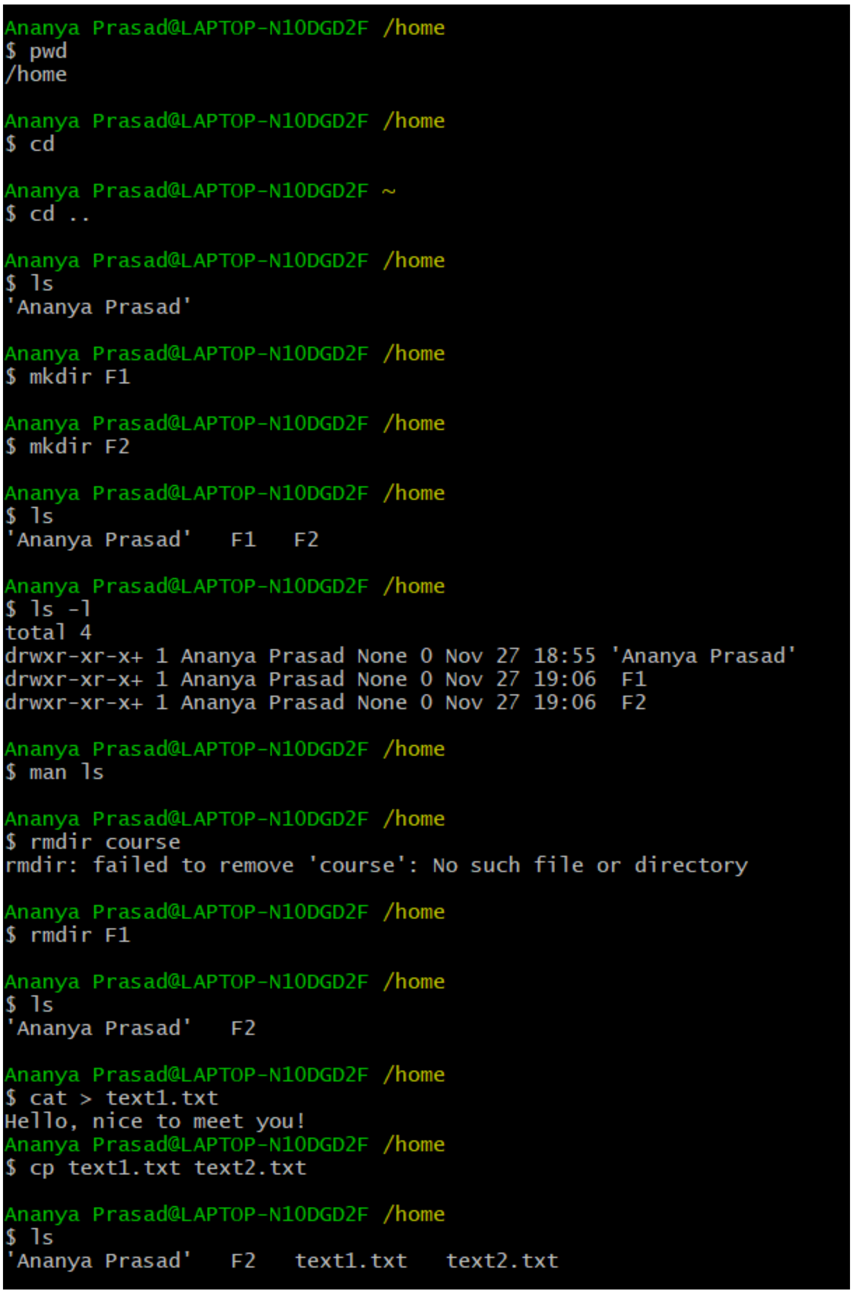
TUTORIAL 2

NAME: Ananya Prasad **PROF:** Dr. Abha Trivedi

