OPERATING SYSTEMS LAB-PROJECT

Student Name:- Charapalli Tharun

Student ID:- 11803413

Email Address:- charapalli.11803413@lpu.in

GitHub Link:- https://github.com/Tharun000/OS-PROJECT

Problem:-1

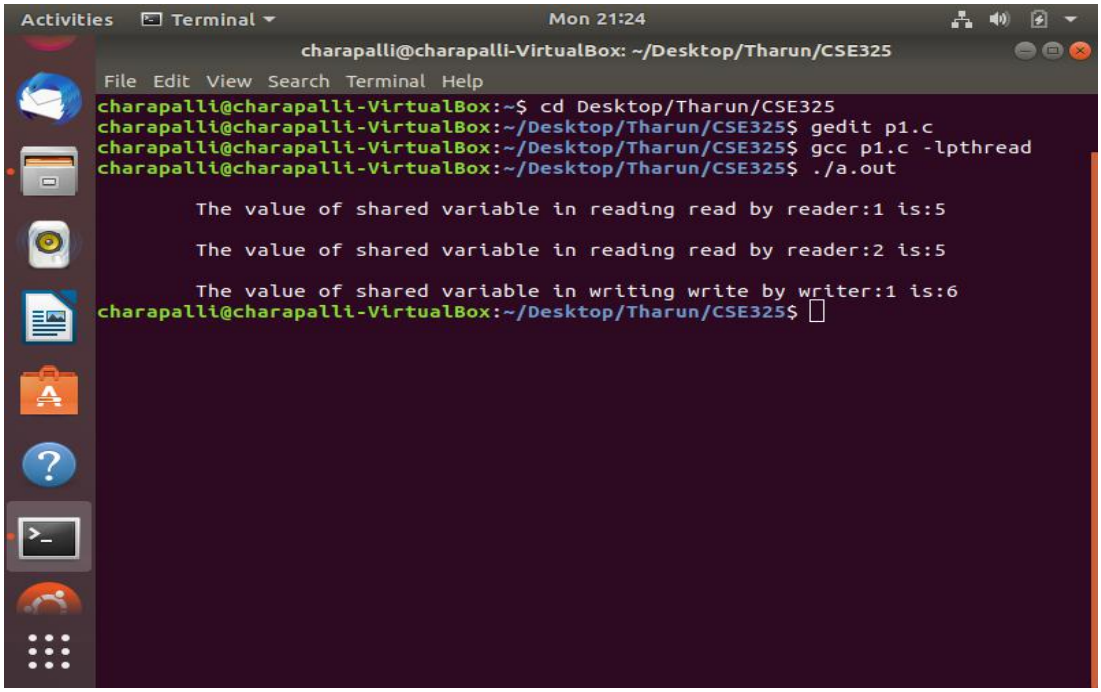
Create a scenario that has three threads. Two threads are reading the value of the shared variable whereas third thread is incrementing the value of the shared variable. If the writer thread is using the shared variable, no reader thread is allowed to use whereas both reader threads can access the shared variable simultaneously. Synchronize the problem

Solution code

```
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>
void* rfun(void *rarg);

void* wfun(void *warg);
int shared=5,readcount=0;
sem_t wrt,mutex;

int main()
{
	sem_init(&wrt,0,1);
	sem_init(&mutex,0,1);
	pthread_t reader[2],writer[1];
	int r_number=1,w_number=1;
	for(int i=0;i<2;i++)
	{
		pthread_create(&reader[i],NULL,rfun,(void
*)&r_number);
		pthread_join(reader[i],NULL);
		r_number+=1;
	}
	for(int i=0;i<1;i++)
```

```
	{
		pthread_create(&writer[i],NULL,wfun,(void
*)&w_number);
		pthread_join(writer[i],NULL);
		r_number+=1;
	}
	sem_destroy(&mutex);
	sem_destroy(&wrt);
}
void* rfun(void *rarg)
{
	int a;
	sem_wait(&mutex);
		readcount+=1;
		if(readcount==1)
		{
			sem_wait(&wrt);
		}
	sem_post(&mutex);
	sleep(1);
	a=shared;
	printf("\n\tThe value of shared variable in reading read
by reader:%d is:%d\n",*((int*)rarg),shared);
	sem_wait(&mutex);
		readcount-=1;
		if(readcount==0)
		{
			sem_post(&wrt);
		}
	sem_post(&mutex);
}
void* wfun(void *warg)
{
	int a;
	sem_wait(&wrt);
		//sem_wait(&mutex);
		a=shared;
		a+=1;
		sleep(1);
		shared=a;
	sem_post(&wrt);
	//sem_post(&mutex);
	printf("\n\tThe value of shared variable in writing
write by writer:%d is:%d\n",*((int*)warg),shared);
}
```

Output snippets



Problem:-2

Create a scenario where there are two threads where one thread is acting as a producer thread producing a random number and the other thread is acting as a consumer thread which printing the random number generated by producer thread. Producer thread can only produce if the global array to place random thread is having an empty index and consumer thread can only consume if there is element in the array. Take a shared variable counter which counts the total no of items in the array at any time. You need to synchronize both the threads using mutex locks for accessing the shared variable counter.

