Adhesh Reghu Kumar
COE18B001

# Operating Systems Lab

## Lab Assignment - 7

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

### Logic Used

This exercise is just implementation of the producer consumer code discussed in class. In this the producer produces a data element, student record from a file in this case, one at a time and stores it in a global buffer array. The consumer reads the produced record from the buffer array and prints it.

The student records are stored in a file "studentData.txt" which contains the student rollNo and name in particular format. The producer code reads from this file one record at a time and updates the buffer array (of student struct).

### Code

```
/*****************************************
* AUTHOR : AdheshR *
* Problem Description: Implement the Producer-Consumer code where one thread produces some data whereas another thread consumes it.
The implementation will illustrate the possible fallacies that can arrive due to asynchronous access of data and reiterate the need for synchronous data access in a multi-threaded setup.*
* Remark: This producer consumer works fine as no concurrency seems to happen between ++ (increment) or -- (decrement). They seem to happen atomically. To see the failing, maybe try to do the increment in steps and sleep in between.
*****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>
#include <string.h>
#define MAX 4096
#define BUFFER_SIZE 10   // indicates the size of buffer in a bounded-buffer setup
```

```c
// Define global variables to be accessed to and from.
struct student
{
    char rollNo[MAX];
    char name[MAX];
};


int counter = 0;
int in = 0;
int out = 0;

// Define the buffer array
struct student buffer[BUFFER_SIZE];

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

int main(int argc, char *argv[])
{
 /*-- Main accepts the filename of student record as input argument from command line
        Validate the arguments -- */

    if(argc!=2)
    {
       printf("Invalid arguments.\n");
       exit(EXIT_FAILURE);
    }

    // Validate the file
    FILE *fp;
    if((fp = fopen(argv[1],"r")) == NULL)
    {
       printf("File does not exist.\n");
       exit(EXIT_FAILURE);
    }

    // Create threads for the producer and consumer
    pthread_t tid[2];

    // Create and init attr to default
    pthread_attr_t attr;
    pthread_attr_init(&attr);
```

```
    // Call the producer thread and also pass the file name
    pthread_create(&tid[0],&attr,producer,argv[1]);

    // Call the consumer thread
    pthread_create(&tid[1],&attr,consumer,NULL);

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

    return 0;
}

void *producer(void *args)
{
    // extract file name from args
    char filename[MAX];
    strcpy(filename,(char *)args);

    // Open file to be read
    FILE *fp = fopen(filename,"r");
    char line[4096];
    int len = 0;

      // Define struct for nextProduced student record
    struct student nextProduced;

    /* -- Producer Code -- */
    while(1)
    {
       while(counter == BUFFER_SIZE);      // buffer is full. wait for consumption

       // Get student data from file
       memset(line,0,4096);
       if(fscanf(fp, "%[^\n]\n", line) == EOF)
             break;

       // Split the record to name and rollno
       strcpy(nextProduced.rollNo,strtok(line," "));
       strcpy(nextProduced.name,strtok(NULL,"\n"));

       // Produce next student item
       buffer[in] = nextProduced;
```

```
        // Increment in pointer & increment counter
        in = (in + 1)%BUFFER_SIZE;
        counter++;
    }
}

void *consumer(void *args)
{
    struct student nextConsumed;

    /* -- Consumer Code -- */
    while(1)
    {
        while(counter == 0);          // buffer is empty. wait for production

        // Consume next student item
        nextConsumed =  buffer[out];

        // Print data in consumed data
        printf("[CONSUMER] Roll No: %s\n",nextConsumed.rollNo);
        printf("[CONSUMER] Name: %s\n",nextConsumed.name);

        // Increment out pointer & decrement counter
        out = (out + 1)%BUFFER_SIZE;

        counter--;
    }
}
```

**Output**

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week7$ gcc -o out 1_produceConsume.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week7$ ./out studentData.txt
[CONSUMER] Roll No: COE18B001
[CONSUMER] Name: Adhesh Reghu Kumar
[CONSUMER] Roll No: COE18B003
[CONSUMER] Name: Akshun Dubey
[CONSUMER] Roll No: COE18B004
[CONSUMER] Name: Anant Patel
[CONSUMER] Roll No: COE18B005
[CONSUMER] Name: Aparajith R
[CONSUMER] Roll No: COE18B006
[CONSUMER] Name: Ashwin BS
[CONSUMER] Roll No: CED18I005
[CONSUMER] Name: R Mukesh
```

2. Extend the producer consumer simulation in Q1 to sync access of critical data using Peterson's  algorithm.

**Logic Used**

The implementation of this is pretty straightforward. We just extend the last problem and add Peterson's algorithm to ensure synchronization.

**Code**

