운영체제론 실습 10주차

정보보호연구실 @ 한양대학교

운영체제론 실습 10주차

1. Mutex
2. Semaphore
3. Producer – Consumer (ver. 2)

프로세스 동기화

- **상호 배제 (Mutual exclusion)**

  - 특정 프로세스가 임계구역에서 실행 중이면,

    다른 프로세스들은 자신들의 임계구역에서 실행될 수 없음

- **진행 (Progress)**

  - 임계구역에 아무 프로세스도 실행되고 있지 않고,

    그들 자신의 임계구역으로 진입하려고 하는 프로세스들이 있다면,

    어느 프로세스가 진입할 수 있는지를 결정해야 되며,

    이 결정은 무한정 연기될 수 없음

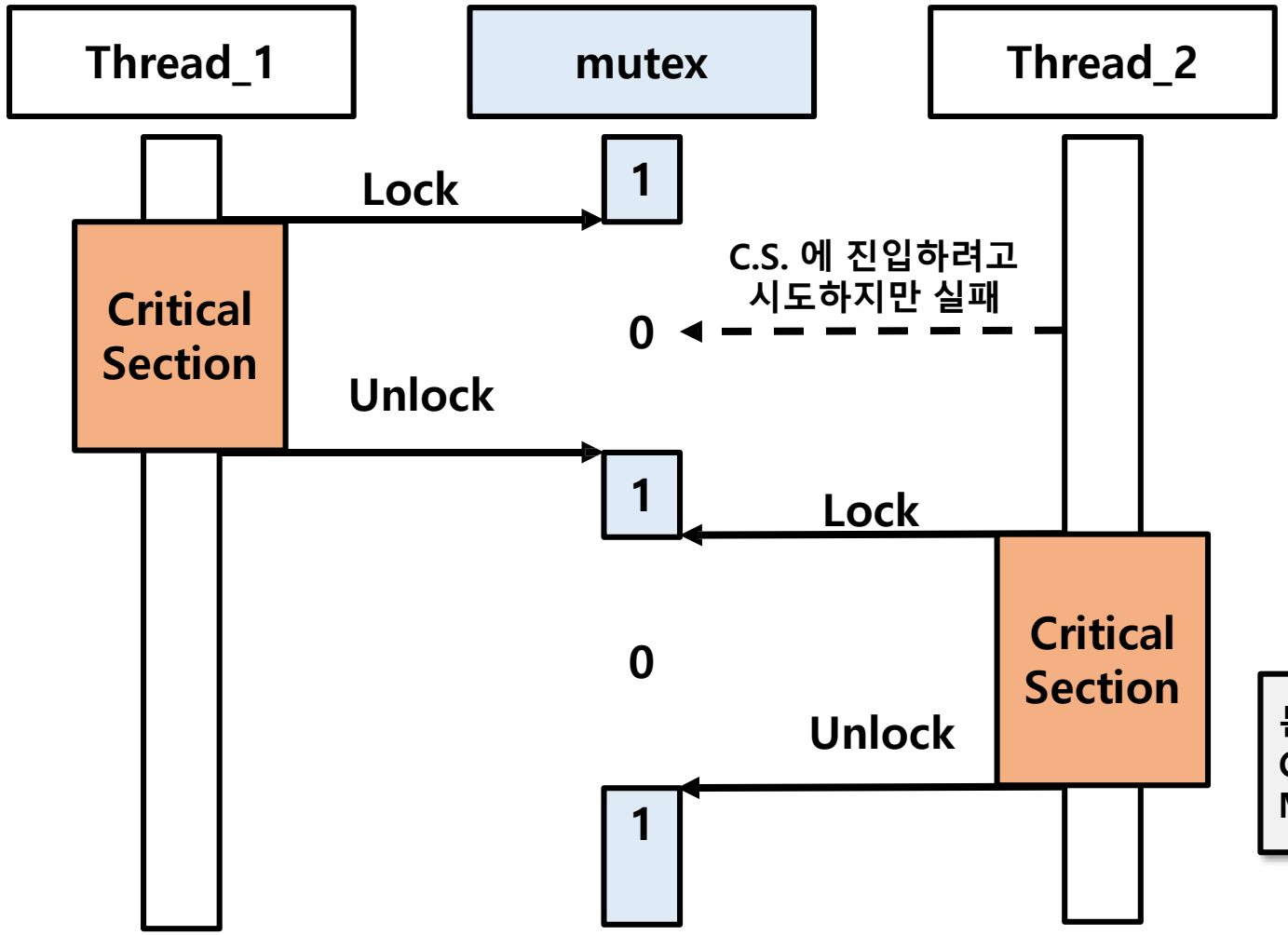
- **한정된 대기 (bounded waiting)**

  - 같은 프로세스가 임계구역을 독점하지 못하게 해야 하며,

    다른 프로세스의 starvation을 막기 위해 한번 하나의 프로세스가 임계 구역에 들어가려고 요청한 이후

    한정된 대기 횟수 안에 해당 프로세스가 임계구역에 들어가야 함

Mutex의 역할: 상호배제(Mutual Exclusion)



