Operating Systems
(Producer consumer problem)

Spring, 2016

Outline

- Objectives
- Programming outlines
- What to do ?

Objectives

- Understanding
  - Produce consumer problem
  - semaphore, mutex

Programming outline -buffer

```
/* buffer.h */
typedef int buffer_item;
#define BUFFER_SIZE 10
```

<관련 함수>

```
#include <pthread.h>
#include <semaphore.h>
pthread_mutex_t mutex;
sem_t empty;
sem_t full;
pthread_mutex_lock(&mutex);
pthread_mutex_unlock(&mutex);
sem_init(&empty, 0, 5);
sem_post(&empty);
sem_wait(&empty);
```

Programming outline –main function

```
#include “buffer.h”

int main(int argc, char *argv[]){

 /* 1. Get command line arguments argv[1], argv[2], argv[3] */
 /* 2. Initialize buffer */
 /* 3. Create producer thread(s) */
 /* 4. Create consumer thread(s) */
 /* 5. Sleep */
 /* 6. Exit */

}
```

How long to sleep before terminating

# of producer threads

# of consumer threads

Programming outline – functions

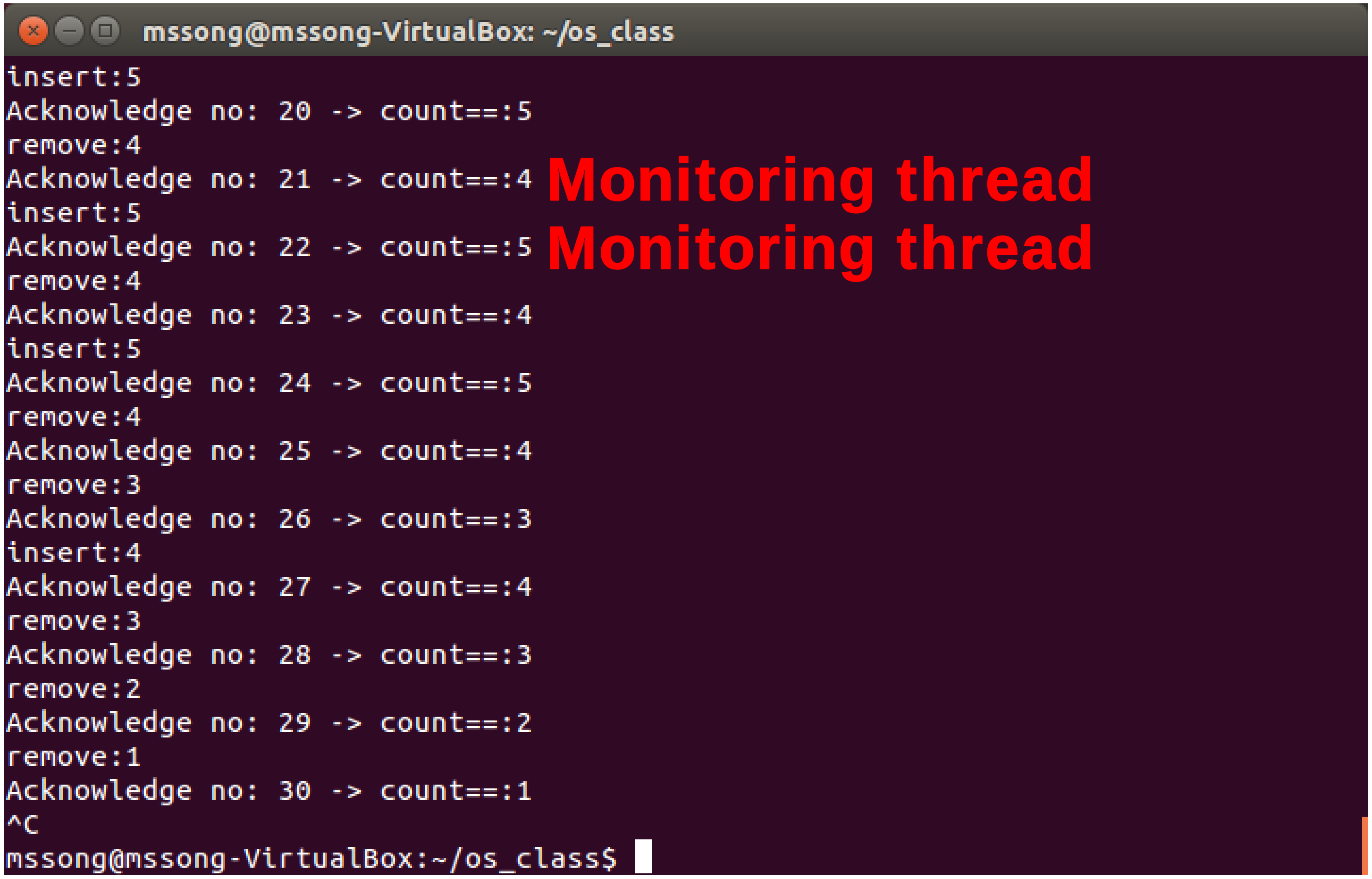
```
#include “buffer.h”

/* the buffer */
buffer_item buffer[BUFFER_SIZE];

int insert_item(buffer_item item){
        /* insert item into buffer
        return 0 if successful, otherwise
        return -1 indicating an error condition */
}

int remove_item(buffer_item *item){
        /* remove an object from buffer placing it in item
        return 0 if successful, otherwise
        return -1 indicating an error condition */
}
```

Threads

```
#include <stdlib.h>                    /* required for rand() */
#include “buffer.h”

void *producer(void *param){
        buffer_item item;

        while(true){
                /* sleep for a random period of time */
                sleep(…);
                /* generate a random number */
                item = rand();
                if(insert_item(item))
                        fprintf(“report error condition”);
                else
                        printf(“producer produced %d\n”, item);
        }
}

void *consumer(void *param){
        buffer_item item;

        while(true){
                /* sleep for a random period of time */
                sleep(…);
                if(remove_item(&item))
                        fprintf(“report error condition”);
                else
                        printf(“consumer consumed %d\n”, item);
                }
}
```

- A monitoring thread
  - This thread monitors the number of items in the buffer
    - Whenever a producer (or a consumer) thread produces or consumes an item, they (both) should be blocked until the monitor thread acknowledges it

<Result>



mssong@mssong-VirtualBox: ~/os_class

```
insert:5
Acknowledge no: 20 -> count==:5
remove:4
Acknowledge no: 21 -> count==:4
insert:5
Acknowledge no: 22 -> count==:5
remove:4
Acknowledge no: 23 -> count==:4
insert:5
Acknowledge no: 24 -> count==:5
remove:4
Acknowledge no: 25 -> count==:4
remove:3
Acknowledge no: 26 -> count==:3
insert:4
Acknowledge no: 27 -> count==:4
remove:3
Acknowledge no: 28 -> count==:3
remove:2
Acknowledge no: 29 -> count==:2
remove:1
Acknowledge no: 30 -> count==:1
^C
mssong@mssong-VirtualBox:~/os_class$
```

What to do?

- What to do ?
  - 강의 노트를 참조해서 producer consumer problem 을 구현할 것
  - 보고서에
    - semaphore, mutex 사용에 대해서 기술할 것
- No submission after the deadline
- Deadline : May 13th