OS Project 4 : Producer-Consumer Problem

Project for Computer Architecture & Operating Systems by Chentao Wu, 2016 Autumn Semester

1. Project Introduction:

In Chapter 3, we developed a model of a system consisting of cooperating sequential processes or threads, all running asynchronously and possibly sharing data. We illustrated this model with the producer-consumer problem, which is representative of operating systems. Specifically, in Section 3.4.1, we described how a bounded buffer could be used to enable processes to share memory.

Let us return to our consideration of the bounded buffer. As we pointed out, our solution allows at most BUFFER.SIZE - 1 items in the buffer at the same time. Suppose we want to modify the algorithm to remedy this deficiency. One possibility is to add an integer variable counter, initialized to 0. counter is incremented every time we add a new item to the buffer and is decremented every time we remove one item from the buffer. The code for the producer process and consumer process can be modified as follows:

Algorithm 1: The producer process

```
while $True$ do
    // produce an item in nextProduced
    while $counter == BUFFER\_SIZE$ do
        ;// do nothing
    end
    $buffer[in] = nextProduced$;
    $in = (in + 1)\%BUFFER\_SIZE$;
    $counter++$;
end
```

Algorithm 2: The consumer process

```
while $True$ do
    while $counter == 0$ do
        ;// do nothing
    end
    $nextConsumed = buffer[out]$;
    $out = (out + 1)\%BUFFER\_SIZE$;
    $counter--$;
    // consume the item in nextConsumed
end
```

2. Project Environment:

VirtualBox with Linux Ubuntu 16.04.
Windows 10

3. Project Realization:

In our solutions to the problems, we use semaphores for synchronization.

Algorithm 3: The structure of the producer process

```
while True do
    ...
    // produce an item in nextp
    ...
    wait(empty);
    wait(mutex);
    ...
    // add nextp to buffer
    ...
    signal(mutex);
    signal(full);
end
```

Algorithm 4: The structure of the consumer process

```
while True do
    wait((full);
    wait(mutex);
    ...
    // remove an item from buffer to nextc
    ...
    signal(mutex);
    signal(empty);
    ...
    // consume the item in nextc
    ...
end
```

Part1 The Buffer

Internally, the buffer will consist of a fixed-size array of type buffer_item (which will be defined using a typef def). The array of buffer_item objects will be manipulated as a circular queue. The definition of buffer_item, along with the size of the buffer, can be stored in a header file such as the following:

```
/* buffer.h */
typedef int buffer.item;
#define BUFFER_SIZE 5
```

The buffer will be manipulated with two functions, insert_item() and remove_item(), which are called by the producer and consumer threads, respectively. A code outlining these functions appears as:

```

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

int ready_pro = 0, ready_con = 0;

int insert_item(buffer_item item){
    /* insert item into buffer
    return 0 if successful, otherwise
    return -1 indicating an error condition */
    try{
        buffer[ready_pro%5] = item;
        ready_pro = (ready_pro+1);
        printf("Insert␣successfully.\n");
        printf("The␣buffer␣is:");
        //output the number in buffer
        for(int j = ready_con; j< ready_pro;++j){
            printf("␣%d",buffer[j%5]);
        }
        printf("\n");
        return 0;
    }
    catch(exception e){
        printf("some␣error␣in␣inserting␣item␣occurs.");
        return -1;
    }
}

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 */

    try{
        ready_con++;
        printf("remove␣%d␣successfully.\n",*item);
        printf("The␣buffer␣is:");
        //output the number in buffer
        for(int j = ready_con; j< ready_pro;++j){
            printf("␣%d",buffer[j%5]);
        }
        printf("\n");
        return 0;
    }
    catch(exception e){
        printf("some␣error␣in␣removing␣item␣occurs.");
        return -1;
    }
}
```

The insert_item() and remov_item() functions will synchronize the producer and consumer using the algorithms before. The buffer will also require an initialization function that initializes the mutualexclusion object mutex along with the empty and full semaphores.

