9. 병행처리

1. 병행 프로세스 동기화(synchronization)

병행 프로세스들이 '공유 데이터'에 동시에 접근하면 데이터 무결성(정확성)에 중대한 문제가 발생할 수 있다. 무결성 유지를 위해 서로 협력하는 프로세스들은 시간에 따라 순차적으로 실행되게 하는 메커니즘을 **'동기화'**라 한다.

◈ **협력 프로세스**

협력 프로세스는 실행 중에 서로 영향을 주고받는 프로세스로 코드나 데이터 등 자원을 공유하게 된다.

---

(1) 동기화란(synchronization)?

컴퓨터 사용에서 '**동기화**'라는 용어는 여러 곳에서 등장한다.

동기화에 대한 개념을 정확하게 집고 넘어가기로 한다.

하드웨어 부분	• 하드웨어적인 동기화는 **시간**을 맞추는 것이다. • 어떤 두 가지 작업을 시간적으로 일치시키거나 같은 간격으로 발생되게 한다. [예] 동기화 DRAM은 CPU의 외부 클럭과 같은 주파수로 작동된다. 비동기식 데이터 전송에서 발생되는 비트열의 간격은 일정하다.
소프트웨어 부분	• 소프트웨어인 동기화는 **형태나 속성**을 일치시키는 것이다. • 동기화 대상은 여러 개의 프로세스들이 공동으로 사용하는 것이다. [예] 채팅 프로그램에서 각각 사용자가 보는 대화 목록은 같아야 한다.

• 동기화는 '공유 자원의 상태 일치'와 **'예측 가능한 상태'**를 위한 것이다.

• 동기화 도구로 세마포어(semaphore)와 모니터(monitor) 기법이 있다.
• 세마포어는 기본형, 모니터는 구조형을 이용하여 동기화를 처리한다.

(2) 병행 프로세스는 '왜 동기화를 해야 하는가?'

어떤 제품을 **생산/판매**하는 개념을 이용하여 **동기화의 중요성**을 살펴보기로 한다.

```
int count = 7; //제품의 현재 수량
void main(){
    make();      //제품 하나를 생산
    sell();      //제품 하나를 판매
}
void make(){ count++; }
void sell(){ count--; }
```

→ 이 프로그램이 순차적으로 수행되면 수행 후에 제품의 수는 7개 그대로이다. 하나를 생산하고 하나를 판매하였기 때문이다.

여기서, make()와 sell()은 변수 count를 공유하는 협력 프로세스이다. 병행 실행되면 위와 같은 결과 7이 될 수 있을 것인가?

① 먼저, count++;와 count--;를 전형적인 **기계어**로 각각 풀어 쓰면 다음과 같다.

count++(생산)	count--(판매)
r1 = count r1 = r1 + 1 count = r1	r2 = count r2 = r2 - 1 count = r2

→ 여기서, r1과 r2는 지역 CPU의 레지스터이다.

- 고급언어 한 줄은 기계어로 여러 줄로 변역될 수 있다.

② **이 기계어들이 다음처럼 병행 수행된다면(서로 혼합되어 수행)**

	기계어 실행 순서	결과 값
T0	**실행 전**	**count = 7**
T1	**생산 r1 = count**	**r1 = 7**
T2	**생산 r1 = r1 + 1**	**r1 = 8**
T3	**판매 r2 = count**	**r2 = 7**
T4	**판매 r2 = r2 - 1**	**r2 = 6**
T5	**생산 count = r1**	**count = 8**
T6	**판매 count = r2**	**count = 6**

→ 실행 전, 'count = 7'이다.
기계어들의 병행 실행 결과는 'count = 6'이다.
하나를 생산하고 하나를 판매하였으면
최종 결과는 'count = 7'이어야 한다.
즉, 데이터 무결성에 중대한 문제가 발생되었다.
시간적으로 동기화가 되지 않았기 때문이다.
결과가 틀렸다는 것은 예측 불가인 것이다.

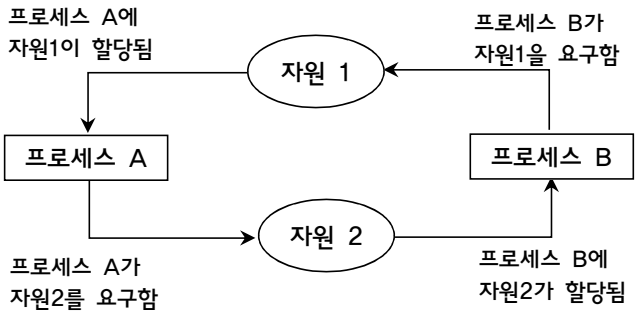
[결론] 한 프로세스가 공유 자료를 취급하고 있으면
다른 프로세스는 공유 자료에 접근할 수 없도록 하면 데이터 무결성이 유지된다.