본 예제에서
Critical Section: 표준출력(stdout)
Mutex: mutex

Pthread_mutex_init() 사용법

- **설명: Attr로 지정하는 속성을 가지고 mutex를 초기화한다.**
  - Mutex: 초기화 하고자 하는 mutex
  - Attr: mutex 속성 (Null 사용하면 기본값)

```
#include <pthread.h>
int pthread_mutex_init(
        pthread_mutex_t *restrict mutex,
        const pthread_mutexattr_t *restrict attr);
```

- **사용 예제**

```
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);
```

Pthread_mutex_lock() 사용법

- **설명: Mutex를 lock한다.**
  - Mutex: lock하고자 하는 mutex

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

- **사용 예제**

```
pthread_mutex_t mutex;
pthread_mutex_lock(&mutex);
```

Pthread_mutex_unlock() 사용법

- **설명: Mutex를 unlock한다.**
  - Mutex: unlock하고자 하는 mutex

```
#include <pthread.h>
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- **사용 예제**

```
pthread_mutex_t mutex;
pthread_mutex_unlock(&mutex);
```

Pthread_mutex_destroy() 사용법

- **설명: Mutex를 소멸시킴으로 mutex를 통해 할당할 수 있었던 공유 자원을 해방시킴**
  - Mutex: 소멸시킬 mutex

```
#include <pthread.h>
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

- **사용 예제**

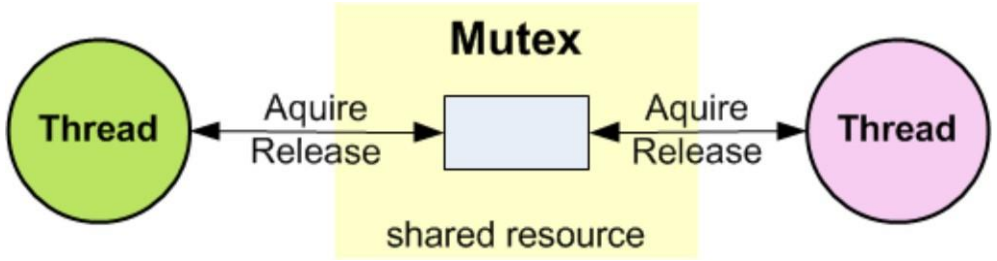
```
pthread_mutex_t mutex;
pthread_mutex_destroy(&mutex);
```

