CLASSICAL SYNCHRONIZATION USING SEMAPHORES

1) AIM: Write a program for bound-buffer(producer-consumer) problem using semaphores.

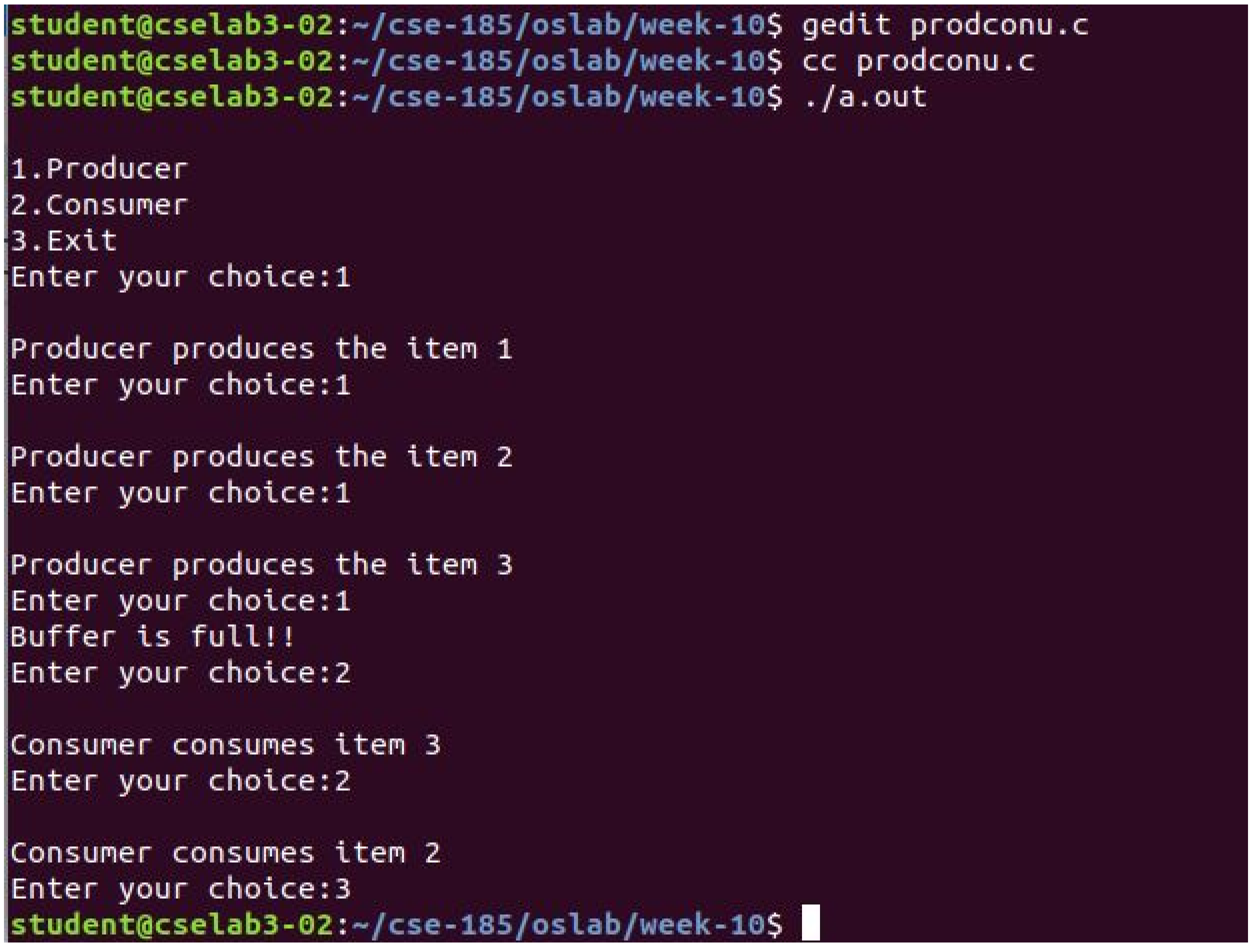
Description: Solution to this problem is, creating two counting semaphores “full” and “empty” to keep track of the current number of full and empty buffers respectively. Producers produce a product and consumers consume the product, but both use of one of the containers each time.

