Christo Pettas

Project Objectives:

The purpose is to learn how to use semaphores to protect a limited size resource. A circular buffer with *n* positions (each position stores 1 character) is to be used to communicate information between two threads (producer and consumer).

Description:
Using a buffer with limited size of 15. The buffer needs to run and wait if the buffer is full from being filled by the producer. While waiting the consumer thread will read buffer so no characters will be over written do to the limited amount of space on the buffer.

This is done by using 3 semaphores to control the buffer flow so everything will run as needed. First you will have the mutex semaphores that will protect critical section. Mutex will be initialized to one. Semaphore empty will be created to stop producer from over writing buffer if consumer has not read item in first position. Empty semaphore will be initialized to 15 (size of buffer). Last semaphore will be to prevent consumer thread from reading an item over again from which it has already read in the past. It will be called full. Full will be initialized to zero.

Producer:
The producer will run till file is finished reading and placed on buffer and then will place a special character onto the end of the buffer to let the consumer thread know that it is finished reading the file.

```
void *producer(void *arg){
                    // printf("Producer_start \n");

                     int j= 0;

                    char newChar;
                    int i = 0;
                    FILE* fp;
                    fp= fopen("mytest.dat", "r");
                    while(fscanf(fp,"%c",&newChar) != EOF){

                    sem_wait(&emp);
                    sem_wait(&mutex);

                       //add the item to the  buffer
                     if(i >= 15){
                       i =0;
                       }
                    buffer[i]= newChar;
                    i++;
```

```
                sem_post(&mutex);
                sem_post(&full);
                }

                buffer[i]='*';
    }
```

Consumer: Consumer thread will read the off the buffer and send to consol to be displayed. Consumer thread will terminate when the special character has been reached on the end of the buffer.

```
void* consumer(){
int i=0;
char ch;
while(buffer[i] != '*'){
        sem_wait(&full);
        sem_wait(&mutex);


        //sleep(1);
        ch = buffer[i];
        if(i == 14)
                i=0;
        else
                i++;
   printf("%c", ch);
        sem_post(&mutex);
        sem_post(&emp);


        //printf("%c", ch);
}
}
```

Main: Main will just initialize the semaphores and then create the threads to run the consumer and the producer and then wait till both are finished to terminate the program.

```
main()
{
      int val =0;
      int err;
      sem_init(&mutex, 0, 1);
      sem_init(&emp, 0, 15);
      sem_init(&full, 0, 0);
```

```
    err = pthread_create(&(producerThread), NULL, &producer, NULL);
    if (err != 0)
        printf("\ncan't create thread :[%s]", strerror(err));
    else
        printf("\n Thread created successfully1\n");

      err = pthread_create(&(consumerThread), NULL, &consumer, NULL);
    if (err != 0)
        printf("\ncan't create thread :[%s]", strerror(err));
    else
        printf("\n Thread created successfully2\n");


pthread_join( producerThread,NULL);
pthread_join( consumerThread,NULL);

  }
```

Conclusion: I have learned how to work with different threads to allow multiple things running at one time. I have seen how powerful threading can be but also understand that protecting the shared memory of the process is also very important