2. 교착상태(deadlock)

교착상태는 컴퓨터 자원을 공유하는 여러 프로세스들이 상대방이 어떤 자원에 접근하는 것을 서로 방해함으로써, **프로세스들이 동시에 기능이 중지**되는 결과를 초래하는 상황을 말한다.

즉, 교착상태는 서로 다른 프로세스들이 자신은 요구한 자원을 할당받아 점유하고 있으면서, 상대방 프로세스에게 할당되어 있는 자원을 상호간에 요구할 때 발생될 수 있다.

다음 그림처럼 프로세스 A와 B는 자신이 사용 중인 자원은 반납하지 않고, 다른 프로세스가 사용하고 있는 자원을 서로 요구할 때 교착상태가 발생될 수 있다.



◈ **교착상태가 발생되기 위한 4가지 필요조건**

시스템에서 교착상태는 다음 4가지 조건이 **동시에 만족**될 때 발생할 수 있다.

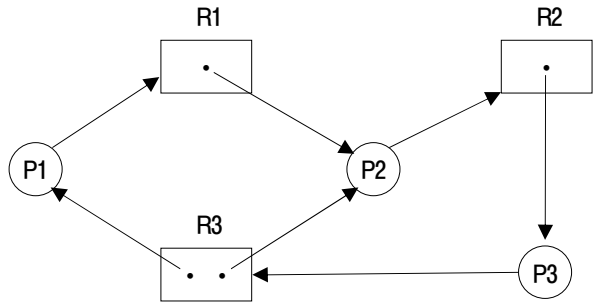
상호배제 **(mutual exclusion)**	적어도 하나의 자원은 **하나의 프로세스만 점유**할 수 있어야 한다. 즉, 비공유 자원이 최소한 한 개가 있으면, 프로세스들은 그 자원이 방출될 때까지 대기해야 한다.**(예 : 프린터)**
비선점 **(non-preemption)**	다른 프로세스가 사용 중인 자원을 **강제로 빼앗을 수 없다.** 즉, 프로세스가 스스로 방출만 한다.
점유와 대기 **(hold & wait)**	프로세스가 적어도 하나의 자원을 점유한 채, 다른 프로세스가 사용 중인 **자원을 추가로 얻기 위해 대기**하고 있다.
순환대기 **(circular wait)**	모든 프로세스가 **체인 형태**로 다른 프로세스가 점유하고 있는 자원을 획득하기 위해 대기한다.

탐구

자원 할당 그래프

교착상태는 방향그래프인 '자원 할당 그래프'를 그려서 쉽게 파악할 수 있다.

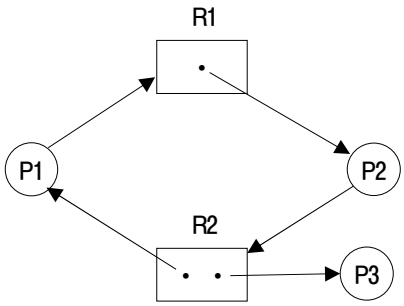
① 교착상태를 가지는 자원 할당 그래프



- p1, p2, p3는 프로세스를 의미하고
- R1, R2, R3는 자원의 종류이고
- 사각형 안의 점은 사용 가능한 자원의 개수이다.
- R1, R2는 1개의 자원이 있고 R3는 2개의 자원을 사용할 수 있다.

- 위의 그래프에는 두 개의 사이클이 존재한다.
  P1 → R1 → P2 → R2 → P3 → R3 → P1
  P2 → R2 → P3 → R3 → P2
- 프로세스 P1, P2, P3는 서로 다른 프로세스가 사용 중인 자원을 요구하고 있으므로 교착상태에 있다.(사용할 수 있는 자원이 없다)

② 사이클이 있지만 교착상태가 아닌 자원 할당 그래프



- 사이클 'P1 → R1 → P2 → R2 → P1'이 존재하지만 프로세스의 교착상태는 아니다.
- 이유는 프로세스 P3이 사용 중인 자원 **R2의 인스턴스를 방출**할 수 있기 때문이다.
  ↓ 프로세스 P3이 종료되면,
- P2는 R2의 인스턴스를 획득할 수 있다.**(사이클 파괴)**

교착상태 예방과 회피 차이점 - 헷갈리는 부분이다.