Solution code:-

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdbool.h>
#include<pthread.h>
void* producer();
void* consumer();
void display();
pthread_mutex_t locker;
int a[10];
static int counter=0,i=1,n;
int main()
{
        pthread_mutex_init(&locker,NULL);
        pthread_t t1,t2;
```

```
	int choice;
	while(true)
	{
		printf("\n");
		printf("\t------------------------*Menu*---------
---------------\n");
		printf("\n");
		printf("\t1) Enter '1' for producing a random
number\n");
		printf("\t2) Enter '2' for consuming a random
number\n");
		printf("\t3) Enter '3' to display global array
\n");
		printf("\t4) Enter '4' to quit the prgram
\n");
		printf("\t---------------------------------------
-----------\n");
		scanf("%d",&choice);
		if(choice==1)
		{
			pthread_create(&t1,NULL,producer,NULL);
			pthread_join(t1,NULL);
		}
		else if(choice==2)
		{
			pthread_create(&t2,NULL,consumer,NULL);
			pthread_join(t1,NULL);
		}
		else if(choice==3)
		{
			display();
		}
		else
		{
			break;
		}
	}
}
void* producer()
{
	pthread_mutex_lock(&locker);
	if(counter>10)
	{
		printf("\tThere is no place to keep random
number\n");
	}
	else
	{
		n=(rand()%10);
		a[i]=n;
```

```
            printf("\n\tRandom number is produced and stored
successfully\n");
            sleep(1);
            i++;
            counter++;
      }

      pthread_mutex_unlock(&locker);
}
void* consumer()
{
      pthread_mutex_lock(&locker);
      if(counter<=0)
      {
            printf("\tThere is no random number inside to
print\n");
      }
      else
      {
            printf("\tCurrently consumed random number is:-
%d\n",n);
            i--;
            sleep(1);
            counter--;
      }
      pthread_mutex_unlock(&locker);
}
void display()
{
      int j;
      if(counter==0)
      {
            printf("\tThere are no elements to display\n");
      }
      printf("\t\n Random numbers in global array\n");
      for(j=1;j<=counter;j++)
      {
            printf("\t\t[%d]\n",a[j]);

      }
}
```

Output Snippets:

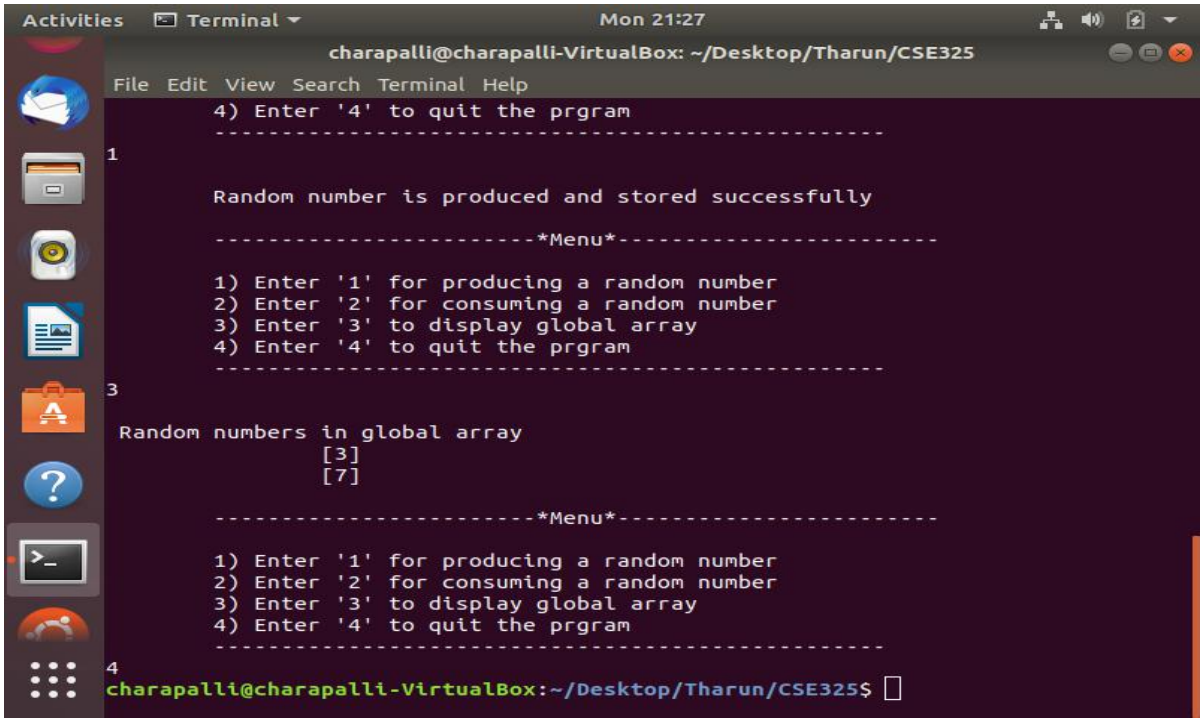
1.