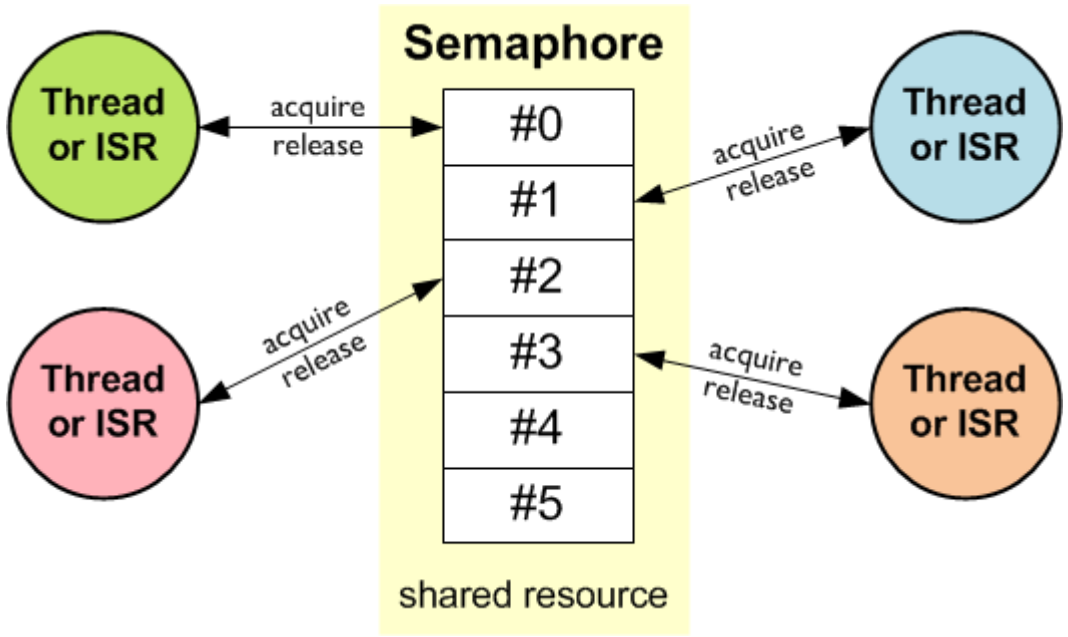
실습1: bounded_waiting.skeleton.c

- **Mutex를 사용하여 출력 문제를 해결하여 보자.**
  - 공유 자원: 표준 출력
  - 각 thread가 내용을 전부 출력하기 전에는 다른 thread가 진입하지 않도록 하는 상호배제를, 어떻게 구현할 것인가?

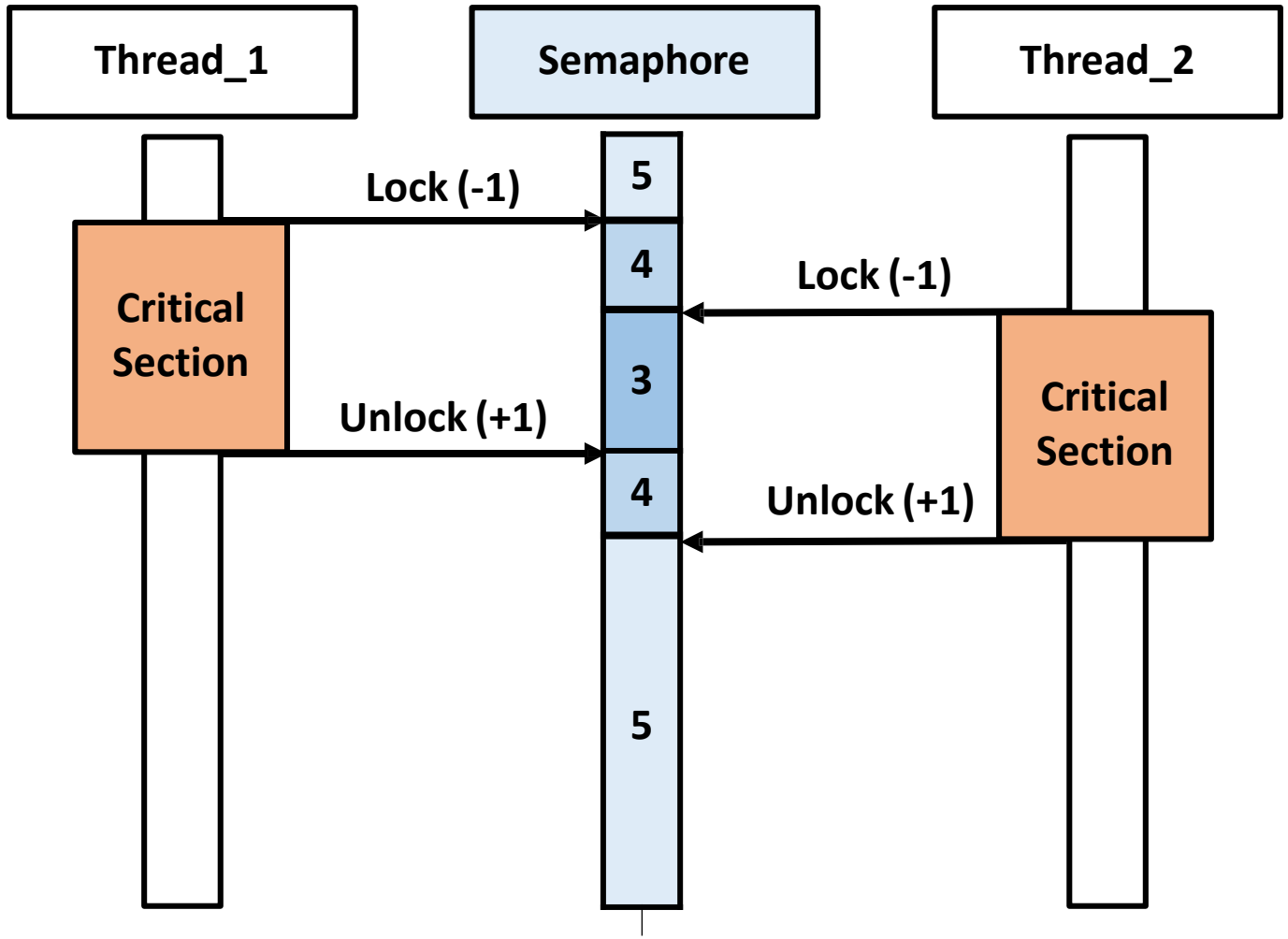
세마포어(Semaphore)