---

① 교착상태 예방((prevention)

교착상태 예방은 **교착상태 발생 4가지 필요조건**(상호배제, 점유와 대기, 순환대기, 비선점) 중 적어도 **어느 하나가 성립되지 않도록** 조치를 취하면 교착상태가 발생되는 것을 예방할 수 있다.

② 교착상태 회피(avoidance)

- 교착상태 회피 전략은 각 프로세스들에게 자신 수행에 필요로 하는 **자원을 요청하는 방법**을 추가로 제시하도록 하는 것이다.
- 예를들면, 각 프로세스에게 자신이 수행되는데 필요한 각 타입의 자원에 대한 최대 수량을 정의하도록 요구하는 것이다.
- 교착상태 회피 전략은 프로세스들이 수행에 필요로 하는 자원의 사용 방법과 최대 수량 등을 미리 알고 있으면 시스템이 교착상태에 들어가지 않도록 알고리즘을 구현할 수 있다는 것이다.

// 교착상태 예방과 회피의 차이점은 대충 정리하면 매우 헷갈리는 부분이 될 수 있다.

예방	독감을 예방하기 위해서는 미리 예방 주사를 맞으면 되고,
회피	독감에 걸리지 않기 위해서는 사람들이 많이 모이는 장소에 가지 않으면 된다.

- 그런데, 교착상태 예방과 회피의 차이점은 위의 독감 예처럼 쉽게 구분되는 것은 아니다.
- 만약, 시험에 출제된다면, 정리를 제대로 한 사람만이 정확하게 맞출 수 있다.

◉ 은행원 알고리즘(Banker's algorithm)

- Dijkstra가 제시한 **교착상태 회피 알고리즘**이다.(1965년)
- 모든 고객들의 대출 요청이 충족되도록 현금을 준비한다.
- 각각의 프로세스는 자원의 최대 사용량을 미리 명시한다.

(시스템에서 각 프로세스가 필요로 하는 자원의 최대 갯수를 알기 어렵다 - 단점)

- 시스템이 항상 안전상태에 있도록 자원할당 요청을 조절하는 방법이다.
- 시스템을 불안전상태에 빠지게 만드는 요청은 거절한다.

(빌려줬을 때 돌려받지 못하게 될 가능성이 있는 요청은 거절)

3. 임계구역(critical section)

- 임계구역은 병행 프로세스들이 **공유하는 데이터 구역**이다.
- 어떤 프로세스가 공유 데이터를 접근하는 중일 때 그 프로세스는 **임계구역**에 있다고 한다.
- 임계구역에서 실행 중인 프로세스는 **인터럽트 상태가 발생되지 않도록** 해야 한다.
- 병행 프로세스들의 임계구역 진입 문제는 다음 **3가지 요건이 충족**되어야 해결된다.

---〈임계구역 문제에 대한 해결책 - 3가지 요건〉---

① **상호배제**(mutual exclusion)
- 하나의 프로세스가 임계구역에 실행중이면, **다른 프로세스는 들어갈 수 없어야 한다.**
- 즉, 상호배제는 어떤 프로세스 p가 자신의 임계구역에서 실행되고 있을 때
- 다른 프로세스들은 그 들 자신의 임계구역에서 실행될 수 없다.

② **진행**(progress)
- 현재, 임계구역에서 실행 중인 프로세스가 없는 상태이고,
- 임계구역으로 진입하려는 프로세스들이 있다면,
- 이들 중 어느 하나만 선택되어 **임계구역으로 진입**할 수 있도록 해야 한다.
- 이 선택이 무한정 연기될 수 없다.
- 즉, **프로그램 수행이 영속적으로 진행**되어야 한다는 뜻이다.

③ **제한된 대기**(bounded waiting)
- 어떤 프로세스도 **제한된 대기 뒤에는 임계구역에 진입**할 수 있어야 한다는 뜻이다.
- 한 번 임계구역에 들어간 프로세스는 다음 번 임계구역에 들어갈 때 제한을 두어야 한다.
- 제한된 대기는 임의의 프로세스가 **기아상태**(starvation)가 되는 것을 방지하려는 것이다.
- 기아상태는 프로세스가 **세마포어 내에서 무한히 기다리는 상태**이다.**(무한봉쇄)**

// 임계구역이 있는 프로그램 구조

```
int a = 1, b = 2, c;
int 임계구역(int k){
   a++; b--;
   c = 3 * k + a + b;
   return c;
}
void main(){
      :
   process1(){ x = 임계구역(4); ... }
   process2(){ x = 임계구역(5); ... }
   process3(){ x = 임계구역(6); ... }
      :
}
```

→ 점선 부분의 코드가 임계구역이다.
(병행 프로세스들이 공유하는 부분이다)

→ 3개의 프로세스는
병행 실행되는 프로세스로 간주한다.