Part2 main() function

The main() function will initialize the buffer and create the separate producer and consumer threads. Once it has created the producer and consumer threads, the mainO function will sleep for a period of time and, upon awakening, will terminate the application. The mainO function will be passed three parameters on the command line:

1. How long to sleep before terminating
2. The number of producer threads
3. The number of consumer threads

```c
int main(){
    //1. Get command line arguments argv[1], argv[2], argv[3]
    int p, c, time;
    scanf("%d%d%d",&time,&p,&c);

    //2. Initialize buffer
    init_semaphores();


    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_t attr1;
    pthread_attr_init(&attr1);
    //3. Create producer thread(s)
    pthread_t pro[p];
    int pthread_num[p];

    for(int i=0;i<p;++i){
        pthread_num[i] = i;
        pthread_create(&pro[i],&attr,producer,&pthread_num[i]);
        printf("Initializing␣producer␣%d␣successfully.\n",i );
    }
    //4. Create consumer thread(s)
    pthread_t con[c];
    int con_num[c];
    for(int i=0;i<c;++i){
        con_num[i] = i;
        pthread_create(&con[i],&attr,consumer,&con_num[i]);
        printf("Initializing␣comsumer␣%d␣successfully.\n",i );
    }
    //5. sleep
    sleep(time);

    //6. Exit
    printf("Exit.\n");
    return 0;
}
```

Part3 Producer and Consumer Threads

The producer thread will alternate between sleeping for a random period of time and inserting a random integer into the buffer. Random numbers will be produced using the rand() function, which produces random irttegers between 0 and RANDMAX. The consumer will also sleep for a random period of time and, upon awakening, will attempt to remove an item from the buffer. An outline of the producer and consumer threads appears as:

```c
void *producer(void *param){
    buffer_item ran;
    int id = *(int *) param;
    time_t t = pthread_self();
    srand(time(&t));

    while(1){
        //sleep for a random period of time
        int time = rand() %6 +1;
        sleep(time);
        //generate a random number
        sem_wait(&empty);
        sem_wait(&mutex);// block it until it is greater than 0, and then --mutex
        printf("Producer %d Sleeping time: %d\n", id, time);
        ran = rand() ;
        printf("Producer %d produced %d \n",id,ran);
        if(insert_item(ran)){
            printf("report error condition");
        }
        printf("\n");
    //  sleep(5);
        sem_post(&mutex); //++mutex, it's an atomic transaction
        sem_post(&full);

    }
}

void *consumer (void *param){
    buffer_item ran;
    int id =  *(int *)param;

    while(1){
        //sleep for a random period of time
        int time = rand() %6 +1;
        sleep(time);
        sem_wait(&full);
        sem_wait(&mutex);// block it until it is greater than 0, and then --mutex
        printf("Consumer %d Sleeping time: %d\n", id, time);
        ran = buffer[ready_con%5];

        printf("Consumer %d starting removing: %d\n", id, ran);
        if(remove_item(&ran)){
            printf("report error condition");
        }
        printf("\n");
    //  sleep(5);
        sem_post(&mutex); //++mutex, it's an atomic transaction
```

```
            sem_post(&empty);
      }

}
```

Part4 Pthreads Semaphores

Pthreads provides two types of semaphores-named and unnamed. For this project, we use unnamed semaphores. The code below illustrates how a semaphore is created:

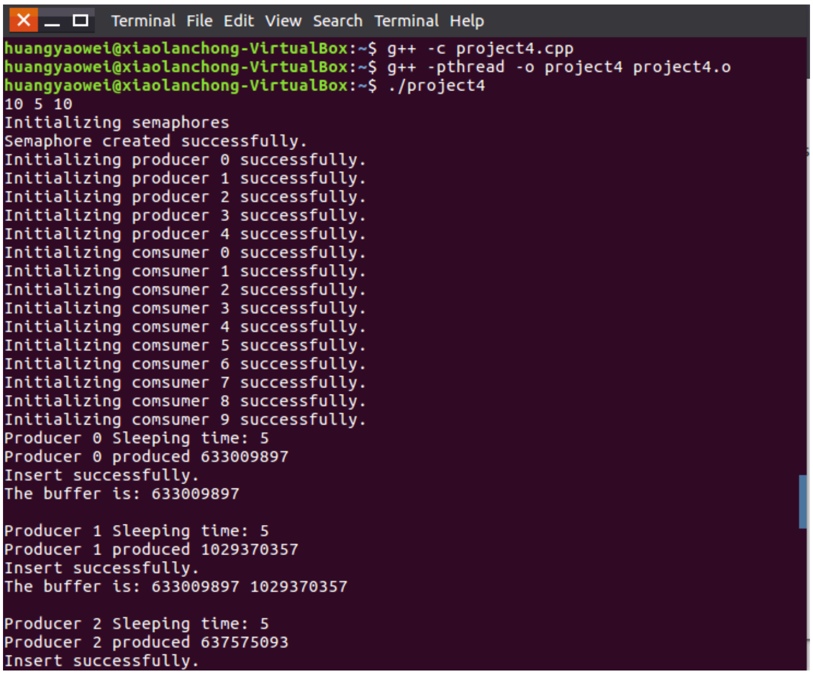
```
void init_semaphores()
{
      printf("Initializing␣semaphores\n");
      // 1. A pointer to the semaphore.
      // 2. A flag indicating the level of sharing. \
               0 means this semaphore can only be shared by threads\
               belonging to the same process that created the semaphore.
      // 3. The semaphore's initial value.
      sem_init(&full, 0, 0);
      sem_init(&empty, 0, BUFFER_SIZE);
      sem_init(&mutex, 0, 1);
      printf("Semaphore␣created␣successfully.\n");
}
```

4. Project Result:

I made some simple tests:

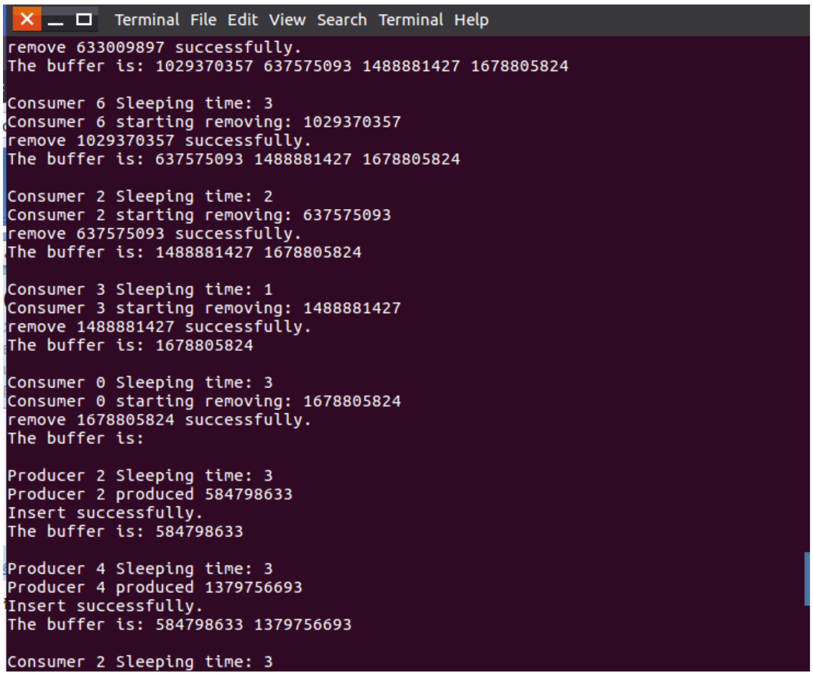
1. Creating 5 producer and 10 consumer in linux.

   When the consumer's number is greater than producer's, the buffer will always be empty.



```
huangyaowei@xiaolanchong-VirtualBox:~$ g++ -c project4.cpp
huangyaowei@xiaolanchong-VirtualBox:~$ g++ -pthread -o project4 project4.o
huangyaowei@xiaolanchong-VirtualBox:~$ ./project4
10 5 10
Initializing semaphores
Semaphore created successfully.
Initializing producer 0 successfully.
Initializing producer 1 successfully.
Initializing producer 2 successfully.
Initializing producer 3 successfully.
Initializing producer 4 successfully.
Initializing comsumer 0 successfully.
Initializing comsumer 1 successfully.
Initializing comsumer 2 successfully.
Initializing comsumer 3 successfully.
Initializing comsumer 4 successfully.
Initializing comsumer 5 successfully.
Initializing comsumer 6 successfully.
Initializing comsumer 7 successfully.
Initializing comsumer 8 successfully.
Initializing comsumer 9 successfully.
Producer 0 Sleeping time: 5
Producer 0 produced 633009897
Insert successfully.
The buffer is: 633009897

Producer 1 Sleeping time: 5
Producer 1 produced 1029370357
Insert successfully.
The buffer is: 633009897 1029370357

Producer 2 Sleeping time: 5
Producer 2 produced 637575093
Insert successfully.
```

图 1: Result: Creating 5 producer and 10 consumer in linux.

