Name: Ashutosh Soni
Id: 2018ucp1505

CST 303 Concurrent and Parallel Programming Lab
ASSIGNMENT-1

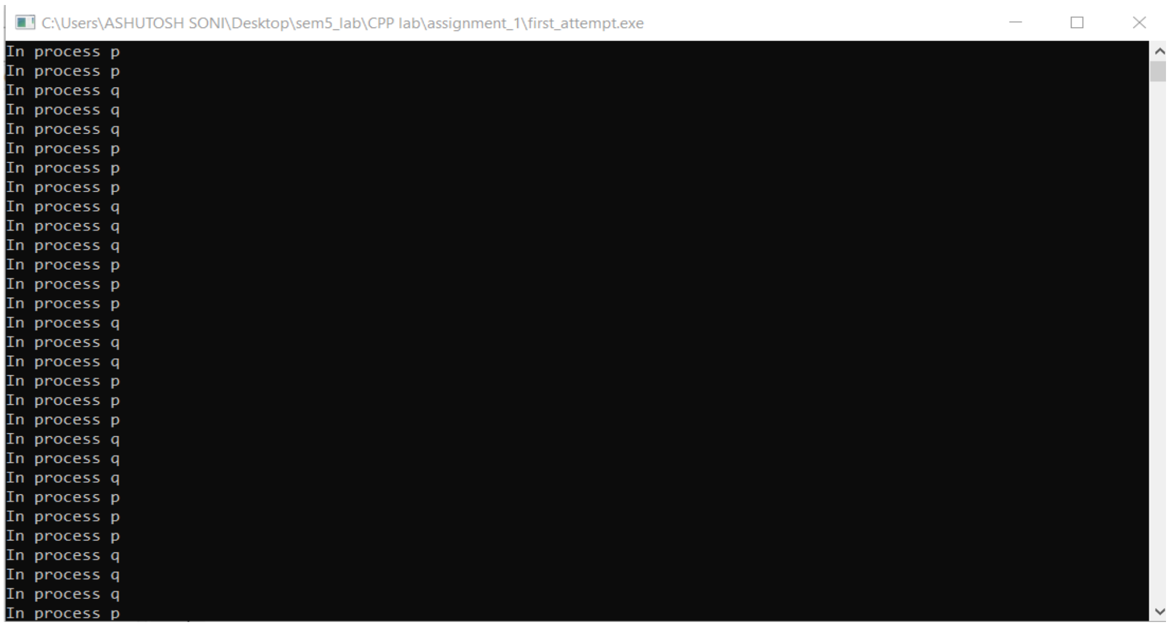
Q1: Implement all the four attempts of Dekker's algorithm to solve critical section problem in C++.
Ans:

First Attempt

```
// First attempt
// Pros
// Mutual Exclusion holds
// Free from deadlock

//cons
// If one process dies other gets blocked.

#include <bits/stdc++.h>
#include <pthread.h>
using namespace std;

int turn = 1;
int x=0;

// Critical Section
void* critical_section(){
	x++;
}

// Process P
void* p(){
	while(1) {
		if(x>=50){
			return NULL;
		}
		cout<<"In process p"<<endl;
		while(turn!=1){

		}
		critical_section();
		turn=2;
	}
}

// Process Q
void* q(){
	while(1){
		if(x>=50){
			return NULL;
```

```
            }
            cout<<"In process q"<<endl;
            while(turn!=2){

            }
            critical_section();
            turn=1;
        }
}

// Loop forever type section
void* start_p(void* arg){
        p();
}

void* start_q(void* arg){
        q();
}

int main(){

        // creation of two thread
        pthread_t pid,qid;
        pthread_create(&pid,NULL,&start_p,NULL);
        pthread_create(&qid,NULL,&start_q, NULL);

        // joining of thread
        pthread_join(pid,NULL);
        pthread_join(qid,NULL);

        // Exit
        pthread_exit(NULL);

        pthread_exit(NULL);

}
```

Output of the Program:



C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_1\first_attempt.exe

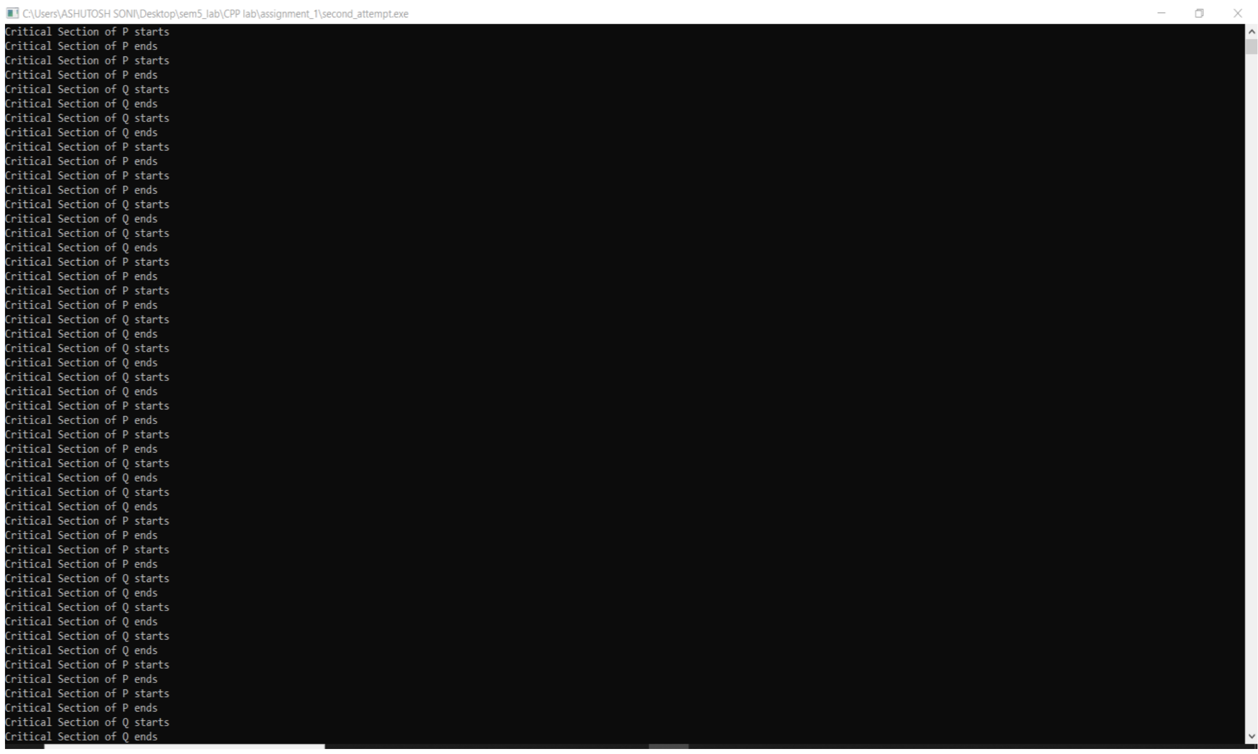
```
In process p
In process p
In process q
In process q
In process q
In process p
In process p
In process p
In process q
In process q
In process q
In process p
In process p
In process p
In process q
In process q
In process q
In process p
In process p
In process p
In process q
In process q
In process q
In process p
In process p
In process p
In process q
In process q
In process q
In process p
```

Second Attempt

```
// Second Attempt

// pros
// No deadlock .
// Free from starvation.

// cons
// Mutual Exclusion principle not holds.

#include<bits/stdc++.h>
#include<pthread.h>
using namespace std;

int x=0;
bool wantp=false,wantq=false;

// Critical Section
void critical_section(){
        x++;
}

// Process p
void* p(){
        while(1){
                if(x>=50){
                        return NULL;
                }
                while(wantq==true){

                }
                wantp=true;
                cout<<"Critical Section of P starts"<<endl;
                critical_section();
                cout<<"Critical Section of P ends"<<endl;
                wantp=false;
        }
}

// Process q
void* q(){
        while(1){
                if(x>=50){
                        return NULL;
                }
                while(wantp==true){

                }
```

```
			wantq=true;
			cout<<"Critical Section of Q starts"<<endl;
			critical_section();
			cout<<"Critical Section of Q ends"<<endl;
			wantq=false;
		}
}

// Loop forever type section
void* start_p(void* arg){
		p();
}

void* start_q(void* arg){
		q();
}

int main(){
		pthread_t pid,qid;
		// creating two thread
		pthread_create(&pid,NULL,&start_p,NULL);
		pthread_create(&qid,NULL,&start_q,NULL);

		// joining of thread
		pthread_join(pid,NULL);
		pthread_join(qid,NULL);

		// Exit
		pthread_exit(NULL);

		return 0;
}
```

Output of the Program:

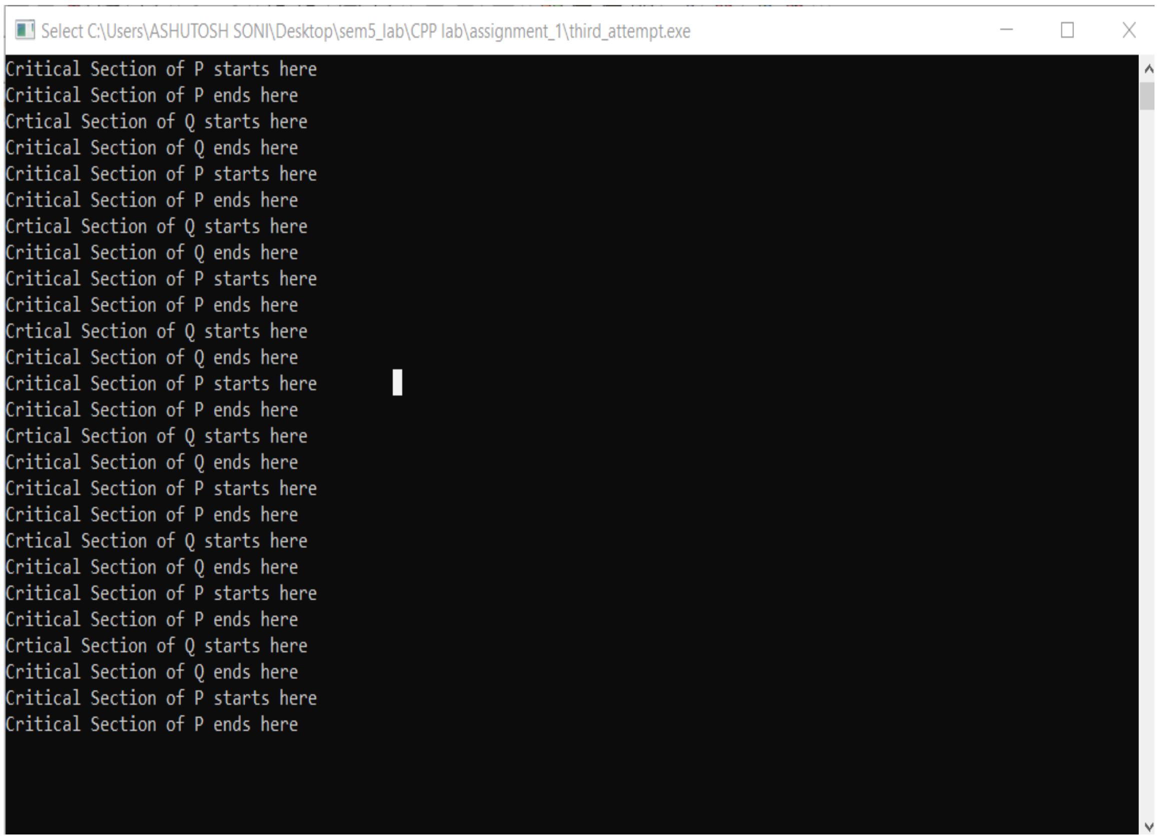

C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_1\second_attempt.exe

```
Critical Section of P starts
Critical Section of P ends
Critical Section of P starts
Critical Section of P ends
Critical Section of Q starts
Critical Section of Q ends
Critical Section of Q starts
Critical Section of Q ends
Critical Section of P starts
Critical Section of P ends
Critical Section of P starts
Critical Section of P ends
Critical Section of Q starts
Critical Section of Q ends
Critical Section of Q starts
Critical Section of Q ends
Critical Section of P starts
Critical Section of P ends
Critical Section of P starts
Critical Section of P ends
Critical Section of Q starts
Critical Section of Q ends
Critical Section of Q starts
Critical Section of Q ends
Critical Section of Q starts
Critical Section of Q ends
Critical Section of P starts
Critical Section of P ends
Critical Section of P starts
Critical Section of P ends
Critical Section of Q starts
Critical Section of Q ends
Critical Section of Q starts
Critical Section of Q ends
Critical Section of P starts
Critical Section of P ends
Critical Section of P starts
Critical Section of P ends
Critical Section of Q starts
Critical Section of Q ends
Critical Section of Q starts
Critical Section of Q ends
Critical Section of Q starts
Critical Section of Q ends
Critical Section of P starts
Critical Section of P ends
Critical Section of P starts
Critical Section of P ends
Critical Section of Q starts
Critical Section of Q ends
```

Third Attempt

```
// Third attempt

// mutual Exclusion satisfied.

// Not free from deadlock.

#include<bits/stdc++.h>
#include<pthread.h>
using namespace std;

bool wantp=false,wantq=false;
int x=0;

// Critical Section
void critical_section(){
        x++;
}

// Process p
void* p(){
        while(1){
                if(x>=50){
                        return NULL;
                }
                wantp=true;
                while(wantq==true){

                }
                cout<<"Critical Section of P starts here"<<endl;
                critical_section();
                cout<<"Critical Section of P ends here"<<endl;
                wantp=false;
        }
}

// Process Q
void* q(){
        while(1){
                if(x>=50){
                        return NULL;
                }
                wantq=true;
                while(wantp==true){
                }
                cout<<"Critical Section of Q starts here"<<endl;
                critical_section();
                cout<<"Critical Section of Q ends here"<<endl;
                wantq=false;
        }
}

// starting of the process
void* start_p(void* arg){
        p();
}

void* start_q(void* arg){
        q();
}

int main(){
        pthread_t pid,qid;
        // Creating two thread
        pthread_create(&pid,NULL,*start_p,NULL);
        pthread_create(&qid,NULL,*start_q,NULL);

        // Joining thread
        pthread_join(pid,NULL);
        pthread_join(qid,NULL);

        // Exit
        pthread_exit(NULL);

        return 0;
}
```

Output of the Program:


Select C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_1\third_attempt.exe

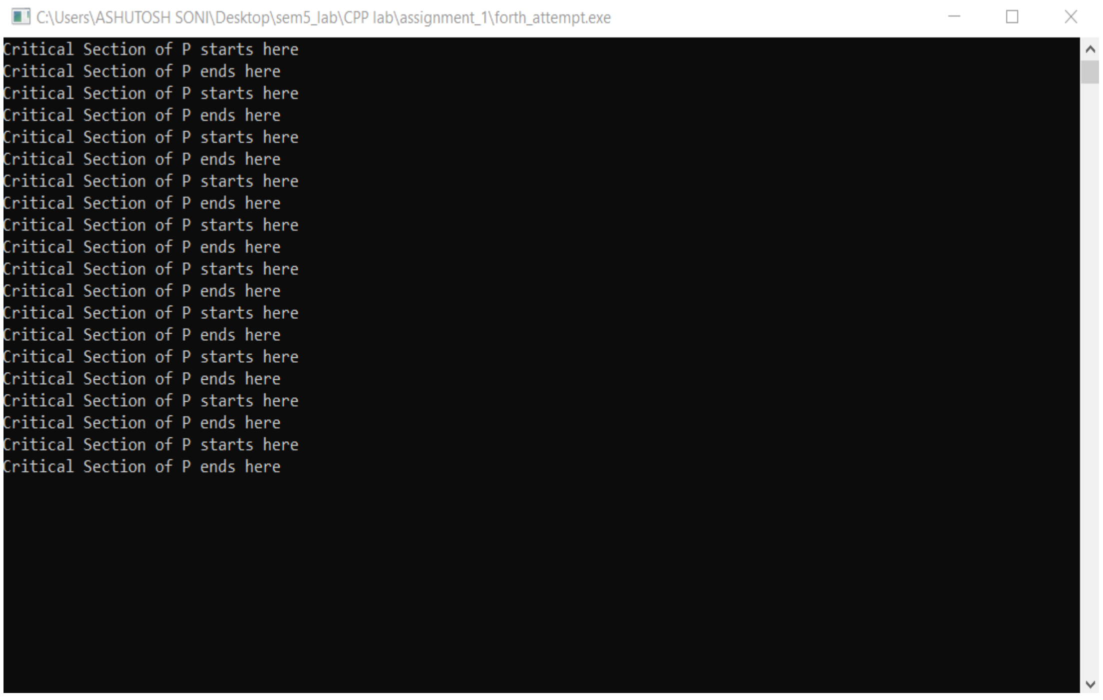
```
Critical Section of P starts here
Critical Section of P ends here
Crtical Section of Q starts here
Critical Section of Q ends here
Critical Section of P starts here
Critical Section of P ends here
Crtical Section of Q starts here
Critical Section of Q ends here
Critical Section of P starts here
Critical Section of P ends here
Crtical Section of Q starts here
Critical Section of Q ends here
Critical Section of P starts here
Critical Section of P ends here
Crtical Section of Q starts here
Critical Section of Q ends here
Critical Section of P starts here
Critical Section of P ends here
Crtical Section of Q starts here
Critical Section of Q ends here
Critical Section of P starts here
Critical Section of P ends here
Crtical Section of Q starts here
Critical Section of Q ends here
Critical Section of P starts here
Critical Section of P ends here
```

Fourth Attempt

```
// Fourth Attempt

// Mutual Exclusion satisfied
// Free from deadlock

// Starvation may happens.

#include<bits/stdc++.h>
#include<pthread.h>
using namespace std;

bool wantp=false,wantq=false;
int x=0;

// Critical Section
void critical_section(){
	x++;
}

// process P
void* p(){
	while(1){
		if(x>=10){
			return NULL;
		}
		wantp=true;
		while(wantq){
			wantp=false;
			wantp=true;
		}
		cout<<"Critical Section of P starts here"<<endl;
		critical_section();
		cout<<"Critical Section of P ends here"<<endl;
		wantp=false;
	}
}

// Process Q
void* q(){
	while(1){
		if(x>=10){
			return NULL;
		}
		wantq=true;
```

```
                while(wantp){
                        wantq=false;
                        wantp=true;
                }
                cout<<"Critical Section of Q starts here"<<endl;
                critical_section();
                cout<<"Critical Section of Q ends here"<<endl;
                wantq=false;
        }
}

// start for join process P
void* start_p(void* arg){
        p();
}

// start for join process Q
void* start_q(void* arg){
        q();
}

int main(){
        pthread_t pid,qid;
        // creating two threads
        pthread_create(&pid,NULL,&start_p,NULL);
        pthread_create(&qid,NULL,&start_q,NULL);

        // Joining  threads
        pthread_join(pid,NULL);
        pthread_join(qid,NULL);

        // Exit
        pthread_exit(NULL);

        return 0;
}
```

Output of the Program:



C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_1\forth_attempt.exe

```
Critical Section of P starts here
Critical Section of P ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of P starts here
Critical Section of P ends here
```

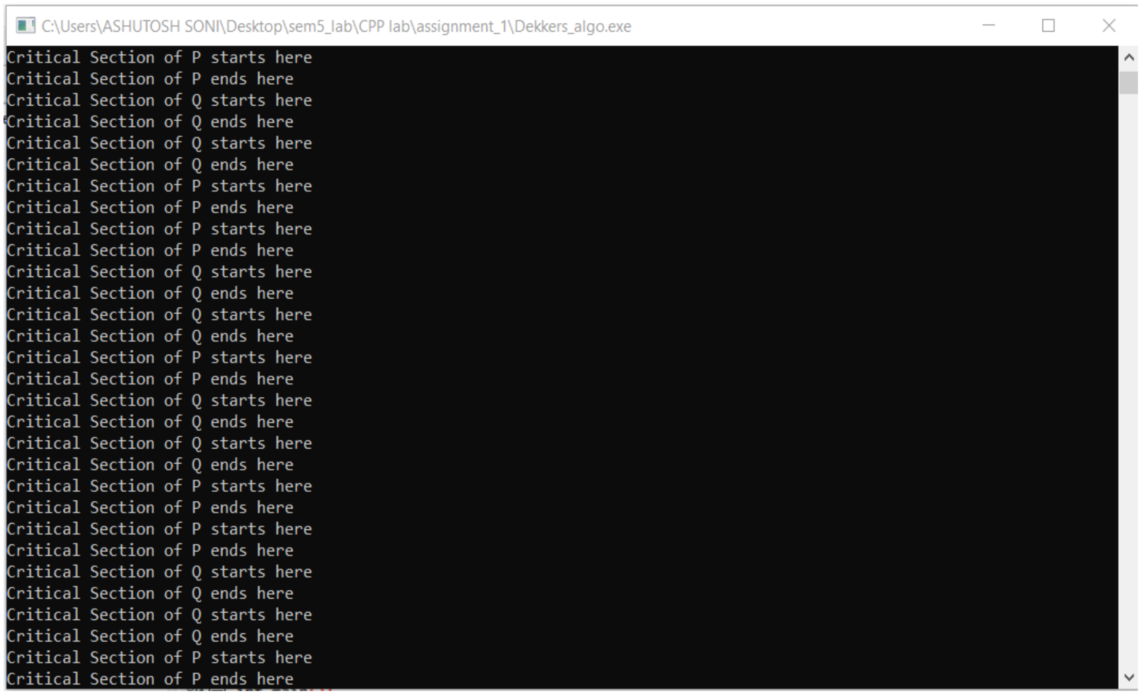
Q2: Implement Dekker's Algorithm for mutual exclusion in C++.
Ans:

Dekkers Algorithm

```
// Dekkers algorithm implementation

// Free from satrvation
// Mutual Exlusion satisfied
// Free from deadlock

#include<bits/stdc++.h>
#include<pthread.h>
using namespace std;

int turn=1;
bool wantp=false,wantq=false;
int x=0;

// Critical Section
void critical_section(){
	x++;
}

// process P
void* p(){
	while(1){
		if(x>=50){
			return NULL;
		}
		wantp=true;
		while(wantq){
			if(turn==2){
				wantp=false;
				while(turn!=1){

				}
				wantp=true;
			}
		}
		cout<<"Critical Section of P starts here"<<endl;
		critical_section();
		cout<<"Critical Section of P ends here"<<endl;
		turn=2;
		wantp=false;
	}
}

// process Q
```

```
void* q(){
        while(1){
                if(x>=50){
                        return NULL;
                }
                wantq=true;
                while(wantp){
                        if(turn==1){
                                wantq=false;
                                while(turn!=2){

                                }
                                wantq=true;
                        }
                }
                cout<<"Critical Section of Q starts here"<<endl;
                critical_section();
                cout<<"Critical Section of Q ends here"<<endl;
                turn=1;
                wantq=false;
        }
}

// start P
void* start_p(void* arg){
        p();
}

// start q
void* start_q(void* arg){
        q();
}


int main(){
        pthread_t pid,qid;
        // creating two thread
        pthread_create(&pid,NULL,&start_p,NULL);
        pthread_create(&qid,NULL,*start_q,NULL);

        // Joining threads
        pthread_join(pid,NULL);
        pthread_join(qid,NULL);

        // Exit
        pthread_exit(NULL);

        return 0;
}
```

Output of the Program:



C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_1\Dekkers_algo.exe

```
Critical Section of P starts here
Critical Section of P ends here
Critical Section of Q starts here
Critical Section of Q ends here
Critical Section of Q starts here
Critical Section of Q ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of Q starts here
Critical Section of Q ends here
Critical Section of Q starts here
Critical Section of Q ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of Q starts here
Critical Section of Q ends here
Critical Section of Q starts here
Critical Section of Q ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of P starts here
Critical Section of P ends here
Critical Section of Q starts here
Critical Section of Q ends here
Critical Section of Q starts here
Critical Section of Q ends here
Critical Section of P starts here
Critical Section of P ends here
```

Name: Ashutosh Soni
Id: 2018ucp1505

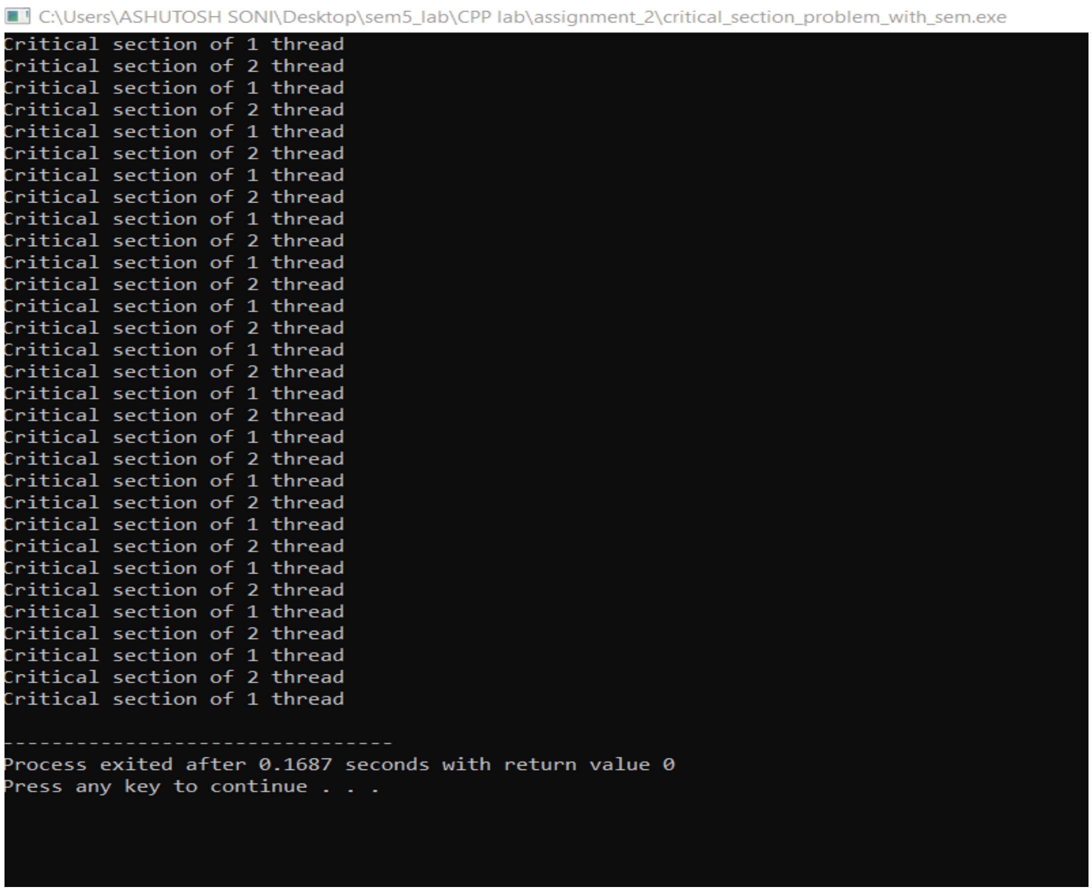
ASSIGNMENT-2

Q1: Implement solution of Critical Section problem with Semaphores (two processes).
Ans:

Critical Section problem with Semaphores (two processes)

```
#include<bits/stdc++.h>
#include<pthread.h>
#include<semaphore.h>
using namespace std;

// Declaration
pthread_t p1,p2;
sem_t semaphore;
int a=0,x=0;

// Critical Section
void critical_section(){
	// Here -1 because lower thread is 2......so to show readability
	cout<<"Critical section of "<<pthread_self()-1<<" thread"<<endl;
	x++;
}

// Process p
void* p1_start(void *arg){
	while(x<30){
		// Non critical section
		a=(a+1)%2;
		sem_wait(&semaphore);
		critical_section();
		sem_post(&semaphore);
	}
}


int main(int argv,char *argc[]){
	// Declaration ......
	pthread_attr_t attr;

	// Initialization of semaphore
	sem_init(&semaphore,0,1);

	// pthread_attr_t initialization
	pthread_attr_init(&attr);
	pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_JOINABLE);
	// creation of process
	int r1=pthread_create(&p1,&attr,p1_start,NULL);
	if(r1){
		cout<<"Error in creating thread"<<endl;
		exit(-1);
	}
	r1=pthread_create(&p2,&attr,p1_start,NULL);
	if(r1){
		cout<<"Error in creating thread"<<endl;
		exit(-1);
	}

	// destroying the pthread_attr
	pthread_attr_destroy(&attr);

	// Joining the process
	r1=pthread_join(p1,NULL);
	if(r1){
		cout<<"Error in joining thread"<<endl;
		exit(-1);
	}
	r1=pthread_join(p2,NULL);
	if(r1){
		cout<<"Error in joining thread"<<endl;
		exit(-1);
	}

	// Exiting pthread
	pthread_exit(NULL);
}
```

Output of the Program:



C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_2\critical_section_problem_with_sem.exe

```
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 1 thread

--------------------------------
Process exited after 0.1687 seconds with return value 0
Press any key to continue . . .
```

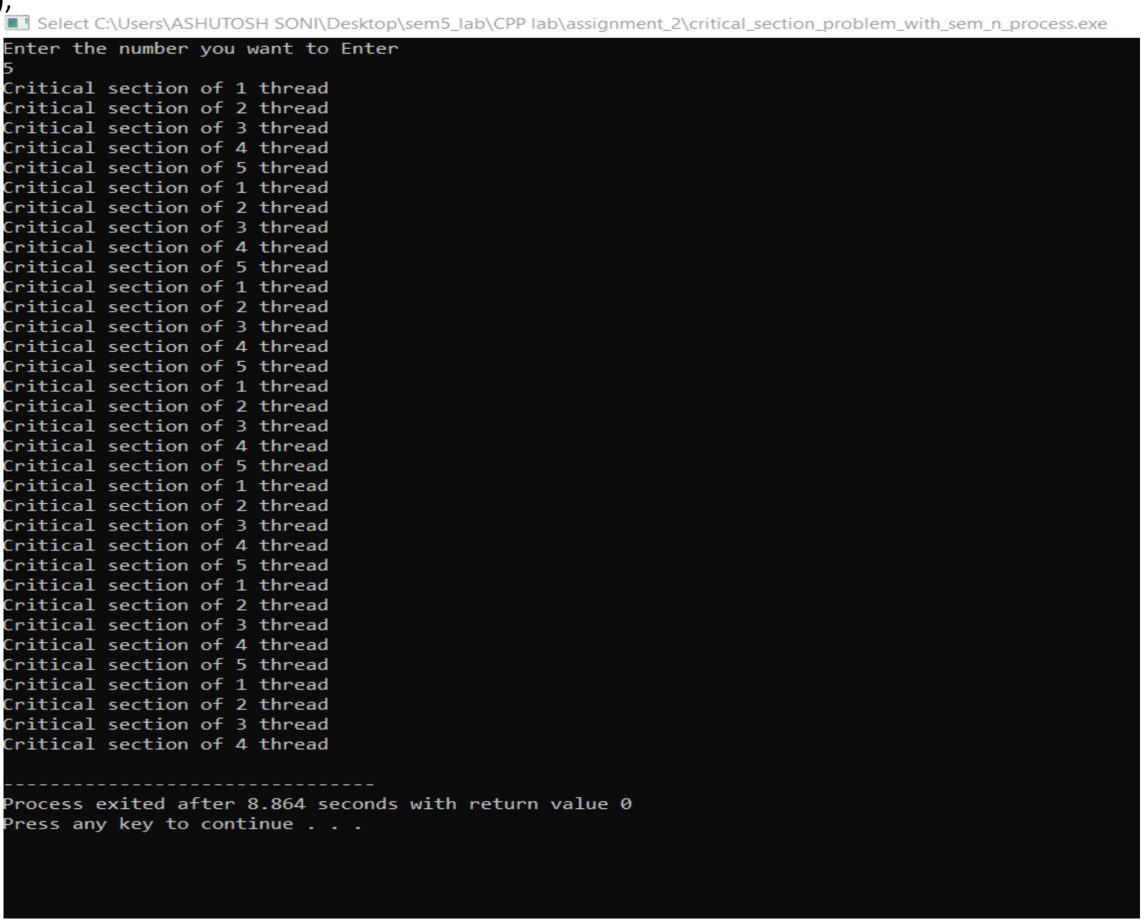
Q2: Implement solution of Critical Section problem with Semaphores (N processes).
Ans:

Critical Section problem with Semaphores (N processes)

```
#include<bits/stdc++.h>
#include<pthread.h>
#include<semaphore.h>
using namespace std;

// Declaration
sem_t semaphore;
int a=0,x=0;

// Critical Section
void critical_section(){
	// Here -1 because lower thread is 2......so to show readability
	cout<<"Critical section of "<<pthread_self()-1<<" thread"<<endl;
	x++;
}

// Process p
void* p1_start(void *arg){
	while(x<30){
		// Non critical section
		a=(a+1)%2;
		sem_wait(&semaphore);
		critical_section();
		sem_post(&semaphore);
	}
}


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

	//declaration
	int r1,N;

	// Taking input of number of process
	cout<<"Enter the number you want to Enter"<<endl;
	cin>>N;

	// Declaration of thread
	pthread_t process[N];

	// Declaration of attribute......
	pthread_attr_t attr;

	// Initialization of semaphore
```

```
        sem_init(&semaphore,0,1);

        // pthread_attr_t initialization
        pthread_attr_init(&attr);
        pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_JOINABLE);

        // creation of process
        for(int i=0;i<N;i++){
                r1=pthread_create(&process[i],&attr,p1_start,NULL);
                if(r1){
                        cout<<"Error in creating thread"<<endl;
                        exit(-1);
                }
        }

        // destroying the pthread_attr
        pthread_attr_destroy(&attr);

        // Joining the process
        for(int i=0;i<N;i++){
                r1=pthread_join(process[i],NULL);
                if(r1){
                        cout<<"Error in joining thread"<<endl;
                        exit(-1);
                }
        }

        // Exiting pthread
        pthread_exit(NULL);
}
```

Output of the Program:



Select C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_2\critical_section_problem_with_sem_n_process.exe

```
Enter the number you want to Enter
5
Critical section of 1 thread
Critical section of 2 thread
Critical section of 3 thread
Critical section of 4 thread
Critical section of 5 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 3 thread
Critical section of 4 thread
Critical section of 5 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 3 thread
Critical section of 4 thread
Critical section of 5 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 3 thread
Critical section of 4 thread
Critical section of 5 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 3 thread
Critical section of 4 thread
Critical section of 5 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 3 thread
Critical section of 4 thread
Critical section of 5 thread
Critical section of 1 thread
Critical section of 2 thread
Critical section of 3 thread
Critical section of 4 thread

--------------------------------
Process exited after 8.864 seconds with return value 0
Press any key to continue . . .
```

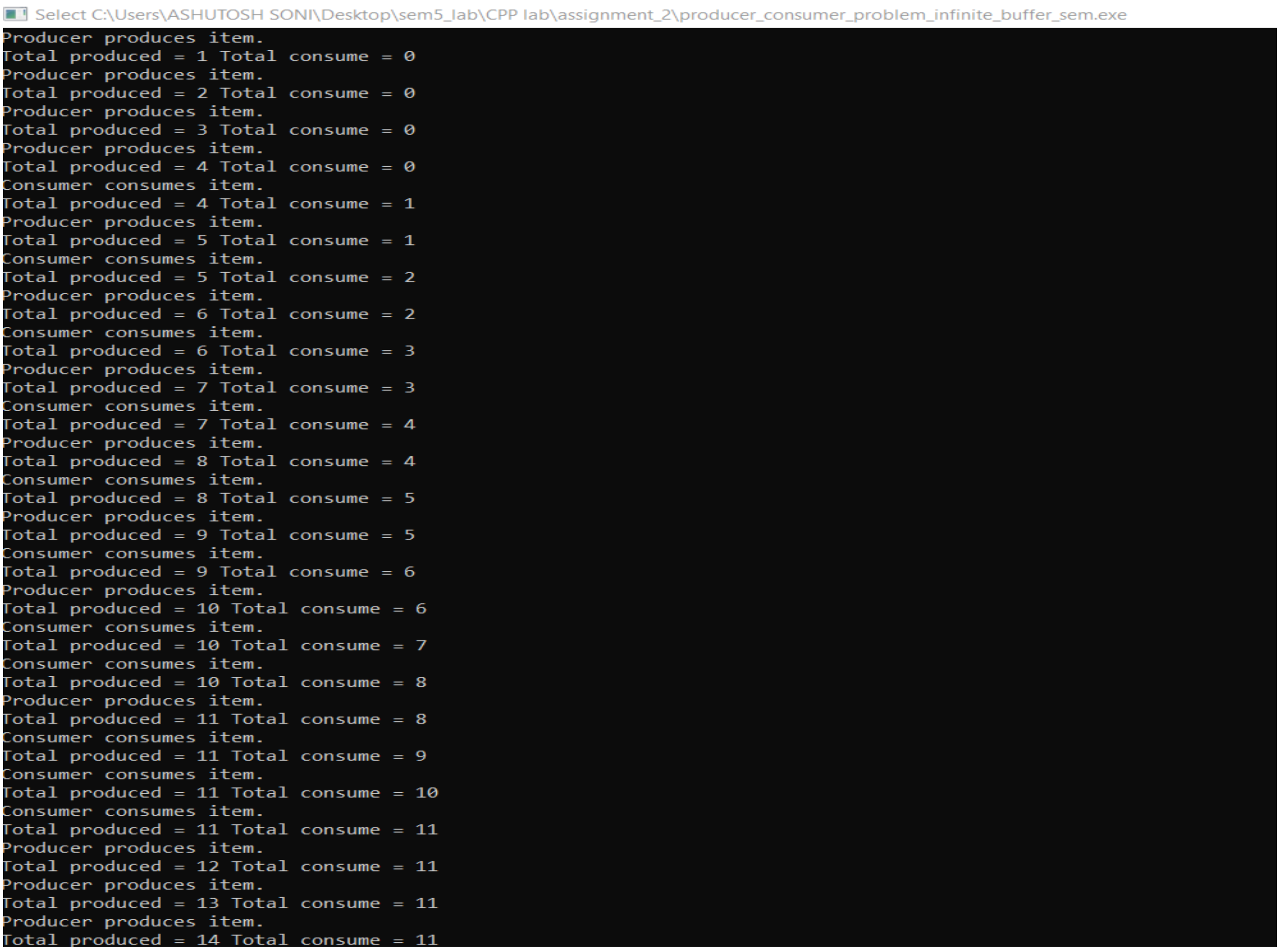
Q3: Implement producer-consumer problem with Semaphores (infinite buffer).
Ans:

Producer-consumer problem with Semaphores (infinite buffer)

```
#include<bits/stdc++.h>
#include<pthread.h>
#include<semaphore.h>
#include <unistd.h>
using namespace std;

// Declaration
int r1,total_produced=0,total_consume=0;

// Semaphore declaration
sem_t notEmpty;

// Producer Section
void* produce(void *arg){
        while(1){
                cout<<"Producer produces item."<<endl;
                cout<<"Total produced = "<<++total_produced<<" Total consume = "<<total_consume*-1<<endl;
                sem_post(&notEmpty);
                sleep(rand()%100*0.01);
        }
}

// Consumer Section
void* consume(void *arg){
        while(1){
                sem_wait(&notEmpty);
                cout<<"Consumer consumes item."<<endl;
                cout<<"Total produced = "<<total_produced<<" Total consume = "<<(--total_consume)*-1<<endl;
                sleep(rand()%100*0.01);
        }
}

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

        // thread declaration
        pthread_t producer,consumer;

        // Declaration of attribute......
        pthread_attr_t attr;

        // semaphore initialization
        sem_init(&notEmpty,0,0);

        // pthread_attr_t initialization
```

```
	pthread_attr_init(&attr);
	pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_JOINABLE);

	// Creation of process
	r1=pthread_create(&producer,&attr,produce,NULL);
	if(r1){
		cout<<"Error in creating thread"<<endl;
		exit(-1);
	}

	r1=pthread_create(&consumer,&attr,consume,NULL);
	if(r1){
		cout<<"Error in creating thread"<<endl;
		exit(-1);
	}

	// destroying the pthread_attr
	pthread_attr_destroy(&attr);

	// Joining the thread
	r1=pthread_join(producer,NULL);
	if(r1){
		cout<<"Error in joining thread"<<endl;
		exit(-1);
	}

	r1=pthread_join(consumer,NULL);
	if(r1){
		cout<<"Error in joining thread"<<endl;
		exit(-1);
	}

	// Exiting thread
	pthread_exit(NULL);

	return 0;
}
```

Output of the Program:



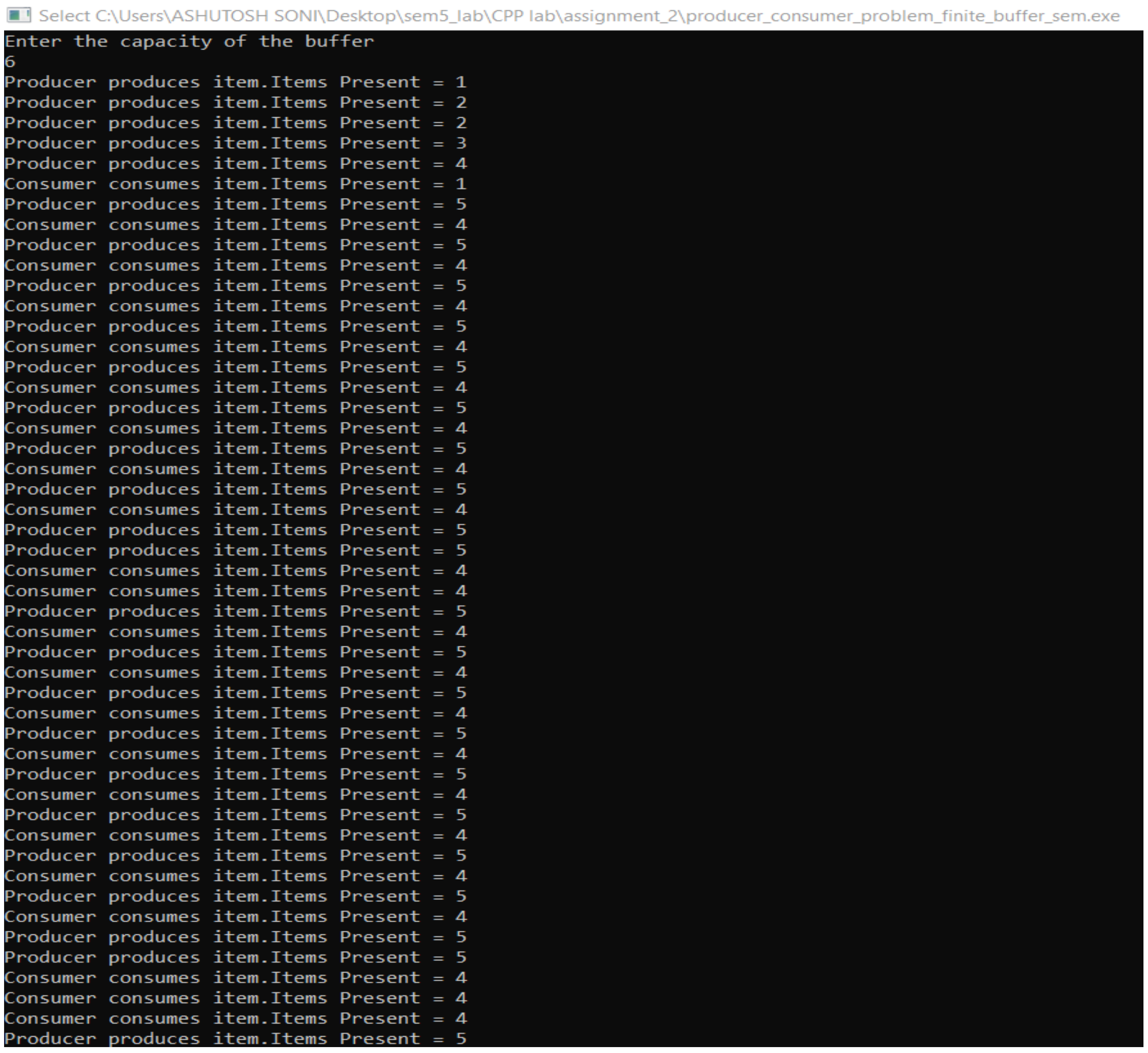
```
Select C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_2\producer_consumer_problem_infinite_buffer_sem.exe
Producer produces item.
Total produced = 1 Total consume = 0
Producer produces item.
Total produced = 2 Total consume = 0
Producer produces item.
Total produced = 3 Total consume = 0
Producer produces item.
Total produced = 4 Total consume = 0
Consumer consumes item.
Total produced = 4 Total consume = 1
Producer produces item.
Total produced = 5 Total consume = 1
Consumer consumes item.
Total produced = 5 Total consume = 2
Producer produces item.
Total produced = 6 Total consume = 2
Consumer consumes item.
Total produced = 6 Total consume = 3
Producer produces item.
Total produced = 7 Total consume = 3
Consumer consumes item.
Total produced = 7 Total consume = 4
Producer produces item.
Total produced = 8 Total consume = 4
Consumer consumes item.
Total produced = 8 Total consume = 5
Producer produces item.
Total produced = 9 Total consume = 5
Consumer consumes item.
Total produced = 9 Total consume = 6
Producer produces item.
Total produced = 10 Total consume = 6
Consumer consumes item.
Total produced = 10 Total consume = 7
Consumer consumes item.
Total produced = 10 Total consume = 8
Producer produces item.
Total produced = 11 Total consume = 8
Consumer consumes item.
Total produced = 11 Total consume = 9
Consumer consumes item.
Total produced = 11 Total consume = 10
Consumer consumes item.
Total produced = 11 Total consume = 11
Producer produces item.
Total produced = 12 Total consume = 11
Producer produces item.
Total produced = 13 Total consume = 11
Producer produces item.
Total produced = 14 Total consume = 11
```

Q4: Implement producer-consumer problem with Semaphores (finite buffer).
Ans:

<u>Producer-consumer problem with Semaphores (finite buffer)</u>

```
#include<bits/stdc++.h>
#include<pthread.h>
#include<semaphore.h>
#include <unistd.h>
using namespace std;

// Declaration
int r1,items=0;

// Semaphore declaration
sem_t notEmpty,notFull;

// Producer Section
void* produce(void *arg){
        while(1){
                sem_wait(&notFull);
                sleep(rand()%100*0.01);
                cout<<"Producer produces item.Items Present = "<<++items<<endl;
                sem_post(&notEmpty);
                sleep(rand()%100*0.01);
        }
}

// Consumer Section
void* consume(void *arg){
        while(1){
                sem_wait(&notEmpty);
                sleep(rand()%100*0.01);
                cout<<"Consumer consumes item.Items Present = "<<--items<<endl;
                sem_post(&notFull);
                sleep(rand()%100*0.01);
        }
}

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

        int N;
        cout<<"Enter the capacity of the buffer"<<endl;
        cin>>N;

        // thread declaration
        pthread_t producer,consumer;

        // Declaration of attribute......
        pthread_attr_t attr;

        // semaphore initialization
        sem_init(&notEmpty,0,0);
        sem_init(&notFull,0,N);

        // pthread_attr_t initialization
        pthread_attr_init(&attr);
        pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_JOINABLE);

        // Creation of process
        r1=pthread_create(&producer,&attr,produce,NULL);
        if(r1){
                cout<<"Error in creating thread"<<endl;
                exit(-1);
        }

        r1=pthread_create(&consumer,&attr,consume,NULL);
        if(r1){
                cout<<"Error in creating thread"<<endl;
                exit(-1);
        }
        // destroying the pthread_attr
        pthread_attr_destroy(&attr);

        // Joining the thread
        r1=pthread_join(producer,NULL);
        if(r1){
                cout<<"Error in joining thread"<<endl;
                exit(-1);
        }

        r1=pthread_join(consumer,NULL);
        if(r1){
                cout<<"Error in joining thread"<<endl;
                exit(-1);
        }

        // Exiting thread
        pthread_exit(NULL);

        return 0;

}
```

Output of the Program:



Select C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_2\producer_consumer_problem_finite_buffer_sem.exe

```
Enter the capacity of the buffer
6
Producer produces item.Items Present = 1
Producer produces item.Items Present = 2
Producer produces item.Items Present = 2
Producer produces item.Items Present = 3
Producer produces item.Items Present = 4
Consumer consumes item.Items Present = 1
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
Producer produces item.Items Present = 5
Consumer consumes item.Items Present = 4
Consumer consumes item.Items Present = 4
Consumer consumes item.Items Present = 4
Producer produces item.Items Present = 5
```

Q5: Implement Merge-sort using Semaphores.
Ans:

Merge-sort using Semaphores

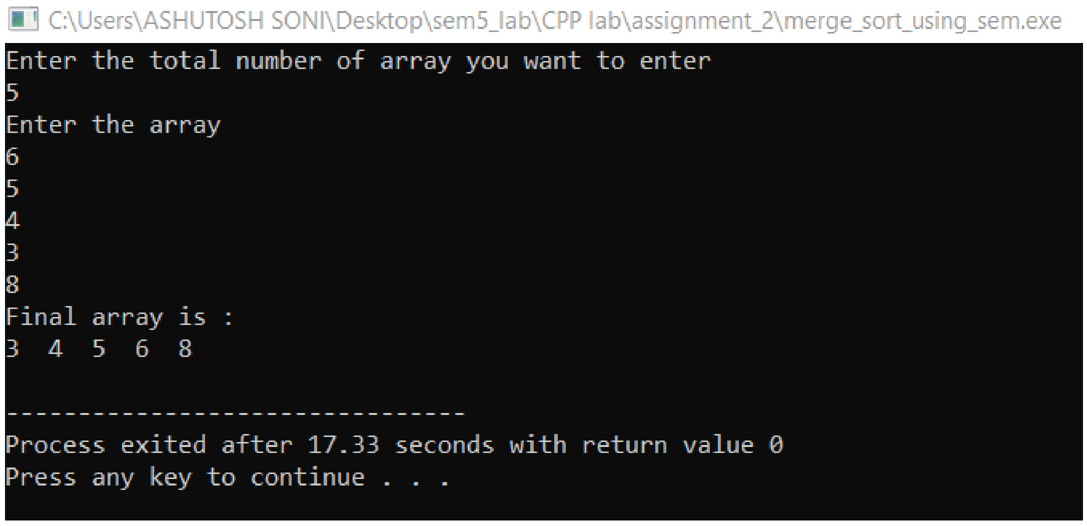
```
// Merge Sort Implementation using Semaphore



#include<bits/stdc++.h>
#include<pthread.h>
#include<semaphore.h>
#include <unistd.h>
using namespace std;

// Decalaration
int r1;
long N;
vector<int> array;

// Declaration of Semaphore
sem_t S1,S2;

// sort first part of array

void* sort_first(void *arg){
        N=*(long* )arg;
        int mid=N/2;
        sort(array.begin(),array.begin()+mid);
        sem_post(&S1);
}

void* sort_second(void *arg){
        N=*(long*)arg;
        int mid=N/2;
        sort(array.begin()+mid,array.end());
        sem_post(&S2);
}

void* merge_array(void *arg){
        N=*(long*)arg;
        int mid=N/2;
        sem_wait(&S1);
        sem_wait(&S2);
        vector<int> left,right;
        for(int i=0;i<mid;i++){
                left.push_back(array[i]);
        }
        for(int i=mid;i<N;i++){
                right.push_back(array[i]);
        }
        int m=left.size(),n=right.size();
```

```
	int i=0,j=0,k=0;
	while(i<m and j<n){
		if(left[i]<=right[j]){
			array[k]=left[i];
			i++;
			k++;
		}
		else{
			array[k]=right[j];
			j++;
			k++;
		}
	}
	while(i<m){
		array[k]=left[i];
		i++;
		k++;
	}
	while(j<n){
		array[k]=right[j];
		k++;
		j++;
	}

	// After merging Final array will be

	cout<<"Final array is : "<<endl;
	for(int i=0;i<N;i++){
		cout<<array[i]<<"  ";
	}
	cout<<endl;

}


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

	// Initialization....
	long N;
	cout<<"Enter the total number of array you want to enter"<<endl;
	cin>>N;
	cout<<"Enter the array"<<endl;
	for(int i=0;i<N;i++){
		int num;
		cin>>num;
		array.push_back(num);
	}

	// Declaration of thread
	pthread_t sort_1,sort_2,merge;
```

```
// Declaration of attribute......
pthread_attr_t attr;

// semaphore initialization
sem_init(&S1,0,0);
sem_init(&S2,0,0);

// pthread_attr_t initialization
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_JOINABLE);

// Creating thread
void *ptr=&N;
r1=pthread_create(&sort_1,&attr,sort_first,ptr);
if(r1){
	cout<<"Error in creating thread"<<endl;
	exit(-1);
}
r1=pthread_create(&sort_2,&attr,sort_second,ptr);
if(r1){
	cout<<"Error in creating thread"<<endl;
	exit(-1);
}
r1=pthread_create(&merge,&attr,merge_array,ptr);
if(r1){
	cout<<"Error in creating thread"<<endl;
	exit(-1);
}

// destroying the pthread_attr
pthread_attr_destroy(&attr);

// Joining the thread
r1=pthread_join(sort_1,NULL);
if(r1){
	cout<<"Error in joining thread"<<endl;
	exit(-1);
}
r1=pthread_join(sort_2,NULL);
if(r1){
	cout<<"Error in joining thread"<<endl;
	exit(-1);
}
r1=pthread_join(merge,NULL);
if(r1){
	cout<<"Error in joining thread"<<endl;
	exit(-1);
}
```

```
        // Exiting thread
        pthread_exit(NULL);

        return 0;
}
```