4. 세마포어(semaphore)

세마포어는 다중 프로그래밍 환경에서 여러 프로세스가 **공유자원에 접근하는 것을 제한**한다.
세마포어는 Dijkstra(다익스트라)가 고안한 것이다.

(1) 세마포어 개요

① 동기화 도구인 세마포어는 **정수형 변수**를 이용한 동기화 기법이다.
→ 즉, 프로세스의 임계구역 접근제어에 세마포어 기법을 이용할 수 있다.
② 세마포어 정수형 변수 S는 초기값 설정 이외는 **연산 P와 V만**으로 접근할 수 있다.
→ P와 V는 네덜란드어 Proberen(검사)과 Verhogen(증가)의 첫 글자이다.

P(s)	while (s<=0) do nothing; s = s - 1;	→ 세마포어(s)가 's<=0'이면 → 프로세스는 임계구역에 진입하지 못하고 대기 상태
V(s)	s = s + 1;	→ 세마포어(s) 값을 1증가시킨다.

---

③ 세마포어(s) 값은 원자적으로 갱신되어야 한다.**(중간에 인터럽트 되어도 안 된다)**
→ 어떤 프로세스가 세마포어 값을 변경하고 있으면 다른 프로세스는 이를 변경할 수 없다.

(2) 세마포어 구현

세마포어를 이용한 프로세스들의 임계구역 접근제어는 다음과 같다.

```
Process :
  Semaphore s = 1;
  P(s);
     [임계구역];
  V(s);
     [나머지 구역];
```

◉ **임계구역 접근제어 기본 원리 - 바쁜 대기 발생**

- 세마포어 **'s = 1'**이면 프로세스는 임계구역에 **진입**하게 된다.(진입 조건)
- 임계구역에 진입하기 전에 프로세스는 's = s - 1'을 수행한다.(P 연산)
  → 's = 0'이 되므로 다른 프로세스들은 임계구역에 들어갈 수 없다.
- 임계구역에서 벗어나면서 프로세스는 's = s + 1'을 수행한다.(V 연산)
  → 's = 1'이 되므로 다른 프로세스들이 임계구역에 들어갈 수 있다.

// 운영체제에서 세마포어 구분

이진 세마포어	이진수 0 또는 1만을 가질 수 있다.
계수 세마포어	제한되지 않은 여러 범위의 값을 가질 수 있다. (제한된 수의 자원에 대한 접근제어에 사용될 수 있다)

5. 모니터(monitor)

세마포어는 효과적인 동기화 기법을 제공하지만, 잘못 사용하면 **타이밍 오류가 발생**할 수 있다.
해서, 고수준 형태의 동기화 구조가 필요하다. 즉, 모니터는 고수준 동기화 구조를 가진다.

다음은 모니터 구조를 Java 코드처럼 표현한 것이다.

```
moniter 모니터명
{
    static int a = 0;                  //변수 정의(전용 데이터)
    int b = 1;
        :
    public entry p1( ){ · · · }        //프로시저 정의(전용 데이터 연산 처리)
    public entry p2( ){ · · · }
    public entry p3( ){ · · · }
                :
}
```

다음은 모니터에 대한 설명이다.
'스레드(프로세스)와 프로시저' 이 두 용어를 잘 구분해서 읽기 바란다.

① 모니터 내부에 정의된 변수는 **모니터 내부에 정의된 프로시저만 접근**할 수 있다.
② 여러 개의 스레드들이 동시에 모니터에 진입하려는 경우,
모니터 구조 자체는 상호배제를 엄격하게 적용한다. 즉, 하나의 스레드만 진입할 수 있다.
모니터 내에 정의된 프로시저에 대한 스레드의 병행 접근은 저절로 해결된다.
③ 따라서, 한 시점에는 **하나의 스레드만 모니터 내부에서 수행**된다.
즉, 프로그래머는 동기화를 위한 코드를 별도로 작성할 필요가 없게 된다.
이유는 모니터 타입 내에 동기화를 위한 코드가 내장되어 있기 때문이다.
④ 모니터는 여러 스레드들이 안전하고 효과적으로 자료를 공유할 수 있도록 한 동기화 기법이다.