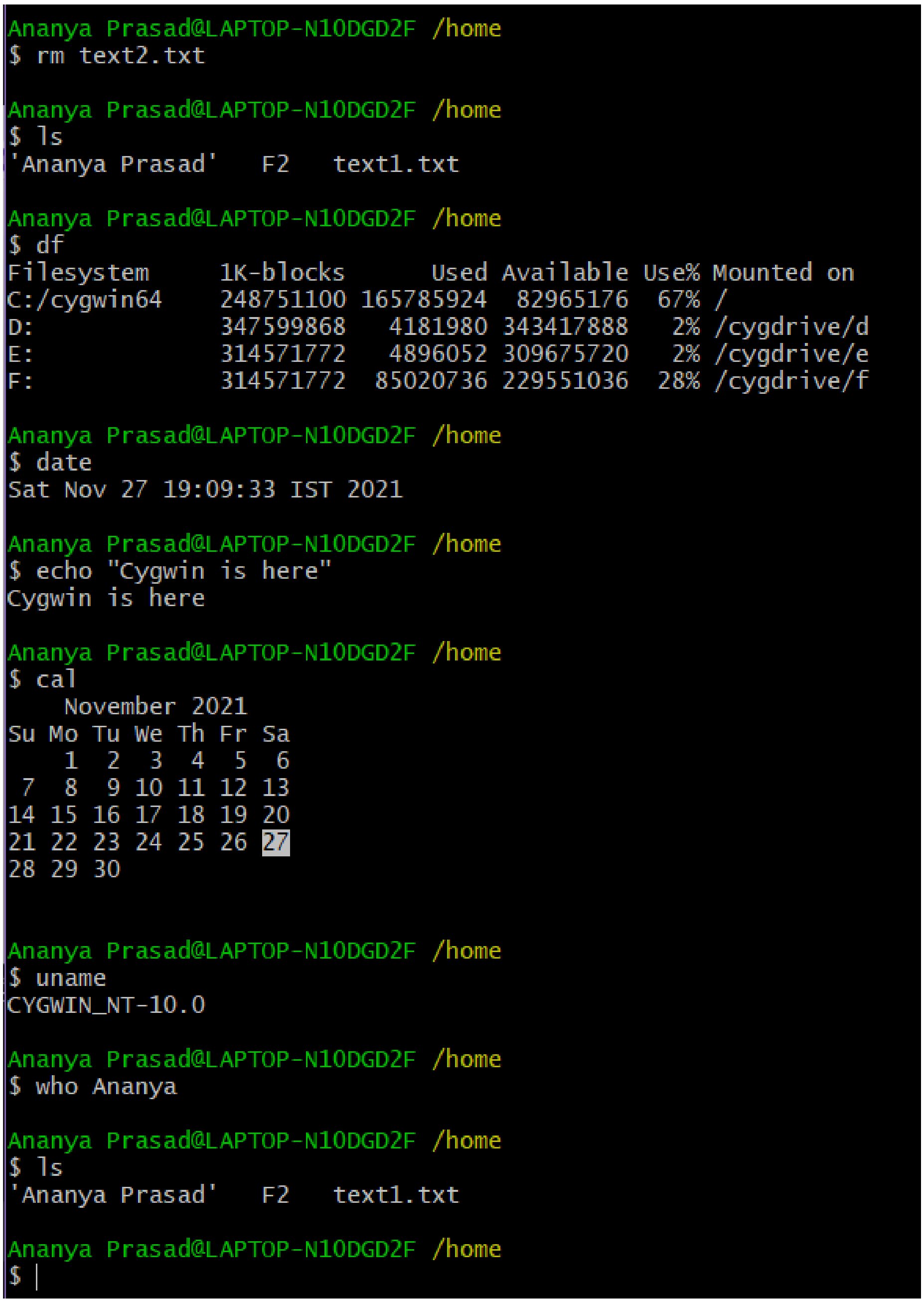
REGNO: 20BCE10093 **SEM/SLOT:** 3/B21+B22+B23

A) Study basic UNIX commands:

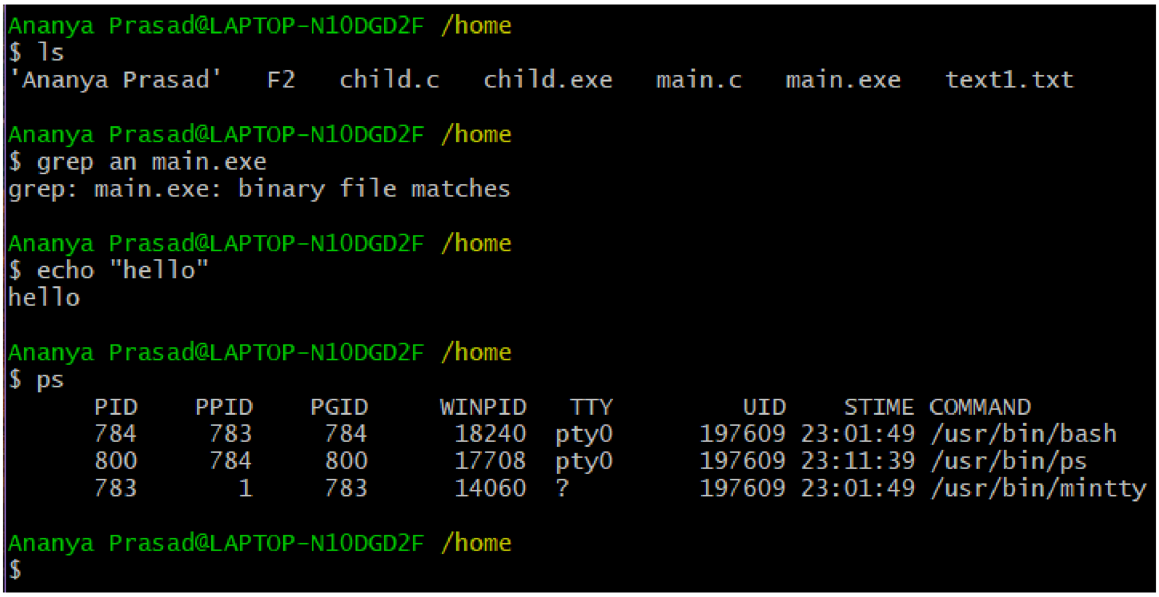
cat, cd, cp, chmod, df, less, ls, mkdir, more, mv, pwd, rmdir, rm, man, uname, who, ps, vi, cal, date, echo, bc, grep



```
Ananya Prasad@LAPTOP-N10DGD2F /home
$ pwd
/home

Ananya Prasad@LAPTOP-N10DGD2F /home
$ cd

Ananya Prasad@LAPTOP-N10DGD2F ~
$ cd ..

Ananya Prasad@LAPTOP-N10DGD2F /home
$ ls
'Ananya Prasad'

Ananya Prasad@LAPTOP-N10DGD2F /home
$ mkdir F1

Ananya Prasad@LAPTOP-N10DGD2F /home
$ mkdir F2

Ananya Prasad@LAPTOP-N10DGD2F /home
$ ls
'Ananya Prasad'   F1   F2

Ananya Prasad@LAPTOP-N10DGD2F /home
$ ls -l
total 4
drwxr-xr-x+ 1 Ananya Prasad None 0 Nov 27 18:55 'Ananya Prasad'
drwxr-xr-x+ 1 Ananya Prasad None 0 Nov 27 19:06  F1
drwxr-xr-x+ 1 Ananya Prasad None 0 Nov 27 19:06  F2

Ananya Prasad@LAPTOP-N10DGD2F /home
$ man ls

Ananya Prasad@LAPTOP-N10DGD2F /home
$ rmdir course
rmdir: failed to remove 'course': No such file or directory

Ananya Prasad@LAPTOP-N10DGD2F /home
$ rmdir F1

Ananya Prasad@LAPTOP-N10DGD2F /home
$ ls
'Ananya Prasad'   F2

Ananya Prasad@LAPTOP-N10DGD2F /home
$ cat > text1.txt
Hello, nice to meet you!
Ananya Prasad@LAPTOP-N10DGD2F /home
$ cp text1.txt text2.txt

Ananya Prasad@LAPTOP-N10DGD2F /home
$ ls
'Ananya Prasad'   F2   text1.txt   text2.txt
```



```
Ananya Prasad@LAPTOP-N10DGD2F /home
$ rm text2.txt

Ananya Prasad@LAPTOP-N10DGD2F /home
$ ls
'Ananya Prasad'   F2   text1.txt

Ananya Prasad@LAPTOP-N10DGD2F /home
$ df