Output of the Program:


C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_2\merge_sort_using_sem.exe

```
Enter the total number of array you want to enter
5
Enter the array
6
5
4
3
8
Final array is :
3  4  5  6  8

--------------------------------
Process exited after 17.33 seconds with return value 0
Press any key to continue . . .
```

Name: Ashutosh Soni
Id: 2018ucp1505

ASSIGNMENT-3

Q1: Implement Critical Section problem using semaphores with a monitor.
Ans:

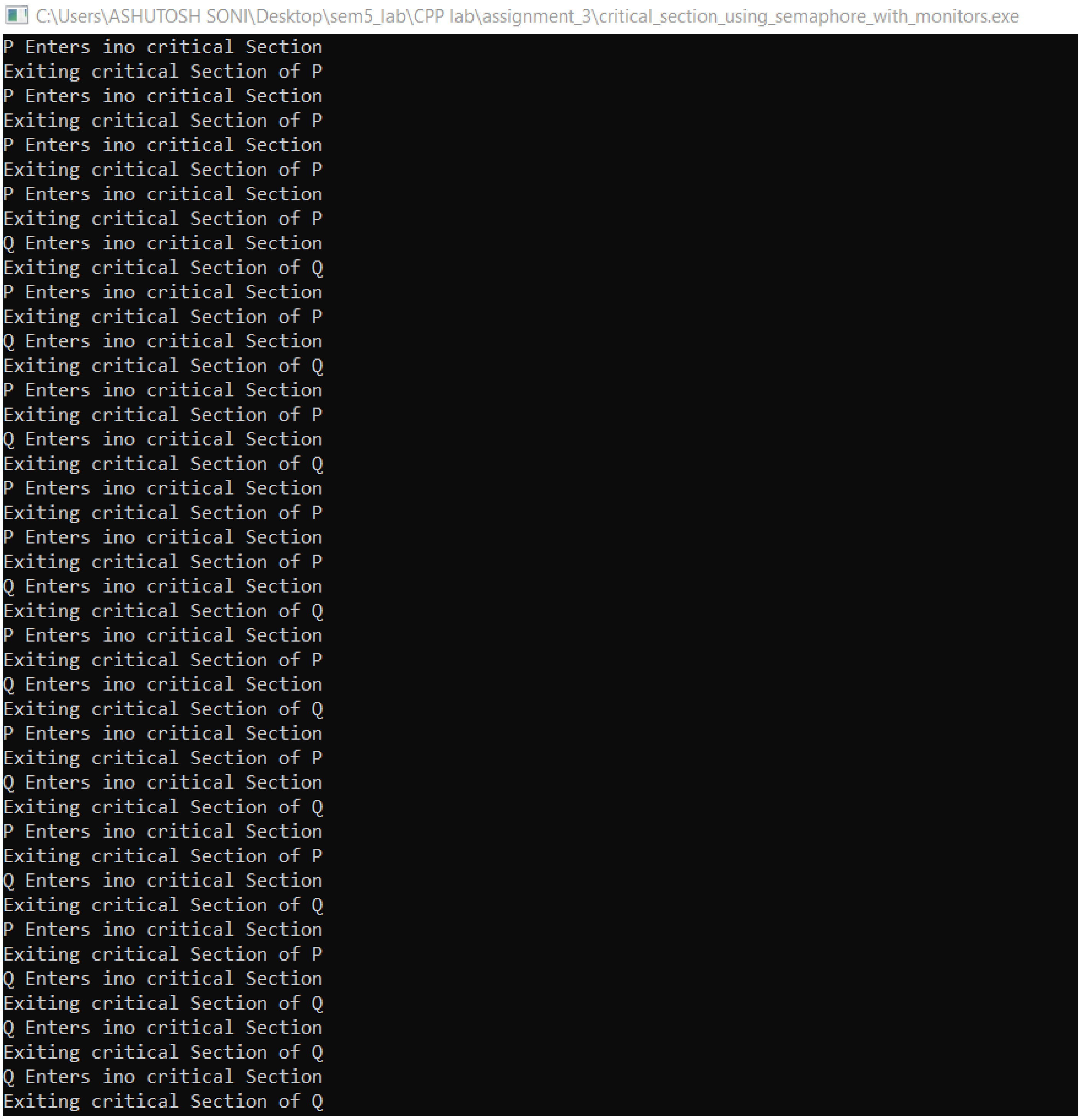
Critical Section problem using semaphores with a monitor

```
// Header file include
#include<bits/stdc++.h>
#include<pthread.h>
using namespace std;

int times=0;
int x=0;

class monitor {
	// Variables
	int s;

	// condition variable for not Zero
	pthread_cond_t notZero;

	// mutex variable for synchronization
	pthread_mutex_t condLock;

	public:
		// Operation wait
		void wait(){
			pthread_mutex_lock(&condLock);
			if(s==0){
				pthread_cond_wait(&notZero,&condLock);
			}
			s=s-1;
			pthread_mutex_unlock(&condLock);
		}

		// Operation Signal
		void signal(){
			pthread_mutex_lock(&condLock);
			s=s+1;
			pthread_cond_signal(&notZero);
			pthread_mutex_unlock(&condLock);
		}

		// Constructor
		monitor(){
			// s=k
```

```
                    s=2;
                    pthread_cond_init(&notZero,NULL);
                    pthread_mutex_init(&condLock,NULL);
          }

          // Destructor
          ~monitor(){
                    pthread_cond_destroy(&notZero);
                    pthread_mutex_destroy(&condLock);
          }
}

// Global Object of Monitor
Sem
;


// Critical Section of the Problem

void critical_section(){
          cout<<"Enters ino critical Section"<<endl;
          x++;
          cout<<"Exiting critical Section of ";
}

//  Main Process  for P and Q

void* process_P(void *arg){

          // Loop Forever
          while(times<100){
                    // Non Critical Section
                    times++;
                    // Wait Operation
                    Sem.wait();
                    cout<<"P ";
                    // Critical Section code
                    critical_section();
                    // Signal Operation
                    cout<<"P"<<endl;
                    Sem.signal();
          }

}

void* process_Q(void *arg){

          // Loop Forever
          while(times<100){
                    // Non Critical Section
```

```
		times++;
		// Wait Operation
		Sem.wait();
		cout<<"Q ";
		// Critical Section code
		critical_section();
		// Signal Operation
		cout<<"Q"<<endl;
		Sem.signal();
	}

}

int main(){
	// Declaration
	pthread_t process_p, process_q;
	pthread_attr_t attr;

	// Initialization
	pthread_attr_init(&attr);
	pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_JOINABLE);

	// Creation
	pthread_create(&process_p,&attr,process_P,NULL);
	pthread_create(&process_q,&attr,process_Q,NULL);

	// Joining
	pthread_join(process_p,NULL);
	pthread_join(process_q,NULL);

	// Destroying
	pthread_attr_destroy(&attr);
	pthread_exit(NULL);

	return 0;
}
```

Output of the program:



```
C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_3\critical_section_using_semaphore_with_monitors.exe
P Enters ino critical Section
Exiting critical Section of P
P Enters ino critical Section
Exiting critical Section of P
P Enters ino critical Section
Exiting critical Section of P
P Enters ino critical Section
Exiting critical Section of P
Q Enters ino critical Section
Exiting critical Section of Q
P Enters ino critical Section
Exiting critical Section of P
Q Enters ino critical Section
Exiting critical Section of Q
P Enters ino critical Section
Exiting critical Section of P
Q Enters ino critical Section
Exiting critical Section of Q
P Enters ino critical Section
Exiting critical Section of P
P Enters ino critical Section
Exiting critical Section of P
Q Enters ino critical Section
Exiting critical Section of Q
P Enters ino critical Section
Exiting critical Section of P
Q Enters ino critical Section
Exiting critical Section of Q
P Enters ino critical Section
Exiting critical Section of P
Q Enters ino critical Section
Exiting critical Section of Q
P Enters ino critical Section
Exiting critical Section of P
Q Enters ino critical Section
Exiting critical Section of Q
P Enters ino critical Section
Exiting critical Section of P
Q Enters ino critical Section
Exiting critical Section of Q
Q Enters ino critical Section
Exiting critical Section of Q
Q Enters ino critical Section
Exiting critical Section of Q
```

Q2: Implement the solution of producer-consumer bounded buffer problem with a monitor.

Ans:

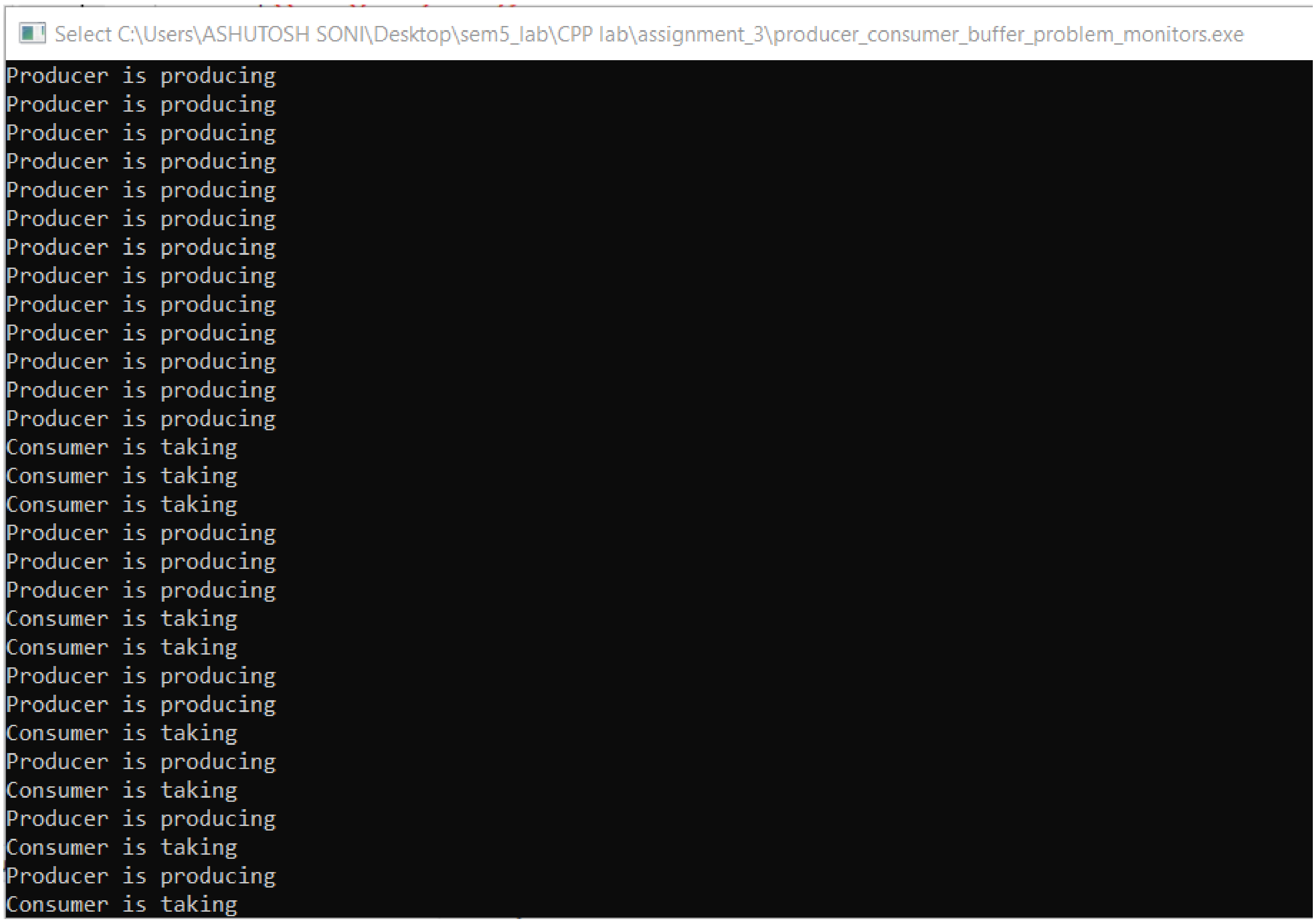
Producer-consumer bounded buffer problem with a monitor

```
// Header file include
#include<bits/stdc++.h>
#include<pthread.h>
#include<unistd.h>
using namespace std;
```

```
int times=0;

class Monitor{
        // buffer for the store
        int buffer=0;

        // capacity of the store
        int capacity;

        // condtion variable for Not Empty and Not Full
        pthread_cond_t notEmpty,notFull;

        // mutex variable for synchorization
        pthread_mutex_t condLock;

        public:

                // Append operation
                void append(){
                        pthread_mutex_lock(&condLock);
                        cout<<"Producer is producing"<<endl;
                        // Wait for buffer to not Full
                        if(buffer==capacity){
                                pthread_cond_wait(&notFull,&condLock);
                        }
                        buffer++;
                        pthread_cond_signal(&notEmpty);
                        pthread_mutex_unlock(&condLock);
                }

                // Take operation
                void take(){
                        pthread_mutex_lock(&condLock);
                        cout<<"Consumer is taking"<<endl;
                        // Wait for Buffer to not Empty
                        if(buffer==0){
                                pthread_cond_wait(&notEmpty,&condLock);
                        }
                        buffer--;
                        pthread_cond_signal(&notFull);
                        pthread_mutex_unlock(&condLock);
                }


                // Constructor
                Monitor(){
                        capacity=25;
                        pthread_cond_init(&notEmpty,NULL);
                        pthread_cond_init(&notFull,NULL);
                pthread_mutex_init(&condLock,NULL);
            }

            // Destructor
            ~Monitor(){
                pthread_cond_destroy(&notEmpty);
                pthread_cond_destroy(&notFull);
                pthread_mutex_destroy(&condLock);
            }

}
// Global variable of monitor where producer is storing and consumer is taking.....
store;


// Produce Function
void* produce(void *arg){
    while(times<1000){
        sleep((rand()%100)*0.01);
        store.append();
        times++;
    }
}


// Consumer Function
void* consume(void *arg){
    while(times<1000){
        sleep((rand()%100)*0.02);
        store.take();
        times++;
    }
}


int main(){

    // Declaration...

    pthread_t producer, consumer;
    pthread_attr_t attr;

    // Initialization

    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_JOINABLE);

    // Creation
    pthread_create(&producer,&attr,produce,NULL);
    pthread_create(&consumer,&attr,consume,NULL);
```

```
        // Destroying
        pthread_attr_destroy(&attr);
        pthread_exit(NULL);

        return 0;


}
```

Output of the program:



Select C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_3\producer_consumer_buffer_problem_monitors.exe

```
Producer is producing
Producer is producing
Producer is producing
Producer is producing
Producer is producing
Producer is producing
Producer is producing
Producer is producing
Producer is producing
Producer is producing
Producer is producing
Producer is producing
Producer is producing
Consumer is taking
Consumer is taking
Consumer is taking
Producer is producing
Producer is producing
Producer is producing
Consumer is taking
Consumer is taking
Producer is producing
Producer is producing
Consumer is taking
Producer is producing
Consumer is taking
Producer is producing
Consumer is taking
Producer is producing
Consumer is taking
```

Q3: Implement the solution of Readers and writers with a monitor.

Ans:

<u>Readers and writers problem with a monitor</u>

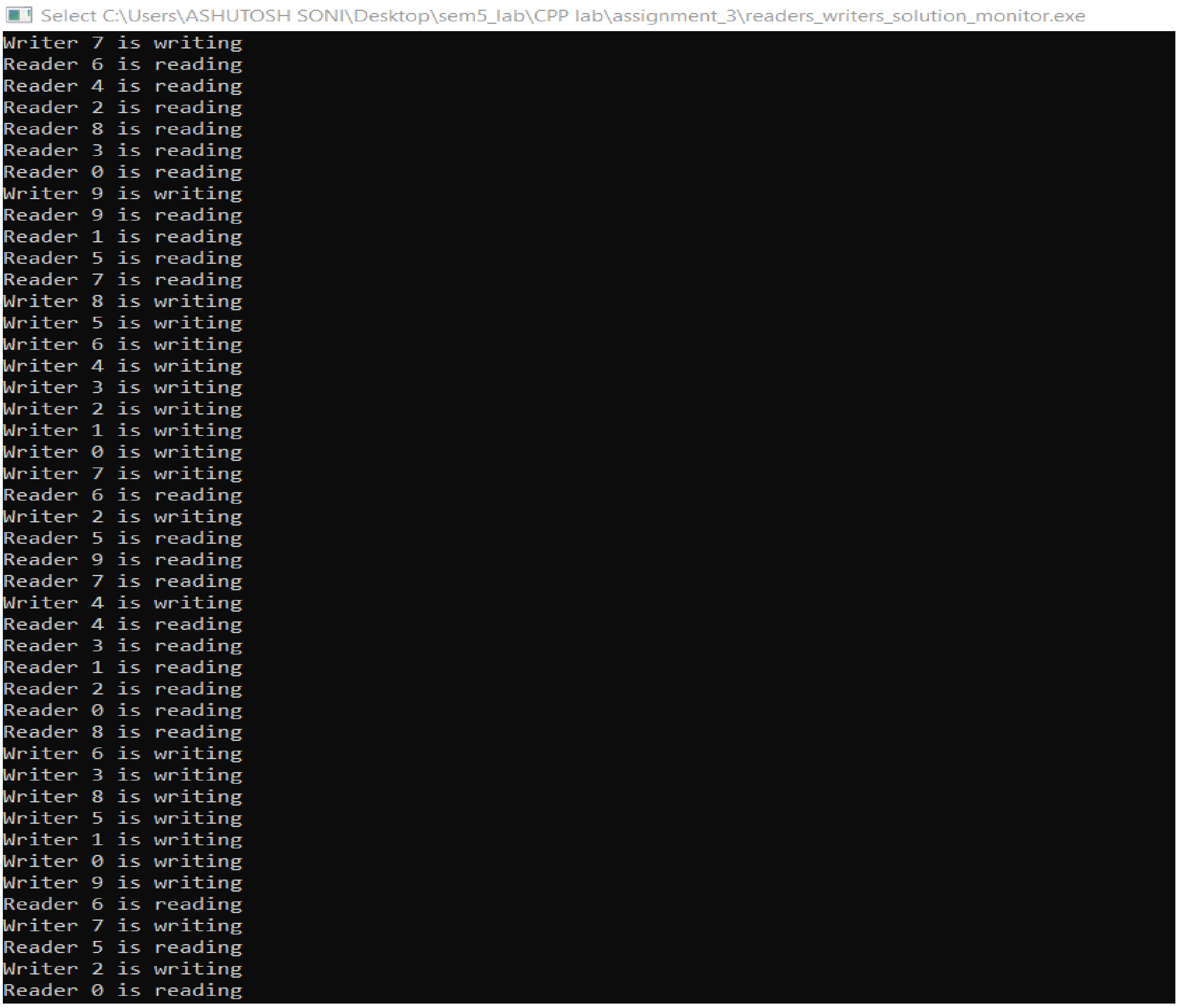
```
// Header file include
#include<bits/stdc++.h>
#include<pthread.h>
#include<unistd.h>
using namespace std;

int items=10;

class monitor {
```

```
// number of readers
int readers;

// number of writers
int writers;

// number of readers waiting
int waitreaders;

// number of writers waiting
int waitwriters;

// condition variable for readers
pthread_cond_t canread;

// condtion variable for writers
pthread_cond_t canwrite;

// mutex for synchornization
pthread_mutex_t condLock;

public:

        // Start read Function
        void start_read(int i){

                pthread_mutex_lock(&condLock);

                if(writers == 1 and waitwriters > 0){
                        waitreaders++;
                        pthread_cond_wait(&canread,&condLock);
                        waitreaders--;
                }

                readers++;
                cout<<"Reader "<< i <<" is reading"<<endl;

                pthread_mutex_unlock(&condLock);

                pthread_cond_broadcast(&canread);

        }

        // End read function
        void end_read(int i){

                pthread_mutex_lock(&condLock);

                if(--readers == 0){
                        pthread_cond_signal(&canwrite);
                }

                pthread_mutex_unlock(&condLock);

        }

        // Start write Function
        void start_write(int i){

                pthread_mutex_lock(&condLock);

                if(writers == 1 or readers > 0){
                        ++waitwriters;
                        pthread_cond_wait(&canwrite,&condLock);
                        --waitwriters;
                }
                writers = 1;
                cout<<"Writer "<<i<<" is writing"<<endl;

                pthread_mutex_unlock(&condLock);

        }

        // End Write Function
        void end_write(int i){

                pthread_mutex_lock(&condLock);

                writers =0;

                if(waitreaders > 0){
                        pthread_cond_signal(&canread);
                }
                else{
                        pthread_cond_signal(&canwrite);
                }

                pthread_mutex_unlock(&condLock);

        }

        // constrcutor
        monitor(){
                readers=0;
                writers=0;
                waitreaders=0;
                waitwriters=0;

                pthread_cond_init(&canread,NULL);
                pthread_cond_init(&canwrite,NULL);
```

```
                        pthread_mutex_init(&condLock,NULL);
                }

                // destructor
                ~monitor(){
                        pthread_cond_destroy(&canread);
                        pthread_cond_destroy(&canwrite);

                        pthread_mutex_destroy(&condLock);
                }

}

// Global Object of monitor class handles readers and writers

library
;

// Reader funciton

void* reader(void *arg){

        int c=0;
        int i = *(int*)arg;

        // Read items
        while(c < items){
                sleep(1);
                library.start_read(i);
                library.end_read(i);
                c++;
        }

}


// Writers function

void* writer(void *arg){

        int c=0;
        int i = *(int*)arg;

        while(c < items){
                sleep(1);
                library.start_write(i);
                library.end_write(i);
                c++;
        }
```

```
}

int main(){

        // Declaration
        pthread_t read[items] ,write[items];
        pthread_attr_t attr;
        int id[items];

        // Initalization
        pthread_attr_init(&attr);
        pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_JOINABLE);

        for(int i=0;i<items;i++){
                id[i]= i;

                // Creating thread

                // for readers
                pthread_create(&read[i],&attr,reader,&id[i]);

                // for writers
                pthread_create(&write[i],&attr,writer,&id[i]);
        }

        // Joining threads

        // readers
        for(int i=0;i<items;i++){
                pthread_join(read[i],NULL);
        }

        // writers
        for(int i=0;i<items;i++){
                pthread_join(write[i],NULL);
        }

        // destroying

        pthread_attr_destroy(&attr);
        pthread_exit(NULL);

        return 0;
}
```

