Synchronization - I

COM301-P Assignment - 7

G S Santhosh Raghul
COE18B045

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

code :

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

#define BUFFER_SIZE 5
#define FULL (in+1)%BUFFER_SIZE==out
#define EMPTY in==out
#define MAX 10

void* producer(void* par);
void* consumer(void* par);

int buf[BUFFER_SIZE],in=0,out=0,done=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;
}
```

```c
void* producer(void* par)
{
   int i=-1;
   while(i<MAX) // producer produces whole numbers from 0 to MAX
   {
       while(FULL); // wait while full - spin wait
       buf[in]=++i; // produce data and put into buffer
       printf("produced data %3d   at index %3d\n",i,in);
       in=(in+1)%BUFFER_SIZE; // update in index
   }
   done=1; // set done to 1 when the producer is done producing
   pthread_exit(0);
}

void* consumer(void* par)
{
   int data;
   while(1)
   {
       while(EMPTY); // wait while empty - spin wait
       data=buf[out]; // consume data from buffer
       printf("consumed data %3d   at index %3d\n",data,out);
       out=(out+1)%BUFFER_SIZE; // update out index
       if(done && EMPTY) // if the producer is done and the buffer has been emptied
           break; // end the loop since all items have been consumed
   }
   pthread_exit(0);
}
```

output :

```
santi@edith:~/OS/L7$ gcc producer-consumer.c
 -pthread
santi@edith:~/OS/L7$ ./a.out
produced data   0   at index   0
produced data   1   at index   1
produced data   2   at index   2
produced data   3   at index   3
consumed data   0   at index   0
consumed data   1   at index   1
consumed data   2   at index   2
consumed data   3   at index   3
produced data   4   at index   4
consumed data   4   at index   4
produced data   5   at index   0
produced data   6   at index   1
produced data   7   at index   2
produced data   8   at index   3
consumed data   5   at index   0
consumed data   6   at index   1
consumed data   7   at index   2
consumed data   8   at index   3
produced data   9   at index   4
produced data  10   at index   0
consumed data   9   at index   4
consumed data  10   at index   0
santi@edith:~/OS/L7$
```

```
santi@edith:~/OS/L7$
santi@edith:~/OS/L7$
santi@edith:~/OS/L7$ ./a.out
produced data   0   at index   0
produced data   1   at index   1
produced data   2   at index   2
produced data   3   at index   3
consumed data   0   at index   0
consumed data   1   at index   1
consumed data   2   at index   2
consumed data   3   at index   3
produced data   4   at index   4
produced data   5   at index   0
produced data   6   at index   1
produced data   7   at index   2
consumed data   4   at index   4
consumed data   5   at index   0
consumed data   6   at index   1
consumed data   7   at index   2
produced data   8   at index   3
produced data   9   at index   4
produced data  10   at index   0
consumed data   8   at index   3
consumed data   9   at index   4
consumed data  10   at index   0
santi@edith:~/OS/L7$
```

```
santi@edith:~/OS/L7$
santi@edith:~/OS/L7$
santi@edith:~/OS/L7$ ./a.out
produced data   0   at index   0
produced data   1   at index   1
produced data   2   at index   2
produced data   3   at index   3
consumed data   0   at index   0
consumed data   1   at index   1
consumed data   2   at index   2
consumed data   3   at index   3
produced data   4   at index   4
produced data   5   at index   0
produced data   6   at index   1
produced data   7   at index   2
consumed data   4   at index   4
consumed data   5   at index   0
consumed data   6   at index   1
consumed data   7   at index   2
produced data   8   at index   3
produced data   9   at index   4
produced data  10   at index   0
consumed data   8   at index   3
consumed data   9   at index   4
consumed data  10   at index   0
santi@edith:~/OS/L7$
```