Filesystem     1K-blocks      Used Available Use% Mounted on
C:/cygwin64    248751100 165785924  82965176  67% /
D:             347599868   4181980 343417888   2% /cygdrive/d
E:             314571772   4896052 309675720   2% /cygdrive/e
F:             314571772  85020736 229551036  28% /cygdrive/f

Ananya Prasad@LAPTOP-N10DGD2F /home
$ date
Sat Nov 27 19:09:33 IST 2021

Ananya Prasad@LAPTOP-N10DGD2F /home
$ echo "Cygwin is here"
Cygwin is here

Ananya Prasad@LAPTOP-N10DGD2F /home
$ cal
    November 2021
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30


Ananya Prasad@LAPTOP-N10DGD2F /home
$ uname
CYGWIN_NT-10.0

Ananya Prasad@LAPTOP-N10DGD2F /home
$ who Ananya

Ananya Prasad@LAPTOP-N10DGD2F /home
$ ls
'Ananya Prasad'   F2   text1.txt

Ananya Prasad@LAPTOP-N10DGD2F /home
$ |
```



```
Ananya Prasad@LAPTOP-N10DGD2F /home
$ ls
'Ananya Prasad'   F2   child.c   child.exe   main.c   main.exe   text1.txt

Ananya Prasad@LAPTOP-N10DGD2F /home
$ grep an main.exe
grep: main.exe: binary file matches

Ananya Prasad@LAPTOP-N10DGD2F /home
$ echo "hello"
hello

Ananya Prasad@LAPTOP-N10DGD2F /home
$ ps
      PID    PPID    PGID     WINPID   TTY         UID    STIME COMMAND
      784     783     784      18240  pty0      197609 23:01:49 /usr/bin/bash
      800     784     800      17708  pty0      197609 23:11:39 /usr/bin/ps
      783       1     783      14060  ?         197609 23:01:49 /usr/bin/mintty

Ananya Prasad@LAPTOP-N10DGD2F /home
$
```