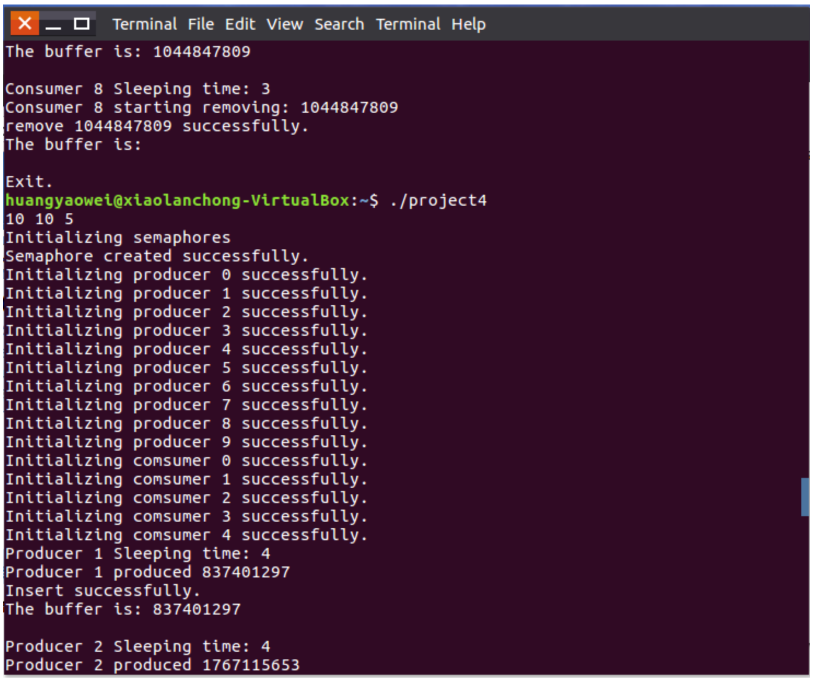


```
remove 633009897 successfully.
The buffer is: 1029370357 637575093 1488881427 1678805824

Consumer 6 Sleeping time: 3
Consumer 6 starting removing: 1029370357
remove 1029370357 successfully.
The buffer is: 637575093 1488881427 1678805824

Consumer 2 Sleeping time: 2
Consumer 2 starting removing: 637575093
remove 637575093 successfully.
The buffer is: 1488881427 1678805824

Consumer 3 Sleeping time: 1
Consumer 3 starting removing: 1488881427
remove 1488881427 successfully.
The buffer is: 1678805824

Consumer 0 Sleeping time: 3
Consumer 0 starting removing: 1678805824
remove 1678805824 successfully.
The buffer is:

Producer 2 Sleeping time: 3
Producer 2 produced 584798633
Insert successfully.
The buffer is: 584798633

Producer 4 Sleeping time: 3
Producer 4 produced 1379756693
Insert successfully.
The buffer is: 584798633 1379756693

Consumer 2 Sleeping time: 3
```

图 2: Result: Creating 5 producer and 10 consumer in linux.