Source Code:

```
#include<stdio.h>

#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()

{

        int n;

        void producer();

        void consumer();

        int wait(int);

        int signal(int);
        printf("\n1.Producer\n2.Consumer\n3.Exit");
        while(1)

        {

                printf("\nEnter your choice:");

                scanf("%d",&n);

                switch(n)

                {

                        case 1: if((mutex==1)&&(empty!=0))

                                        producer();

                                else
                                        printf("Buffer is full!!");

                                break;

                        case 2: if((mutex==1)&&(full!=0))

                                        consumer();

                                else

                                        printf("Buffer is empty!!");
                                break;



                        case 3: exit(0);

                                break;
                        }
        }

                        return 0;

}

int wait(int s)

{

return (--s);

}

int signal(int s)

{

return(++s);

}


void producer()

{

mutex=wait(mutex);
```

```
full=signal(full);

empty=wait(empty);

x++;

printf("\nProducer produces the item %d",x);

mutex=signal(mutex);
}
void consumer(){
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\nConsumer consumes item %d",x);
x--;
mutex=signal(mutex);
}
```

Output:



```
student@cselab3-02:~/cse-185/oslab/week-10$ gedit prodconu.c
student@cselab3-02:~/cse-185/oslab/week-10$ cc prodconu.c
student@cselab3-02:~/cse-185/oslab/week-10$ ./a.out

1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:3
student@cselab3-02:~/cse-185/oslab/week-10$
```

2) **Aim:**Write a program for Dining philosophers problem using semaphores.

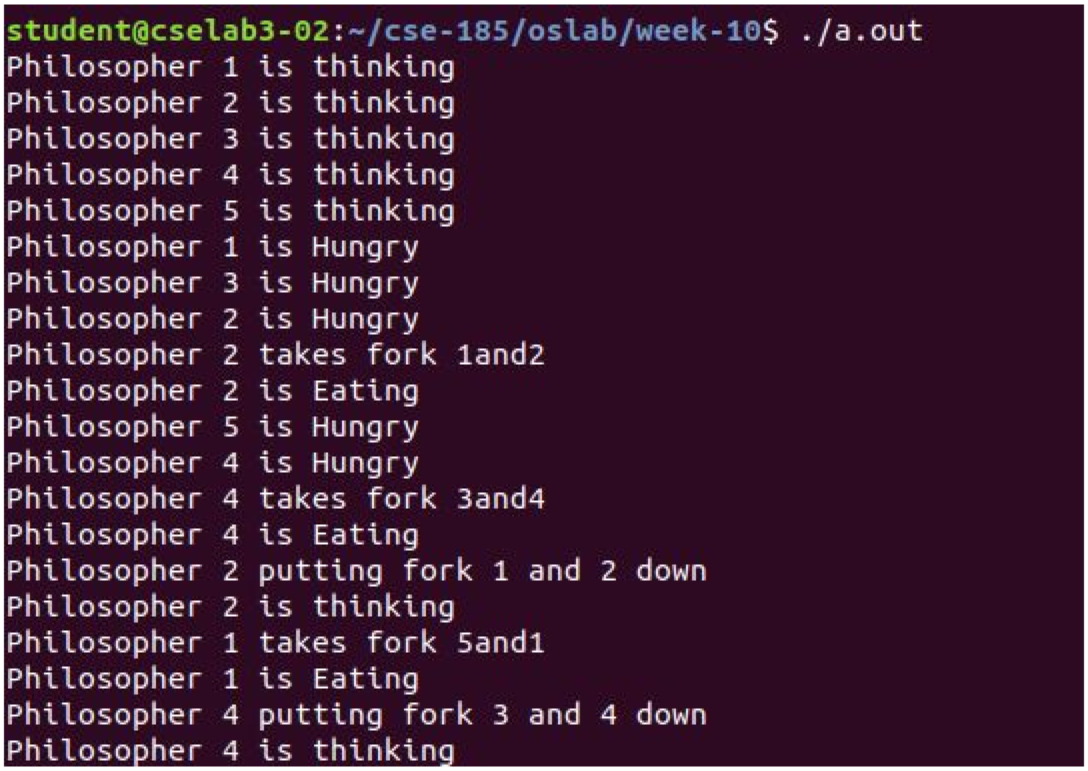
Description: The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pickup the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both. This problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.