- **세마포어는 공유 자원에 접근하는 것을 제어하기 위한 방법**
  - 정수 값을 가지는 변수를 이용하여 공유자원에 접근할 수 있는 최대 허용치 만큼 접근을 허용함
- **Mutex vs Semaphore**
  - Mutex는 하나의 공유 자원을 보호하기 위해 사용하지만
  - Semaphore는 제한된 일정 개수를 가지는 자원을 보호하고 관리한다.





세마포어(Semaphore)



sem_init() 사용법

- **설명: Value의 값을 가진 semaphore를 초기화한다. (Unnamed Semaphore)**
  - **Sem:** 초기화 하고자 하는 semaphore
  - **Pshared:** 0 값이면 스레드 간 공유되고, 아니면 프로세스 간 공유됨
  - **Value:** semaphore의 초기값을 지정함

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

- **사용 예제**

```
sem_t sem;
sem_init(&sem, 0, 1);
```

sem_open() 사용법

- **설명: Value의 값을 가진 semaphore를 생성 및 초기화한다. (Named Semaphore)**
  - **name:** 생성하고자 하는 semaphore의 **이름**
  - **oflag:** 파일 관련 옵션, **생성에는 O_CREAT 사용**
  - **mode:** 생성되는 파일에 대한 접근 권한 부여, **주로 0666 사용**
  - **value:** semaphore의 초기값을 지정함

```
#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>
sem_t *sem_open(const_char *name, int oflag, mode_t mode, unsigned int value);
```

- **사용 예제**

```
sem_t *sem;
sem = sem_open("mysem", O_CREAT, 0666, 3);
```

sem_wait() 사용법

- **설명: semaphore의 값을 감소시킨다. (즉 lock을 수행함)**
  - **Sem:** lock 하고자 하는 semaphore

```
#include <semaphore.h>  i
nt sem_wait(sem_t *sem);
```

- **사용 예제**

```
sem_wait(&sem); /* unnamed semaphore */
sem_wait(sem); /* named semaphore */
```

sem_post() 사용법

- **설명: semaphore의 값을 1 증가시킨다. (즉 unlock을 수행함)**
  - **Sem:** unlock 하고자 하는 semaphore

```
#include <semaphore.h>  i
nt sem_post(sem_t *sem);
```

- **사용 예제**

```
sem_post(&sem); /*unnamed semaphore*/
sem_post(sem); /*named semaphore*/
```

sem_destroy() 사용법

- **설명: *unnamed* semaphore를 파괴한다.**
  - **Sem:** 파괴하고자 하는 semaphore

```
#include <semaphore.h>
int sem_destroy(sem_t *sem);
```

- **사용 예제**

```
sem_destroy(&sem);
```

sem_unlink() 사용법

- **설명: *named* semaphore를 파괴한다.**
  - **name:** 파괴하고자 하는 semaphore의 **이름** (open때 정한 이름)

```
#include <semaphore.h>
int sem_unlink(const char *name);
```

