Assignment-7
Synchronization

Subash Mylraj
(CED18I051)

30 November 2020

Question 1: Simulate the Producer Consumer code discussed in the class.

Code:

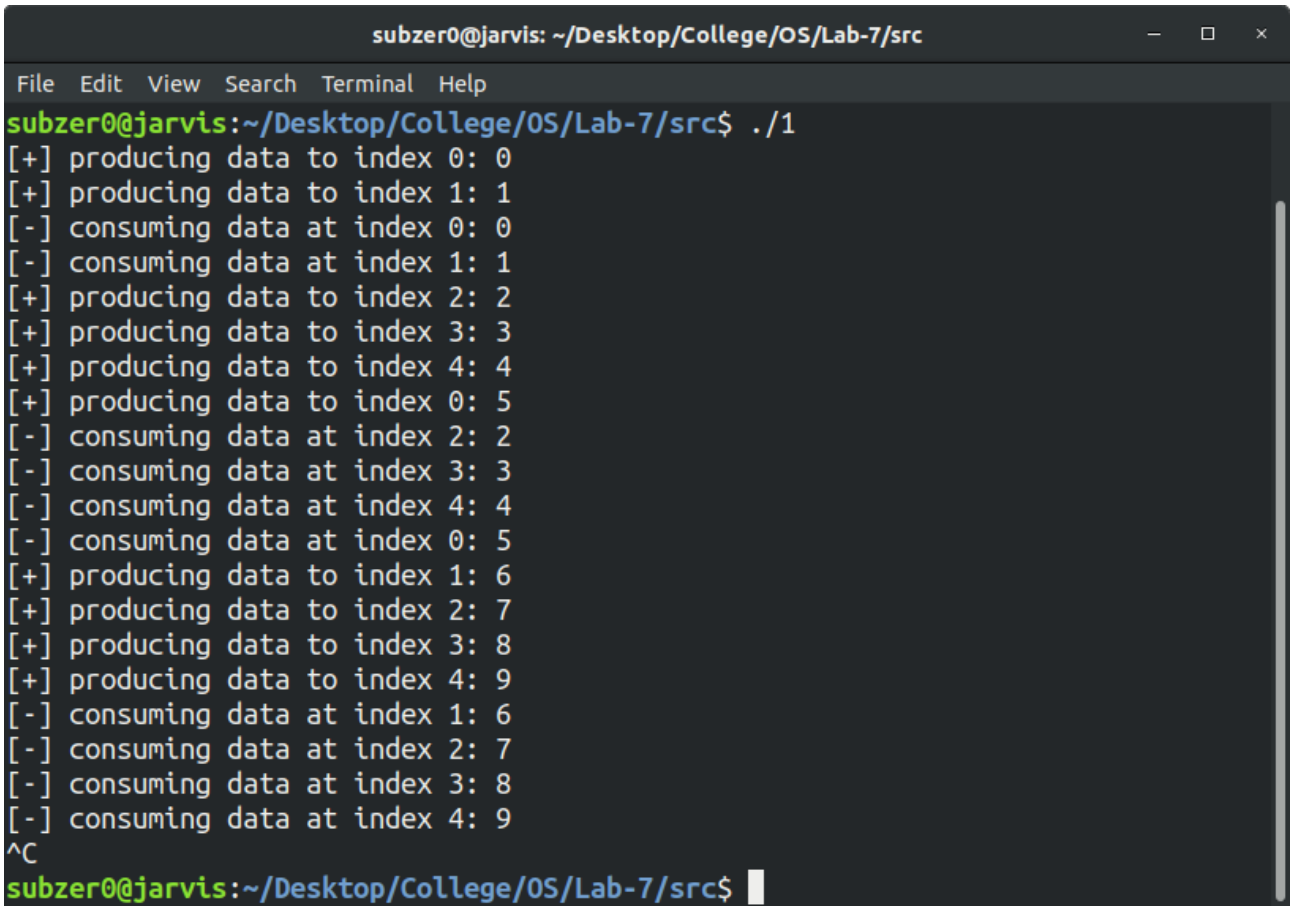
```
#include<stdio.h>
#include<sys/wait.h>
#include<pthread.h>
#define N 5

void *producer(void *args);
void *consumer(void *args);

int buf[N];
int in = 0, out = 0;

void *producer(void *args){

   int data = 0,i = 0;
   while(i < 10){

      while((in+1) % N == out);  // spin lock

      buf[in] = data++;
      printf("[+] producing data to index %d: %d\n", in, buf[in]);
      in = (in + 1) % N;
      i++;
   }
   pthread_exit(0);
}

void *consumer(void *args){

   int data;
   while(1){

      while(in == out);

      data = buf[out];
      printf("[-] consuming data at index %d: %d\n", out, data);
      out = (out+1) % N;
   }
   pthread_exit(0);
}

int main(){

   pthread_t tid[2];
```

```
    pthread_create(&tid[0], NULL, producer, NULL);
    pthread_create(&tid[1], NULL, consumer, NULL);

    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);

    return 0;
}
```

Explanation:

This program is a simulation of a solution to the producer consumer problem. This solution uses a circular queue, where the producer produces until the queue is full and the consumer consumes until the queue is empty. Once the queue is full, the consumer stops producing.