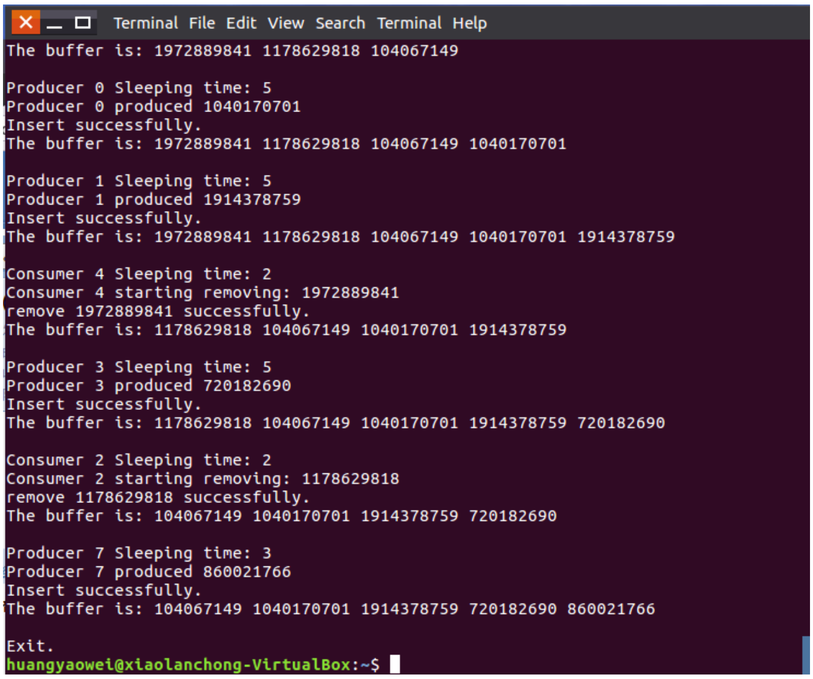
2. Creating 10 producer and 5 consumer in linux.

   When the producer's number is greater than consumer's, the buffer will always be full.



```
The buffer is: 1044847809

Consumer 8 Sleeping time: 3
Consumer 8 starting removing: 1044847809
remove 1044847809 successfully.
The buffer is:

Exit.
huangyaowei@xiaolanchong-VirtualBox:~$ ./project4
10 10 5
Initializing semaphores
Semaphore created successfully.
Initializing producer 0 successfully.
Initializing producer 1 successfully.
Initializing producer 2 successfully.
Initializing producer 3 successfully.
Initializing producer 4 successfully.
Initializing producer 5 successfully.
Initializing producer 6 successfully.
Initializing producer 7 successfully.
Initializing producer 8 successfully.
Initializing producer 9 successfully.
Initializing comsumer 0 successfully.
Initializing comsumer 1 successfully.
Initializing comsumer 2 successfully.
Initializing comsumer 3 successfully.
Initializing comsumer 4 successfully.
Producer 1 Sleeping time: 4
Producer 1 produced 837401297
Insert successfully.
The buffer is: 837401297

Producer 2 Sleeping time: 4
Producer 2 produced 1767115653
```

图 3: Result: Creating 10 producer and 5 consumer in linux.

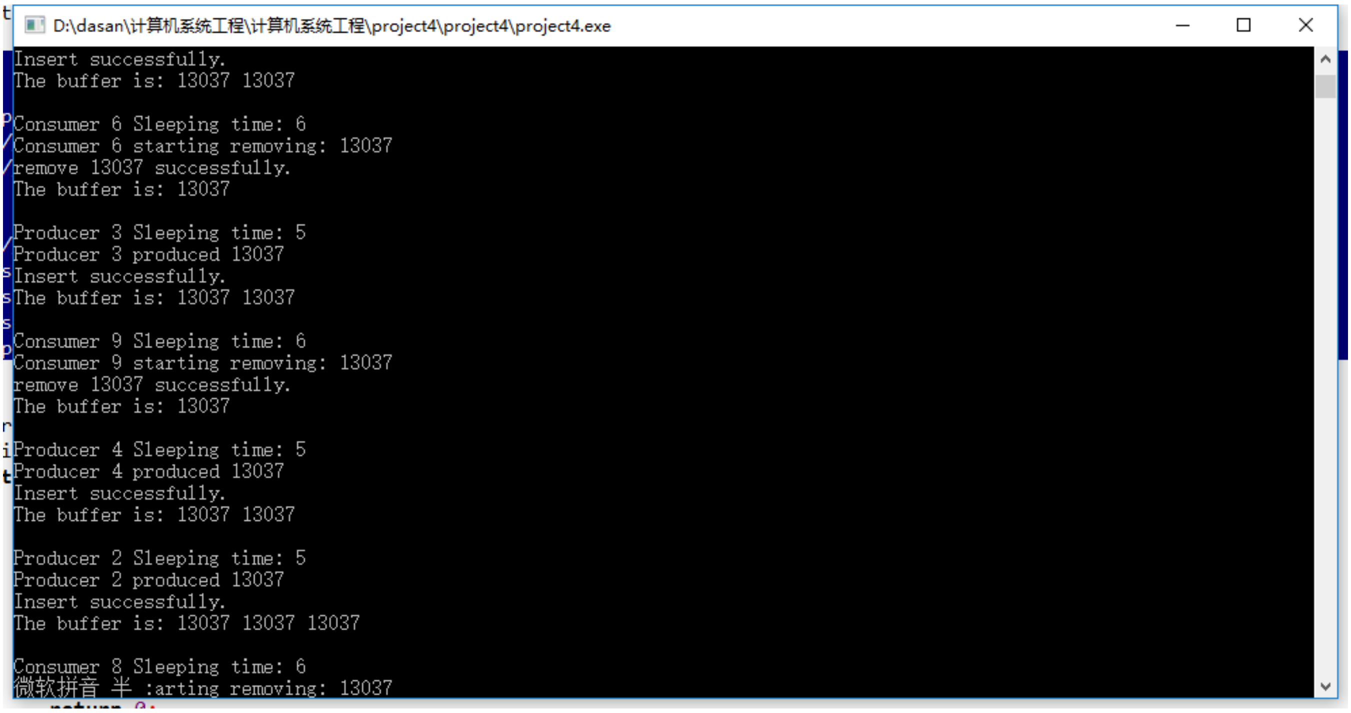


```
The buffer is: 1972889841 1178629818 104067149

Producer 0 Sleeping time: 5
Producer 0 produced 1040170701
Insert successfully.
The buffer is: 1972889841 1178629818 104067149 1040170701

Producer 1 Sleeping time: 5
Producer 1 produced 1914378759
Insert successfully.
The buffer is: 1972889841 1178629818 104067149 1040170701 1914378759

Consumer 4 Sleeping time: 2
Consumer 4 starting removing: 1972889841
remove 1972889841 successfully.
The buffer is: 1178629818 104067149 1040170701 1914378759

Producer 3 Sleeping time: 5
Producer 3 produced 720182690
Insert successfully.
The buffer is: 1178629818 104067149 1040170701 1914378759 720182690

Consumer 2 Sleeping time: 2
Consumer 2 starting removing: 1178629818
remove 1178629818 successfully.
The buffer is: 104067149 1040170701 1914378759 720182690

Producer 7 Sleeping time: 3
Producer 7 produced 860021766
Insert successfully.
The buffer is: 104067149 1040170701 1914378759 720182690 860021766

Exit.
huangyaowei@xiaolanchong-VirtualBox:~$
```

图 4: Result: Creating 10 producer and 5 consumer in linux.

5. The problem I have met

I have spent the most time at creating the random number in windows, the code in windows is the same to the one in the linux, but I have found that the random number created by producer process will always be the same at a time. Like this:


D:\dasan\计算机系统工程\计算机系统工程\project4\project4\project4.exe

```
Insert successfully.
The buffer is: 13037 13037

Consumer 6 Sleeping time: 6
Consumer 6 starting removing: 13037
remove 13037 successfully.
The buffer is: 13037

Producer 3 Sleeping time: 5
Producer 3 produced 13037
Insert successfully.
The buffer is: 13037 13037

Consumer 9 Sleeping time: 6
Consumer 9 starting removing: 13037
remove 13037 successfully.
The buffer is: 13037

Producer 4 Sleeping time: 5
Producer 4 produced 13037
Insert successfully.
The buffer is: 13037 13037

Producer 2 Sleeping time: 5
Producer 2 produced 13037
Insert successfully.
The buffer is: 13037 13037 13037

Consumer 8 Sleeping time: 6
微软拼音 半 :arting removing: 13037
```

图 5: Result: problem in windows.

And I have thought a lot of method to solve it, like use pthread_self() or random number in main thread to be the seed of the srand(time(&t)) of other thread, but it didn't work.I think it may due to the random mechanism in windows. The time seed is the thread's running time rather than the current system's time, so in each thread the random number created are the same.

6. Harvest

Through this project I learned a lot knowledge about process synchronization and how to use semaphore to realize it. And dealing with the Producer-Consumer problem makes me understand the synchronization more deeply. The operating system is a very fun thing to manipulate, and solving problems makes me very proudable.

