OS01 4주차 과제 - 201702081

- Producer / Consumer Problem ~ Mutex & Conditional Variables

1. 코드 이미지 및 설명

```
/* You need to make some variables (mutex, condition variables). */
/* Reference below variables, functions. */
pthread_cond_t fill = PTHREAD_COND_INITIALIZER;
pthread_cond_t use = PTHREAD_COND_INITIALIZER;
```

fill : 버퍼 내에 item이 추가되는걸 기다림 / 추가되었음을 알림 에 사용할 Condition Variable

use : 버퍼 내에 공간이 생기는걸 기다림 / 생겼음을 알림 에 사용할 Condition Variable

```
/* homework */
// return buffer's value using get_ptr if successful,
// otherwise, -1
int get()
{
    // 빼갈 게 없으면, item이 생길때까지 기다림
    while(count <= 0)
        pthread_cond_wait(&fill, &m_id);

    // 빼가고 카운터 감소, get 포인터 증가
    if(count > 0) {
        int got = buffer[get_ptr];
        count--;
        get_ptr = (get_ptr + 1) % MAX;
        pthread_cond_signal(&use);          // 빈공간이 생겼음을 알림
        return got;
    }

    // 빼갈 게 없음
    else return -1;
}
```

get 에서는, 버퍼에 뺄 item이 없을 때(count <= 0) 추가될 때까지 wait를 통해 기다리고,

추가되었다면 그것을 뺀 다음에 카운터를 감소, 탐색 인덱스를 하나 증가시킨다. Circular Buffer이므로, 모듈러 연산을 통해 인덱스를 계산해준다.

이후에는 빈 공간이 생겼음을 use 조건변수를 매개하여 알린다.

```c
/* homework */
// return buffer's value using put_ptr if successful,
// otherwise, -1
int put(int val)
{
    // 꽉차서 못넣으면, 빈공간 생길때까지 기다림
    while(count >= MAX)
        pthread_cond_wait(&use, &m_id);

    // 넣고 카운터 증가, put 포인터 증가
    if(count < MAX) {
        buffer[put_ptr] = val;
        int stored = buffer[put_ptr];
        count++;
        put_ptr = (put_ptr + 1) % MAX;
        pthread_cond_signal(&fill);         // item을 넣었음을 알림
        return stored;
    }

    // 꽉차서 못넣음
    else return -1;
}
```

put 에서는, 버퍼에 item이 꽉 차서 더 넣을 수가 없을 때(count >= MAX) 비워질 때까지 wait를 통해 기다리고,

비워졌다면 현재 인덱스에 데이터를 넣은 다음에 카운터를 증가, 탐색 인덱스를 하나 증가시킨다. Circular Buffer이므로, 모듈러 연산을 통해 인덱스를 계산해준다.

이후에는 item을 넣었음을 fill 조건변수를 매개하여 알린다.

```c
void *producer(void *arg)
{
    pthread_mutex_lock(&m_id);
    int id = prod_id++;
    pthread_mutex_unlock(&m_id);
    for (int i = 0; i < PROD_ITEM; ++i) {
        usleep(10);

        /* homework */
        /*----------*/
        pthread_mutex_lock(&m_id);
        int ret = put(i);

        /*----------*/
        /* homework */
        if (ret == -1) {
            printf("can't put, becuase buffer is full.\n");
        } else {
            printf("producer %d PUT %d\n", id, ret);
        }
        pthread_mutex_unlock(&m_id);
    }
}
```

```c
void *consumer(void *arg)
{
    pthread_mutex_lock(&m_id);
    int id = cons_id++;
    pthread_mutex_unlock(&m_id);
    for (int i = 0; i < CONS_ITEM; ++i) {
        usleep(10);

        /* homework */
        /*----------*/
        pthread_mutex_lock(&m_id);
        int ret = get();

        /*----------*/
        /* homework */
        if (ret == -1) {
            printf("can't get, becuase buffer is empty.\n");
        } else {
            printf("consumer %d GET %d\n", id, ret);
        }
        pthread_mutex_unlock(&m_id);
    }
}
```

producer, consumer에는 함수호출 ~ 결과출력 까지의 Mutual Exclusion을 보장하기 위하여 Lock, Unlock을 각각 걸어주었다.

2. 실행 결과

```
▶ ./pc
producer 1 PUT 0
consumer 1 GET 0
producer 2 PUT 0
consumer 4 GET 0
producer 2 PUT 1
producer 1 PUT 1
consumer 4 GET 1
consumer 2 GET 1
producer 2 PUT 2
producer 1 PUT 2
consumer 6 GET 2
consumer 1 GET 2
producer 2 PUT 3
consumer 3 GET 3
producer 1 PUT 3
producer 3 PUT 0
consumer 2 GET 3
producer 1 PUT 4
producer 2 PUT 4
producer 3 PUT 1
consumer 6 GET 0
consumer 3 GET 4
consumer 5 GET 4
producer 3 PUT 2
consumer 7 GET 1
consumer 5 GET 2
producer 3 PUT 3
consumer 7 GET 3
producer 3 PUT 4
```

```
▶ ./pc | grep "^consumer"; ./pc | grep "^producer"; ./pc | wc -l
consumer 2 GET 0
consumer 6 GET 0
consumer 5 GET 1
consumer 2 GET 1
consumer 4 GET 0
consumer 3 GET 2
consumer 7 GET 3
consumer 3 GET 1
consumer 6 GET 2
consumer 5 GET 4
consumer 4 GET 2
consumer 7 GET 3
consumer 1 GET 3
consumer 1 GET 4
producer 3 PUT 0
producer 3 PUT 1
producer 2 PUT 0
producer 3 PUT 2
producer 1 PUT 0
producer 2 PUT 1
producer 1 PUT 1
producer 2 PUT 2
producer 3 PUT 3
producer 1 PUT 2
producer 2 PUT 3
producer 3 PUT 4
producer 2 PUT 4
producer 1 PUT 3
producer 1 PUT 4
      29
```

모든 경우가 잘 나왔음을 확인할 수 있었다.

- 느낀 점

  어렵다 시스템프로그래밍 2 같다

  근데 어려운만큼 보람은 있어서 좋다