[Tip] 컴퓨터 언어 자바는 세마포어를 지원하지 않는다. 그러나 프로그래머가 직접 구현은 할 수 있다.
자바(Java), Mesa, Concurrent Pascal과 같은 언어는 모니터 개념의 병행성 구조를 제공한다.
그리고, 자바의 모든 객체는 자신과 관련된 모니터를 가진다.
즉, 자바의 스레드가 **synchronized** 블록에 진입하므로 객체의 모니터를 얻게 된다.

기출문제 분석

1. 운영체제에서 교착상태(deadlock)가 발생하기 위한 필요조건에 해당되지 않는 것은? [2014년 지방 9급]

① 상호배제(mutual exclusion)
② 점유와 대기(hold and wait)
③ 선점(preemption)
④ 순환대기(circular wait)

♤ **교착상태(deadlock)가 발생하기 위한 필요조건**

- 교착상태 발생 필요조건 : 상호배제, 비선점, 점유와 대기, 순환대기
- 시스템에서 교착상태는 4가지 조건이 **동시에 만족될 때 발생**할 수 있다.
- 교착상태는 **프로세스들이 동시에 기능이 중지**되는 결과를 초래하는 상황을 말한다.

정답 : ③

2. 교착상태가 발생하기 위해서 만족해야 하는 조건들에 대한 설명으로 옳지 않은 것은? [2016년 국가 9급]

① 상호배제(mutual exclusion) 조건 :한 프로세스에 의해 점유된 자원은 다른 프로세스가 사용할 수 없다.
② 점유와 대기(hold and wait) 조건 : 이미 하나 이상의 자원을 점유한 프로세스가 다른 프로세스에 의해 점유된 자원을 요청하며 대기하고 있다.
③ 비선점(no preemption) 조건 : 프로세스가 점유한 자원을 그 프로세스로부터 강제로 빼앗을 수 있다.
④ 순환대기(circular wait) 조건 : 프로세스 간에 닫힌 체인(closed chain)이 존재하여, 체인 내의 각 프로세스는 체인 내의 다른 프로세스에 의해 소유되어 있는 자원을 요청하며 대기하고 있다.

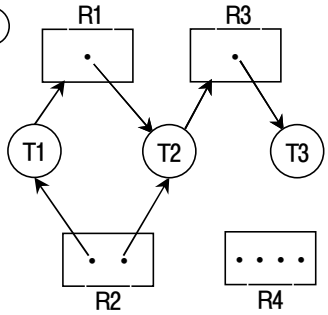
♤ **교착상태(deadlock)**

- 비선점(no preemption) 조건 : 프로세스가 점유한 자원을 강제로 빼앗을 수 없다.

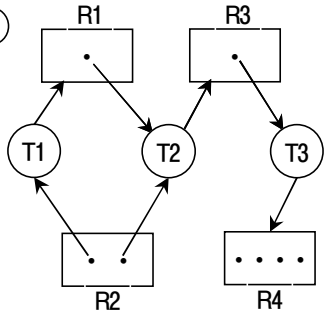
정답 : ③

3. 다음 중 교착상태인 시스템 자원 할당 그래프에 해당하는 것만을 모두 고르면? (단, 자원 할당 그래프에서 T_i는 쓰레드, R_j는 자원(사각형), $T_i \rightarrow R_j$는 요청 간선, $R_j \rightarrow T_i$는 할당 간선이고 각 자원 R_j는 한 개 이상의 인스턴스(사각형 내의 점)를 가진다) [2022년 국회 9급]