Source code:

```
#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4};

sem_tmutex;

sem_tS[N];

void test(int phnum)
{

        if (state[phnum] == HUNGRY && state[LEFT] != EATING && state[RIGHT] !=
EATING)
        {

                // state that eating
                state[phnum] =EATING;
                sleep(2);
}
                printf("Philosopher %d takes fork %dand%d\n",phnum + 1, LEFT + 1, phnum +1);
                printf("Philosopher %d is Eating\n", phnum+1);
                sem_post(&sem_tS[phnum]); // sem_post(&S[phnum]) has no effect during
takefork used to wake up hungry philosophers during putfork

        }

}



// take up chopsticks

void take_fork(int phnum)

{

        sem_wait(&sem_tmutex);

        state[phnum] = HUNGRY; // state that hungry

        printf("Philosopher %d is Hungry\n", phnum +1);

        test(phnum); // eat if neighbours are not eating

        sem_post(&sem_tmutex);

        sem_wait(&sem_tS[phnum]); // if unable to eat wait to be signalled sleep(1);
}

void put_fork(int phnum)

{

        sem_wait(&sem_tmutex);

        state[phnum] = THINKING;

        printf("Philosopher %d putting fork %d and %d down\n", phnum + 1, LEFT + 1, phnum +
1);
        printf("Philosopher %d is thinking\n", phnum + 1);
        test(LEFT);
        test(RIGHT);

        sem_post(&sem_tmutex);

}
```

```
void* philospher(void* num)

{

        while (1)
         {

                int* i = num;

                sleep(1);

                take_fork(*i);

                sleep(0);

                put_fork(*i);
        }

}

int main()

{

        int i;

        pthread_t thread_id[N];

        sem_init(&sem_tmutex, 0, 1);

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

                sem_init(&sem_tS[i], 0, 0);

        for (i = 0; i < N; i++)
        {

                pthread_create(&thread_id[i], NULL,philospher, &phil[i]);

                printf("Philosopher %d is thinking\n", i + 1);

        }
        for (i = 0; i < N; i++)

                pthread_join(thread_id[i], NULL);
        return 0;

}
```

Output:



```
student@cselab3-02:~/cse-185/oslab/week-10$ ./a.out
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 3 is Hungry
Philosopher 2 is Hungry
Philosopher 2 takes fork 1and2
Philosopher 2 is Eating
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 4 takes fork 3and4
Philosopher 4 is Eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5and1
Philosopher 1 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
```

3) **Aim:**Write a program for Reader-Writers problem using semaphores.

Description: Readers-Writers Problem. The readers-writers problem relates to an object such as a file that is shared between multiple processes . ... The readers-writers problem is used to manage synchronization so that there are no problems with the object data.