7. Code

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <time.h>
#include <iostream>

using namespace std;

typedef int buffer_item;
#define BUFFER_SIZE 5

buffer_item buffer[BUFFER_SIZE];

sem_t full, empty, mutex;

void init_semaphores()
{
    printf("Initializing␣semaphores\n");
    // 1. A pointer to the semaphore.
    // 2. A flag indicating the level of sharing. \
          0 means this semaphore can only be shared by threads\
          belonging to the same process that created the semaphore.
    // 3. The semaphore's initial value.
    sem_init(&full, 0, 0);
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&mutex, 0, 1);
    printf("Semaphore␣created␣successfully.\n");
}

int ready_pro = 0, ready_con = 0;
int insert_item(buffer_item item){
    try{
        buffer[ready_pro%5] = item;
        ready_pro = (ready_pro+1);
        printf("Insert␣successfully.\n");
        printf("The␣buffer␣is:");
        for(int j = ready_con; j< ready_pro;++j){
            printf("␣%d",buffer[j%5]);
        }
        printf("\n");
        return 0;
    }
    catch(exception e){
        printf("some␣error␣in␣inserting␣item␣occurs.");
        return -1;
    }
}

void *producer(void *param){
    buffer_item ran;
    //sem_wait(&mutex);
    int id = *(int *) param;
    printf("Producer␣%d␣seed␣%lu␣\n",id,pthread_self());
    time_t t = pthread_self()%10000;
    srand(time(&t));
    //sem_post(&mutex);
    while(1){
        //sleep for a random period of time
        int time = rand() %6 +1;
        sleep(time);
        //generate a random number
        sem_wait(&empty);
        sem_wait(&mutex);// block it until it is greater than 0, and then --mutex
        printf("Producer␣%d␣Sleeping␣time:␣%d\n", id, time);
        ran = rand() ;
        printf("Producer␣%d␣produced␣%d␣\n",id,ran);
        if(insert_item(ran)){
            printf("report␣error␣condition");
        }
        printf("\n");
    //  sleep(5);
        sem_post(&mutex); //++mutex, it's an atomic transaction
        sem_post(&full);

    }
}

int remove_item(buffer_item *item){
    try{
        ready_con++;
        printf("remove␣%d␣successfully.\n",*item);
        printf("The␣buffer␣is:");
        for(int j = ready_con; j< ready_pro;++j){
            printf("␣%d",buffer[j%5]);
        }
        printf("\n");
        return 0;
    }
    catch(exception e){
        printf("some␣error␣in␣removing␣item␣occurs.");
        return -1;
    }
}

void *consumer (void *param){
    buffer_item ran;
    int id =  *(int *)param;


    while(1){
        //sleep for a random period of time
        int time = rand() %6 +1;
        sleep(time);
        sem_wait(&full);
        sem_wait(&mutex);// block it until it is greater than 0, and then --mutex
        printf("Consumer␣%d␣Sleeping␣time:␣%d\n", id, time);
        ran = buffer[ready_con%5];

        printf("Consumer␣%d␣starting␣removing:␣%d\n", id, ran);
        if(remove_item(&ran)){
            printf("report␣error␣condition");
        }
        printf("\n");
    //  sleep(5);
        sem_post(&mutex); //++mutex, it's an atomic transaction
        sem_post(&empty);
    }

}


int main(){
    //1. Get command line arguments argv[1], argv[2], argv[3]
    int p, c, time;
    scanf("%d%d%d",&time,&p,&c);

    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_t attr1;
    pthread_attr_init(&attr1);
    //2. Initialize buffer
    init_semaphores();

    //3. Create producer thread(s)
```

```c
    pthread_t pro[p];
    int pthread_num[p];

    for(int i=0;i<p;++i){
        pthread_num[i] = i;
        pthread_create(&pro[i],&attr,producer,&pthread_num[i]);
        printf("Initializing␣producer␣%d␣successfully.\n",i );
    }
    //4. Create consumer thread(s)
    pthread_t con[c];
    int con_num[c];
    for(int i=0;i<c;++i){
        con_num[i] = i;
        pthread_create(&con[i],&attr,consumer,&con_num[i]);
        printf("Initializing␣comsumer␣%d␣successfully.\n",i );
    }
    //5. sleep
    sleep(time);

    //6. Exit
    printf("Exit.\n");
    return 0;
}
```