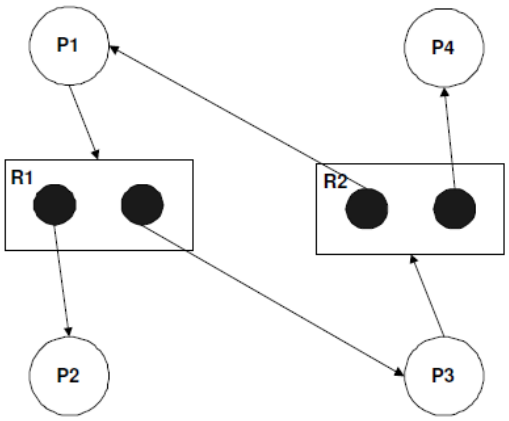
1. 모니터에 진입하려고 처음 시도하는 프로세스보다 대기 중인 프로세스에게 우선권을 줘야 하는가? 그렇지 않으면 그 반대로 우선권을 줘야 하는가? **어떤 프로세스에게 우선권을 줘야 하는지를 기술하고 그 이유를 상세히 설명하시오**.

[Answer]
진입하려는 프로세스에게 우선권을 줄 경우 대기 중인 프로세스는 계속 대기하는 문제가 발생한다. 따라서 대기 중인 프로세스에게 우선권을 줘야 한다.

2. 임의의 시스템 X 의 현재 상태는 다음과 같다고 가정하자. **현재 시스템이 deadlock 상태인지 기술하고 그 이유를 상세히 설명**하시오.
   (1) 자원 유형은 2 가지가 존재한다: R1, R2
   (2) 각 자원 유형별로 2 개의 instance 가 존재한다.
   (3) 현재 시스템에는 총 4 개의 프로세스가 존재한다: P1, P2, P3, P4
   (4) 현재 자원 할당 현황은 다음과 같다.
      - R1 유형의 instance 는 각각 P2, P3 프로세스에게 할당
      - R2 유형의 instance 는 각각 P1, P4 프로세스에게 할당
   (5) 현재 자원 요청 현황은 다음과 같다.
      - P1 프로세스는 R1 유형의 자원 요청
      - P3 프로세스는 R2 유형의 자원 요청

[Answer]
현재 시스템 X 의 Resource Allocation 그래프는 아래와 같다. 자원 요청 처리를 <P2, P1, P4, P3> 순서로 처리할 경우 deadlock 이 발생하지 않는다.



3. 다음 Pseudo 코드는 다양한 버그를 포함하고 있다. 버그를 수정하시오.

```
Semaphore mutex = 0; // semaphore initialization

// returns true if the item was added, false otherwise
Bool add_item_to_queue(queue_t queue, item_t item) {
    Wait(mutex);
    if (is_full(queue)) { // check if the queue is full
        return false;
    }
    append_to_queue(queue, item); // add an item to the queue
    Signal(mutex);
    return true;
    Signal(mutex);
}
```

[Answer]

```
~~Semaphore mutex = 0; // semaphore initialization~~
Semaphore mutex = 1; // semaphore initialization

// returns true if the item was added, false otherwise
Bool add_item_to_queue(queue_t queue, item_t item) {
    Wait(mutex);
    if (is_full(queue)) {
        Signal(mutex);
        return false;
    }
    append_to_queue(queue, item);
    Signal(mutex);
    return true;
    ~~Signal(mutex);~~
}
```

4. 다음은 2 개의 프로세스(e.g., P0, P1)에 대한 Critical Section 문제를 해결하기 위한 솔루션을 나타낸다. **아래 코드는 정상적으로 동작하는가? 만약 그렇지 않다는 그 원인을 상세히 기술하시오.** (단, LOAD/STORE 명령은 Atomic 하게 동작한다고 가정한다).

enum { P0, P1 } turn = P0; // turn is a global shared variable	
P0's code	P1's code

// functions to enter the critical section
void critical_section_enter_P0() {
  while (turn != P0) continue;
}
// functions to leave the critical section
void critical_section_leave_P0() {
  turn = P1;
}
	// functions to enter the critical section
	void critical_section_enter_P1() {
	  while (turn != P1) continue;
	}
	// functions to leave the critical section
	void critical_section_leave_P1() {
	  turn = P0;
	}

[Answer]

No. P0 프로세스가 CR 코드 완료 후 다시 CR 에 진입하기 위해서는 반드시 P1 프로세스가 CR 코드를 수행한 뒤에만 가능함. 따라서 Progress 를 만족하지 못함.

critical_section_enter_P0();
critical_section_leave_P0();
// may block even though t1 is not in the critical section
critical_section_enter_P0();
critical_section_leave_P0();

5. 프로세스 P1, P2, P3 의 우선순위는 P1 = 1, P2 = 5, P3 = 10 (값이 클수록 우선 순위가 높음)로 설정되어 있으며, 각 프로세스가 수행하는 코드 및 주요 특성은 다음과 같다.