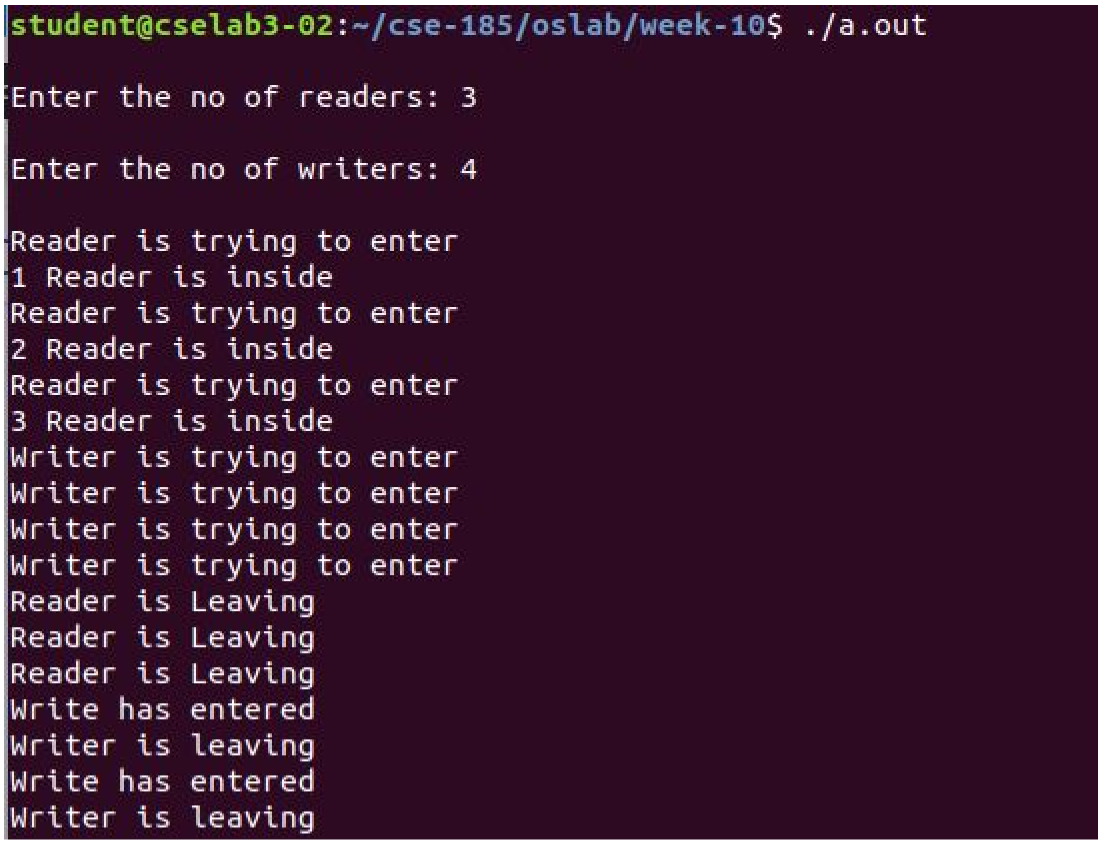
Source code-

```
#include<pthread.h>

//#include<semaphore.h>

#include<stdio.h>

#include<stdlib.h>

pthread_mutex_t x,wsem;

pthread_t tid;

int readcount;

void intialize()

{

        pthread_mutex_init(&x,NULL);

        pthread_mutex_init(&wsem,NULL);

        readcount=0;

}

void * reader (void * param)

{

        int waittime;

        waittime = rand() % 5;
        printf("\nReader is trying to enter");

        pthread_mutex_lock(&x);

        readcount++;
        if(readcount==1)

                pthread_mutex_lock(&wsem);
```

```
	printf("\n%d Reader is inside",readcount);

	pthread_mutex_unlock(&x);
	 sleep(waittime);

	pthread_mutex_lock(&x);

	readcount--;

	if(readcount==0)

		pthread_mutex_unlock(&wsem);

	pthread_mutex_unlock(&x);
	printf("\nReader is Leaving");

}

void * writer (void * param)

{

	int waittime;

	waittime=rand() % 3;

	printf("\nWriter is trying to enter");

	pthread_mutex_lock(&wsem);

	printf("\nWrite has entered");

	sleep(waittime);

	pthread_mutex_unlock(&wsem);

	printf("\nWriter is leaving");

	sleep(30);

	exit(0);

}

int main()

{	int n1,n2,i;

	printf("\nEnter the no of readers: ");
```

```
        scanf("%d",&n1);

        printf("\nEnter the no of writers: ");
        scanf("%d",&n2);

        for(i=0;i<n1;i++)

                pthread_create(&tid,NULL,reader,NULL);
        for(i=0;i<n2;i++)

                pthread_create(&tid,NULL,writer,NULL);

        sleep(30);

        exit(0);


}
```

Output :



```
student@cselab3-02:~/cse-185/oslab/week-10$ ./a.out

Enter the no of readers: 3

Enter the no of writers: 4

Reader is trying to enter
1 Reader is inside
Reader is trying to enter
2 Reader is inside
Reader is trying to enter
3 Reader is inside
Writer is trying to enter
Writer is trying to enter
Writer is trying to enter
Writer is trying to enter
Reader is Leaving
Reader is Leaving
Reader is Leaving
Write has entered
Writer is leaving
Write has entered
Writer is leaving
```

https://www.cs.jhu.edu/~yairamir/cs418/os4/sld003.htm