

# 프로세스 동기화

## 동기화란?

### 동기화

- “동기화”라는 단어는 사용하는 상황에 따라 다른 의미를 가진다.
- Wikipedia: 오케스트라에서 지휘자와 같이 시스템을 조화롭게 작동시키기 위해 이벤트를 조정하는 것이다. 컴퓨터 과학에서 동기화는 프로세스 동기화 또는 데이터 동기화를 의미한다.

### 데이터 동기화

- Wikipedia: 데이터세트의 복사본들을 일관성 있게 유지하거나 데이터의 무결성을 유지하는 프로세스이다.

### 프로세스 동기화

- Wikipedia: 스레드 또는 프로세스 동기화는 임계구역(공유자원을 사용하는 공간)에서 둘 이상의 프로세스 또는 스레드가 동시에 실행되지 않도록 보장하는 메커니즘이다.
- geeksforgeeks: 멀티 프로세스 시스템에서 여러 프로세스가 제어되고, 예측 가능한 방식으로 공유 자원에 접근할 수 있도록 조정하는 것
- [책] 쉽게 배우는 운영체제: 프로세스끼리 통신을 하는 경우 누가 먼저 작업할지, 작업이 언제 끝날지 등을 서로 알려주어야 하는데, 이를 동기화라고 한다.

# 동기화란?

## 동기화

- “동기화”라는 단어는 사용하는 상황에 따라 다른 의미를 가진다.
- Wikipedia: 오케스트라에서 지휘자와 같이 시스템을 조화롭게 작동시키기 위해 이벤트를 조정하는 것이다. 컴퓨터 과학에서 동기화는 프로세스 동기화 또는 데이터 동기화를 의미한다.

## 데이터 동기화

- Wikipedia: 데이터세트의 복사본들을 일관성 있게 유지하거나 데이터의 무결성을 유지하는 프로세스이다.

## 프로세스 동기화

- Wikipedia: 스레드 또는 프로세스 동기화는 임계구역(공유자원을 사용하는 공간)에서 둘 이상의 프로세스 또는 스레드가 **동시에 실행되지 않도록 보장**하는 메커니즘이다.
- geeksforgeeks: **멀티 프로세스 시스템**에서 여러 프로세스가 **제어**되고, 예측 가능한 방식으로 **공유 자원**에 접근할 수 있도록 조정하는 것
- [책] 쉽게 배우는 운영체제: **프로세스끼리 통신**을 하는 경우 누가 먼저 작업할지, 작업이 언제 끝날지 등을 서로 알려주어야 하는데, 이를 동기화라고 한다.

## 용어 정리

### 공유자원 ✓

- 여러 프로세스가 공동으로 이용하는 변수, 메모리, 파일 등을 말한다.



### 경쟁 상태( Race Condition ) ✓

- 2개 이상의 프로세스가 공유 자원을 병행적으로 읽거나 쓰는 상황이다.
- ✓ 동일한 코드를 실행하거나 동일한 메모리 또는 공유 변수에 접근하고 있는 모든 프로세스가 자신의 값이 맞다고 경쟁하는 것을 말하는 바람직하지 않는 상태이다.

### 임계구역( Critical Section )

- 프로그램에서 임계자원을 이용하는 부분이다.
- 다수의 프로세스가 접근 가능한 영역이지만, 한 순간에 하나의 프로세스만 사용할 수 있는 영역이다.

# 프로세스 간 통신

## (Inter Process Communication)

## 프로세스 간 통신의 종류

### 1. 프로세스 내부 데이터 통신

하나의 프로세스 내에 2개 이상의 스레드가 존재하는 경우의 통신이다. 프로세스 내부의 스레드는 전역 변수나 파일을 이용하여 데이터를 주고 받는다.

### 2. 프로세스 간 데이터 통신

같은 컴퓨터에 있는 여러 프로세스끼리 통신하는 경우로, 공용 파일 또는 운영체제가 제공하는 파이프를 사용하여 통신한다.