Output:



Question 2: Extend the producer consumer simulation in Q1 to sync access of critical data using Petersons algorithm.

Code:

```
#include<stdio.h>
#include<sys/wait.h>
#include<pthread.h>
#include<unistd.h>
```

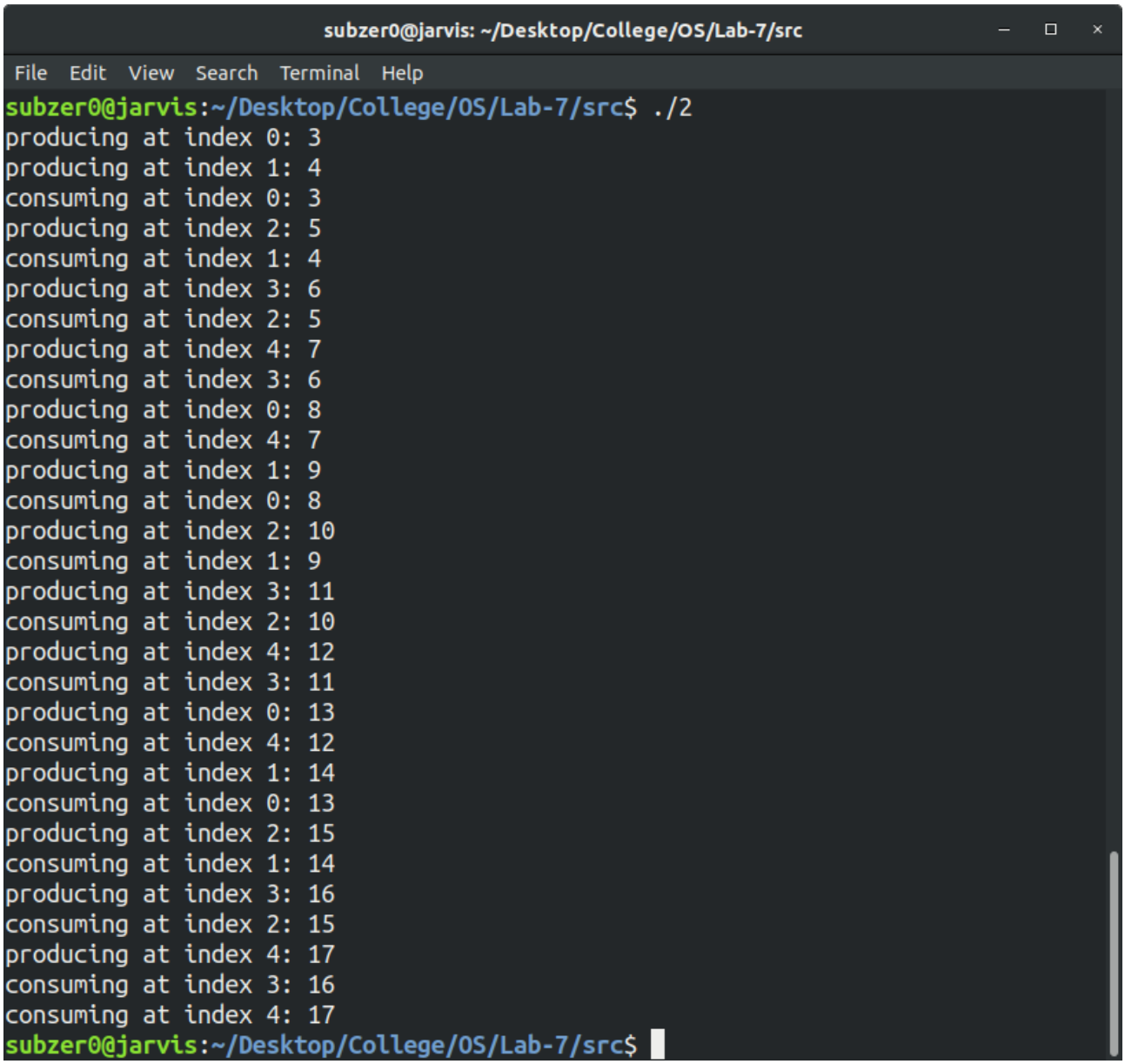
```c
#define N 5

void *producer(void *args);
void *consumer(void *args);

int buf[N];
int in = 0,out = 0,counter = 0;
int flag[] = {0,0};
int turn = 0;

void lock(int self){

    // Set flag[self] = 1 saying you want to acquire lock
    flag[self] = 1;

    // But, first give the other thread the chance to
    // acquire lock
    turn = 1-self;

    // Wait until the other thread looses the desire
    // to acquire lock or it is your turn to get the lock.
    while (flag[1-self]==1 && turn==1-self);
}

// Executed after leaving critical section
void unlock(int self){

    // You do not desire to acquire lock in future.
    // This will allow the other thread to acquire
    // the lock.
    flag[self] = 0;
}

int main(){

   pthread_t tid[2];

   pthread_create(&tid[0], NULL, producer, NULL);
   pthread_create(&tid[1], NULL, consumer, NULL);
   pthread_join(tid[0], NULL);
   pthread_join(tid[1], NULL);
   return 0;
}

void *producer(void *args){

   int data = 3,i = 0;
   while(i<15){

      while(counter == N) ;
      lock(0);
      buf[in] = data++;
      printf("producing at index %d: %d\n", in, buf[in]);
      in = (in+1)%N;
      i++;
      counter++;
      unlock(0);
   }
   pthread_exit(0);
}

void *consumer(void *args){

   int data,i = 0;

   while(i<15){
```

```
        while(counter == 0);
        lock(1);
        data = buf[out];
        printf("consuming at index %d: %d\n", out, buf[out]);
        out = (out+1)%N;
        i++;
        counter--;
        unlock(1);
        //sleep(1);
    }
    pthread_exit(0);
}
```

