운영체제 및 실습

분 반	00
학 과	컴퓨터공학과
학 번	201502049
이 름	노 효 근

1. 문제 해결 방법

1) 소스코드

- producer

```
void *producer(void *args){
        int i;
        int producerID = ++producerCount;
        int item;

        for(i = 0; i < PRODUCE_TIME; i++){
                item = (random() % 100) + producerID;
                usleep(item);

                //생산 세마포어 변수 produce_ok를 함수를 사용해 1 감소
                sem_wait(&produce_ok);
                /*생산 하는 도중에 소비자나 다른 생산자가 버퍼에 접근하지 못하도록
                  버퍼 세마포어 변수 buffer_lock을 함수를 사용해 1 감소*/
                sem_wait(&buffer_lock);
                if(insertItem(item) == 0)
                        printf("%s[PRODUCER_ERROR]insertItem Failed%s\n", T_RED, T_DEFAULT);
                else
                        printf("%sproducer %d add CQ %d%s\n", T_YELLOW, producerID, item, T_DEFAULT);
                /*생산이 끝나면 소비자나 다른 생산자가 버퍼에 접근할 수 있도록
                  버퍼 세마포어 변수 buffer_lock을 함수를 사용해 1 증가*/
                sem_post(&buffer_lock);
                /*소비자가 소비할 수 있도록
                소비 세마포어변수 consume_ok를 함수를 사용해 1 증가*/
                sem_post(&consume_ok);

        }
}
```

sem_wait(&s) 함수는 semWait(s)와 동일하고, 세마포어 값을 1 감소시키고 세마포어 s의 값이 0 이하면 프로세스 쓰레드가 임계영역으로 들어가지 못하게 하는 역할을 한다. item을 삽입하기 전에 세마포어 변수 produce_ok가 1 감소해야하므로 sem_wait(&produce_ok);를 해주고, 생산하는 도중에 소비자나 다른 생산자가 버퍼에 접근하지 못하도록 해야하므로sem_wait(&buffer_lock);을 해준다.
생산이 끝나면 소비자나 다른 생산자가 버퍼에 접근할 수 있도록 하기 위해 buffer_lock을 1증가시켜야 하므로 sem_post(&buffer_lock)을 해준다. sem_post() 함수는 signal(s)와 동등한 역할을 하고 세마포어의 잠금을 해제한다.
세마포어 변수 consume-_ok의 값을 1 증가시켜야 하므로 sem_post(&consume_ok)를 해주면 된다.

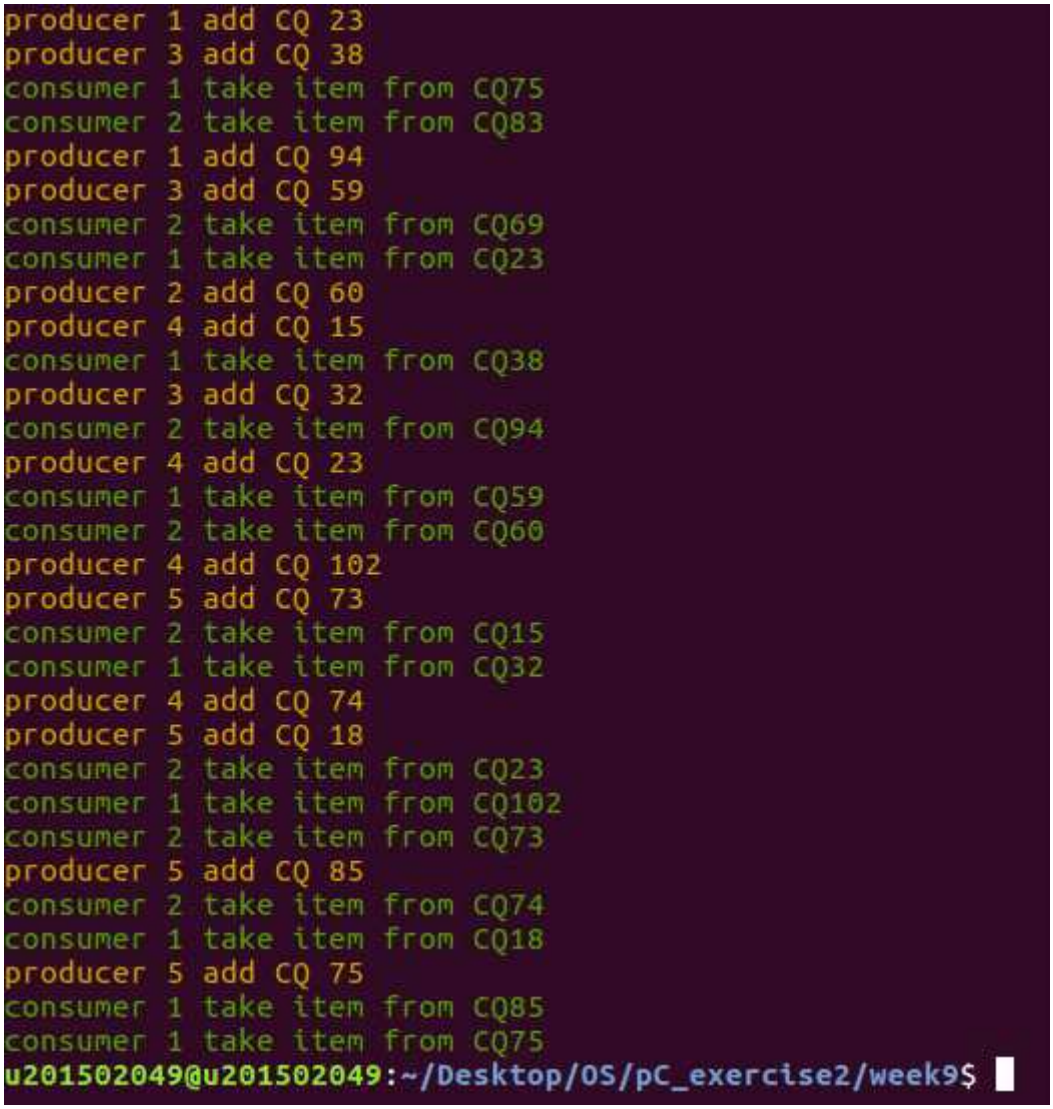
- consumer

```
void *consumer(void *args){
        int i;
        int consumerID = ++consumerCount;
        int time;

        for(i = 0; i < CONSUME_TIME; i++){
                int item;
                time = (random() % 100) + consumerID;
                usleep(time);

                //소비 세마포어 변수 consume_ok를 함수를 사용해 1 감소
                sem_wait(&consume_ok);
                /*생산자와 동일, 생산자나 다른 소비자가 버퍼에 접근하지 못하도록
                버퍼 세마포어 변수 buffer_lock을 함수를 사용해 1 감소*/
                sem_wait(&buffer_lock);
                if((item = takeItem()) == -1)
                        printf("%s[CONSUMER_ERROR]takeItem Failed%s\n", T_RED, T_DEFAULT);
                else
                        printf("%sconsumer %d take item from CQ%d%s\n", T_GREEN, consumerID, item, T_DEFAULT);
                /*생산자와 동일, 생산자나 다른 소비자가 버퍼에 접근할 수 있도록
                버퍼 세마포어 변수 buffer_lock을 함수를 사용해 1 증가*/
                sem_post(&buffer_lock);
                /*소비자가 버퍼 안의 물건을 하나 꺼내 소비했으므로
                생산 세마포어변수 produce_ok를 함수를 사용해 1 증가*/
                sem_post(&produce_ok);
        }
}
```