### 3. 네트워크를 이용한 데이터 통신

여러 컴퓨터가 네트워크로 연결되어 있을 때도 통신이 가능한데, 이 경우 프로세스는 소켓을 이용하여 데이터를 주고받는다. 이처럼 소켓을 이용하는 프로세스 간 통신을 네트워킹이라고 한다. 다른 컴퓨터에 있는 함수를 호출하여 통신하는 원격 프로시저 호출도 여기에 해당한다.

## 프로세스간 통신의 분류

### 통신 방향에 따른 분류

프로세스 간 통신은 동시에 실행되는 프로세스끼리 데이터를 주고받는 작업을 의미한다.

- 양방향 통신

데이터를 동시에 양쪽 방향으로 전송할 수 있는 구조. 일반적인 통신은 모두 양방향 통신이다. 예) 소켓 통신

- 반양방향 통신

데이터를 양쪽 방향으로 전송할 수 있지만 동시 전송은 불가능하고 특정 시점에 한쪽 방향으로만 전송할 수 있는 구조. 예) 무전기

- 단방향 통신

모스 신호처럼 한쪽 방향으로만 데이터를 전송할 수 있는 구조. 예) 전역 변수, 파이프

### 통신 구현 방식에 따른 분류

전역 변수의 경우 받는쪽에서는 언제 데이터가 올지 모르기 때문에, 반복적으로 값을 점검해야 한다. 이처럼 수시로 데이터가 전송되었는지 확인하기 위해 반복문을 무한 실행하며 기다리는 것을 바쁜 대기(busy waiting)이라고 한다. 만약 데이터가 도착했음을 알려주는 동기화(synchronization)을 사용하면, 바쁜 대기 문제를 해결할 수 있다. 예) 메신저 알림

- 대기가 있는 통신

동기화를 지원하는 통신 방식. 데이터를 받는 쪽은 데이터가 도착할 때까지 자동으로 대기 상태에 머물러 있다.

- 대기가 없는 통신

동기화를 지원하지 않는 통신 방식. 데이터를 받는 쪽은 바쁜 대기를 사용하여 데이터가 도착했는지 여부를 직접 확인한다.

# 동기화가 필요한 이유



## 1. 데이터의 일관성과 무결성을 유지하기 위해

- 데이터의 무결성이란 데이터가 전송, 처리되는 모든 과정에서 변경되거나 손상되지 않고 완전성, 정확성, 일관성을 유지함을 보장하는 특성이다.(정확하고 유효한 데이터 유지)
- 데이터의 일관성이란 서로 다른 위치에 보관된 동일한 데이터가 일치하는지에 대한 여부를 나타낸다.