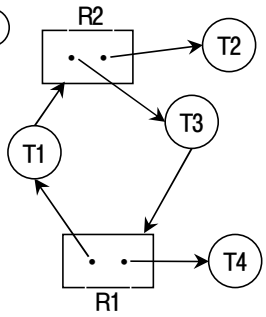
ㄱ



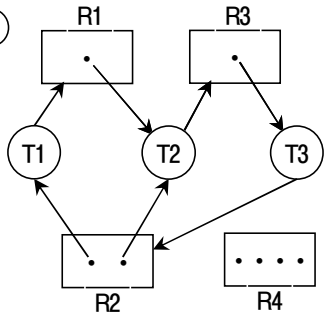
ㄴ



ㄷ

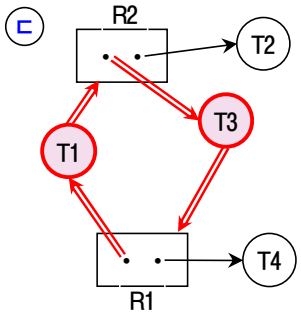
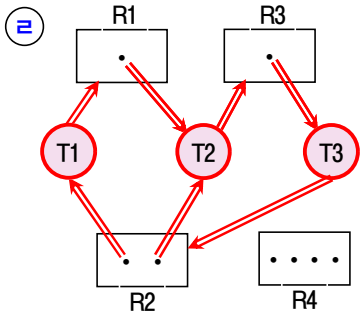


ㄹ



① ㄱ ② ㄴ ③ ㄹ
④ ㄱ, ㄴ ⑤ ㄷ, ㄹ

♤ 자원 할당 그래프

ㄷ	ㄹ
	
・사이클이 **존재**하지만 **교착상태가 아니다.** ・이유는 T4가 사용 중인 자원 **R1의 인스턴스를 방출**할 수 있기 때문이다.	・사이클이 **존재하므로 교착상태이다.** ・사이클 : T1→R1→T2→R3→T3→R2→T1,T2

정답 : ③

4. 운영체제의 세마포어(semaphore)에 대한 설명으로 옳지 않은 것은? [2022년 국가 9급]

① 프로세스간 상호배제(mutual exclusion)의 원리를 보장하는데 사용된다.
② 여러 개의 프로세스가 동시에 그 값을 수정하지 못한다.
③ 세마포어에 대한 연산은 수행 중에 인터럽트 될 수 있다.
④ 세마포어는 플래그 변수와 그 변수를 검사하거나 증감시키는 연산들로 정의된다.

♤ **세마포어**

• 세마포어에 대한 연산은 수행 중에 **인터럽트** 될 수 **있다.(×) → 인터럽트** 될 수 **없다.**

// 세마포어 개요

① 동기화 도구인 세마포어는 **정수형 변수**를 이용한 동기화 기법이다.
→ 즉, 프로세스의 임계구역 접근제어에 세마포어 기법을 이용할 수 있다.
② 세마포어 정수형 변수 S는 초기값 설정 이외는 **연산 P와 V만**으로 접근할 수 있다.
→ P와 V는 네덜란드어 Proberen(검사)과 Verhogen(증가)의 첫 글자이다.

P(s)	while (s<=0) do nothing;	→ 세마포어(s)가 's<=0'이면
	s = s - 1;	→ 프로세스는 임계구역에 진입하지 못하고 대기 상태
V(s)	s = s + 1;	→ 세마포어(s) 값을 1증가시킨다.

③ 세마포어(s) 값은 원자적으로 갱신되어야 한다.**(중간에 인터럽트 되어도 안 된다)**
→ 어떤 프로세스가 세마포어 값을 변경하고 있으면 다른 프로세스는 이를 변경할 수 없다.

// 세마포어 구현

```
Process :
  Semaphore s = 1;
  P(s);
     [임계구역];
  V(s);
     [나머지 구역];
```

◉ **임계구역 접근제어 기본 원리 - 바쁜 대기 발생**
• 세마포어 **'s = 1'**이면 프로세스는 임계구역에 **진입**하게 된다.(진입 조건)
• 임계구역에 진입하기 전에 프로세스는 's = s - 1'을 수행한다.(P 연산)
→ 's = 0'이 되므로 다른 프로세스들은 임계구역에 들어갈 수 없다.
• 임계구역에서 벗어나면서 프로세스는 's = s + 1'을 수행한다.(V 연산)
→ 's = 1'이 되므로 다른 프로세스들이 임계구역에 들어갈 수 있다.

• 세마포어는 다중 프로그래밍 환경에서 여러 프로세스가 **공유자원에 접근하는 것을 제한**한다.
• 세마포어는 Dijkstra(다익스트라)가 고안한 것이다.

// 운영체제에서 세마포어 구분

이진 세마포어	이진수 0 또는 1만을 가질 수 있다.
계수 세마포어	제한되지 않은 여러 범위의 값을 가질 수 있다. (제한된 수의 자원에 대한 접근제어에 사용될 수 있다)

정답 : ③

5. 교착상태(deadlock)를 해결할 수 있는 방법으로 적당하지 않은 것은? [2015년 서울 9급]

① 프로세스들이 필요로 하는 자원에 대해 배타적인 통제권을 갖게 한다.
② 자원에 선형으로 고유번호를 할당하고, 각 프로세스는 현재 점유한 자원의 고유번호보다 큰 번호 방향으로만 자원을 요구하도록 한다.
③ 한 프로세스가 실행되는 데 필요한 모든 자원을 할당한 후 실행시킨다.
④ 자원을 점유하고 있는 프로세스가 다른 자원을 요구할 때, 점유하고 있는 자원을 반납하고 요구하도록 한다.