- **사용 예제**

```
sem_unlink("mysem");
```

예제1: semaphore.c

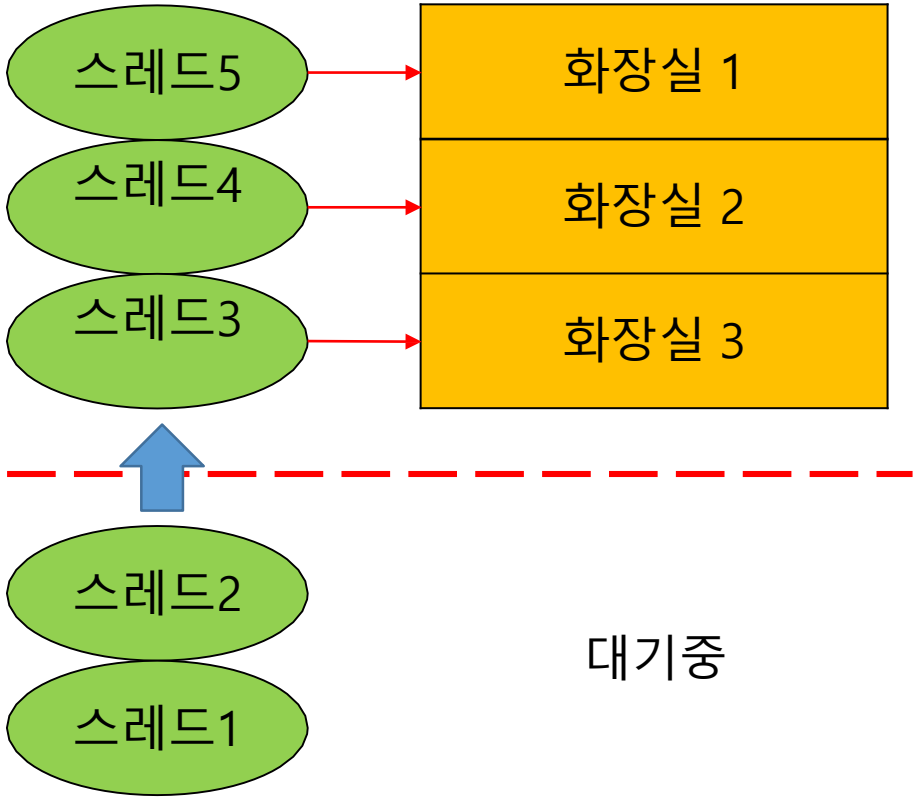
```c
int main()
{

    pthread_t thread_id[N];
    sem_init(&semaphore, 0, 3);
    printf("Semaphore test Start!\n");
    /*
    * 세 개의 자식 스레드를 생성한다.
    */
    for (int i = 0; i < N; i++) {
        pthread_create(&thread_id[i], NULL, toilet, arg+i);
    }

    sleep(5);
    alive = 0;
    /*
    * 자식 스레드가 종료될 때까지 기다린다.
    */
    for (int i = 0; i < N; i++) {
        pthread_join(thread_id[i], NULL);
    }
    printf("모든 스레드가 화장실 이용을 끝냈습니다.\n");
    // 세마포어 객체 소멸
    sem_destroy(&semaphore);

    return 0;
}
```

```c
void *toilet(void *arg)
{
    while (alive) {
        int temp = *((int *)arg);
        sem_wait(&semaphore);
        /*
        * 임계구역 시작: R, B, G
        */
        printf("스레드 %d가 화장실을 이용하기 시작합니다.\n", temp);
        sleep(rand()%2+1); // 화장실 이용하는 시간이라 가정
        printf("스레드 %d가 화장실 이용을 마쳤습니다.\n", temp);
        /*
        * 임계구역 종료
        */
        sem_post(&semaphore);
        usleep(1);
    }
    pthread_exit(NULL);
}
```

