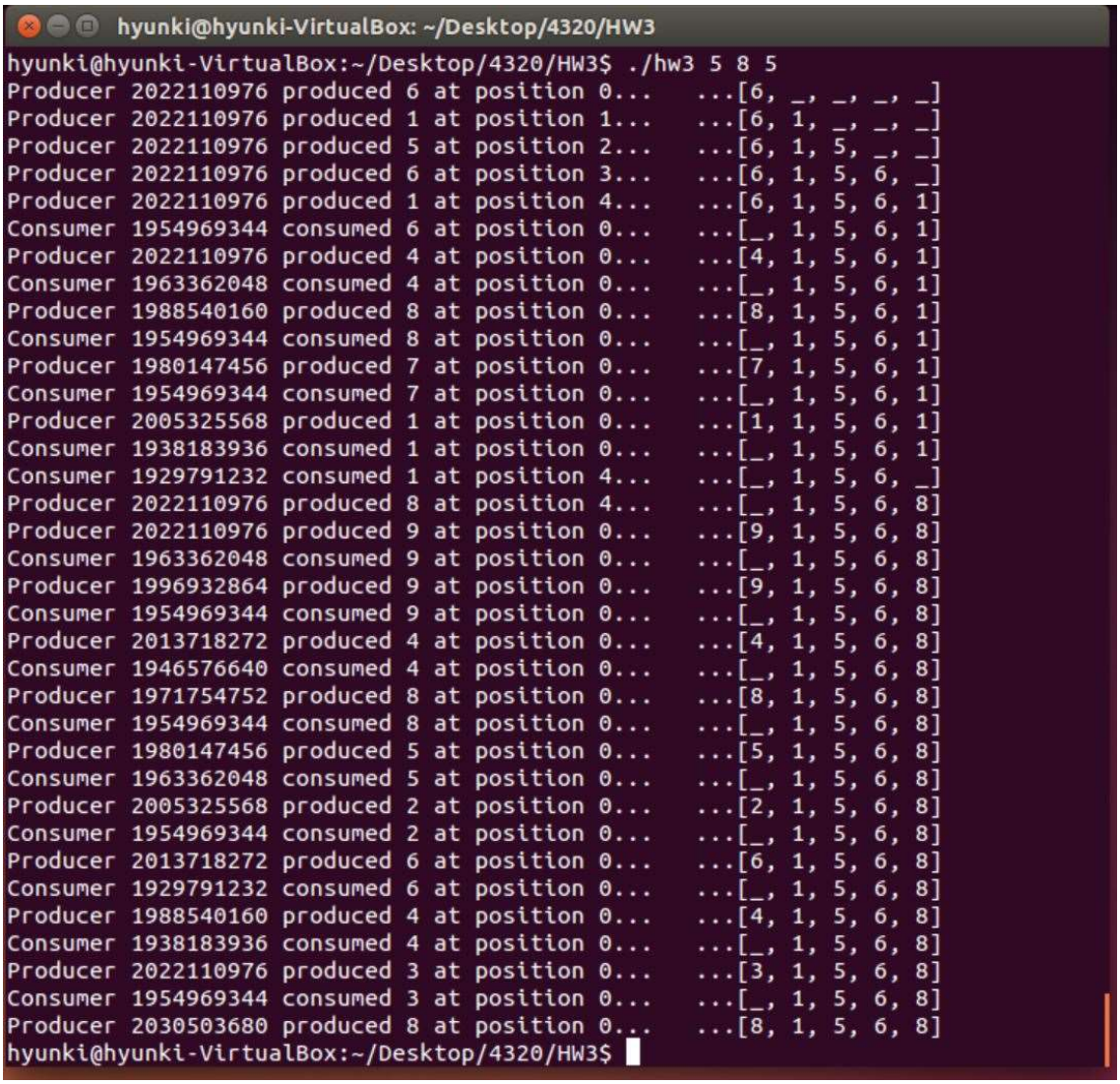
HomeWork3
CSC4320

#002-34-4677
Hyunki Lee

1. Screenshot of output



2. C source code

```
#include "buffer.h"
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <time.h>



buffer_item buffer[BUFFER_SIZE];

sem_t empty;
sem_t full;
pthread_mutex_t mutex;
int i = 0;
int leftPos = 0;
```

```
int rightPos = BUFFER_SIZE-1;
int numOfItemInBuffer;
int pos = 0;



void *producer(void *param);
void *consumer(void *param);

int insert_item(buffer_item item)
{
	int check = 0;
	/* Acquire Empty Semaphore */
	sem_wait(&empty);

	/* Acquire mutex lock to protect buffer */
	pthread_mutex_lock(&mutex);

	/* Insert item into buffer */
```

```
if(numOfItemInBuffer == 0){
        pos = 0;
        leftPos = 0;
        rightPos = BUFFER_SIZE-1;
        buffer[pos] = item;
        numOfItemInBuffer++;
}
else if(numOfItemInBuffer == BUFFER_SIZE){
        check = -1;
}
else if(leftPos <= rightPos){
        pos = BUFFER_SIZE - rightPos;
        rightPos--;
        buffer[pos] = item;
        numOfItemInBuffer++;
}
else{
```

```
            pos = leftPos - 1;
            leftPos--;
            buffer[pos] = item;
            numOfItemInBuffer++;
        }

        printf("Producer %u produced %d at position %d...     ...[",
(unsigned int)pthread_self(), item, pos);
        for( i =0; i < BUFFER_SIZE; i++){
            if(buffer[i] == 0){
                printf("_");
            }else{
                printf("%d", buffer[i]);
            }
            if(i != (BUFFER_SIZE-1)){
                printf(", ");
            }
        }
```

```
        printf("]₩n");

        /* Release mutex lock and full semaphore */
        pthread_mutex_unlock(&mutex);
        sem_post(&full);

        return check;
}

int remove_item(buffer_item *item)
{
        int check = 0;
        /* Acquire Full Semaphore */
        sem_wait(&full);

        /* Acquire mutex lock to protect buffer */
        pthread_mutex_lock(&mutex);
```

```
/* remove an object from buffer placing it in item */
if(numOfItemInBuffer == 0){
	check = -1;
}
else if(leftPos <= rightPos){
	pos = leftPos;
	leftPos++;
	*item = buffer[pos];
	buffer[pos] = 0;
	numOfItemInBuffer--;
}
else{
	pos = (BUFFER_SIZE-rightPos)-1;
	rightPos++;
	*item = buffer[pos];
	buffer[pos] = 0;
	numOfItemInBuffer--;
}
```

```
