1. 국수를 아주 좋아하는 철학자 5명이 모였습니다.
그들은 원형 테이블에 둘러 앉아 "삶이란 무엇인가?"라는 질문에 대한 해답을 고민하기 시작했습니다.
원형 테이블의 중앙에는 국수가 놓여 있었고, 테이블에는 모두 다섯 개의 젓가락이 놓여 있었습니다.
즉, 철학자들 한 명의 왼쪽과 오른쪽에 각각 한 개의 젓가락이 놓여 있다는 뜻입니다.
따라서 철학자들은 왼쪽과 오른쪽 양쪽의 젓가락을 이용하여 국수를 먹을 수 있습니다.
국수는 무한리필되기 때문에 5명의 철학자들은 해답을 찾을 때까지 생각을 하기로 했습니다.
하지만 어느날 그들은 모두 굶어 죽었습니다. 왜 그랬을까요?

   이 문제는 자원(젓가락)을 공유하는 프로세스(철학자)의 동기화 문제로 유명한 메타포어인
"철학자들의 저녁식사" 문제입니다.
위 문제의 설명으로 적절한 것을 모두 고르시오.

   1) 시계 방향으로 돌아가면서 먹기로 약속을 정했으면 아무도 굶어죽지 않았을 것이다.
   2) 모두가 왼쪽에 있는 젓가락을 먼저 집고, 오른쪽 젓가락을 나중에 집기로 약속을 한다면, 아무도 굶어죽지 않을 것이다.
   3) 인접한 두 철학자들이 서로 동시에 식사를 하지 않기로 약속을 한다면, 아무도 굶어죽지 않을 것이다.
   4) 홀수 번호의 철학자는 먼저 왼쪽 젓가락을 집고, 다음에 오른쪽 젓가락을 집도록 하고, 반대로 짝수 번호인 철학자는 오른쪽 젓가락을 먼저 집고, 다음에 왼쪽 젓가락을 집도록 하면 아무도 굶어죽지 않을 것이다.

2. Counting Semaphore에 대한 설명으로 가장 틀린 것은?
   1) 여러 프로세스 간의 동기화 문제를 해결하는 데 사용할 수 있다.
   2) 여러 프로세스의 상호배제(mutual exclusion)를 지원하기 위해서 세마포어의 초기값을 n으로 지정할 수 있다.
   3) 주로 사용 가능한 자원의 인스턴스가 여러 개인 경우에 카운팅 세마포어를 사용할 수 있다.
   4) 카운팅 세마포어를 사용하면 deadlock과 starvation 문제도 해결할 수 있다.

3. 다음 중 동기화 문제 해결을 위한 monitor 방법에 대한 설명으로 가장 옳지 않은 것은?
   1) 모니터 내부의 자원(변수)에 접근하고자 하는 프로세스는 반드시 모니터 내부의 함수를 호출해야 한다.
   2) 모니터 내부의 함수를 사용하는 프로세스는 상호배제(mutual exclusion)가 보장된다.
   3) 모니터에서는 동기화를 위해 wait()와 signal()을 사용하고, signal() 함수를 호출하면 모니터 큐에서 대기한다.
   4) 자바에서는 모니터락을 지원하므로 synchronized 키워드로 간단하게 임계구역을 지정할 수 있다.

4. 주니온은 서울 출장을 가기 위해 KTX 열차표를 온라인으로 발매했는데, 해당 좌석에 가보니 동일한 시간에 동일한 좌석으로 발매된 열차표를 가진 승객이 있었다. 이 경우 KTX 온라인 발권 시스템에 어떤 문제가 발생했다고 보는 것이 가장 합리적일까?

   1) 좌석표 데이터에 대한 접근을 할 때 mutual exclusion을 제대로 보장해 주지 못했을 것이다.

   2) 여러 개의 발권 쓰레드가 deadlock에 빠져 progress 조건을 만족하지 못했을 것이다.

   3) 주니온의 발권 처리 프로세스의 우선순위가 낮아 starvation이 발생했을 것이다.

   4) 발권 처리 과정에서 bounded waiting을 하느라 중복 발행을 하게 되었을 것이다.

