운영체제론 9주차 실습

정보보호연구실 @ 한양대학교

HANYANG UNIVERSITY

운영체제론 실습 9주차

- 프로세스 동기화
- Peterson 솔루션
- 스핀락
- Producer — Consumer 문제

프로세스 동기화

상호 배제 (Mutual Exclusion)

- 특정 프로세스가 임계 구역에서 실행 중일 때, 다른 프로세스는 자신의 임계 구역에 진입할 수 **없음**

진행 (Progress)

- 현재 임계 구역을 실행 중인 프로세스가 없고, 자신의 임계 구역에 들어가려 하는 프로세스들이 있다면 **어느 프로세스가 임계 구역에 진입할지 결정**해야 하며, 이 결정은 **무한히 연기될 수 없음**

한정된 대기 (Bounded Waiting)

- 한 프로세스가 임계 구역을 독점할 수 없도록, 다른 프로세스의 기아 현상(starvation)을 막기 위해

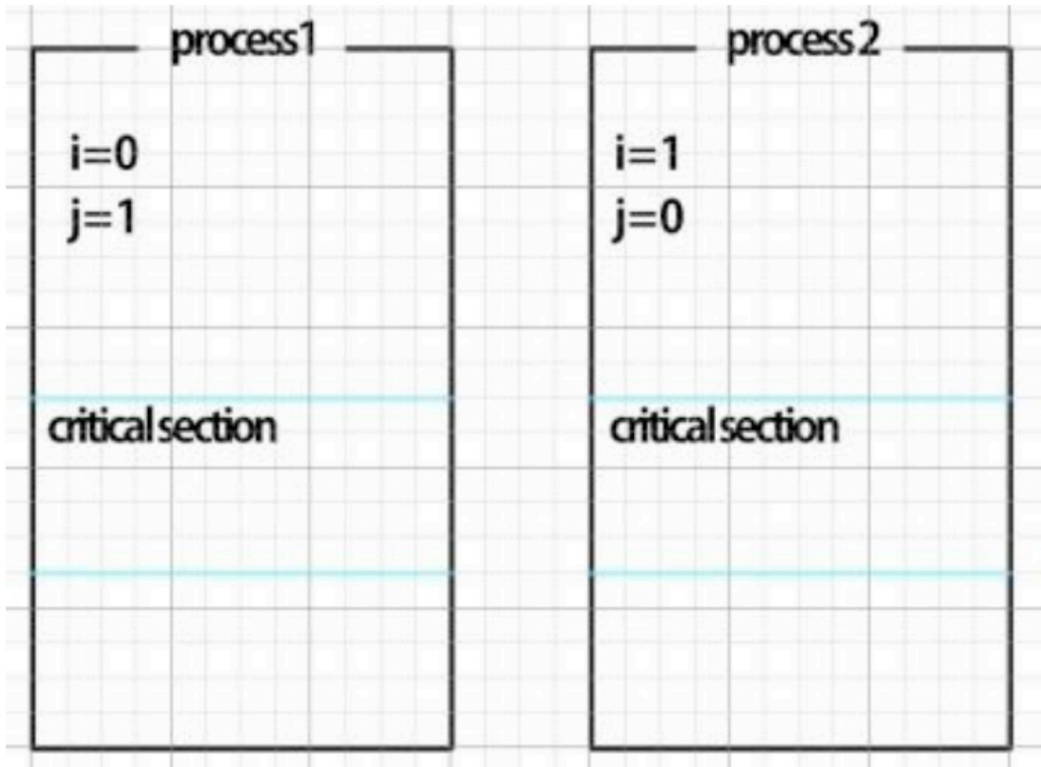
- 한 프로세스가 임계 구역 진입을 요청했다면, **한정된 대기 횟수 안에** 진입해야 함

Peterson 솔루션

peterson.c

```c
void *worker(void *arg)
{
    int i = *(int *)arg;
    int j = (i+1)%2;
    int k;

    while (alive) {
        flag[i] = 1;
        turn = j;
        while (flag[j] && turn == j);
        /*
         * 임계구역 시작: A 또는 B 문자를 한 줄에 40개씩 10줄 출력한다.
         */
        for (k = 0; k < 400; ++k) {
            printf("%c", 'A'+i);
            if ((k+1) % 40 == 0)
                printf("\n");
        }
        /*
         * 임계구역 종료
         */
        flag[i] = 0;
    }
    pthread_exit(NULL);
}
```



- **flag:** 어떤 process가 임계구역으로 진입하고 싶은지 알려줌
- **turn:** 누구의 차례인지 알려줌

스핀락 (Spin Lock)