```
santi@edith:~/OS/L7$
santi@edith:~/OS/L7$
santi@edith:~/OS/L7$ ./a.out
produced data   0   at index   0
produced data   1   at index   1
produced data   2   at index   2
produced data   3   at index   3
consumed data   0   at index   0
produced data   4   at index   4
consumed data   1   at index   1
consumed data   2   at index   2
consumed data   3   at index   3
consumed data   4   at index   4
produced data   5   at index   0
produced data   6   at index   1
produced data   7   at index   2
produced data   8   at index   3
consumed data   5   at index   0
consumed data   6   at index   1
consumed data   7   at index   2
consumed data   8   at index   3
produced data   9   at index   4
consumed data   9   at index   4
produced data  10   at index   0
consumed data  10   at index   0
santi@edith:~/OS/L7$
```

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

code :

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

#define BUFFER SIZE 5
#define FULL (in+1)%BUFFER SIZE==out
#define EMPTY in==out
#define MAX 10
#define PRODUCER 0
#define CONSUMER 1
#define FALSE 0
#define TRUE 1

void* producer(void* par);
void* consumer(void* par);

int in=0,out=0,turn=0,flag[2]={0,0},buf[BUFFER SIZE],end=FALSE;


int main()
{
   printf("note: actual order of execution may vary from the order of printf\n");
   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* par)
{
   int i=-1,index;
   while(i<MAX) // producer produces whole numbers from 0 to MAX
   {
   // wait while full
       while(FULL);
   // petersons solution lock for mutual exclusion
       flag[PRODUCER]=TRUE;
       turn=CONSUMER;
       while (flag[CONSUMER] && turn==CONSUMER);
   // critical section - produce data in this case
       buf[in]=++i;
       index=in; // save index for printing
       in=(in+1)%BUFFER SIZE;
   // unlock
       flag[PRODUCER]=FALSE;
   // remainder section
       printf("produced data %3d   at index %3d\n",i,index);
   }
   pthread exit(0);
}

void* consumer(void* par)
{
   int data,index;
   while(data<MAX)
   {
   // wait while empty
       while(EMPTY);
   // petersons solution lock for mutual exclusion
       flag[CONSUMER]=TRUE;
       turn=PRODUCER;
       while (flag[PRODUCER] && turn==PRODUCER);
   // critical section - consume data in this case
       data=buf[out];
       index=out; // save index for printing
       out=(out+1)%BUFFER_SIZE;
```

```
    // unlock
        flag[CONSUMER]=FALSE;
    // remainder section
        printf("consumed data %3d   at index %3d\n",data,index);
        if(EMPTY && end)
            break;
    }
    pthread exit(0);
}
```

output :

Here, the order of printf may be different from the order of production or consumption. For example, “consumed data 5” message got printed before the “produced data 5” message. To get the correct order, we may put the printf statement also inside the critical section.