        printf("Consumer %u consumed %d at position %d...     ...[",(unsigned int)pthread_self(), *item, pos);
        for( i = 0; i < BUFFER_SIZE; i++){
                if(buffer[i] == 0){
                        printf("_");
                }else{
                        printf("%d", buffer[i]);
                }
                if(i != (BUFFER_SIZE-1)){
                        printf(", ");
                }
        }
        printf("]₩n");

        /* Release mutex lock and empty semaphore */
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
```

```
    return check;
}


int main(int argc, char *argv[])
{
    /* Get command line arguments argv[1],argv[2],argv[3] */
    if(argc !=4){
        fprintf(stderr, "USAGE:./hw3 <sleeptime> <# of producer threads> <# of consumer threads>₩n");
    }
    int sleepTime = atoi(argv[1]);
    int num_Pro_Threads = atoi(argv[2]);
    if(num_Pro_Threads < 1){
        fprintf(stderr, "USAGE: <# of producer threads> should be larger than 1.₩n");
    }
    int num_Con_Threads = atoi(argv[3]);
```

```
    if(num_Con_Threads < 1){
        fprintf(stderr, "USAGE: <# of consumer threads> should be larger than 1.₩n");
    }

    /* Initialize buffer related synchronization tools */
    int j;
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    srand(time(NULL));

    /* Create producer threads based on the command line input */
    pthread_t pro[num_Pro_Threads];
    for(j = 0; j < num_Pro_Threads; j++){
        pthread_create(&pro[j], NULL, producer, NULL);
    }
    /* Create consumer threads based on the command line
```

```
input */
        pthread_t con[num_Con_Threads];
        for(j = 0; j < num_Con_Threads; j++){
                pthread_create(&con[j], NULL, consumer, NULL);
        }

        /* Sleep for user specified time based on the command
line input */
        sleep(sleepTime*3);

        return 0;
}

void *producer(void *param)
{
        /* producer thread that calls insert_item() */
      int ranNum = rand() % 10;
```

```
        while(1){
                sleep(ranNum);
                buffer_item item = (rand() % 9)+1;
                insert_item(item);
        }



}

void *consumer(void *param)
{
         int ranNum = rand() % 10;
        /* consumer thread that calls remove_item() */
        while(1){
                sleep(ranNum);
                buffer_item item;
                remove_item(&item);
        }
```

}