## 2. 경쟁 상태를 피하기 위해 ✓

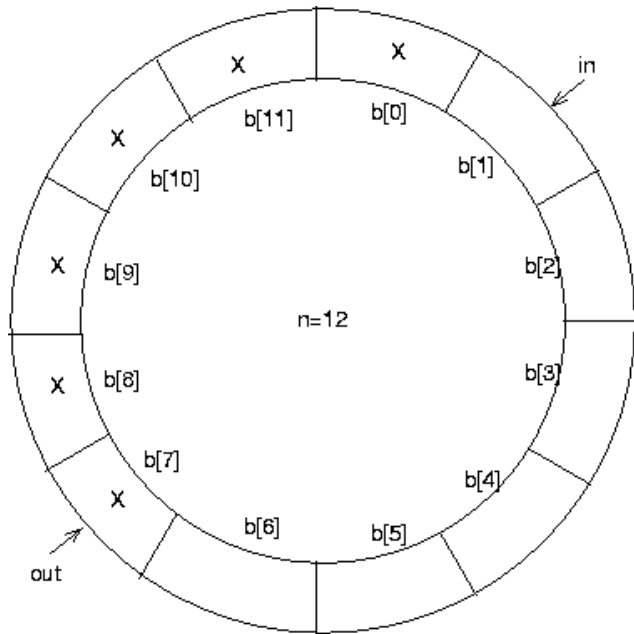
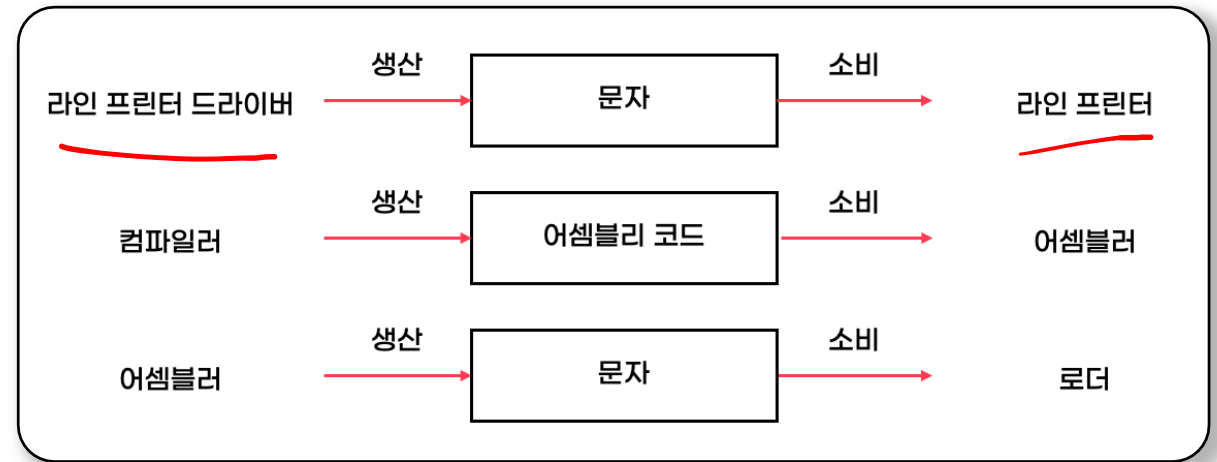
- 2개 이상의 프로세스가 공유 자원을 병행적으로 읽거나 쓰는 상황에서 모든 프로세스가 자신의 값이 맞다고 말하는 바람직 하지 않은 “경쟁 상태”를 피해야한다.

## 3. 생산자-소비자 문제를 해결하기 위해

- 생산자-소비자 문제는 임계구역과 관련된 전통적인 문제이다.
- 다음과 같은 상황에서 문제가 발생한다.
  1. 생산자가 이미 채워진 버퍼에 더 채우려고 할 때
  2. 소비자가 빈 버퍼에서 데이터를 꺼내려고 할 때
  3. 생산자와 소비자가 동시에 접근할 때

## 생산자-소비자 문제

- 다음과 같은 상황에서 문제가 발생한다.
  - 생산자가 이미 채워진 버퍼에 더 채우려고 할 때
  - 소비자가 빈 버퍼에서 데이터를 꺼내려고 할 때
  - 생산자와 소비자가 동시에 접근할 때



```
Producer() {  
    input(buf);  
    sum = sum + 1;  
}
```

sum 변수 읽기 (sum = 6)

sum + 1 → sum = 7

sum 쓰기

```
Consumer() {  
    output(buf);  
    sum = sum - 1;  
}
```

sum 변수 읽기 (sum = 6)

sum - 1 → sum = 5

sum 쓰기

## 공유 자원과 임계구역

```
AddMoney () {  
  
    read (money)  
    money = money + 5  
    write (a)  
  
}
```



### 공유 자원

- 공유 자원은 여러 프로세스가 공동으로 이용하는 변수, 메모리, 파일 등을 말한다.
- 공유 자원은 공동으로 이용되기 때문에 누가 언제 데이터를 읽거나 쓰느냐에 따라 그 결과가 달라질 수 있다.
- 2개 이상의 프로세스가 공유 자원을 병행적으로 읽거나 쓰는 상황을 “경쟁 조건(race condition) ” 이 발생했다고 한다.