P1	**P2**	**P3**
<code sequence A> **Wait(X)** **<Critical Section>** **Signal(X)** **<code sequence B>**	**<code sequence A>** **Wait(X)** **Wait(Y)** **<Critical Section>** **Signal(Y)** **Signal(X)** **<code sequence B>**	**<code sequence A>** **Wait(Y)** **Wait(X)** **<Critical Section>** **Signal(X)** **Signal(Y)** **<code sequence B>**
- Semaphore X 와 Y 는 각각 "1"로 설정됨 - <code sequence A> 실행에 걸리는 시간: 2 time units - <code sequence B> 실행에 걸리는 시간: 3 time units - <Critical Section> 실행에 걸리는 시간: 4 time units - Wait(), Signal() 및 Context Switch 에 소비되는 시간: 0 time unit - P1, P2, P3 은 time 0, time 3, time 10 에서 실행을 시작함		

Preemptive Priority Scheduling 알고리즘을 사용한다고 가정했을 때 각 프로세스의 실행 Diagram 을 작성하시오. Diagram 에서 사용될 notation 은 다음과 같다.

- A: <code sequence A> 실행 중
- B: <code sequence B> 실행 중
- X: 프로세스가 Semaphore X 를 획득하여 현재 Critical Section 을 수행하고 있음
- Y: 프로세스가 Semaphore Y 를 획득하여 현재 Critical Section 을 수행하고 있음
- 2: 프로세스가 Semaphore X, Y 를 모두 획득하여 현재 Critical Section 을 수행하고 있음
- Blank: 프로세스가 현재 실행되고 있지 않음 (for any reason)

Time	00	01	02	03	04	05	06	07	08	09
P1										
P2										
P3										

Time	10	11	12	13	14	15	16	17	18	19
P1										
P2										
P3										

Time	20	21	22	23	24	25	26	27	28	29
P1										
P2										
P3										

[Answer]

Time	00	01	02	03	04	05	06	07	08	09
P1	A	A	X			X	X	X		
P2				A	A				2	2
P3										

Time	10	11	12	13	14	15	16	17	18	19
P1										
P2			2	2						
P3	A	A			2	2	2	2	B	B

Time	20	21	22	23	24	25	26	27	28	29
P1					B	B	B			
P2		B	B	B						
P3	B									

6. 프로세스 A 와 B 는 세마포어 R 과 S 를 이용하여 상호 동기화를 수행한다. R = 0, S = 1 로 초기화되어 있으며, 프로세스 A 와 B 는 각각 다음과 같은 코드를 수행한다고 했을 때 **다음 물에 대해 True 또는 False 를 선택하고 그 이유를 상세히 기술하시오**.

Process A	**Process B**
Wait(R)	**Operation B.1**
Wait(S)	**Signal(R)**
Operation A.1	**Operation B.2**
Signal(S)	**Wait(S)**
Operation A.2	**Operation B.3**
Signal(R)	**Signal(S)**

a. **[5]** (True/False) Operation A.1 은 Operation B.1 이 끝나기 전에는 시작될 수 없다
b. **[5]** (True/False) Operation A.1 과 Operation B.2 는 동시에 수행될 수 없다.
c. **[5]** (True/False) Operation A.1 과 Operation B.3 는 동시에 수행될 수 없다.
d. **[5]** (True/False) Operation A.2 와 Operation B.3 은 동시에 수행될 수 없다
e. **[5]** (True/False) 프로세스 A 와 B 는 Deadlock 에 빠진다.

[Answer]

a. True, R 이 0 으로 초기화되어 있기 때문에 프로세스 B 가 Signal(R)를 수행하기 전에는 Wait(R)가 완료될 수 없다
b. False, 프로세스 B 가 Signal(R) 실행 후, R 과 S 는 각각 1 로 설정되기 때문에 프로세스 A 는 Wait(R), Wait(S)를 수행할 수 있으며, 따라서 Operation A.1 과 Operation B.2 는 동시에 실행 가능하다.
c. True, 프로세스 A 가 Wait(S)를 실행했기 때문에 프로세스 A 가 Signal(S)를 실행하기 전에는 프로세스 B 는 Wait(S)를 완료할 수 없으며 따라서, Operation A.1 과 Operation B.3 은 동시에 수행될 수 없다.
d. False, 프로세스 A 의 Signal(S) 실행이 완료된 후 프로세스 B 가 Wait(S)를 호출할 경우 S 값이 1 로 설정되기 때문에 프로세스 B 는 Block 되지 않고 Operation B.3 을 수행할 수 있으며 따라서 Operation A.2 와 Operation B.3 은 동시에 실행될 수 있다.
e. False, 프로세스 B 는 "Hold and Wait" 조건을 만족하지 않기 때문에 Deadlock 에 빠지지 않는다.

7. 다음은 Peterson's Solution 과 같이 2 개의 프로세스를 대상으로 Critical Section 문제를 해결하기 위한 솔루션의 Pseudo-Code 이다. 다음 코드의 정상 동작 여부를 기술하고 그 이유를 상세히 기술하시오

```
boolean flag[2];
while(1) {
    flag[i] = true;
    while (flag[j]);
    …
    Critical Section
    …

    flag[i] = false;
    …
    Remainder Section
    …
}
```

[Answer]

두 프로세스가 동시에 flag[i] = true 부분을 수행할 경우 모든 프로세스가 while 루프에서 빠져 나올 수 없기 때문에 progress 조건을 만족하지 못함.