B) USE of FORK():

Implement the C program in which main program accepts an integer array. Main program uses the fork system call to create a new process called a child process. Parent process sorts an integer array and passes the sorted array to child process through the command line arguments of execve system call. The child process uses execve system call to load new program that uses this sorted array for performing the binary search to search the particular item in the array.

MAIN

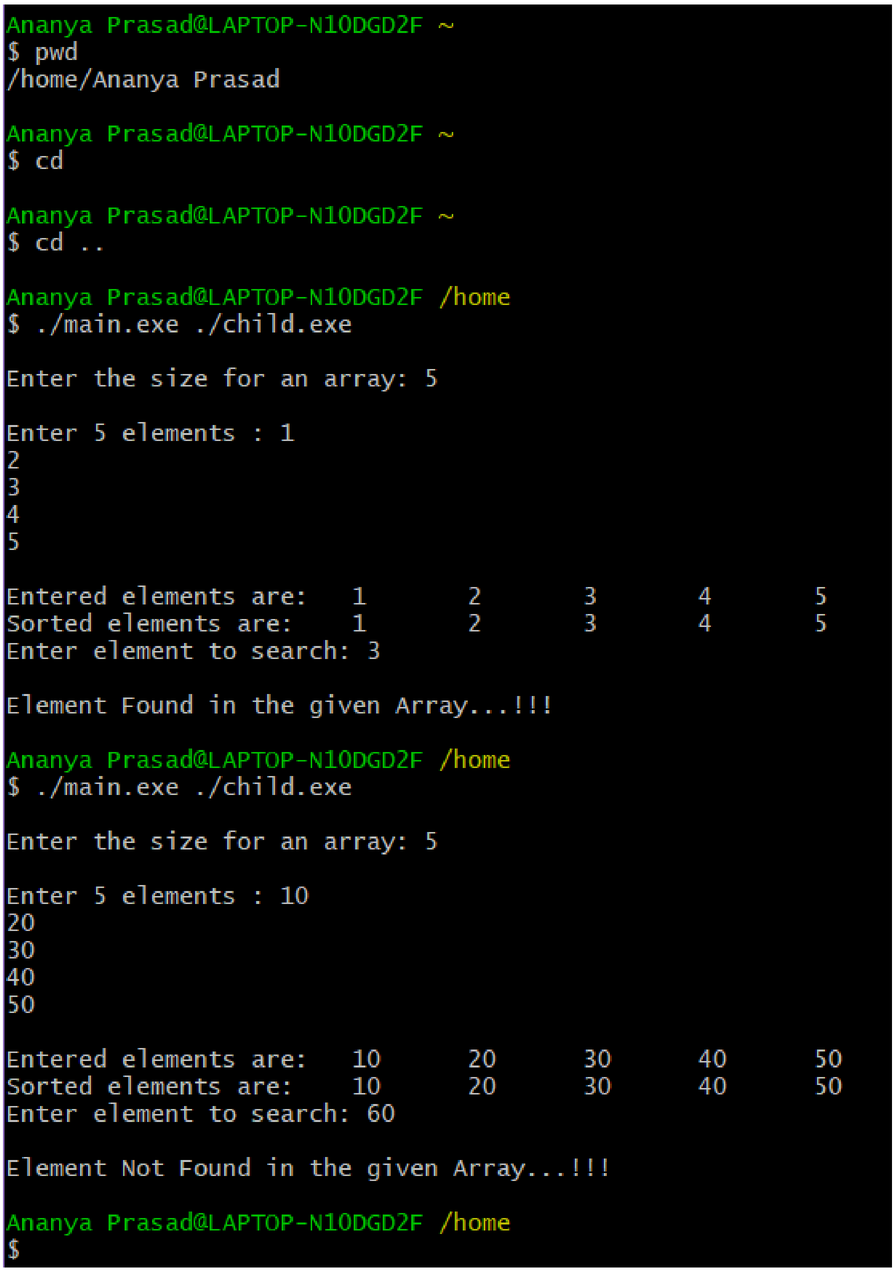
```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

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

{

  int val[10],ele;

  pid_t pid;

  char* cval[10];

  char *newenviron[] = { NULL };

  int i,j,n,temp;

  printf("\nEnter the size for an array: ");

    scanf("%d",&n);

    printf("\nEnter %d elements : ", n);

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

    scanf("%d",&val[i]);


  printf("\nEntered elements are: ");

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

    printf("\t%d",val[i]);


  for(i=1;i<n;i++)

  {

    for(j=0;j<n-1;j++)

    {
        if(val[j]>val[j+1])
        {
            temp=val[j];
            val[j]=val[j+1];
            val[j+1]=temp;
        }
    }
}
printf("\nSorted elements are: ");
for(i=0;i<n;i++)
    printf("\t%d",val[i]);

printf("\nEnter element to search: ");
scanf("%d",&ele);
val[i] = ele;
for (i=0; i < n+1; i++)
    {
         char a[sizeof(int)];
    snprintf(a, sizeof(int), "%d", val[i]);

        cval[i] = malloc(sizeof(a));
        strcpy(cval[i], a);
}
cval[i]=NULL;

pid=fork();
if(pid==0)
{
    execve(argv[1], cval, newenviron);
        perror("Error in execve call...");
}
```

```
}
```

CHILD

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[],char *en[])
{
    int i,j,c,ele;
    int arr[argc];

    for (j = 0; j < argc-1; j++)
    {
        int n=atoi(argv[j]);
        arr[j]=n;
    }
    ele=atoi(argv[j]);
    i=0;
    j=argc-1;
    c=(i+j)/2;
    while(arr[c]!=ele && i<=j)
    {
        if(ele > arr[c])
            i = c+1;
        else
            j = c-1;

        c = (i+j)/2;
    }
    if(i<=j)
        printf("\nElement Found in the given Array...!!!\n");
    else
```

```
        printf("\nElement Not Found in the given Array...!!!\n");

