HARISH K M
COE18B066

Operating System
Assignment-7

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

Approach:

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.

Code:

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

void *producer(void *param);
void *consumer(void *param);

int buf[N];
int in=0,out=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 *param)
{
      int data=3,i=0;
      while(i<10)
      {
            while((in+1)%N==out);

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

```
void *consumer(void *param)
{
      int data;
      while(1)
      {
            while(in==out);

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

Output:

```
harish@harish-ubuntu:~/Desktop/os/asst7$ gcc pro_con.c -lpthread
harish@harish-ubuntu:~/Desktop/os/asst7$ ./a.out
producing index 0
producing index 1
producing index 2
consuming index 0
consuming index 1
consuming index 2
producing index 3
producing index 4
producing index 0
producing index 1
consuming index 3
consuming index 4
consuming index 0
consuming index 1
producing index 2
producing index 3
producing index 4
consuming index 2
consuming index 3
consuming index 4
^C
harish@harish-ubuntu:~/Desktop/os/asst7$
```

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

Approach:

This program is an extension to the previous program. Petersons algorithm is used to attain synchronization between the producer and the consumer. This is because, the producer acquires lock, produces an item and increments the counter variable everytime and releases its lock and then the consumer acquires lock consumes,decrements the counter and unlocks. As the operation upon

the counter variable is not atomic, its a critical section operation, which we resolve by using Peterson's solution for 2 processes.

Code:

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

void *producer(void *param);
void *consumer(void *param);

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 *param)
{
      int data=3,i=0;
      while(i<10)
      {
            while(counter==N);

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

void *consumer(void *param)
{
      int data,i=0;

      while(i<10)
      {
            while(counter==0);

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

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.

Approach:

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.
In the consumer N number of lines are read from dictionary and put into a buffer and the consumer checks for the word in N number of lines when given a lock once.This happens until all lines are exhausted.

Code:

```
#include<stdio.h>
#include<sys/wait.h>
#include<pthread.h>
#include<string.h>
#define N 5
void *producer(void *param);
void *consumer(void *param);

struct dict
```

```
{
     char word[30];
     char mean1[30],mean2[30];
}buf[N];

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

int in=0,out=0,end=0;
int j=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];
     strcpy(search_word,"book");

     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 *param)
{
     int i=0;
     FILE *fptr=fopen("dict.txt","r");
```

```
	while(1)
	{

		lock(0);
		j=0;
		while(j<5)
		{
			//printf("@\n");
			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].mean1[k++]=ch;
			}
			buf[j].mean1[k]='\0';
			k=0;
			while((ch=fgetc(fptr))!='\n')
			{
				if(ch==EOF)
				{
					buf[j].mean2[k]='\0';
					end=1;
					unlock(0);
					return NULL;
				}
				buf[j].mean2[k++]=ch;
			}
			buf[j].mean2[k]='\0';
			j++;

		}
		int k;

		++i;
		unlock(0);
	}

	pthread_exit(0);
}

void *consumer(void *param)
{
	int i=0;

	while(1)
```

```
	{

		lock(1);
		//printf("cons");
		int k=0;
		while(k<j)
		{
			//printf("#");
			if(strcmp(buf[k].word,search_word)==0)
			{
				printf("Word found:\n");
				puts(buf[k].word);
				printf("Meaning 1:\n");
				puts(buf[k].mean1);
				printf("Meaning 2:\n");
				puts(buf[k].mean2);
				unlock(1);
				return NULL;
			}
			k++;
		}

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

	}

	pthread_exit(0);
}
```

Sample dict.txt

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
letter,something in envelope,an alphabet

Output:

```
harish@harish-ubuntu:~/Desktop/os/asst7$ gcc dict.c -lpthread
harish@harish-ubuntu:~/Desktop/os/asst7$ ./a.out page
Word found:
page
Meaning 1:
compilation of words
Meaning 2:
virtual memory
harish@harish-ubuntu:~/Desktop/os/asst7$ ./a.out class
Word found:
class
Meaning 1:
thing in c
Meaning 2:
where students learn
harish@harish-ubuntu:~/Desktop/os/asst7$ ./a.out light
Word not found!
harish@harish-ubuntu:~/Desktop/os/asst7$
```