```
/*****************************************
* AUTHOR : AdheshR *
* Problem Description: Implement the Producer-Consumer code but with Petersons synchronization algorithm.*
*****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>
#include <string.h>
#define MAX 4096

#define BUFFER_SIZE 10                    // indicates the size of buffer in a bounded-buffer setup

// Define global variables to be accessed to and from.
struct student
{
    char rollNo[MAX];
    char name[MAX];
};

int counter = 0;
int in = 0;
int out = 0;

// Define the buffer array
struct student buffer[BUFFER_SIZE];

// Define variables for peterson algo
```

```c
int flag[2];
int turn;

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

int main(int argc, char *argv[])
{
    /*-- Main accepts the filename of student record as input argument from
command line
        Validate the arguments -- */

    if(argc!=2)
    {
       printf("Invalid arguments.\n");
       exit(EXIT_FAILURE);
    }

    // Validate the file
    FILE *fp;
    if((fp = fopen(argv[1],"r")) == NULL)
    {
       printf("File does not exist.\n");
       exit(EXIT_FAILURE);
    }

    // Create threads for the producer and consumer
    pthread_t tid[2];

    // Create and init attr to default
    pthread_attr_t attr;
    pthread_attr_init(&attr);

    // Call the producer thread and also pass the file name
    pthread_create(&tid[0],&attr,producer,argv[1]);

    // Call the consumer thread
    pthread_create(&tid[1],&attr,consumer,NULL);

    pthread_join(tid[0],NULL);
    pthread_join(tid[1],NULL);
    return 0;
}
```

```c
void *producer(void *args)
{
    // extract file name from args
    char filename[MAX];
    strcpy(filename,(char *)args);

    // Open file to be read
    FILE *fp = fopen(filename,"r");
    char line[4096];
    int len = 0;

      // Define struct for nextProduced student record
    struct student nextProduced;

    /* -- Producer Code -- */
    while(1)
    {
       while(counter == BUFFER_SIZE);  // buffer is full. wait for consumption

       /* -- Entry Section -- */
       flag[0] = 1,turn = 1;

       while(flag[1] == 1 && turn == 1);

       /* -- Critical Section -- */

       // Get student data from file
       memset(line,0,4096);
       if(fscanf(fp, "%[^\n]\n", line) == EOF)
       {
             flag[0] = 0;
             break;
       }

       // Split the record to name and rollno
       strcpy(nextProduced.rollNo,strtok(line," "));
       strcpy(nextProduced.name,strtok(NULL,"\n"));

       // Produce next student item
       buffer[in] = nextProduced;

       // Increment in pointer & increment counter
```

```
        in = (in + 1)%BUFFER_SIZE;
        counter++;

        /* -- Exit Section -- */
        flag[0] = 0;
    }
}

void *consumer(void *args)
{
    struct student nextConsumed;

    /* -- Consumer Code -- */
    while(1)
    {
        while(counter == 0);                            // buffer is empty. wait
for production

        /* -- Entry Section -- */
        flag[1] = 1, turn = 0;

        while(flag[0] == 1 && turn == 0);

        /* -- Critical Section -- */

        // Consume next student item
        nextConsumed =  buffer[out];

        // Print data in consumed data
        printf("Roll No: %s\n",nextConsumed.rollNo);
        printf("Name: %s\n",nextConsumed.name);

        // Increment out pointer & decrement counter
        out = (out + 1)%BUFFER_SIZE;
        counter--;

        /* -- Exit Section -- */
        flag[1] = 0;
    }
}
```

**Output**

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week7$ gcc -o out 2_peterson.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week7$ ./out studentData.txt
Roll No: COE18B001
Name: Adhesh Reghu Kumar
Roll No: COE18B003
Name: Akshun Dubey
Roll No: COE18B004
Name: Anant Patel
Roll No: COE18B005
Name: Aparajith R
Roll No: COE18B006
Name: Ashwin BS
Roll No: CED18I005
Name: R Mukesh
```

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.