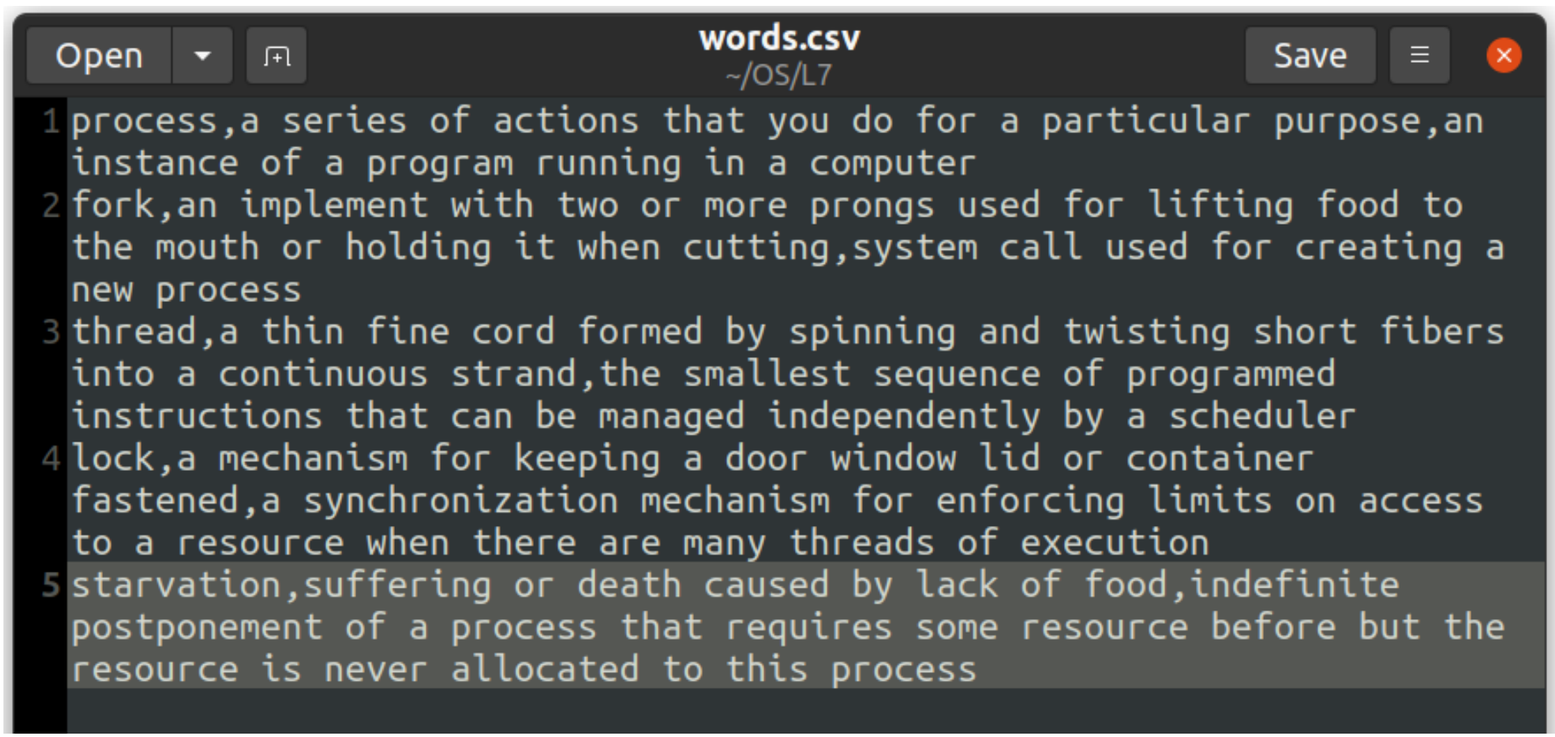
```
santi@edith:~/OS/L7$ gcc petersons-solution.c -pthread
santi@edith:~/OS/L7$ ./a.out
note: actual order of execution may vary from the order of printf
produced data   0   at index   0
produced data   1   at index   1
produced data   2   at index   2
produced data   3   at index   3
produced data   4   at index   4
consumed data   0   at index   0
consumed data   1   at index   1
consumed data   2   at index   2
consumed data   3   at index   3
consumed data   4   at index   4
consumed data   5   at index   0
produced data   5   at index   0
produced data   6   at index   1
produced data   7   at index   2
produced data   8   at index   3
produced data   9   at index   4
produced data  10   at index   0
consumed data   6   at index   1
consumed data   7   at index   2
consumed data   8   at index   3
consumed data   9   at index   4
consumed data  10   at index   0
santi@edith:~/OS/L7$
```

```
santi@edith:~/OS/L7$ ./a.out
note: actual order of execution may vary from the order of printf
produced data   0   at index   0
produced data   1   at index   1
produced data   2   at index   2
produced data   3   at index   3
produced data   4   at index   4
consumed data   0   at index   0
produced data   5   at index   0
consumed data   1   at index   1
consumed data   2   at index   2
consumed data   3   at index   3
consumed data   4   at index   4
consumed data   5   at index   0
consumed data   6   at index   1
produced data   6   at index   1
produced data   7   at index   2
produced data   8   at index   3
consumed data   7   at index   2
consumed data   8   at index   3
consumed data   9   at index   4
produced data   9   at index   4
produced data  10   at index   0
consumed data  10   at index   0
santi@edith:~/OS/L7$
```

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.

Note: This can be implemented using either Mutex locks or Petersons algorithm.

I have used Peterson's algorithm. Words are stored in words.csv file. It’s content is shown below.



code :

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

#define BUFFER_SIZE 5
#define FULL (in+1)%BUFFER_SIZE==out
#define EMPTY in==out
#define WORD_SIZE 16
#define MEANING_SIZE 128
#define MAX 10
#define PRODUCER 0
#define CONSUMER 1
#define FALSE 0
#define TRUE 1

struct dict
{
   char word[WORD_SIZE],meaning_1[MEANING_SIZE],meaning_2[MEANING_SIZE];
};

void* producer(void* par);
void* consumer(void* par);

int in=0,out=0,turn=0,flag[2]={0,0},end=FALSE,found=FALSE;
struct dict buf[BUFFER_SIZE];
char search_key[30];

int main(int argc,char* argv[])
{
   if(argc!=2)
   {
       fprintf(stderr,"invalid usage!\ncorrect usage: %s word\n",argv[0]);
       return 1;
   }
   printf("note: actual order of execution may vary from the order of printf\n");
```

```
   pthread t tid[2];
   strcpy(search key,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);
   if(!found)

   {
       fprintf(stderr,"\033[1;31mword not found\n");
       return 1;
   }
   return 0;
}

void* producer(void* par)
{
   char ch;
   int k,index;
   FILE* dictionary file;
   if((dictionary file=fopen("words.csv","r"))==NULL)
       perror("file not found");
   while(TRUE)
   {
   // wait while full
       while(FULL);

   // petersons solution lock for mutual exclusion

       flag[PRODUCER]=TRUE;
       turn=CONSUMER;
       while(flag[CONSUMER] && turn==CONSUMER);
```

```
// critical section - produce data in this case
    // get word
    k=0;
    while((ch=fgetc(dictionary file))!=',')
        buf[in].word[k++]=ch;
    buf[in].word[k]='\0';

    // get meaning 1
    k=0;
    while((ch=fgetc(dictionary file))!=',')
        buf[in].meaning 1[k++]=ch;
    buf[in].meaning 1[k]='\0';
    // get meaning 2
    k=0;


    while((ch=fgetc(dictionary_file))!=EOF)
    {
        if(ch=='\n')
            break;
        buf[in].meaning 2[k++]=ch;
    }
    buf[in].meaning 2[k]='\0';
    index=in;
    in=(in+1)%BUFFER SIZE;

// unlock
    flag[PRODUCER]=FALSE;

// remainder section
    printf("produced \033[1;36m'%s'\033[0m at index %2d\n",buf[index].word,index);
    if(ch==EOF || found)
        break;
}
end=TRUE;
pthread exit(0);
}
```

```
void* consumer(void* par)
{
   struct dict temp;
   int index;
   while(TRUE)
   {
   // wait while empty
       while(EMPTY && !end);
   // petersons solution lock for mutual exclusion
       flag[CONSUMER]=TRUE;
       turn=PRODUCER;
       while(flag[PRODUCER] && turn==PRODUCER);
   // critical section - consume data in this case
       temp=buf[out];
       index=out;
       out=(out+1)%BUFFER SIZE;
   // unlock
       flag[CONSUMER]=FALSE;
   // remainder section
       printf("consumed \033[1;35m'%s'\033[0m at index %2d\n",temp.word,index);
       if(strcmp(temp.word,search key)==0)
       {
   printf("word \033[1;33m'%s'\033[0m found\n      \033[1m├──── %s\n      └────
%s\n\033[0m",temp.word,temp.meaning 1,temp.meaning 2);
           found=TRUE;
       }
       if(found || (EMPTY && end)) // if the word has been found or all the words have been consumed
           break; // exit
   }
   pthread exit(0);
}
```

output :

```
santi@edith:~/OS/L7$ gcc dictionary.c -pthread
santi@edith:~/OS/L7$ ./a.out
invalid usage!
correct usage: ./a.out word
santi@edith:~/OS/L7$ ./a.out hello
note: actual order of execution may vary from the order of printf
produced 'process' at index  0
consumed 'process' at index  0
consumed 'fork' at index  1
produced 'fork' at index  1
produced 'thread' at index  2
consumed 'thread' at index  2
produced 'lock' at index  3
consumed 'lock' at index  3
produced 'starvation' at index  4
consumed 'starvation' at index  4
word not found
santi@edith:~/OS/L7$ ./a.out thread
note: actual order of execution may vary from the order of printf
produced 'process' at index  0
consumed 'process' at index  0
consumed 'fork' at index  1
produced 'fork' at index  1
produced 'thread' at index  2
consumed 'thread' at index  2
produced 'lock' at index  3
word 'thread' found
      ├─── a thin fine cord formed by spinning and twisting short fibers into a continuous strand
      └─── the smallest sequence of programmed instructions that can be managed independently by a scheduler
produced 'starvation' at index  4
santi@edith:~/OS/L7$
```

```
santi@edith:~/OS/L7$ ./a.out fork
note: actual order of execution may vary from the order of printf
produced 'process' at index  0
produced 'fork' at index  1
produced 'thread' at index  2
produced 'lock' at index  3
consumed 'process' at index  0
consumed 'fork' at index  1
word 'fork' found
      ├──── an implement with two or more prongs used for lifting food to the mouth or holding it when cutting
      └──── system call used for creating a new process
produced 'starvation' at index  4
santi@edith:~/OS/L7$ ./a.out starvation
note: actual order of execution may vary from the order of printf
produced 'process' at index  0
consumed 'process' at index  0
consumed 'fork' at index  1
produced 'fork' at index  1
produced 'thread' at index  2
consumed 'thread' at index  2
consumed 'lock' at index  3
produced 'lock' at index  3
produced 'starvation' at index  4
consumed 'starvation' at index  4
word 'starvation' found
      ├──── suffering or death caused by lack of food
      └──── indefinite postponement of a process that requires some resource before but the resource is never allocated to this process
santi@edith:~/OS/L7$
```