# 뮤텍스 & 세마포어

이성호

# 교착상태

코딩중

점유중

대기

임계영역

대기

게임중

점유중

# 임계영역(Critical Section)

## 교착상태 발생 가능 영역

네가지 조건

1. 상호 배제

2. 점유 대기

3. 비선점

4. 순환대기

# 상호배제(MUTual EXclusion)

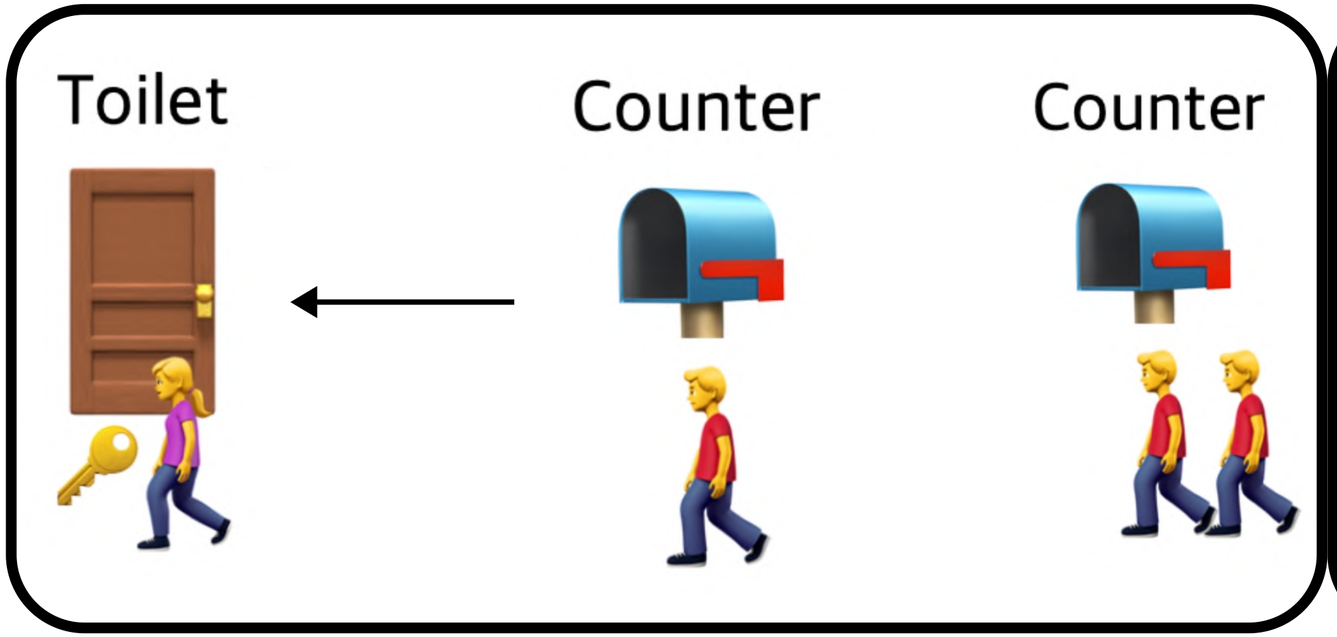
프로세스들이 필요로하는 자원에 대해 하나의 프로세스가 배타적인 통제권, 즉 독점권을 요구하는 것