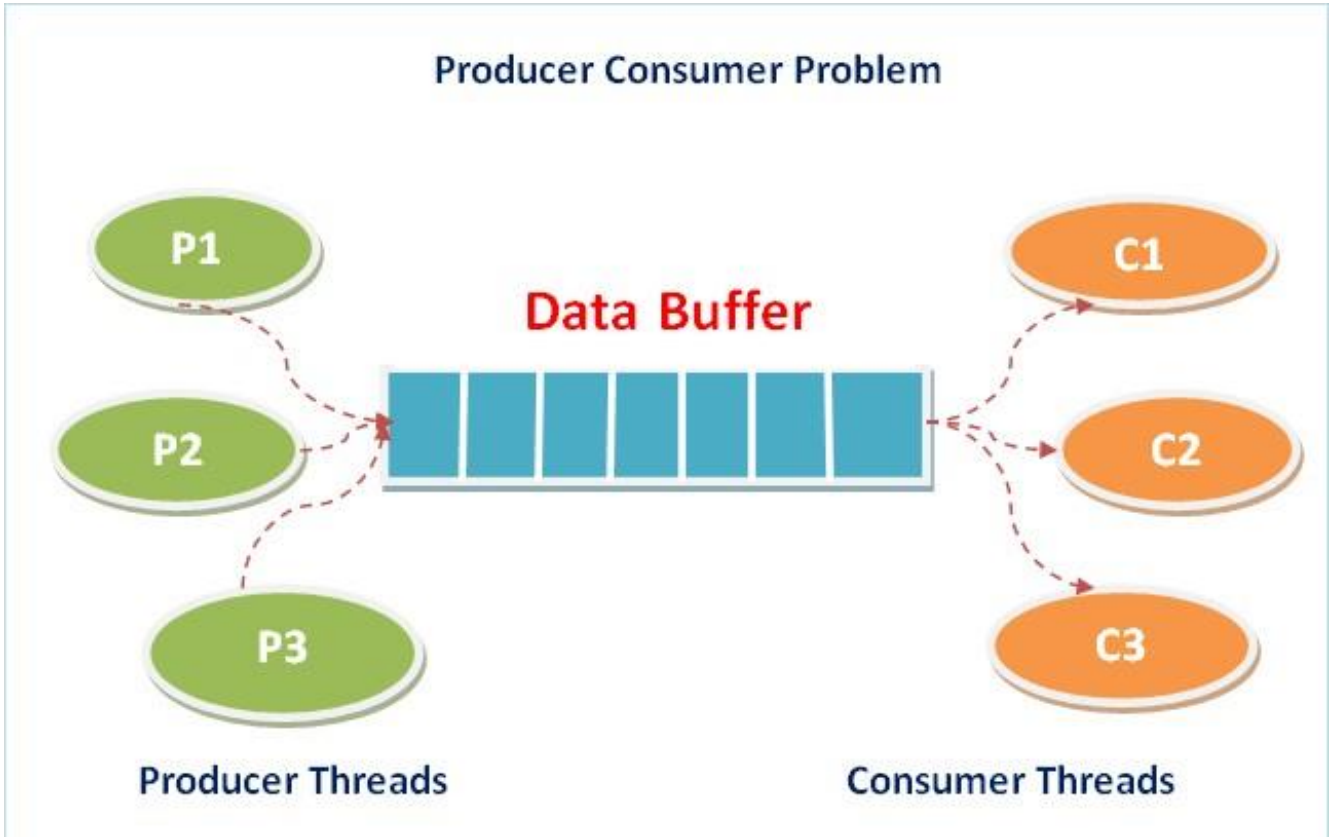
Critical Section

예제1: semaphore.c

```
Semaphore test Start!
스레드 5가 화장실을 이용하기 시작합니다.
스레드 4가 화장실을 이용하기 시작합니다.
스레드 3가 화장실을 이용하기 시작합니다.
스레드 4가 화장실 이용을 마쳤습니다.
스레드 2가 화장실을 이용하기 시작합니다.
스레드 5가 화장실 이용을 마쳤습니다.
스레드 3가 화장실 이용을 마쳤습니다.
스레드 4가 화장실을 이용하기 시작합니다.
스레드 1가 화장실을 이용하기 시작합니다.
스레드 2가 화장실 이용을 마쳤습니다.
스레드 5가 화장실을 이용하기 시작합니다.
스레드 5가 화장실 이용을 마쳤습니다.
스레드 3가 화장실을 이용하기 시작합니다.
스레드 4가 화장실 이용을 마쳤습니다.
스레드 2가 화장실을 이용하기 시작합니다.
스레드 1가 화장실 이용을 마쳤습니다.
스레드 5가 화장실을 이용하기 시작합니다.
스레드 3가 화장실 이용을 마쳤습니다.
스레드 4가 화장실을 이용하기 시작합니다.
스레드 5가 화장실 이용을 마쳤습니다.
스레드 4가 화장실 이용을 마쳤습니다.
스레드 2가 화장실 이용을 마쳤습니다.
스레드 1가 화장실을 이용하기 시작합니다.
스레드 1가 화장실 이용을 마쳤습니다.
모든 스레드가 화장실 이용을 끝냈습니다.
```



실습2: bounded_buffer.skeleton.c (ver. 2)



- **다중 Semaphore를 사용하여 producer-consumer 문제를 해결하여 보자.**
  - 제공한 코드에 적절한 수정을 가해야 함
  - 어떤 코드가 남고, 어떤 코드가 필요없어지는가?
  - printf 함수는 임계 구역 바깥으로
- **Mutex + Semaphore의 조합**
  - 2개의 semaphore 사용
  - 각 semaphore의 역할과 초기값은 무엇인가?
  - 어떤 공유변수를, 누구로부터 보호해야 하는가?