**CPP Lab**

**Assignment-4**

**Name-Ashish Goyal**

**Id-2016ucp1100**

**Batch-A (1, 2)**

1. **Implement solution of Critical Section problem with Semaphores (two processes).**

**CODE-**

//gcc a1.c -lpthread -lrt

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

sem\_t semaphore;

int common=0;

void\* thread(void\* arg)

{

char\* msg=(char\*)arg;

while(1)

{

sem\_wait(&semaphore); //waiting

printf("\n%s-Entered critical section\n",msg);

//int value;

//sem\_getvalue(semaphore,value) ;

//printf("%d\n",value);

sleep(1); //critical section

common++;

printf("\ncommon value-%d\n",common);

printf("\n%s-Exiting critical section\n",msg);

sem\_post(&semaphore); //signal

}

}

int main()

{

sem\_init(&semaphore,0,1); //binary semaphore

pthread\_t t1,t2;

char\* msg1="Thread1";

char\* msg2="Thread2";

pthread\_create(&t1,NULL,thread,msg1);

sleep(3);

pthread\_create(&t2,NULL,thread,msg2);

pthread\_join(t1,NULL);

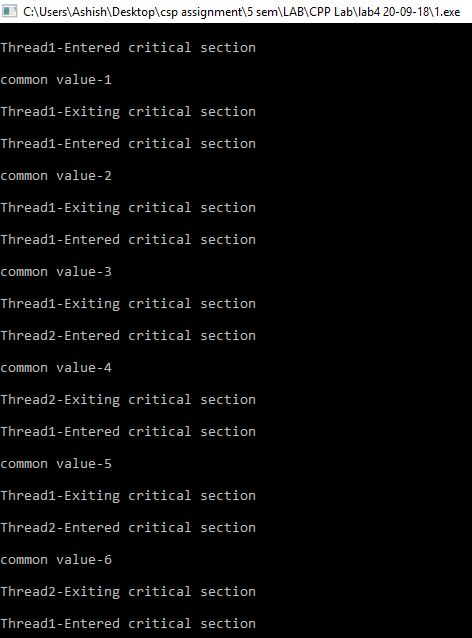
pthread\_join(t2,NULL);

sem\_destroy(&semaphore);

return 0;

}

**OUTPUT-**



1. **Implement solution of Critical Section problem with Semaphores (N processes).**

**CODE-**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

sem\_t semphore;

int common=0;

void\* thread(void\* arg)

{

int message=(int)arg;

while(1)

{

sem\_wait(&semphore);

printf("\nThread %d-Entered critical section\n",message);

sleep(3);

common++;

printf("\ncommon value-%d\n",common);

printf("\nThread %d-Exiting critical section\n",message);

sem\_post(&semphore);

}

}

int main()

{

sem\_init(&semphore,0,1);

pthread\_t thr[10];

int i;

for(i=0;i<10;i++)

{

pthread\_create(&thr[i],NULL,thread,i);

sleep(2);

}

for(i=0;i<10;i++)

{

pthread\_join(thr[i],NULL);

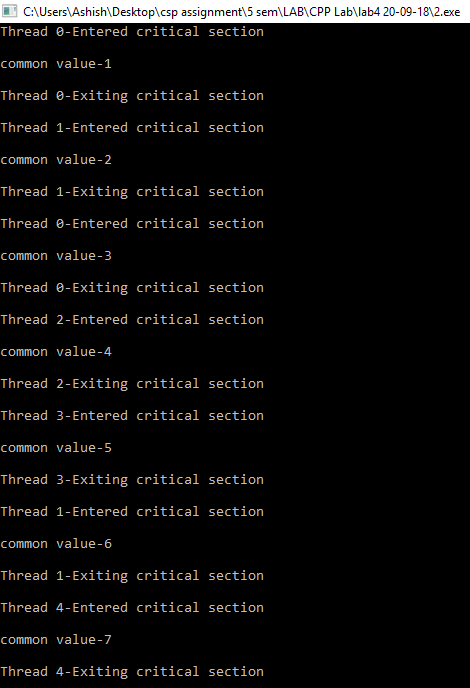
}

sem\_destroy(&semphore);

return 0;