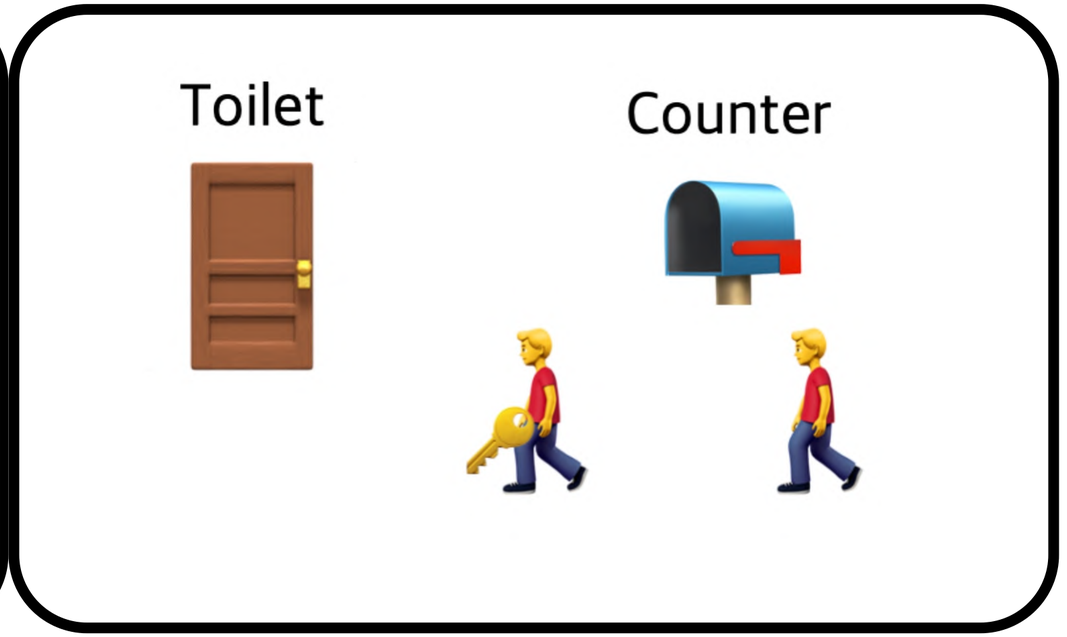
# Mutex

**여러 스레드를 실행하는 환경에서 자원에 대한 접근을 제한을 강제하기 위한 동기화 매커니즘**

**화장실 : 임계영역**
**사람 : 스레드**

**FIFO : Queue**



# Mutex

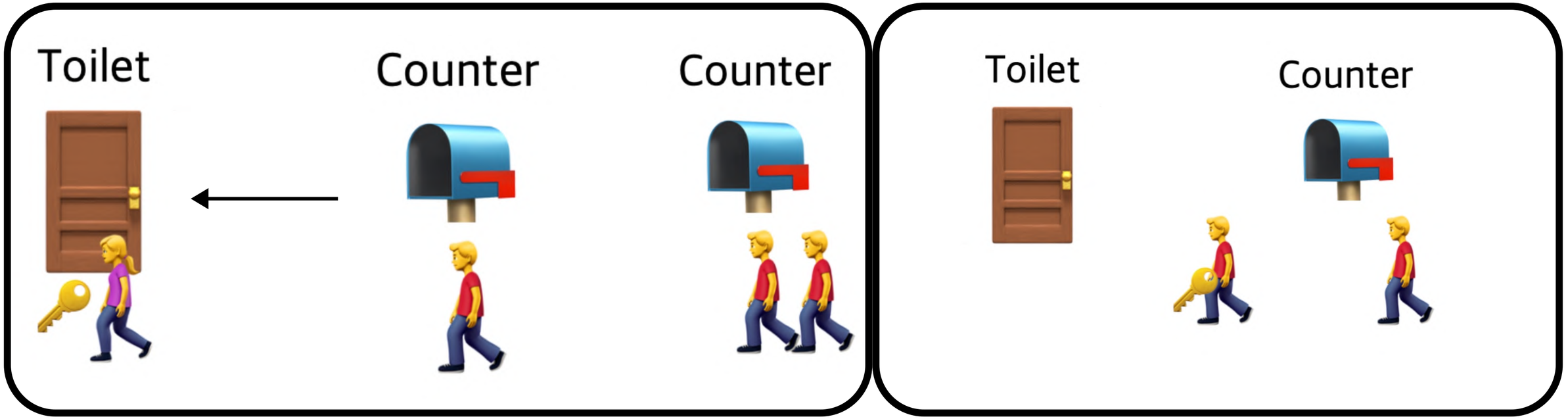
시스템 전반의 성능에 영향을 주고 싶지 않고 길게 처리해야하는 작업인 경우에 주로 사용된다. 주로 스레드 작업에서 많이 사용된다.

lock()

## Sleep 상태

unlock()

