```c
int produce(MessageBuffer **buffer, int sender_id, char *data)
{
    if (is_full(**buffer))
    {
        printf("full!\n\n");
        return -1;
    }

    if (strlen(data) > 100)
    {
        printf("len(data) > 100\n\n");
        return -1;
    }

    /*---------------------------------------*/
    /* TODO 3 : produce message              */
    s_wait();
    strcpy((*buffer)->messages[(*buffer)->in].data, (char *)data);
    (*buffer)->messages[(*buffer)->in].sender_id = sender_id;
    (*buffer)->in = ((*buffer)->in + 1) % BUFFER_SIZE;
    s_quit();
    /* TODO 3 : END                          */
    /*---------------------------------------*/

    printf("produce message\n");
    return 0;
}
```

s_wait() & s_quit()의 사용위치는 Produce 함수에서 다음과 같습니다.

두개의 프로세스 (produce & consumer)가 서로 공동으로 접근하는 변수 값은 버퍼에 해당하는 부분으로, 한쪽에서 값을 넣어주면, 다른 한쪽에서는 값을 사용하는 방식으로 활용이 이뤄져야 합니다.

특히 현재 실습에서 세마포를 사용하는 가장 중요한 목적은 프로세스들 사이에서 경쟁관계(race condition) 없이 잘 사용하는 것입니다.

공유자원의 경우 다른 프로세스가 동시에 접근을 한다면, 어느 프로세스에 의해 변경된 자료가 나중에 수정된 자료인지 모르게 되는 문제가 생기며, 이로 인해 데이터의 각종 오류가 산재하게 됩니다. 그래서 세마포어 라는 일종의 변수를 두어 제어를 하게 되는 것인데요.

(실습 영상에서 보여준 num의 값이 세마포를 사용하지 않았을 때 race condition으로 뒤엉킨 사례를 해결해 나가는 목적)

```c
int consume(MessageBuffer **buffer, Message **message)
{

    if (is_empty(**buffer))
    {
        return -1;
    }
    /*--------------------------------------*/
    /* TODO 4 : consume message             */
    memory_segment = shmat(shmid, NULL, 0);
    *message = (Message *)memory_segment;
    s_wait();
    strcpy((*message)->data, (*buffer)->messages[(*buffer)->out].data);
    (*message)->sender_id = (*buffer)->messages[(*buffer)->out].sender_id;
    (*buffer)->out = ((*buffer)->out + 1) % BUFFER_SIZE;
    s_quit();

    /* TODO 4 : END                         */
    /*--------------------------------------*/
    return 0;
}
```

s_wait() & s_quit()의 사용위치는 Consumer 함수에서 다음과 같습니다.

사용 목적은 producer 때와 동일하며, 마찬가지로 race condition을 방지하는 형태로 활용됩니다.