### 임계구역

- 공유 자원 접근 순서에 따라 실행 결과가 달라지는 프로그램의 영역을 임계구역(critical section)이라고 한다.
- 프로세스 실행 상황에서 공유할 수 없는 자원이 있는 부분을 임계구역으로 지정하지 않으면 문제가 발생할 수 있다.
- 임계구역에서는 한 순간에 한 프로세스만 접근이 가능하다.

## 임계구역 해결 조건

어떤 방법이든 다음의 세 가지 조건을 만족해야 한다.

### 1. 상호 배제 (mutual exclusion)

- 한 프로세스가 임계구역에 들어가면 다른 프로세스는 임계구역에 들어갈 수 없다.

### 2. 한정 대기(bounded waiting)

- 어떤 프로세스도 무한 대기(infinite postpone)하지 않아야 한다. 즉 특정 프로세스가 임계구역에 진입하지 못하면 안 된다.

### 3. 진행의 융통성(progress flexibility)

- 한 프로세스가 다른 프로세스의 진행을 방해해서는 안 된다.
- 번갈아가며 사용하기로 했다고 하더라도 임계 구역을 사용하는 프로세스가 없다면 기다리지 않고 사용할 수 있도록 해야함.

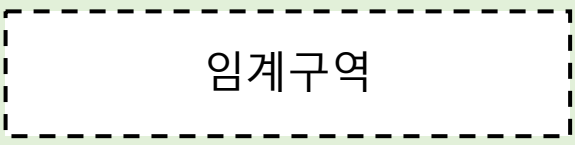
# 임계구역 해결 방법

## 1. 뮤텝스

- 프로세스가 자원을 사용하면 자원을 잠그고, 모두 사용하면 잠금을 해제하는 잠금 및 잠금해제 메커니즘이다.

```
Object myObject = new Object(); Mutex  
mutex = myObject.getMutex(); mutex.free();
```

```
while (true) {  
    acquire lock
```



임계구역

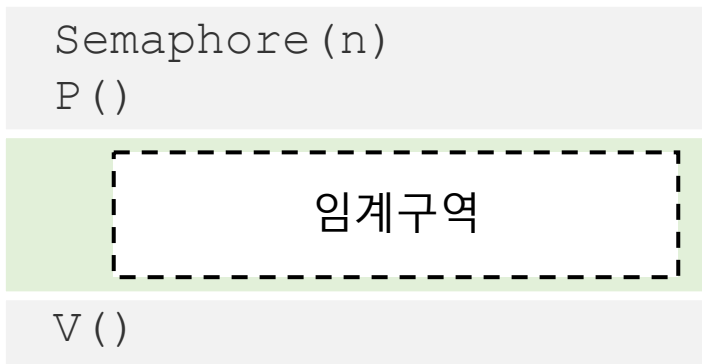
```
    release lock  
}
```

```
acquire() {  
    while (available)  
        ; /* busy wait */  
    available = false;  
}
```

```
release() {  
    available = true;  
}
```

## 2. 세마포어 ( Semaphore )

- 세마포어는 깃발이라는 뜻이다. 옛날에는 기차길에서 통행 가능 여부를 깃발을 통해서 나타냈고, 이 깃발을 세마포어라고 했다. ( 즉 이 겹치는 기차길 부분이 임계구역인 것이다. )
- Semaphore는 공유 가능한 자원의 수를 나타내는 정수 값이다.
- P ( wait )과 V ( Signal ) 두 개의 원자적 함수를 사용하며, 해당 함수를 통해 세마포어에 접근할 수 있다.
  - P : 자원을 요청하는 함수이다. 사용 가능한 자원이 있다면 자원의 수를 감소시킨 후 임계구역에 진입하고, 자원이 없다면 기다린다.
  - V : 자원을 반납하고 기다리는 프로세스에게 신호를 보내는 함수이다. 자원의 수를 증가시킨다. Block&WakeUp 방식의 세마포어에서는 대기 중인 프로세스가 있을 때 대기 리스트에서 프로세스를 꺼낸다. 꺼내진 프로세스는 준비 리스트에 추가된다.
- Binary Semaphore와 Counting Semaphore로 나뉜다.
  - Binary Semaphore: Semaphore가 0과 1의 값만 가질 수 있다. 이는 잠금을 구현하는 데 사용되며, 뮤텍스로도 알려져있다.
  - Counting Semaphore: 범위를 지정할 수 있는 Semaphore이다.