## WakeUp상태



# Mutex

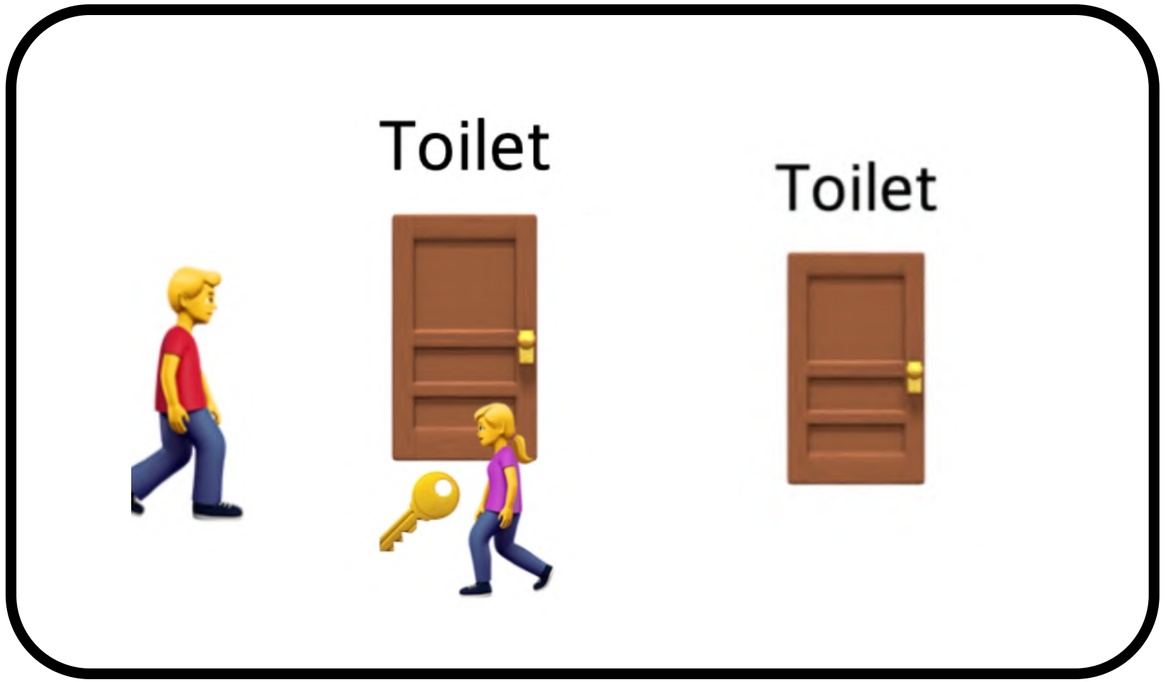
```
do {
  wait (mutex);

    // Critical section

  signal (mutex);

    // Remainder section
} while (TRUE);
```

# SpinLock

임계영역이 언락되어 진입이 가능해질 때까지 루프를 돌면서 재시도하여 스레드가 CPU를 점유하고 있는 상태

짧은 시간 안에 진입할 수 있는 경우 문맥 교환 비용이 들지 않으므로 효율을 높일 수 있다 (ContextSwitching X)

## 멀티 코어 프로세스

# SpinLock

```
wait(S) {
  while (S <= 0);  // 자원이 없다면 while 루프를 돌며 대기를 함.

  S--;  // 자원을 획득함.
}

signal(S) {
  S++;  // 자원을 해제함.
}
```

# Semaphore

**공유 자원을 하나의 스레드만 접근할 수 있게 하는 방식이었던 뮤텍스와 달리 세마포어는 지정된 개수만큼의 스레드가 동시에 접근하도록 동기화할 수 있게 하는 방식 (멀티프로그래밍환경)**

Toilet

Num of Empty rooms

3

Toilet

Num of Empty rooms

0

# Semaphore

```
wait() {
    S−;
    if (S < 0) {
        add this process to Queue;
        sleep();
    }
}

signal() {
    S++;
    if(S <= 0) {
        remove a process p from Queue;
        wakeup(p);
    }
}
```

# Binary Semaphore

## 임계 구역 문제를 해결하는데 사용하며 자원이 하나이기 때문에 뮤텍스로 사용할 수 있는 세마포어

| 바이너리 세마포어 | 뮤텍스 |
| --- | --- |
| 신호 전달 메커니즘 기반으로 동작 | 잠금 메커니즘 기반으로 동작 |
| 현재 스레드보다 우선순위가 높은 스레드가 바이너리 세마포어를 해제하고 잠글 수 있음 | 뮤텍스를 획득한 스레드는 크리티컬 섹션에서 나갈 때만 뮤텍스 해제 가능 |
| 값은 `wait()` , `signal()` 에 따라 변경 | 값이 locked, unlocked 으로 수정 |
| 여러 개의 스레드가 동시에 이진 세마포어를 획득 가능 | 한 번에 하나의 스레드만 뮤텍스를 획득 가능 |
| 소유권이 없다 | 뮤텍스를 소유한 스레드만 잠금을 해제 가능하므로 소유권이 있음 |
| 다른 스레드/프로세스가 잠금 해제가 가능하기 때문에 뮤텍스보다 빠르다. | 획득한 스레드만 잠금 해제가 가능하므로 바이너리 세마포어보다 느리다. |

# Monitor

**임계 구역을 지켜내기 위한 방법인 상호 배제를 프로그램으로 구현한 것이며, 이진 세마포어만 가능**

**Mutex Lock(or Binary Semaphore) + Condition Variable**

## 세마포어에서의 이슈

**wait, signal 명시적 사용**

## Condition Variable

**notify()**
**notifyAll()**
**wait()**

# Synchronized

**자바의 모든 인스턴스는 Monitor를 가지고 있으며 (Object 내부)Monitor를 통해 Thread 동기화**