```
Activities  Terminal  Mon 21:27
charapalli@charapalli-VirtualBox: ~/Desktop/Tharun/CSE325
File  Edit  View  Search  Terminal  Help

        ------------------------*Menu*------------------------

        1) Enter '1' for producing a random number
        2) Enter '2' for consuming a random number
        3) Enter '3' to display global array
        4) Enter '4' to quit the prgram
        ------------------------------------------------
1

        Random number is produced and stored successfully

        ------------------------*Menu*------------------------

        1) Enter '1' for producing a random number
        2) Enter '2' for consuming a random number
        3) Enter '3' to display global array
        4) Enter '4' to quit the prgram
        ------------------------------------------------
1

        Random number is produced and stored successfully

        ------------------------*Menu*------------------------

        1) Enter '1' for producing a random number
        2) Enter '2' for consuming a random number
        3) Enter '3' to display global array
        4) Enter '4' to quit the prgram
```

2

```
Activities  Terminal  Mon 21:27
charapalli@charapalli-VirtualBox: ~/Desktop/Tharun/CSE325
File  Edit  View  Search  Terminal  Help

        1) Enter '1' for producing a random number
        2) Enter '2' for consuming a random number
        3) Enter '3' to display global array
        4) Enter '4' to quit the prgram
        ------------------------------------------------
2
        Currently consumed random number is:-6

        ------------------------*Menu*------------------------

        1) Enter '1' for producing a random number
        2) Enter '2' for consuming a random number
        3) Enter '3' to display global array
        4) Enter '4' to quit the prgram
        ------------------------------------------------
1