♤ **교착상태 해결**

• 프로세스들이 필요로 하는 자원에 대해 **배타적인** 통제권을 갖게 한다.**(×)**
→ 자원 사용이 배타적이면 교착상태를 해결할 수 없다.
→ 예를 들면, 프린터라는 자원은 동시에 서로 다른 두 가지를 출력할 수 없다.

정답 : ①

6. 다중 쓰레드(multi thread) 프로그래밍을 할 때 다음 C언어의 변수들 중에서 임계구역(critical section)에 해당하는 것은? [2015년 서울 9급]

① 매크로변수(macro variable)
② 지역변수(local variable)
③ 함수인자(argument)
④ 전역변수(global variable)

♤ **임계구역**

```
int count = 7;              //임계구역, 모든 thread는 전역변수를 공유한다.
void main()
{
    make();
    sell();
}
void make(){ count++; }     //thread
void sell(){ count--; }     //thread
```

정답 : ④

7. 프로세스(process)나 스레드(thread)들이 공유자원에 하나 이상의 수정(write 또는 modify) 연산을 포함하는 동시 접근을 할 때 그 접근 부분들을 임계구역(critical section)이라 한다. 이를 보호하기 위한 병행 프로세스 동기화 기법으로 옳은 것은? [2018년 국회 9급]

① 인터럽트(interrupt)
② 선점 스케줄링(preemptive scheduling)
③ 문맥교환(context switching)
④ 상호배제(mutual exclusion)
⑤ 교착상태(deadlock)

♤ **임계구역 – 병행 프로세스 동기화 기법**

- 임계구역은 병행 프로세스들이 **공유하는 데이터 구역**이다.
- 어떤 프로세스가 공유 데이터를 접근하는 중일 때 그 프로세스는 **임계구역**에 있다고 한다.
- 임계구역에서 실행 중인 프로세스는 **인터럽트 상태가 발생되지 않도록** 해야 한다.

◈ **상호배제(mutual exclusion)**

- 하나의 프로세스가 임계구역에 실행중이면, **다른 프로세스는 들어갈 수 없어야 한다.**
- 즉, 상호배제는 어떤 프로세스 p가 자신의 임계구역에서 실행되고 있을 때
- 다른 프로세스들은 그 들 자신의 임계구역에서 실행될 수 없다.

정답 : ④

8. 임계지역(critical section) 문제에 대한 해결책이 가져야 하는 성질로 가장 옳지 않은 것은? [2018년 서울 9급]

① 한 번에 한 프로세스만이 임계지역을 수행하도록 해야 한다.
② 프로세스는 자신이 임계지역을 수행하지 않으면서 다른 프로세스가 임계지역을 수행하는 것을 막으면 안된다.
③ 프로세스의 임계지역 진입은 유한시간 내에 이루어져야 한다.
④ 임계지역 문제의 해결책에서는 프로세스의 수행 속도에 대해 적절한 가정을 할 수 있다.

♤ **임계지역 문제**

- 임계지역 문제의 해결책에서는 프로세스의 수행 속도에 대해 적절한 가정을 할 수 있다.**(×)**
  → 임계지역 문제 해결책과 프로세스의 수행 속도는 연관이 없다.

정답 : ④

9. 임계구역에 대한 설명으로 옳은 것은? [2021년 국가 9급]

① 임계구역에 진입하고자 하는 프로세스가 무한대기에 빠지지 않도록 하는 조건을 진행의 융통성(progress flexibility)이라 한다.

② 자원을 공유하는 프로세스들 사이에서 공유자원에 대해 동시에 접근하여 변경할 수 있는 프로그램 코드 부분을 임계영역(critical section)이라 한다.

③ 한 프로세스가 다른 프로세스의 진행을 방해하지 않도록 하는 조건을 한정 대기(bounded waiting)라 한다.

④ 한 프로세스가 임계구역에 들어가면 다른 프로세스는 임계구역에 들어갈 수 없도록 하는 조건을 상호배제(mutual exclusion)라 한다.