5. Dijkstra가 제안한 동기화 문제에 대한 소프트웨어 솔루션으로, 경쟁 상황이 발생하는 임계 영역을 가지는 여러 개의 프로세스에 대해 상호 배제를 보장하는 두 개의 연산인 P()와 V()를 제공하는 해결책을 무엇이라고 할까?

   1) 뮤텍스

   2) 세마포어

   3) 모니터

   4) 라이브니스

6. Multi-Level Feedback Queue 스케줄링 알고리즘은 여러 개의 우선순위가 다른 ready queue를 사용하여 하나의 ready queue에서 처리를 마치지 못한 프로세스를 우선순위가 더 높은 ready queue에 feedback 시켜줌으로서 우선순위를 높여준다. 이것을 우리는 노화(aging) 기법이라고 부르기도 한다. aging 기법이 필요한 이유와 가장 관련이 높은 것은?

   1) mutual exclutioin

   2) progress

   3) bounded waiting

   4) critical section

7. Process Synchronization에 대한 설명으로 가장 옳은 것은?

   1) 어떤 Scheduler를 사용하더라도 race condition은 기피할 수 없으므로 Mutual Exclusion, Progress, Bounded Waiting 을 고려해서 Critical Section Problem 을 해결해야 한다.

   2) Semaphore 는 정수 변수로서 오로지 wait과 signal 연산만을 통해서 접근할 수 있으며 반드시 0으로 초기화를 해 주어야 한다.

   3) Spinlock은 busy waiting을 하며 lock을 획득하는 방식이므로 멀티코어 시스템에서는 비효율적이므로 사용하면 좋지 않다.

   4) Monitor는 모니터 내부에서는 항상 하나의 프로세스만이 활성화되도록 보장해 주므로, 프로그래머가 동기화 제약 조건을 명시적으로 프로그래밍해야 할 필요가 없다는 장점이 있다.

8. Semaphore를 사용했을 때 발생할 수 있는 문제점으로 옳은 것을 모두 고르시오.

   1) 반드시 Spinlock(busy waiting)이 발생한다.

   2) Deadlock & Starvation이 발생할 수 있다.

   3) wait(), signal()을 순서에 맞게 사용하지 않으면 mutual exclusion 문제가 발생할 수 있다.

   4) mutual exclusion 문제는 절대로 발생할 수가 없다.

9. Peterson's algorithm과 같은 software적인 해결책은 SMP (Symmetric Multiprocessor System) 환경과 같은 modern computer system에서는 제대로 작동할 수 있다는 보장이 없다. 그 해결책으로 하드웨어적인 instruction이나 mutex, semaphore와 같은 소프트웨어적인 API를 사용하여 프로세스 간 동기화를 보장하는 방법을 사용한다. ① 이런 기법들은 모두 이것을 획득하도록 하여 critical section을 보호한다. 이것 혹은 이것을 이용하는 기법을 무엇이라 하는가? ② 동기화를 위한 이런 방법을 구현하는 데 있어서, test_and_set()이나 compare_and_swap()과 같은 하드웨어 instruction이나 mutex/semaphore의 wait(), signal() 구현은 모두 이 성질을 만족해야만 한다. 이 성질을 무엇이라 하는가?

   위에서 밑줄 친 부분에 들어갈 용어로 가장 알맞게 짝지어진 것은?

   1) locking, atomicity

   2) busy waiting, atomicity

   3) locking, integrity

   4) busy waiting, integrity

10. 다음 Java 코드 중에서 mutual exclution이 보장되기 어려운 코드를 모두 고르시오.

1)
```
class Counter {
    public static int count = 0;
    synchronized public static void increment() {
        Counter.count++;
    }
}
```

2)
```
class Counter {
    public static int count = 0;
    public static Object obj = new Object();
    public static void increment() {
        synchronized (obj) {
            Counter.count++;
        }
    }
}
```

3)
```
class Counter {
    public static int count = 0;
    public static void increment() {
        synchronized (this) {
            Counter.count++;
        }
    }
}
```

4)
```
class Counter {
    public static int count = 0;
    public static void increment() {
        synchronized (new String("lock") {
            Counter.count++;
        }
    }
}
```

Answers (indended by the Question Provider):

1) 1
2) 4
3) 3
4) 1
5) 2
6) 3
7) 4
8) 2, 3
9) 1
10) 3, 4