produce함수와 코드는 동일하고 함수 안에 들어가는 인자만 변경하였다.
sem_wait(&s) 함수는 semWait(s)와 동일하고, 세마포어 값을 1 감소시키고 세마포어 s의 값이 0 이하면 프로세스 쓰레드가 임계영역으로 들어가지 못하게 하는 역할을 한다.item을 삽입하기 전에 consume_ok의 값을 1 증가시켜야하므로 sem_wait(&consume_ok) 를 해주고 생산자나 다른 소비자가 버퍼에 접근하지 못하도록 임계영역을 설정해줘야 하므로 sem_wait(&buffer_lock)을 해준다.
item을 빼고 난 다음에는 소비자나 다른 생산자가 버퍼에 접근할 수 있도록 하기 위해 sem_post(&buffer_lock) 함수를 사용하여 buffer_lock을 1 증가시킨다.
sem_post() 함수는 signal(s)와 동한 역할을 하고 세마포어의 잠금을 해제한다.
마지막으로 소비자가 버퍼 안의 물건을 하나 꺼내 소비했으므로 생산 세마포어 변수 produce_ok를 1증가시키기 위해 sem_post(&produce_ok)를 해주면 된다.
이 함수가 실행된 이후에도 세마포어의 값이 0 이하면 임계영역에 들어갈 수 없다.

2) 실행화면

```
u201502049@u201502049:~/Desktop/OS/pC_exercise2/week9$ ls
cq.c  cq.h.gch  main    main.o    semaphore.c  semaphore.h.gch
cq.h  cq.o      main.c  Makefile  semaphore.h  semaphore.o
u201502049@u201502049:~/Desktop/OS/pC_exercise2/week9$ ./main
producer 1 add CQ 78
consumer 1 take item from CQ78
producer 2 add CQ 17
consumer 2 take item from CQ17
producer 3 add CQ 96
producer 4 add CQ 39
producer 5 add CQ 91
consumer 1 take item from CQ96
consumer 2 take item from CQ39
producer 1 add CQ 93
producer 2 add CQ 23
producer 3 add CQ 30
consumer 2 take item from CQ91
producer 4 add CQ 94
producer 5 add CQ 64
consumer 1 take item from CQ93
consumer 2 take item from CQ23
producer 1 add CQ 41
producer 2 add CQ 28
consumer 1 take item from CQ30
consumer 2 take item from CQ94
producer 2 add CQ 32
producer 3 add CQ 75
consumer 2 take item from CQ64
consumer 1 take item from CQ41
producer 1 add CQ 83
producer 2 add CQ 69
consumer 1 take item from CQ28
consumer 2 take item from CQ32
producer 1 add CQ 23
producer 3 add CQ 38
consumer 1 take item from CQ75
consumer 2 take item from CQ83
```


producer 1 add CQ 23
producer 3 add CQ 38
consumer 1 take item from CQ75
consumer 2 take item from CQ83
producer 1 add CQ 94
producer 3 add CQ 59
consumer 2 take item from CQ69
consumer 1 take item from CQ23
producer 2 add CQ 60
producer 4 add CQ 15
consumer 1 take item from CQ38
producer 3 add CQ 32
consumer 2 take item from CQ94
producer 4 add CQ 23
consumer 1 take item from CQ59
consumer 2 take item from CQ60
producer 4 add CQ 102
producer 5 add CQ 73
consumer 2 take item from CQ15
consumer 1 take item from CQ32
producer 4 add CQ 74
producer 5 add CQ 18
consumer 2 take item from CQ23
consumer 1 take item from CQ102
consumer 2 take item from CQ73
producer 5 add CQ 85
consumer 2 take item from CQ74
consumer 1 take item from CQ18
producer 5 add CQ 75
consumer 1 take item from CQ85
consumer 1 take item from CQ75
u201502049@u201502049:~/Desktop/OS/pC_exercise2/week9$