### Algorithm 1: Semaphore Wait Operation

```
Function wait(S) is
    if S > 0 then
        /* do not block the process
        S ← S - 1;
        return;
    else
        /* block the calling process
        sleep;
    end
end
```

### Algorithm 2: Semaphore Signal Operation

```
Function signal(S) is
    if there are processes sleeping on S then
        select a process to wake up;
        wake up the selected process;
        return;
    else
        /* no process is waiting on S
        S ← S + 1;
        return;
    end
end
```

### 3. 스핀락

- Wikipedia: 스핀락은 임계구역 진입이 불가능할 때 진입이 가능할 때까지 루프를 돌며 재시도하는 방식(바쁜 대기) 으로 구현된 락을 말한다.
- 운영체제의 스케줄링을 지원을 받지 않고 잠금이 해제될 때까지 루프를 수행하며 CPU를 소모한다.

```
while (true) {  
    acquire lock
```

임계구역

```
    release lock  
}
```

```
acquire() {  
    while (available)  
        ; /* busy wait */  
    available = false;  
}
```

```
release() {  
    available = true;  
}
```

#### ▲ 뮉텍스 방식으로 구현한 스핀락 ▲

```
Semaphore S = 1  
wait(S)
```

임계구역

```
signal(S)
```

```
wait(S) {  
    while (S <= 0)  
        ; /* busy wait */  
    S--;  
}
```

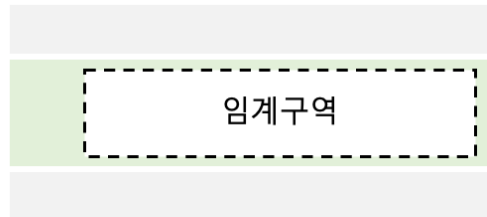
```
signal(S) {  
    S++;  
}
```

#### ▲ 세마포어 방식으로 구현한 스핀락 ▲

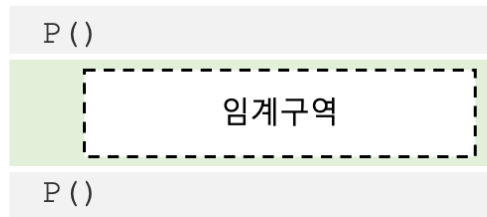


## 4. 모니터

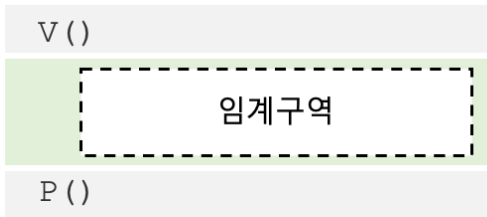
- 세마포어의 가장 큰 문제는 잘못된 사용으로 인해 임계구역이 보호받지 못한다는 것이다.
- 공유 자원을 사용할 때 모든 프로세스가 세마포어 알고리즘을 따른다면, 굳이 P()와 V()를 사용할 필요 없이 자동으로 처리하면 된다. 이를 실제로 구현한 것이 모니터 ( monitor )이며, 뮉텍스나 세마포어보다 높은 레벨의 동기화 형태이다.
- 모니터는 공유 자원을 숨기고 공유 자원에 접근하기 위한 인터페이스만 제공함으로써 자원을 보호하고 프로세스 간에 동기화를 시킨다.