}
```



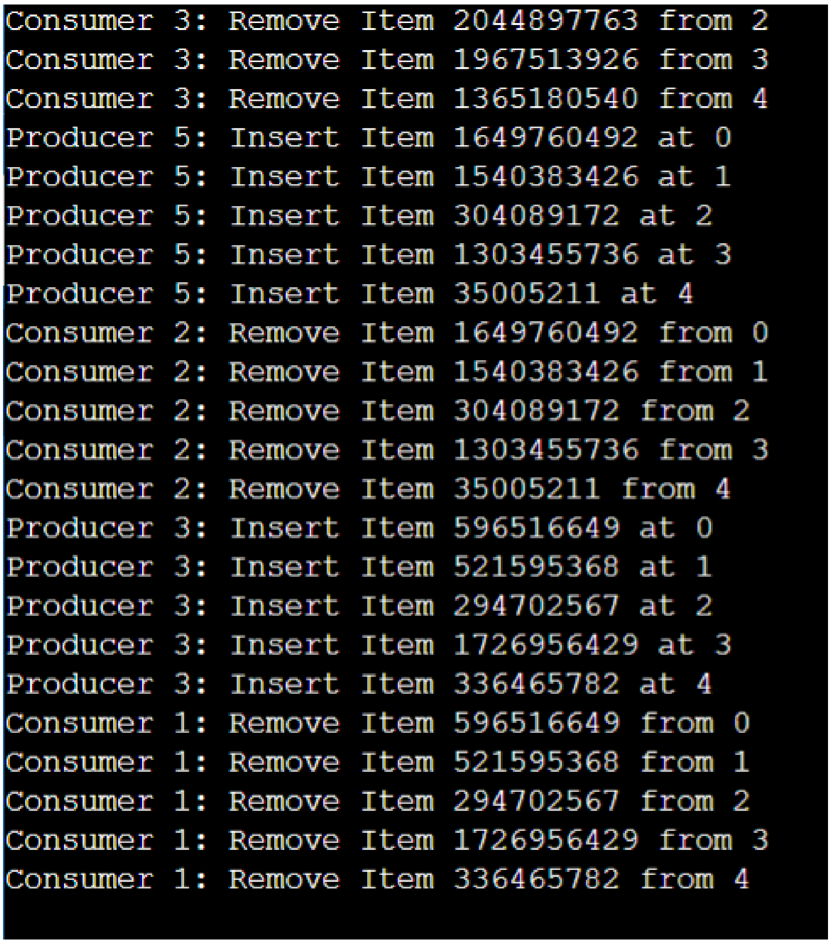
```
Ananya Prasad@LAPTOP-N10DGD2F ~
$ pwd
/home/Ananya Prasad

Ananya Prasad@LAPTOP-N10DGD2F ~
$ cd

Ananya Prasad@LAPTOP-N10DGD2F ~
$ cd ..

Ananya Prasad@LAPTOP-N10DGD2F /home
$ ./main.exe ./child.exe

Enter the size for an array: 5

Enter 5 elements : 1
2
3
4
5

Entered elements are:   1       2       3       4       5
Sorted elements are:    1       2       3       4       5
Enter element to search: 3

Element Found in the given Array...!!!

Ananya Prasad@LAPTOP-N10DGD2F /home
$ ./main.exe ./child.exe

Enter the size for an array: 5

Enter 5 elements : 10
20
30
40
50

Entered elements are:   10      20      30      40      50
Sorted elements are:    10      20      30      40      50
Enter element to search: 60

Element Not Found in the given Array...!!!