        Random number is produced and stored successfully

        ------------------------*Menu*------------------------

        1) Enter '1' for producing a random number
        2) Enter '2' for consuming a random number
        3) Enter '3' to display global array
        4) Enter '4' to quit the prgram
        ------------------------------------------------
```

3.



Problem:-3

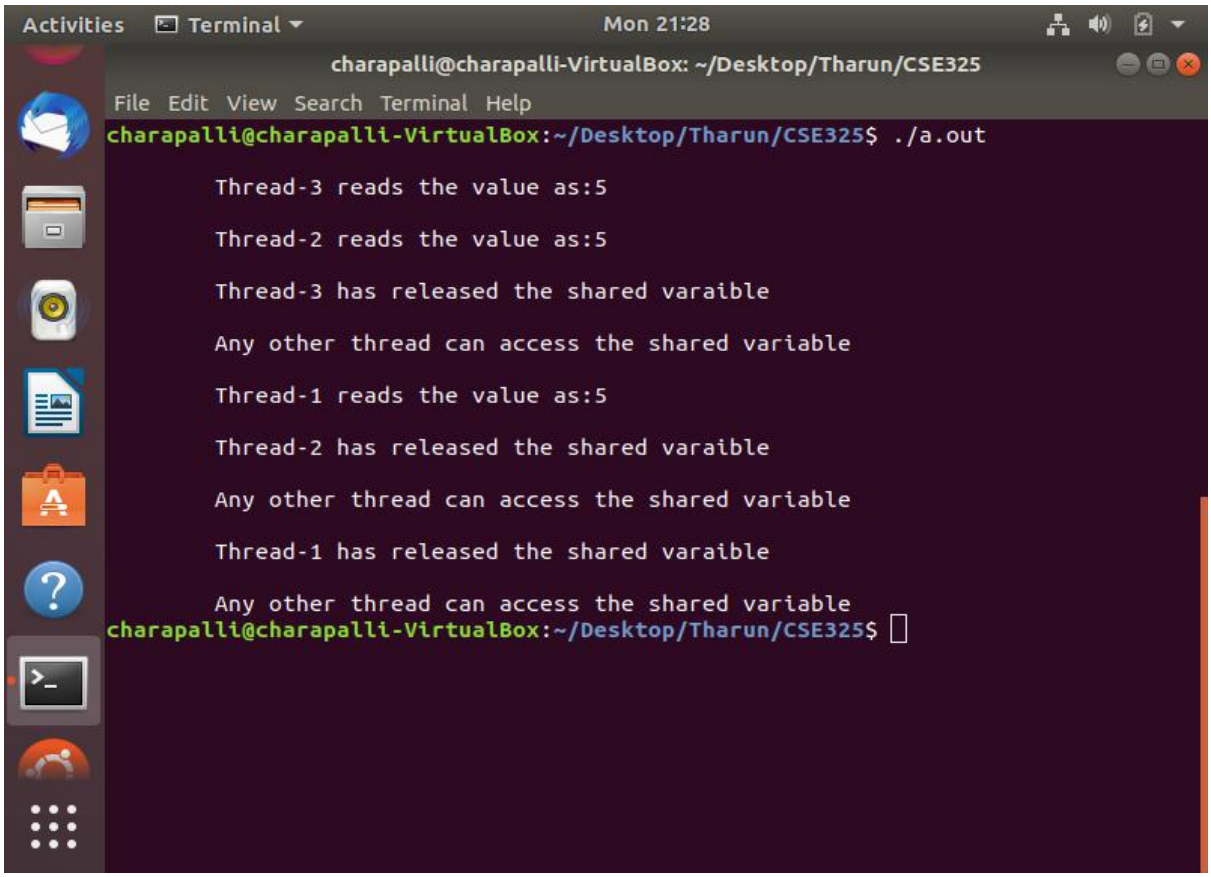
Wap that has three threads. Using semaphore, allow only two threads to access the shared variable at one time.

Solution Code

```
#include<stdio.h>
#include<unistd.h>
#include<semaphore.h>
#include<pthread.h>
void *fun1();
void *fun2();
void *fun3();
sem_t s;
int shared=5,count=0;
int main()
{
        sem_init(&s,0,2);
        pthread_t t1,t2,t3;
        pthread_create(&t1,NULL,fun1,NULL);
        pthread_create(&t2,NULL,fun2,NULL);
        pthread_create(&t3,NULL,fun3,NULL);
        pthread_join(t1,NULL);
        pthread_join(t2,NULL);
        pthread_join(t3,NULL);
}
void* fun1()
```

```
{

            int a,l;
      sem_wait(&s);
            count+=1;
            a=shared;
            printf("\n\tThread-1 reads the value as:%d\n",a);
            sleep(1);
      sem_post(&s);
            printf("\n\tThread-1 has released the shared
varaible\n");
printf("\n\tAny other thread can access the shared variable
\n");
}
void* fun2()
{

            int a;
      sem_wait(&s);
            count+=1;
            a=shared;
            printf("\n\tThread-2 reads the value as:%d\n",a);
            sleep(1);
      sem_post(&s);
            printf("\n\tThread-2 has released the shared
varaible\n");
            printf("\n\tAny other thread can access the
shared variable \n");

}

void* fun3()
{
      int a;
      sem_wait(&s);
            count+=1;
            a=shared;
            printf("\n\tThread-3 reads the value as:%d\n",a);
            sleep(1);
      sem_post(&s);
            printf("\n\tThread-3 has released the shared
varaible\n");
            printf("\n\tAny other thread can access the
shared variable \n");


}
```

Output Snippets



Problem:-4

A parent process creates a child process. The child process after its creation will send a message “Hello parent, this is child process” to its parent through pipe. Once the message is received by the parent, the parent will execute and print “This is Parent process”.

Solution Code

```
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdlib.h>
int main()
{
        int fd[2],n;
        char buffer[100];
        pid_t p;
        pipe(fd);
        p=fork();
        if(p>0)
```

```
        {
                wait(NULL);
                close(fd[1]);
                n=read(fd[0],buffer,100);
                printf("\n\tThis is parent process\n");

                printf("\t\n");
                write(1,buffer,n);

        }
        else if(p==0)
        {
                char msg[100]="        Hello parent, this is
child process\n";
                close(fd[0]);
                printf("\n\tPassing message to parent[Child
Process]\n");
                printf("\n\t-------------------------------------
--------");
                write(fd[1],msg,50);
        }
}
```

Output Snippets