세마포어를 사용하지 않고 바로 임계 구역에 들어가면, 임계구역을 보호할 수 없다.

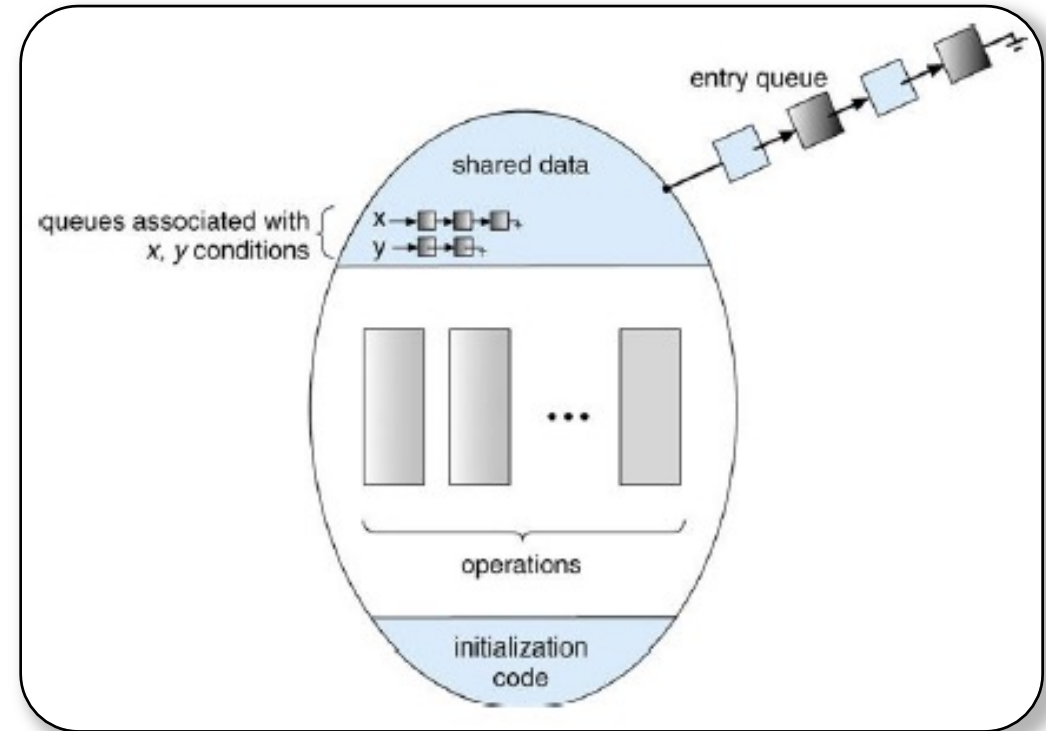


P() 를 두 번 사용하여 wake\_up 신호가 발생하지 않는다. 따라서 무한 대기에 빠진다.



P()와 V()를 반대로 사용하면 상호 배제가 보장되지 않아 임계 구역을 보호할 수 없다.