}

**OUTPUT-**



1. **Implement producer-consumer problem with Semaphores (infinite buffer).**

**CODE-**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <pthread.h>

#include <semaphore.h>

#define SIZE 100000 // size of the queue(queue of product)

int queue[SIZE];

int tail,head;

sem\_t notfull,notempty;

void \*producer(void \*arg)

{

int val;

do

{

int val=rand()%100;

printf("produced product number: %d\n",val);

queue[tail]=val;

tail=(tail+1)%SIZE;

sem\_post(&notempty); /\* increment number of products in the queue \*/

sleep(rand()%5); /\* rest after product produce \*/

}while(val);

}

void \*consumer(void \*arg)

{

int val;

do

{

sem\_wait(&notempty); /\* wait for products in queue \*/

val=queue[head];

head=(head+1) % SIZE;

printf("consumed product number: %d\n", val);

sleep(rand()%5); /\* consume the product \*/

}while(val);

}

int main(int argc,char \*argv[])

{

pthread\_t product,consume;

sem\_init(&notfull,0,SIZE); //middle zero means sharing b/w thread

sem\_init(&notempty,0,0);

pthread\_create(&product,NULL,producer,NULL);

pthread\_create(&consume,NULL,consumer,NULL);

pthread\_join(product,NULL);

pthread\_join(consume,NULL);

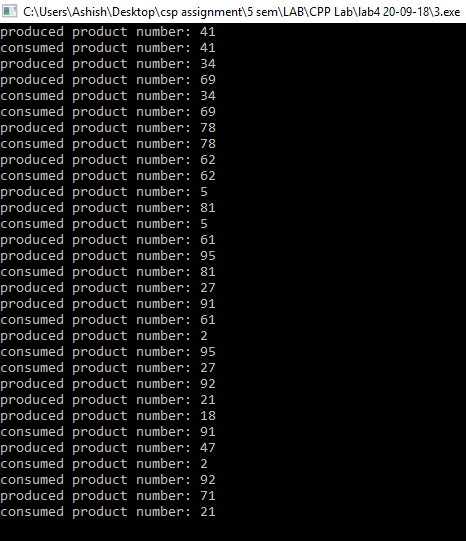
sem\_destroy(&notfull);

sem\_destroy(&notempty);

return 0;

}

**OUTPUT-**

****

1. **Implement producer-consumer problem with Semaphores (finite buffer)**

**CODE-**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <pthread.h>

#include <semaphore.h>

#define SIZE 20 // size of the queue(queue of product)

int queue[SIZE];

int tail,head;

sem\_t notfull,notempty;

void \*producer(void \*arg)

{

int val;

do

{

int val=rand()%100;

printf("produced product number: %d\n",val);

sem\_wait(&notfull); /\* wait for empty place in queue \*/

queue[tail]=val;

tail=(tail+1)%SIZE;

sem\_post(&notempty); /\* increment number of products in the queue \*/

sleep(rand()%4); /\* rest after product produce \*/

}while(val);

}

void \*consumer(void \*arg)

{

int val;

do

{

sem\_wait(&notempty); /\* wait for products in queue \*/

val=queue[head];

head=(head+1) % SIZE;

printf("consumed product number: %d\n", val);

sem\_post(&notfull); /\* increase free spaces in the queue\*/

sleep(rand()%5); /\* consume the product \*/

}while(val);

}

int main(int argc,char \*argv[])

{

pthread\_t product,consume;

sem\_init(&notfull,0,SIZE); //middle zero means sharing b/w thread

sem\_init(&notempty,0,0);

pthread\_create(&product,NULL,producer,NULL);

pthread\_create(&consume,NULL,consumer,NULL);

pthread\_join(product,NULL);

pthread\_join(consume,NULL);

sem\_destroy(&notfull);

sem\_destroy(&notempty);

return 0;

}

**OUTPUT-**