**Logic Used**

In this question we have to implement a dictionary using the producer-consumer code. The producer reads the words with their primary and secondary meaning from an input file "dictRecords.txt" and stores them into a dictionary one at a time. The producer after inserting the word and its meanings to the dictionary, then passes the word (alone) to the consumer side. The consumer receives the word and searches through the dictionary for its meaning (primary and secondary) and displays them on STDOUT.

The synchronization of shared access of the word is achieved through Peterson's solution (which is discussed in Question 2).

Let us take a look at the "dictRecords.txt" file.

dictRecords.txt | 1_produceConsume.c | 2_peterson.c | 3_dict.c

```
cup;a small bowl-shaped container;an ornamental trophy in the form of a cup;
bag;a flexible container with an opening at the top, used for carrying things;one's particular interest or taste.;
run;move at a speed faster than a walk;pass or cause to pass quickly in a particular direction;
journey;an act of travelling from one place to another;travel somewhere;
roar;a full, deep, prolonged cry uttered by a lion or other large wild animal;a loud outburst of laughter;
jog;run at a steady gentle pace;nudge or knock slightly;
asd;entle pace;nudge or knock slightly;
```

Here the format of each data record is:
*Word; Primary meaning; Secondary meaning*

**Code:**

```
/*****************************************
* AUTHOR : AdheshR *
* Problem Description: Dictionary of atleast 20 words*
*****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>
#include <string.h>
#define MAX 4096
#define RANGE 100

#define BUFFER_SIZE 10  // indicates the size of buffer in a bounded-buffer setup

// Define global variables to be accessed to and from.
struct dict
{
    char word[MAX];
    char pri_mean[MAX];
    char sec_mean[MAX];
};

int counter = 0;
int in = 0;
int out = 0;

// Define the buffer array
char *buffer[BUFFER_SIZE];

// Define variables for peterson algo
int flag[2];
int turn;

// Define a linear dictionary
int pos = 0;
struct dict dictionary[RANGE];
```

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

int main(int argc, char *argv[])
{
    /*-- Main accepts the filename of student record as input argument from
command line
        Validate the arguments -- */

    if(argc!=2)
    {
       printf("Invalid arguments.\n");
       exit(EXIT_FAILURE);
    }

    // Validate the file
    FILE *fp;
    if((fp = fopen(argv[1],"r")) == NULL)
    {
       printf("File does not exist.\n");
       exit(EXIT_FAILURE);
    }

    // Create threads for the producer and consumer
    pthread_t tid[2];

    // Create and init attr to default
    pthread_attr_t attr;
    pthread_attr_init(&attr);

    // Call the producer thread and also pass the file name
    pthread_create(&tid[0],&attr,producer,argv[1]);

    // Call the consumer thread
    pthread_create(&tid[1],&attr,consumer,NULL);

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

    return 0;
}
```

```
void *producer(void *args)
{
    // extract file name from args
    char filename[MAX];
    strcpy(filename,(char *)args);

    // Open file to be read
    FILE *fp = fopen(filename,"r");
    char line[4096];
    int len = 0;

      // Define struct for nextProduced student record
    struct dict nextProduced;

    /* -- Producer Code -- */
    while(1)
    {
       while(counter == BUFFER_SIZE);  // buffer is full. wait for consumption

       /* -- Entry Section -- */
       flag[0] = 1,turn = 1;

       while(flag[1] == 1 && turn == 1);

       /* -- Critical Section -- */

       // Get word record from file
       memset(line,0,4096);
       if(fscanf(fp, "%[^\n]\n", line) == EOF)
       {
             flag[0] = 0;
             break;
       }

       // Split the dictionary record
       strcpy(nextProduced.word,strtok(line,";"));
       strcpy(nextProduced.pri_mean,strtok(NULL,";"));
       strcpy(nextProduced.sec_mean,strtok(NULL,"\n"));

       // Insert into dictionary
       printf("[PRODUCER] Word: %s\n",nextProduced.word);
       strcpy(dictionary[pos].word,nextProduced.word);
       printf("[PRODUCER] Primary meaning: %s\n",nextProduced.pri_mean);
```

```
        strcpy(dictionary[pos].pri_mean,nextProduced.pri_mean);
        printf("[PRODUCER] Secondary meaning: %s\n",nextProduced.sec_mean);
        strcpy(dictionary[pos++].sec_mean,nextProduced.sec_mean);