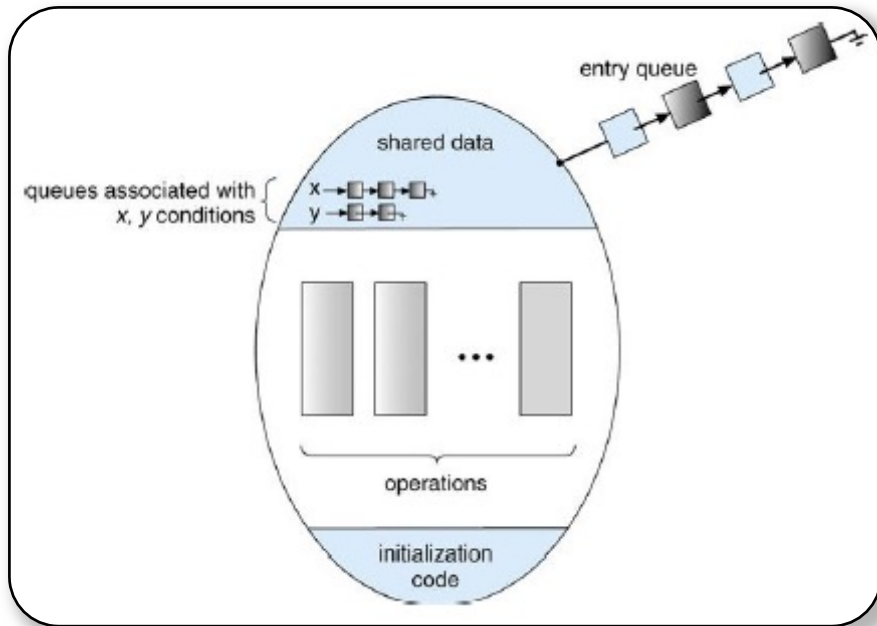
세마포어에서 실수로 인해 발생할 수 있는 문제점



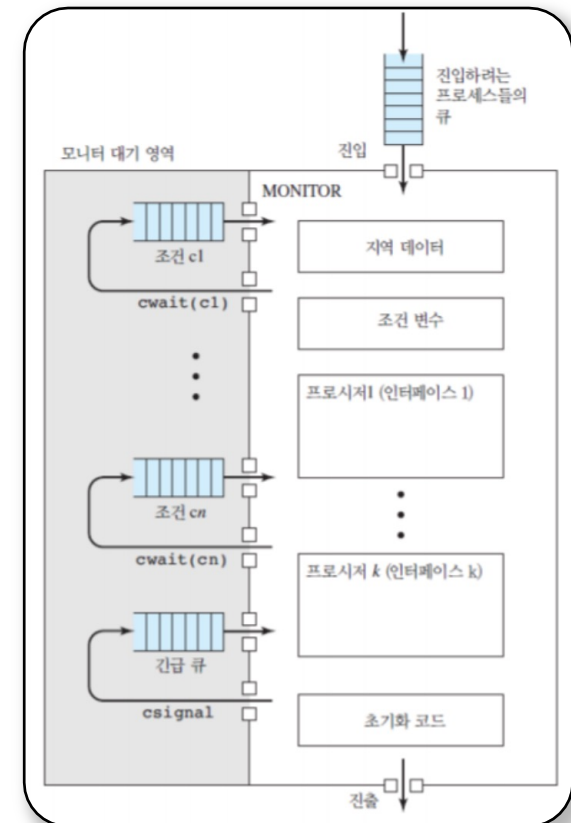
이미지 출처: <https://cs-ssupport.tistory.com/m/428>

## 4. 모니터

- Monitor 내부에 진입하기 위해서는 Monitor lock을 얻어야 하며, 프로세스는 Monitor 내부에 선언된 프로시저(메서드)를 통해서만 공유자원에 접근이 가능하다.
- Monitor에 존재하는 여러 프로시저들은 동시에 실행될 수 없다.
- Monitor 내부에서는 "오직 하나의 프로세스"만이 active상태가 된다.
- Monitor 구조 자체만으로는 동기화 수행이 부족하기 때문에, Conditional Variables를 사용하며, Conditional Variables은 wait()과 signal() 메서드를 사용해서만 접근이 가능하다.



```
monitor <monitor-name>{  
  <shared variable declarations>  
  procedure body P1(...){  
    ...  
  }  
  procedure body P2(...){  
    ...  
  }  
  procedure body P3(...){  
    ...  
  }  
  ....  
  initialization code(...){  
    ...  
  }  
}
```



## 퀴즈

### 1. 세마포어(Semaphore)에 관한 설명 중 틀린 것은?

- 1 상호배제 문제를 해결하기 위하여 사용한다.
- 2 정수의 변수로서 양의 값만을 가진다.
- 3 여러 개의 프로세스가 동시에 그 값을 수정하지 못한다.
- 4 자원의 수를 감소시키고 자원이 없을 경우 대기하는 wait() 함수와 자원의 수를 증가시키고 대기하는 프로세스를 깨우는 signal() 원자적 함수가 있다.