Explanation:

This program is an extension to the previous program. Petersons algorithm is used to attain synchronization between the producer and the consumer. The variable counter is used to keep track of the status of the buffer. If it is full, then the producer does not produce anything. If it is empty, then the consumer does not consume anything.

Output:



Question 3: Dictionary Problem: Let the producer set up a dictionary of at least 20 words with three attributes (Word, Primary meaning, Secondary meaning) and let the consumer search for the word and retrieve its respective primary and secondary meaning.

Code:

```
#include<stdio.h>
#include<sys/wait.h>
#include<pthread.h>
#include<string.h>
#define N 5

void *producer(void *args);
void *consumer(void *args);

struct dict{
   char word[30];
   char meaning_1[30], meaning_2[30];
}buf[N];

int flag[] = {0, 0};
int turn = 0;
char search_word[30];

int in = 0,out = 0,end = 0;
int j = 0;

// acquire lock
void lock(int id){

   // set the flag to let the other thread know that this thread wants to acquire a lock
   flag[id] = 1;

   // give the other thread a chance first
   turn = 1-id;

   // wait till the other thread finishes
   while (flag[1-id] == 1 && turn == 1-id) ;
}

// release lock
void unlock(int id){

   // set the flag to let the other thread know that this thread does not need the lock anymore
   flag[id] = 0;
}


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

   pthread_t tid[2];
   strcpy(search_word, argv[1]);

   pthread_create(&tid[0],NULL,producer,NULL);
   pthread_create(&tid[1],NULL,consumer,NULL);

   pthread_join(tid[0],NULL);
   pthread_join(tid[1],NULL);

   return 0;
}
```

```
void *producer(void *args){

   int i=0;
   FILE *fptr=fopen("dict.txt","r");
   while(1){
      lock(0);
      j=0;
      while(j<5){

         char ch;
         char word[20];
         int k=0;

         while((ch=fgetc(fptr))!=','){
            buf[j].word[k++]=ch;
         }

         buf[j].word[k]='\0';
         k=0;

         while((ch=fgetc(fptr))!=','){
            buf[j].meaning_1[k++]=ch;
         }

         buf[j].meaning_1[k]='\0';
         k=0;

         while((ch=fgetc(fptr))!='\n'){
            if(ch==EOF){
               buf[j].meaning_2[k]='\0';
               end=1;
               unlock(0);
               return NULL;
            }
            buf[j].meaning_2[k++]=ch;
         }

         buf[j].meaning_2[k]='\0';
         j += 1;
      }
      int k;
      i += 1;
      unlock(0);
   }
   pthread_exit(0);
}

void *consumer(void *args){

   int i=0;

   while(1){

      lock(1);
      int k=0;
      while(k<j){
         if(strcmp(buf[k].word,search_word)==0){
            printf("[*] Word found: %s\n", buf[k].word);
            printf("[1] Meaning 1: %s\n", buf[k].meaning_1);
            printf("[2] Meaning 2: %s\n", buf[k].meaning_2);
            unlock(1);
            return NULL;
         }
         k += 1;
```

```
      }

      i += 1;
      if(end==1){
         printf("[!] Word not found!\n");
         return NULL;
      }
      unlock(1);

   }

   pthread_exit(0);
}
```

Dictionary:

```
book,compilation of pages,read material
class,thing in c,where students learn
page,compilation of words,virtual memory
word,number of bits depends on sys arch, compilation of letters
```

Explanation:

dict.txt contains a list of words with 2 of its meanings. This file contains the data in csv format (comma separated values). This program searches for a word in this list. The producer parses the file and puts the word structure in the buffer. The consumer pulls the words from the buffer and checks whether the pulled word is the key.

Output: