualExclusion
15
16          t1Inside = true;  // enterMutualExclusion
17
18          // 임계 영역 코드
19
20          t1Inside = false; // exitMutualExclusion
21
22          // 임계 영역 외부 코드
23
24      } // 바깥쪽 while 문 끝
25
26  } // 스레드 T1 끝
27
28  스레드 T2:
29
30  void main() {
31
32      while (!done)
33      {
34          while (t1Inside); // enterMutualExclusion
35
36          t2Inside = true;  // enterMutualExclusion
37
38          // 임계 영역 코드
39
40          t2Inside = false; // exitMutualExclusion
41
42          // 임계 영역 외부 코드
43
44      } // 바깥쪽 while 문 끝
45
46  } // 스레드 T2 끝
```

상호 배제 문제에 대한 소프트웨어 해결책

- 세 번째 버전(교착 상태 야기)
  - 임계 영역 테스트 전 자신의 플래그 설정
    - 상호 배제 보장
  - [문제점3 - 교착 상태]
    - 두 스레드가 동시에 자신의 플래그 설정 가능
    - While문만 반복

예제 5-8 상호 배제 구현 – 버전 3

```
01  시스템:
02
03  boolean t1WantsToEnter = false;
04  boolean t2WantsToEnter = false;
05
06  startThreads(); // 두 스레드를 초기화하고 시작함
07
08  스레드 T1:
09
10  void main()
11  {
12      while (!done)
13      {
14          t1WantsToEnter = true;  // enterMutualExclusion
15
16          while (t2WantsToEnter); // enterMutualExclusion
17
18          // 임계 영역 코드
19
20          t1WantsToEnter = false; // exitMutualExclusion
21
22          // 임계 영역 외부 코드
23
24      } // 바깥쪽 while 문 끝
25
26  } // 스레드 T1 끝
27
28  스레드 T2:
29
30  void main()
31  {
32      while (!done)
33      {
34          t2WantsToEnter = true;  // enterMutualExclusion
35
36          while (t1WantsToEnter); // enterMutualExclusion
37
38          // 임계 영역 코드
39
40          t2WantsToEnter = false; // exitMutualExclusion
41
42          // 임계 영역 외부 코드
43
44      } // 바깥쪽 while 문 끝
45
46  } // 스레드 T2 끝
```

상호 배제 문제에 대한 소

- 데커의 알고리즘(적합한 해결책)
  - 선호 스레드 (favored thread) 개념 사용
    - 임계 영역에 여러 스레드 접근 시 먼저 진입 가능
    - 충돌 문제 해결
  - 상호 배제 보장
  - 교착 상태, 무기한 연기 방지

예제 5-10 상호 배제를 구현하는 데커의 알고리즘

```
01  시스템:
02
03  int favoredThread = 1;
04  boolean t1WantsToEnter = false;
05  boolean t2WantsToEnter = false;
06
07  startThreads(); // 두 스레드를 초기화하고 시작함
08
09  스레드 T1:
10
11  void main()
12  {
13      while (!done)
14      {                                   enterMutualExclusion()
15          t1WantsToEnter = true;
16
17          while (t2WantsToEnter)
18          {
19              if (favoredThread == 2)
20              {
21                  t1WantsToEnter = false;
22                  while (favoredThread == 2); // 바쁜 대기
23                  t1WantsToEnter = true;
24              } // if 문끝
25
26          } // while 문끝
27
28          // 임계 영역 코드
29
30          favoredThread = 2;
31          t1WantsToEnter = false;         exitMutualExclusion()
32
33          // 임계 영역 외부 코드
34
35      } // 바깥쪽 while 문끝
36
37  } // 스레드 T1 끝
38
39  스레드 T2:
40
41  void main()
42  {
43      while (!done)
44      {                                   enterMutualExclusion()
45          t2WantsToEnter = true;
46
47          while (t1WantsToEnter)
48          {
49              if (favoredThread == 1)
50              {
51                  t2WantsToEnter = false;
52                  while (favoredThread == 1); // 바쁜 대기
53                  t2WantsToEnter = true;
54              } // if 문끝
55
56          } // while 문끝
57
58          // 임계 영역 코드
59
60          favoredThread = 1;
61          t2WantsToEnter = false;         exitMutualExclusion()
62
63          // 임계 영역 외부 코드
64
65      } // 바깥쪽 while 문끝
66
67  } // 스레드 T2 끝
```

세마포어

- 세마포어
  - 상호 배제를 이루는 또 다른 방법
  - 보호변수
    - 초기화 후 P와V 중 한 연산에서만 접근, 수정
    - 스레드가 임계 영역 사용 원할 시: P연산(wait; 대기연산) 호출
    - 임계 영역을 나오고 싶을 시: V연산(signal; 신호연산) 호출
- 세마포어를 사용한 상호 배제
  - 바이너리 세마포어
    - 한번에 하나의 스레드가 임계 영역을 사용 가능
    - P연산(wait)
      - 대기(wait)하고 있는 스레드가 없을 때, 임계 영역 허용
      - P연산 호출 시, 세마포어 값 '0'으로 감소
      - 다른 스레드가 P호출 시, 해당 스레드는 블록됨
    - V연산(signal)
      - 스레드가 임계 영역 수행을 마친 후 호출
      - V연산 호출 시, 세마포어 값 '1'로 증가
      - 대기 하고 있던 스레드 임계 영역 사용

세마포어

예제 5-15 세마포어를 사용한 상호 배제

```
01 시스템:
02
03 // 세마포어를 생성하고 값을 1로 초기화함
04 Semaphore occupied = new Semaphore(1);
05
06 startThreads(); // 두 스레드를 초기화하고 시작함
07
08 스레드 Tx:
09
10 void main()
11 {
12     while (!done)
13     {
14         P(occupied); // 대기
15
16         // 임계 영역 코드
17
18         V(occupied); // 신호
19
20         // 임계 영역 외부 코드
21     } // while 문 끝
22 } // 스레드 Tx
```

If S > 0
    S = S-1
Else
    호출하는 스레드를 세마포어의 대기 스레드 큐에 넣음

If S를 대기하는 스레드가 있으면
    세마포어의 큐에서 대기하는 '다음' 스레드 시작함
Else
    S = S+1

세마포어

- 카운팅 세마포어
  - '0' 이상의 정수로 초기화되는 세마포어
  - 동일한 자원들이 있는 풀에서 자원을 할당할 때 사용
  - 풀에 있는 자원의 수와 같은 값으로 초기화
  - P호출 시, 세마포어 값 '1' 감소
    - 풀의 자원이 추가적으로 할당되어 스레드에서 사용
  - V호출 시, 세마포어 값 '1' 증가
    - 자원을 풀로 돌려주고, 이 자원을 다른 스레드에 할당 할 수 있음
  - 세마포어가 '0'까지 줄어들었을 때
    - 스레드가 P를 호출하면, V를 통해 자원을 풀로 반납할 때까지 대기