### 2. 모니터(Monitor)에 관한 설명 중 틀린 것은?

- 1 공유 데이터와 이 데이터를 처리하는 프로시저로 구성된다.
- 2 모니터 외부의 프로세스는 모니터 내부의 데이터를 직접 접근할 수 없다.
- 3 모니터에서는 Wait()과 Signal() 연산이 사용된다.
- 4 한순간에 여러 프로세스가 모니터에 동시에 집입하여 자원을 공유할 수 있다.

### 3. 임계구역에 진입할 수 없을 때 바쁜대기를 하도록 구현된 락 기법은 무엇인가?

- 1 Metext 2 Spinlock 3 Monitor 4 Semaphore

## 퀴즈

### 1. 세마포어(Semaphore)에 관한 설명 중 틀린 것은?

- 1 상호배제 문제를 해결하기 위하여 사용한다.
- 2 정수의 변수로서 양의 값만을 가진다. → 음수도 가능하지만, 음수일 경우 자원을 요청하고 대기한다.
- 3 여러 개의 프로세스가 동시에 그 값을 수정하지 못한다.
- 4 자원의 수를 감소시키고 자원이 없을 경우 대기하는 wait() 함수와 자원의 수를 증가시키고 대기하는 프로세스를 깨우는 signal() 원자적 함수가 있다.

### 2. 모니터(Monitor)에 관한 설명 중 틀린 것은?

- 1 공유 데이터와 이 데이터를 처리하는 프로시저로 구성된다.
- 2 모니터 외부의 프로세스는 모니터 내부의 데이터를 직접 접근할 수 없다.
- 3 모니터에서는 Wait()과 Signal() 연산이 사용된다.
- 4 한순간에 여러 프로세스가 모니터에 동시에 집입하여 자원을 공유할 수 있다. → 모니터 내부에서는 한 번에 하나의 프로세스만 실행된다.

### 3. 임계구역에 진입할 수 없을 때 바쁜대기를 하도록 구현된 락 기법은 무엇인가?

- 1 Metext 2 Spinlock 3 Monitor 4 Semaphore

## 참고

1. 전반적인 개념 - [책] 운영체제 - 그림으로 배우는 원리와 구조, [책] 쉽게 배우는 운영체제

2. 동기화 정의 - “[OS] Synchronization 동기화란?”

<https://en.wikipedia.org/wiki/Synchronization>

[https://en.wikipedia.org/wiki/Synchronization\\_\(computer\\_science\)](https://en.wikipedia.org/wiki/Synchronization_(computer_science))

<https://ooeunz.tistory.com/94>

3. 세마포어

<https://www.geeksforgeeks.org/synchronization-by-using-semaphore-in-python/>

[https://en.wikipedia.org/wiki/Semaphore\\_\(programming\)](https://en.wikipedia.org/wiki/Semaphore_(programming))

4. 뮤텍스와 세마포어의 차이

<https://www.shiksha.com/online-courses/articles/mutex-vs-semaphore-what-are-the-differences/>

5. 뮤텍스 예시

<https://codegym.cc/groups/posts/220-whats-the-difference-between-a-mutex-a-monitor-and-a-semaphore>

6. 동기화가 필요한 이유

<https://www.geeksforgeeks.org/introduction-of-process-synchronization/>

7. 데이터 무결성 정의

<http://terms.tta.or.kr/dictionary/dictionaryView.do?subject=%EB%8D%B0%EC%9D%B4%ED%84%B0+%EB%AC%B4%EA%B2%B0%EC%84%B1>

8. 데이터 일관성 정의

[https://en.wikipedia.org/wiki/Data\\_consistency](https://en.wikipedia.org/wiki/Data_consistency)

9. 뮤텍스, 스핀락, 세마포어

<https://cs-ssupport.tistory.com/m/428>