♤ **임계구역**

① 임계구역에 진입하고자 하는 프로세스가 무한대기에 빠지지 않도록 하는 조건을 **진행의 융통성**(progress flexibility)이라 한다.**(×)** → **제한된 대기**(bounded waiting)

② 자원을 공유하는 프로세스들 사이에서 공유자원에 대해 **동시**에 접근하여 **변경할 수 있는** 프로그램 코드 부분을 임계영역(critical section)이라 한다.**(×)** → **동시**에 **변경할 수 없는**

③ 한 프로세스가 다른 프로세스의 진행을 방해하지 않도록 하는 조건을 **한정 대기**(bounded waiting)라 한다.**(×)** → **진행**(progress)

정답 : ④

10. 임계구역(critical region)에 대한 설명으로 옳지 않은 것은? [2021년 군무원 9급]

① 하나의 프로세스만 사용해야 임계구역 내의 자원의 무결성을 보장할 수 있다.

② 동시에 다수의 프로세스가 병렬적으로 실행할 수 있도록 하여 실행 효율성을 높일 수 있는 영역이다.

③ 임계구역을 정의하기 위해서는 상호배제(mutual exclusion) 기법이 필요하다.

④ 세마포어(P(s), V(s))는 임계구역 내에 하나의 프로세스만 허용하도록 하는 용도로 사용하는 기술이다.

♤ **임계구역**

- 동시에 **다수**의 프로세스가 **병렬적으로 실행**할 수 있도록 하여 실행 효율성을 높일 수 있는 영역이다.(×) → 하나의 프로세스가 임계구역에 실행중이면, **다른 프로세스는 들어갈 수 없어야 한다.**
- 임계구역은 병행 프로세스들이 **공유하는 데이터 구역**이다.
- 어떤 프로세스가 공유 데이터를 접근하는 중일 때 그 프로세스는 **임계구역**에 있다고 한다.

정답 : ②

11. 은행원 알고리즘(banker's algorithm)이 교착상태를 해결하는 방법은? [2022년 지방 9급]

① 예방 ② 회피 ③ 검출 ④ 회복

♤ **은행원 알고리즘 – 교착상태 회피**

- Dijkstra가 제시한 **교착상태 회피 알고리즘**이다.(1965년)
- 모든 고객들의 대출 요청이 충족되도록 **현금**을 **준비**한다.
- 각각의 프로세스는 자원의 최대 사용량을 미리 명시한다.

(시스템에서 각 프로세스가 필요로 하는 자원의 최대 갯수를 알기 어렵다 – 단점)

- 시스템이 항상 안전상태에 있도록 자원할당 요청을 조절하는 방법이다.
- 시스템을 불안전상태에 빠지게 만드는 요청은 거절한다.

(빌려줬을 때 돌려받지 못하게 될 가능성이 있는 요청은 거절)

교착상태 회피	• 교착상태 **회피** 전략은 각 프로세스들에게 자신 수행에 필요로 하는 **자원**을 **요청**하는 방법을 추가로 제시하도록 하는 것이다. • 예를들면, 각 프로세스에게 자신이 수행되는데 필요한 각 타입의 자원에 대한 최대 수량을 정의하도록 요구하는 것이다. • 교착상태 회피 전략은 프로세스들이 수행에 필요로 하는 자원의 사용 방법과 최대 수량 등을 미리 알고 있으면 시스템이 교착상태에 들어가지 않도록 알고리즘을 구현할 수 있다는 것이다.

정답 : ②

12. 프로세스 동기화 문제를 해결하기 위한 방법인 세마포어(semaphore) 알고리즘에 대한 설명으로 옳지 않은 것은? [2014년 계리직]

① 세마포어 알고리즘은 상호배제 문제를 해결할 수 없다.
② 세마포어 변수는 일반적으로 실수형 변수를 사용하지 않는다.
③ 세마포어 알고리즘은 P 연산(wait 연산)과 V 연산(signal 연산)을 사용한다.
④ P 연산과 V 연산의 구현 방법에 따라 바쁜 대기(busy waiting)를 해결할 수 있다.

♤ **세마포어**

- 세마포어 알고리즘은 상호배제 문제를 해결할 수 없다.**(×)**
  → 세마포어 알고리즘은 상호배제 문제를 해결할 수 있다.
  → 즉, 프로세스의 임계구역 상호배제에 세마포어 기법을 이용할 수 있다.
- 동기화 도구인 세마포어는 정수형 변수를 이용한 동기화 기법이다.

정답 : ①