        // Produce next word
        buffer[in] = nextProduced.word;

        // Increment in pointer & increment counter
        in = (in + 1)%BUFFER_SIZE;
        counter++;

        /* -- Exit Section -- */
        flag[0] = 0;
        sleep(1);
    }
}

void *consumer(void *args)
{
    struct dict nextConsumed;

    /* -- Consumer Code -- */
    while(1)
    {
        while(counter == 0);          // buffer is empty. wait for production

        /* -- Entry Section -- */
        flag[1] = 1, turn = 0;

        while(flag[0] == 1 && turn == 0);

        /* -- Critical Section -- */

        // Consume next student item
        strcpy(nextConsumed.word,buffer[out]);

        // Only the word gets passed from the producer to the consumer
        printf("[CONSUMER] Word: %s\n",nextConsumed.word);

        // Find the primary meaning and secondary meaning from the global
dictionary - linear search
        int i=0;
        for(i=0;i<RANGE;++i)
```

```
        {
                if(strcmp(nextConsumed.word,dictionary[i].word) == 0)
                {
                        strcpy(nextConsumed.pri_mean,dictionary[i].pri_mean);
                        strcpy(nextConsumed.sec_mean,dictionary[i].sec_mean);
                        break;
                }
        }

        printf("[CONSUMER] Primary meaning: %s\n",nextConsumed.pri_mean);
        printf("[CONSUMER] Secondary meaning: %s\n",nextConsumed.sec_mean);

        // Increment out pointer & decrement counter
        out = (out + 1)%BUFFER_SIZE;
        counter--;

        /* -- Exit Section -- */
        flag[1] = 0;
    }
}
```

## Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week7$ ./out dictRecords.txt
[PRODUCER] Word: cup
[PRODUCER] Primary meaning: a small bowl-shaped container
[PRODUCER] Secondary meaning: an ornamental trophy in the form of a cup;
[CONSUMER] Word: cup
[CONSUMER] Primary meaning: a small bowl-shaped container
[CONSUMER] Secondary meaning: an ornamental trophy in the form of a cup;
[PRODUCER] Word: bag
[PRODUCER] Primary meaning: a flexible container with an opening at the top, used for carrying things
[PRODUCER] Secondary meaning: one's particular interest or taste.;
[CONSUMER] Word: bag
[CONSUMER] Primary meaning: a flexible container with an opening at the top, used for carrying things
[CONSUMER] Secondary meaning: one's particular interest or taste.;
[PRODUCER] Word: run
[PRODUCER] Primary meaning: move at a speed faster than a walk
[PRODUCER] Secondary meaning: pass or cause to pass quickly in a particular direction;
[CONSUMER] Word: run
[CONSUMER] Primary meaning: move at a speed faster than a walk
[CONSUMER] Secondary meaning: pass or cause to pass quickly in a particular direction;
[PRODUCER] Word: journey
[PRODUCER] Primary meaning: an act of travelling from one place to another
[PRODUCER] Secondary meaning: travel somewhere;
[CONSUMER] Word: journey
[CONSUMER] Primary meaning: an act of travelling from one place to another
[CONSUMER] Secondary meaning: travel somewhere;
[PRODUCER] Word: roar
[PRODUCER] Primary meaning: a full, deep, prolonged cry uttered by a lion or other large wild animal
[PRODUCER] Secondary meaning: a loud outburst of laughter;
[CONSUMER] Word: roar
[CONSUMER] Primary meaning: a full, deep, prolonged cry uttered by a lion or other large wild animal
[CONSUMER] Secondary meaning: a loud outburst of laughter;
[PRODUCER] Word: jog
[PRODUCER] Primary meaning: run at a steady gentle pace
[PRODUCER] Secondary meaning: nudge or knock slightly;
[CONSUMER] Word: jog
[CONSUMER] Primary meaning: run at a steady gentle pace
[CONSUMER] Secondary meaning: nudge or knock slightly;
[PRODUCER] Word: asd
[PRODUCER] Primary meaning: entle pace
[PRODUCER] Secondary meaning: nudge or knock slightly;
[CONSUMER] Word: asd
[CONSUMER] Primary meaning: entle pace
[CONSUMER] Secondary meaning: nudge or knock slightly;
```

---