운영체제 과제#2

#1. 생산자-소비자로 구성된 응용프로그램 만들기

```
/Users/songsubin/Desktop/as
songsubin@songsubin-ui-Macl
on ; exit;
producer : wrote 0
        consumer : read 0
producer : wrote 1
        consumer : read 1
producer : wrote 2
producer : wrote 3
        consumer : read 2
producer : wrote 4
producer : wrote 5
        consumer : read 3
        consumer : read 4
        consumer : read 5
        consumer : read 6
producer : wrote 6
        consumer : read 7
        consumer : read 8
        consumer : read 9
producer : wrote 7
producer : wrote 8
producer : wrote 9
```

```
songsubin@songsubin-ui-MacBookA
on ; exit;
producer : wrote 0
        consumer : read 0
producer : wrote 1
        consumer : read 1
        consumer : read 2
producer : wrote 2
        consumer : read 3
producer : wrote 3
        consumer : read 4
        consumer : read 5
producer : wrote 4
        consumer : read 6
producer : wrote 5
producer : wrote 6
producer : wrote 7
        consumer : read 7
producer : wrote 8
        consumer : read 8
producer : wrote 9
        consumer : read 9
```

```
// write n into the shared memory
void mywrite(int n) {
   /* [Write here] */
   sem_wait(&semWrite);
   pthread_mutex_lock(&critical_section);   // 뮤텍스 잠금
   queue[wptr] = n; //버퍼에 데이터 쓰기
   wptr = (wptr + 1) % N_COUNTER; // write pointer 이동
   pthread_mutex_unlock(&critical_section); // 뮤텍스 해제
   sem_post(&semRead); //공유 버퍼에 새로운 데이터가 추가된거 소비자 스레드한테 알리기
}
```

```
int myread() {
   /* [Write here] */
   sem_wait(&semRead);
   pthread_mutex_lock(&critical_section); // 뮤텍스 잠금

   int n = queue[rptr]; //rptr 이 가리키는 위치에서 데이터 읽어옴
   rptr = (rptr + 1) % N_COUNTER; //읽기 작업이 끝나고 다음 위치로 이동

   pthread_mutex_unlock(&critical_section);
   sem_post(&semWrite); // 빈공간 사용이 가능하다는 신호
```

```
    return n;
}
```

```
/* [Write here] */
    sem_init(&semWrite, 0, N_COUNTER); // semWrite 초기화
    sem_init(&semRead, 0, 0); // 0 으로 semRead 초기화
```

#2. 소프트웨어로 문을 만드는 방법

1. 소프트웨어 기반 동기화 방식(알고리즘)과 간략한 설명

소프트웨어 기반 동기화 방식에는 많은 종류가 있는데, 대표적인 방식에는 뮤텍스, 세미포어, 모니터, 스핀락, 피터슨의 알고리즘 등이 있다.

뮤텍스는 하나의 쓰레드와 프로세스에 의해 소유가 가능한 키를 기반으로 한 상호 배제 기법이며, 뮤텍스는 임계 구역에 대한 접근을 동기화하여 여러 스레드가 동시에 접근하지 못하도록 하는 특징이 있다.

세미포어는 시그널 메커니즘을(현재 공유자원에 접근이 가능한 쓰레드이며, 프로세스의 수를 나타내는 값을 두어) 사용하여 상호 배제를 달성하는 기법이다. 이는 특정 자원에 접근할 수 있는 스레드 또는 프로세스의 수를 제어하는 값을 유지한다. 일반적으로 두 가지 유형이 있는데, 카운팅 세마포어와 이진 세마포어가 있다.

모니터는 Mutex 와 세미포어에서 발생하는 오류를 해결하는 고급 언어 구조이다. 즉, 더 높은 단계의 동기화 형태이며, monitor 는 동기화 되고 있는 데이터, monitor lock 와 하나 이상의 조건 변수를 포함한다.

스핀락(spinlock)은 공유 자원에 대한 접근을 제어하기 위한 동기화 메커니즘이며, 바쁜 대기와 컨텍스트 스위칭 오버헤드가 낮아 짧은 시간 동안 자원을 보호할 때 유용하다.

피터슨의 알고리즘은 두 개의 프로세스 간의 상호 배제를 보장하기 위해 설계되었기 때문에, 프로세스가 두 개인 경우에만 적용이 가능하다. 피터슨 알고리즘은 상호배제, 진행조건, 한정된 대기 조건을 만족하는 알고리즘이다.

2. 내가 선택한 동기화 방식과 구현

내가 선택한 동기화 방식은 스핀락이다. 스핀락은 간단하면서도 효율적인 동기화 방식인데, 특히 짧은 시간 동안 임계 구역을 보호할 때 유용하다. 또한 스핀락은 바쁜 대기를 사용하여 컨텍스트 스위칭 오버헤드가 낮아 성능이 중요한 상황에서 효율적이라는 장점이 존재한다. 하지만 수업시간에 스핀락은 단일 CPU 를 가진 운영체제에서 비효율적이라는 점을 배웠고, 나는 스핀락이 얼마나 비효율적인지 알고 싶어서 이 동기화 방식을 선택하게 되었다.

[둘의 성능 비교를 위한 코드]

```
// 실행시간 측정
double measure_execution_time() {
    pthread_t t[2]; // thread structure
    struct timespec start, end;
```

```
    // 현재 시간을 CLOCK_MONOTONIC 기준으로 가지고 와서 start 에 저장한다.
    clock_gettime(CLOCK_MONOTONIC, &start);

    // 생산자와 소비자 스레드 생성
    pthread_create(&t[0], NULL, producer, NULL);
    pthread_create(&t[1], NULL, consumer, NULL);

    for(int i=0; i<2; i++)
        pthread_join(t[i], NULL);

    //end 에 서장
    clock_gettime(CLOCK_MONOTONIC, &end);

    // 시작 시간과 끝 시간의 차이를 계산
    long seconds = end.tv_sec - start.tv_sec;
    long nanoseconds = end.tv_nsec - start.tv_nsec;
    return seconds + nanoseconds * 1e-9;
}
/*설명 */
struct timespec start, end;
```

시작시간과 끝시간을 저장하는 구조체이다.

```
clock_gettime(CLOCK_MONOTONIC, &start);
```

현재 시간을 CLOCK_MONOTONIC 기준으로 측정하여 start 에 저장한다.

```
pthread_create(&t[0], NULL, producer, NULL);
pthread_create(&t[1], NULL, consumer, NULL);
```

생산자-소비자 thread 생성

```
for(int i=0; i<2; i++)
        pthread_join(t[i], NULL);
```

두 개의 thread 가 종료될 때까지 대기함

[original_mutex 실행결과] **[spin_lock 실행결과]**

```
producer : wrote 4
producer : wrote 5
        consumer : read 2
producer : wrote 6
        consumer : read 3
        consumer : read 4
producer : wrote 7
producer : wrote 8
        consumer : read 5
        consumer : read 6
        consumer : read 7
producer : wrote 9
        consumer : read 8
        consumer : read 9
producer : wrote 0
        consumer : read 0
producer : wrote 1
        consumer : read 1
        consumer : read 8
producer : wrote 2
producer : wrote 3
producer : wrote 4
producer : wrote 5
        consumer : read 3
        consumer : read 4
producer : wrote 6
producer : wrote 7
        consumer : read 5
producer : wrote 8
producer : wrote 9
        consumer : read 6
        consumer : read 7
        consumer : read 8
        consumer : read 9
producer : wrote 0
        consumer : read 0
producer : wrote 1
        consumer : read 1
producer : wrote 2
producer : wrote 3
        consumer : read 2
producer : wrote 4
        consumer : read 3
        consumer : read 4
        consumer : read 1
producer : wrote 5
        consumer : read 2
producer : wrote 6
        consumer : read 3
producer : wrote 7
        consumer : read 4
        consumer : read 5
producer : wrote 8
producer : wrote 9
평균시간 : 0.55792 seconds.
```

```
producer : wrote 3
producer : wrote 4
        consumer : read 3
producer : wrote 5
        consumer : read 4
        consumer : read 5
producer : wrote 6
producer : wrote 7
        consumer : read 6
        consumer : read 7
producer : wrote 8
        consumer : read 8
        consumer : read 5
producer : wrote 9
producer : wrote 0
        consumer : read 0
producer : wrote 1
        consumer : read 1
        consumer : read 8
producer : wrote 2
producer : wrote 3
producer : wrote 4
        consumer : read 3
producer : wrote 5
producer : wrote 6
        consumer : read 4
        consumer : read 5
producer : wrote 7
        consumer : read 6
        consumer : read 7
        consumer : read 4
        consumer : read 5
producer : wrote 8
producer : wrote 9
producer : wrote 0
        consumer : read 0
producer : wrote 1
        consumer : read 1
producer : wrote 2
producer : wrote 3
        consumer : read 2
producer : wrote 4
        consumer : read 3
producer : wrote 5
producer : wrote 6
        consumer : read 4
producer : wrote 7
        consumer : read 5
producer : wrote 8
        consumer : read 6
producer : wrote 9
        consumer : read 7
        consumer : read 8
        consumer : read 9
평균시간 : 0.56602 seconds.
```

실행결과를 보기 위해서 평균시간을 구했는데 스핀락이랑 실제 pthread_mutex 차이가 크지 않아서 의아했다. 수업시간에 배운 내용에 따르면 스핀락은 단일 CPU 를 가진 운영체제에서 비효율적이라는 점을 배웠는데.. .. 결과를 보고서 내가 잘못 구현했나 생각하고 뭔가 잘못 구현을 한 것이라고 생각했다. 사실 아직도 맞는 결과인지는 모르겠다. 구글링을 한 결과 스핀락은 임계구간이 긴 구역이 특히 비효율적이라는 결과를 봤는데 내가 구현한 건 짧은 임계구간이라서 비슷한 건가 싶었다. ㅠㅠ...

#3. 소프트웨어로 문을 만드는 방법

3-1) page.c 코드를 컴파일하고 실행시켜보세요. 실행시킬때 time 명령어와 함께 실행하여 실행 시간을 측정해보세요 (e.g., time ./a.out)

```
songsubin@songsubin-ui-MacBookAir real % gcc -o page page.c -lpthread
songsubin@songsubin-ui-MacBookAir real % time ./page
./page  0.40s user 0.01s system 47% cpu 0.842 total
```

3-2) 컴파일된 파일은 pagesize 변수를 인자값으로 받습니다. 값을 변경해가며 실행시간의 변화를 확인해보세요 (e.g., time ./a.out 1024)

```
songsubin@songsubin-ui-MacBookAir real % time ./page
./page  0.40s user 0.01s system 47% cpu 0.842 total
songsubin@songsubin-ui-MacBookAir real % time ./page 1024
./page 1024  0.40s user 0.00s system 98% cpu 0.407 total
songsubin@songsubin-ui-MacBookAir real % time ./page 2048
./page 2048  0.53s user 0.00s system 99% cpu 0.538 total
songsubin@songsubin-ui-MacBookAir real % time ./page 4096
./page 4096  0.67s user 0.00s system 99% cpu 0.677 total
songsubin@songsubin-ui-MacBookAir real % time ./page 8192
./page 8192  0.97s user 0.01s system 98% cpu 0.989 total
songsubin@songsubin-ui-MacBookAir real % time ./page 100000
./page 100000  0.14s user 0.00s system 98% cpu 0.143 total
songsubin@songsubin-ui-MacBookAir real % time ./page 6144
./page 6144  0.66s user 0.00s system 99% cpu 0.671 total
songsubin@songsubin-ui-MacBookAir real % time ./page 7168
./page 7168  0.38s user 0.00s system 98% cpu 0.389 total
```

3-3) 입력 값이 특정 값에 이르면 (i.e., 실제 page size) 갑자기 실행시간이 확 증가합니다!이를 통해 여러분 컴퓨터의 페이지 크기를 확인해보세요. 혹시 시간 값의 차이가 잘 보이지 않거나, 실행에 이상이 있으시면 #define된 각종 값들을 변경해보세요.

- ➔ 8192로 예상한다.
- ➔ 4096에서 8192로 넘어갈 때 시간이 크게 증가했기 때문이다.

3-4) 리눅스(or macOS)에서 여러분 컴퓨터에 설정된 페이지 크기를 확인하는 명령어가 무엇인지 찾아보시고, 여러분이 찾아낸 값과 일치하는지 비교해보세요. 거꾸로 실제 값을 확인 해놓고 값을 찾아보는 것도 좋은 방법이 되겠네요. 예를 들어 실제 값이 1000이라면, 999, 1000, 1001 이렇게 실행시키면서요!

- ➔ 페이지 크기를 확인하는 명령어 : getconf
- ➔
```
songsubin@songsubin-ui-MacBookAir ~ % getconf PAGE_SIZE
16384
```
- ➔
```
songsubin@songsubin-ui-MacBookAir real % time ./page 16384
./page 16384  0.73s user 0.00s system 99% cpu 0.739 total
```

뭔가 결과가 의아했다.. 페이지 크기를 확인하기 이전에 위에서 답했듯이, 8192가 맞는 줄 알았다. 2048에서 4096으로 했을 때, CPU 시간 변화가 크지 않았다. 근데 4096에서 8192로 넘어갔을 때 실행시간 변화가 큰 것을 확인할 수 있었고, 8192가 실제 페이지 사이즈 인줄 알았다. 더 확실하게 하기 위해서, 6144와 7168로 바이트를 주면서 좁혀갔다..근데 왜 6144에서 7168로 가면 CPU 시간이 줄어드는지 모르겠다. 그리고 실제로 페이지 크기를 확인하기 위해서 터미널에 명령어를 줬는데, 16384가 나와서..내가 예상한 값과 전혀 다르다는 것을 확인했고 실제로 page.c에서 16384를 줬는데.. 8192를 줬을 때보다 시간이 적게 나와서 혼란스러운 결과였다.