Output of the Program:



Select C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_3\readers_writers_solution_monitor.exe

```
Writer 7 is writing
Reader 6 is reading
Reader 4 is reading
Reader 2 is reading
Reader 8 is reading
Reader 3 is reading
Reader 0 is reading
Writer 9 is writing
Reader 9 is reading
Reader 1 is reading
Reader 5 is reading
Reader 7 is reading
Writer 8 is writing
Writer 5 is writing
Writer 6 is writing
Writer 4 is writing
Writer 3 is writing
Writer 2 is writing
Writer 1 is writing
Writer 0 is writing
Writer 7 is writing
Reader 6 is reading
Writer 2 is writing
Reader 5 is reading
Reader 9 is reading
Reader 7 is reading
Writer 4 is writing
Reader 4 is reading
Reader 3 is reading
Reader 1 is reading
Reader 2 is reading
Reader 0 is reading
Reader 8 is reading
Writer 6 is writing
Writer 3 is writing
Writer 8 is writing
Writer 5 is writing
Writer 1 is writing
Writer 0 is writing
Writer 9 is writing
Reader 6 is reading
Writer 7 is writing
Reader 5 is reading
Writer 2 is writing
Reader 0 is reading
```

Q4: Implement the solution of Dining philosophers with a monitor.
Ans:

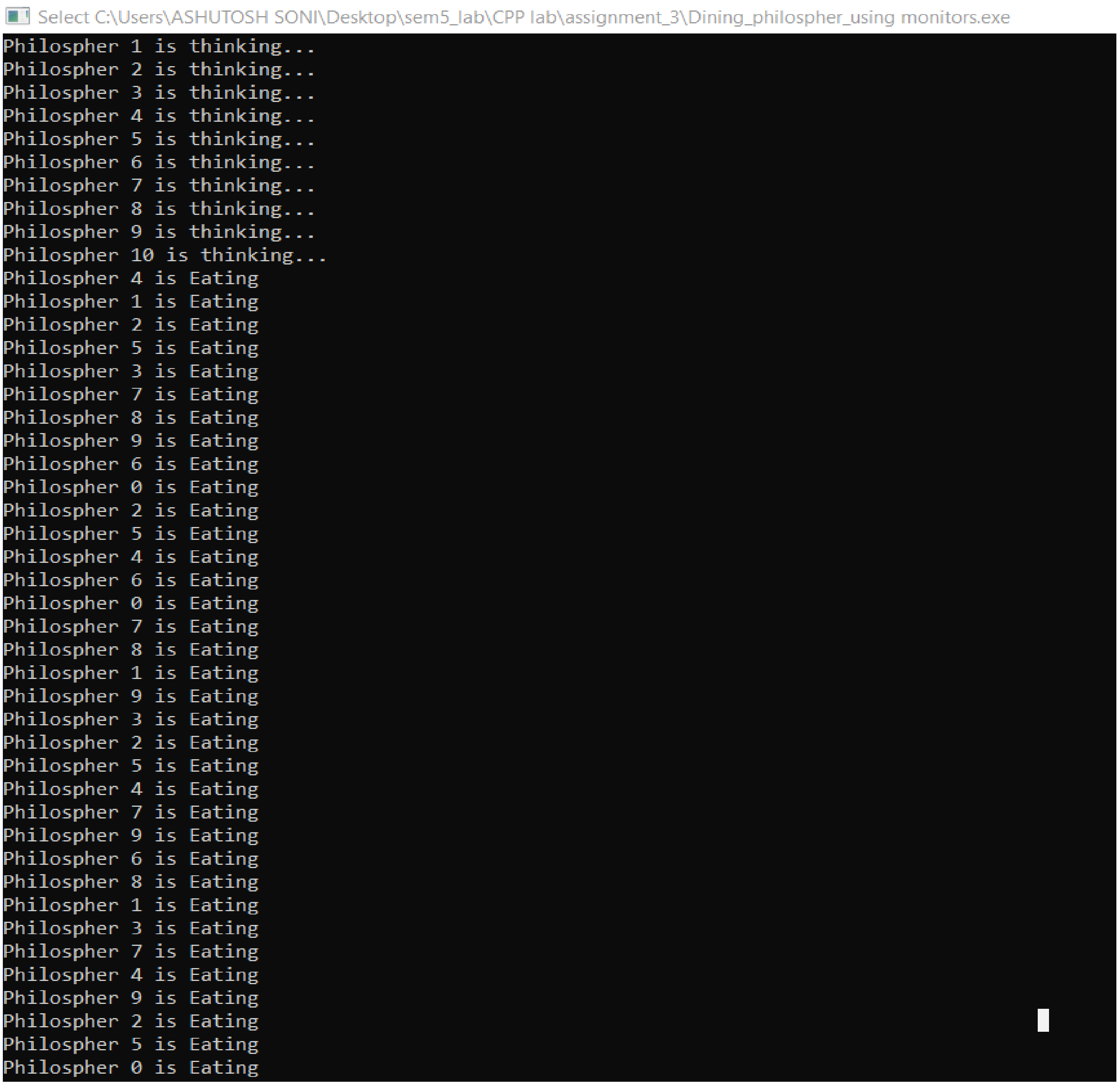
<u>Dining philosophers Problem with a monitor</u>

```
// Header file include
#include<bits/stdc++.h>
#include<pthread.h>
#include<unistd.h>
using namespace std;

#define N 10
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4)%N
#define RIGHT (phnum + 1)%N
```

```
// Philospher index
int phil[N];
int times=200;

class monitor {

        // state of the philospher
        int state[N];

        // Philospher condition variable
        pthread_cond_t phcond[N];

        // mutex variable for synchronization
        pthread_mutex_t condLock;

        public:

                // Test for the desired condtion
                // i.e. Left and Right philospher are not reading
                void test(int phnum){

                        if(state[(phnum+1)%5] != EATING and state[(phnum+4)%5] != EATING and state[phnum]
==HUNGRY){
                                state[phnum] = EATING;

                                pthread_cond_signal(&phcond[phnum]);
                        }

                }

                // Take Fork function
                void take_fork(int phnum){

                        pthread_mutex_lock(&condLock);

                        // Indicates it is hungry
                        state[phnum]=HUNGRY;

                        // test for condition
                        test(phnum);

                        // If unable to eat.. wait for the signal
                        if(state[phnum]!=EATING){
                                pthread_cond_wait(&phcond[phnum],&condLock);
                        }
                        cout<<"Philospher "<<phnum<<" is Eating"<<endl;

                        pthread_mutex_unlock(&condLock);
                }

                // Put Fork function
                void put_fork(int phnum){

                        pthread_mutex_lock(&condLock);

                        // Indicates that I am thinking
                        state[phnum]=THINKING;

                        test(RIGHT);
                        test(LEFT);

                        pthread_mutex_unlock(&condLock);

                }

                // constructor
                monitor(){

                        for(int i=0;i<N;i++){
                                state[i] = THINKING;
                        }

                        for(int i=0;i<N;i++){
                                pthread_cond_init(&phcond[i],NULL);
                        }

                        pthread_mutex_init(&condLock,NULL);
                }

                // destructor
                ~monitor(){

                        for(int i=0;i<N;i++){
                                pthread_cond_destroy(&phcond[i]);
                        }

                        pthread_mutex_destroy(&condLock);
                }


}

// Global Object of the monitor
phil_object;

void* philospher(void *arg){
        int c=0;
        while(c<times){
```

```
		int i = *(int*)arg;
		sleep(1);
		phil_object.take_fork(i);
		sleep(0.5);
		phil_object.put_fork(i);
		c++;
	}
}

int main(){

	// Declaration...
	pthread_t thread_id[N];
	pthread_attr_t attr;

	// Initialization...
	pthread_attr_init(&attr);
	pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_JOINABLE);

	for(int i=0;i<N;i++){
		phil[i]=i;
	}

	// Creating...
	for(int i=0;i<N;i++){
		pthread_create(&thread_id[i],&attr,philospher,&phil[i]);
		cout<<"Philospher "<<i+1<<" is thinking..."<<endl;
	}

	// Joining....
	for(int i=0;i<N;i++){
		pthread_join(thread_id[i],NULL);
	}

	// Destroying
	pthread_attr_destroy(&attr);
	pthread_exit(NULL);

	return 0;
}
```

Output of the program:


Select C:\Users\ASHUTOSH SONI\Desktop\sem5_lab\CPP lab\assignment_3\Dining_philospher_using monitors.exe

```
Philospher 1 is thinking...
Philospher 2 is thinking...
Philospher 3 is thinking...
Philospher 4 is thinking...
Philospher 5 is thinking...
Philospher 6 is thinking...
Philospher 7 is thinking...
Philospher 8 is thinking...
Philospher 9 is thinking...
Philospher 10 is thinking...
Philospher 4 is Eating
Philospher 1 is Eating
Philospher 2 is Eating
Philospher 5 is Eating
Philospher 3 is Eating
Philospher 7 is Eating
Philospher 8 is Eating
Philospher 9 is Eating
Philospher 6 is Eating
Philospher 0 is Eating
Philospher 2 is Eating
Philospher 5 is Eating
Philospher 4 is Eating
Philospher 6 is Eating
Philospher 0 is Eating
Philospher 7 is Eating
Philospher 8 is Eating
Philospher 1 is Eating
Philospher 9 is Eating
Philospher 3 is Eating
Philospher 2 is Eating
Philospher 5 is Eating
Philospher 4 is Eating
Philospher 7 is Eating
Philospher 9 is Eating
Philospher 6 is Eating
Philospher 8 is Eating
Philospher 1 is Eating
Philospher 3 is Eating
Philospher 7 is Eating
Philospher 4 is Eating
Philospher 9 is Eating
Philospher 2 is Eating
Philospher 5 is Eating
Philospher 0 is Eating
```