만일 다른 프로세스가 임계 구역 (lock) 사용 시,
무한한 루프를 돌며 임계 구역의 반환을 기다리는 형태

- context switching을 하지 않고 진입을 시도하는 형태
- **Busy Waiting** 상태
- 임계 구역의 실행 시간이 **매우 짧은 경우** 유용
- 임계 구역의 실행 시간이 길 경우 CPU 시간을 많이 소모하게 됨

compare_and_swap

atomic_compare_exchange_strong (object, expected, desired) -> boolean

- object == expected 라면 object의 값을 desired 로 바꾸고, true 리턴
- object != expected 라면 expected의 값을 object의 값으로 바꾸고, false 리턴
- 값 비교부터 값 변경까지 atomic하게 (uninterrupted)하게 수행
- 다른 방식으로는 test_and_set 함수가 있음
- **atomic_compare_exchange_weak**: object == expected 때 false 리턴이 발생할 수 있으나, 발생 확률이 낮고 상대적으로 빠르기 때문에 **거짓 판단이 크게 중요치 않은 조건문**에 사용

스핀락 구현하기

```
atomic_int lock = 0;
int expected = 0;
...
while(!atomic_compare_exchange_weak(&lock, &expected, 1))
    expected = 0;
```

함수로 인해 expected의 값이 1로 바뀐 상황
expected를 0로 되돌림

while 루프가 돌아가는 조건: 함수의 반환값이 **false**
= lock과 expected(0)의 값이 다르다
= 현재 lock의 값은 1이다
= 다른 프로세스가 임계 구역 실행 중

루프를 빠져나왔다: 함수의 반환값이 **true**
= lock과 expected(0)의 값이 같음
= lock의 값이 0이었으며, 현재 1로 바뀜
(임계 구역에 진입하기 위해 다른 프로세스 막기)

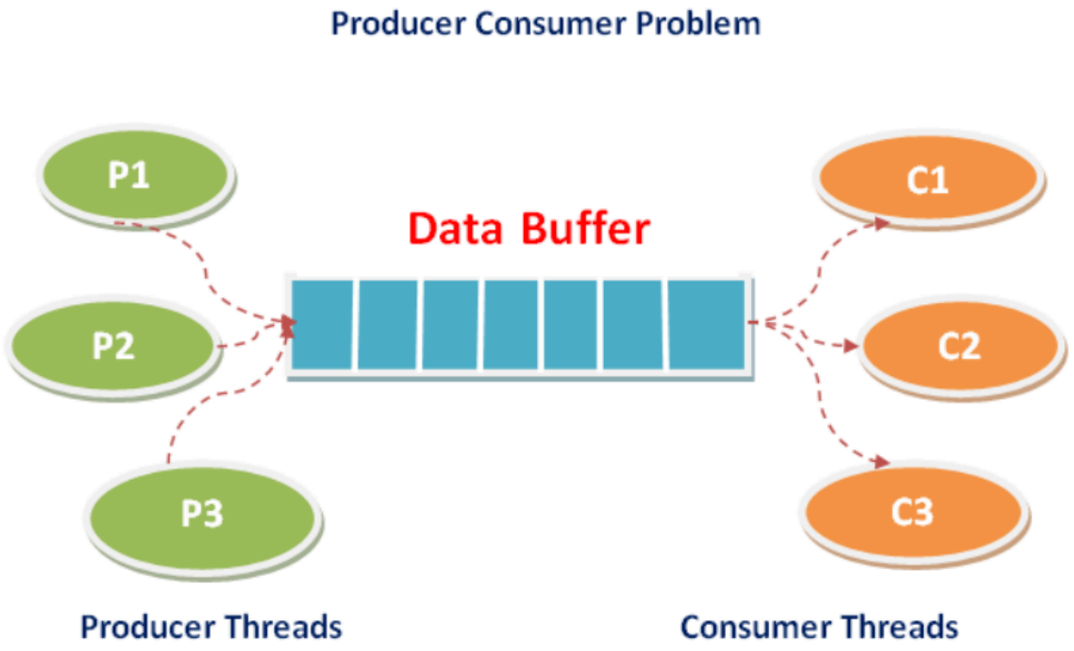
Producer—Consumer 문제

Bounded_buffer 문제 중 하나

- 생산자는 buffer에 만든 아이템을 저장하고, 소비자는 저장된 아이템을 꺼내온다.

- 꽉 찬 buffer에 아이템을 더 저장하거나, 빈 buffer에서 아이템을 꺼내오는 것은 불가능하다.

- 동기화를 하지 않은 채로 실행할 경우, 두 생산자가 같은 자리에 아이템을 저장하거나
  두 소비자가 같은 아이템을 가져오는 상황이 발생할 수 있다.



오늘의 실습

스핀락을 사용하여 두 프로그램의 프로세스 동기화 시도하기

- bound_nosync.c 파일을 이론 수업 강의 자료의 솔루션을 참고하여 동기화
  - **색색의 문자열들이 각자 섞이지 않고 출력되도록** 해 보자.
- bound_buffer_nosync.c 파일 동기화
  - **생산자와 소비자가 문제 없이 아이템을 주고 받을수 있도록** 해 보자.
  - 생산자: 임계 구역에서 **아이템을 버퍼에 추가**한다
  - 소비자: 임계 구역에서 **아이템을 버퍼에서 꺼내**온다