Ananya Prasad@LAPTOP-N10DGD2F /home
$
```

C.1) Bound Buffer Problem (Producer - Consumer)

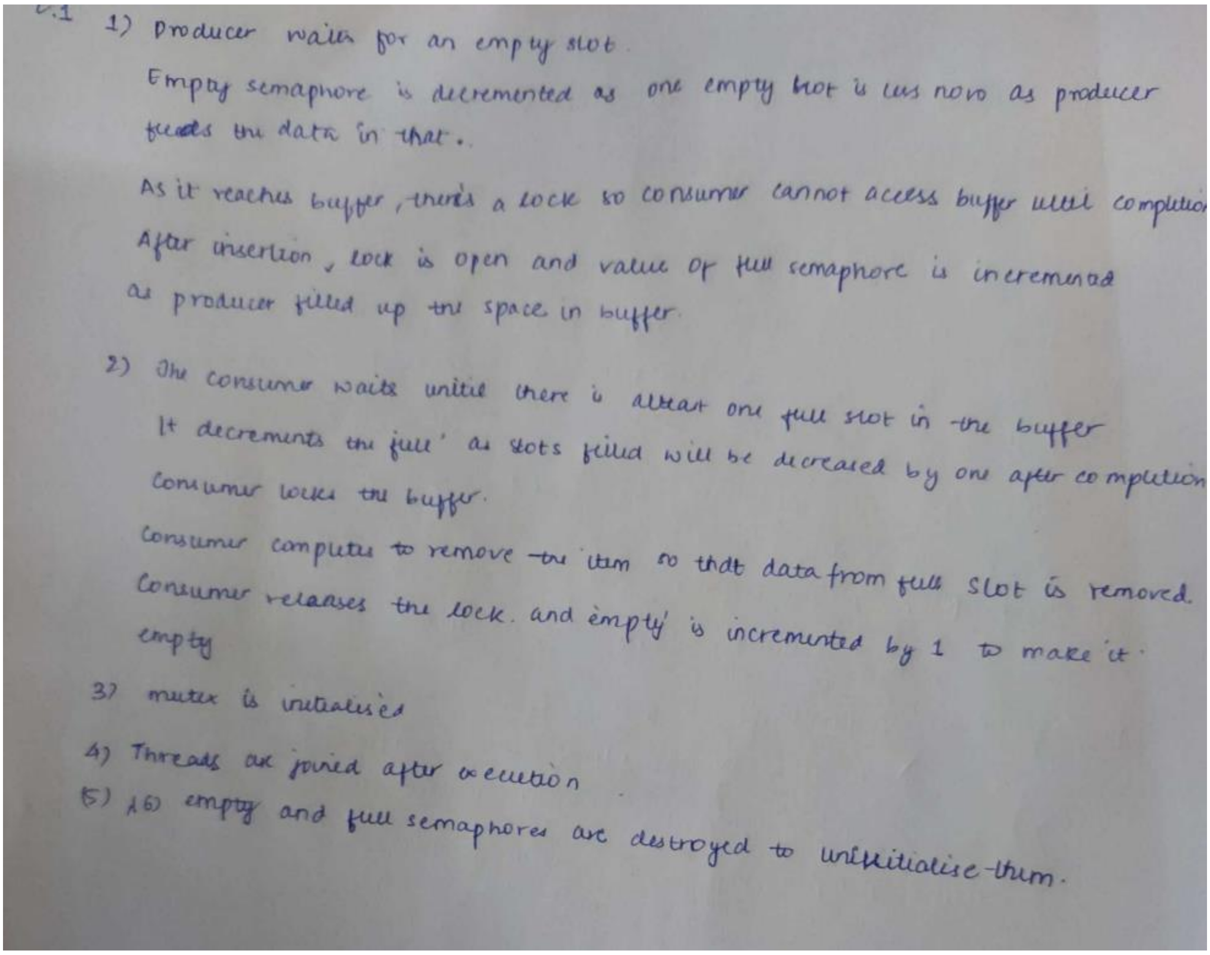
```
#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>
```

```
/*
This program provides a possible solution for producer-consumer problem using mutex and semaphore.
I have used 5 producers and 5 consumers to demonstrate the solution. You can always play with these values.
*/

#define MaxItems 5 // Maximum items a producer can produce or a consumer can consume
#define BufferSize 5 // Size of the buffer

sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;

void *producer(void *pno)       ……………………………………………………………………1
{
   int item;
   for(int i = 0; i < MaxItems; i++) {
      item = rand(); // Produce an random item
      sem_wait(&empty);
      pthread_mutex_lock(&mutex);
      buffer[in] = item;
```

```
        printf("Producer %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);
        in = (in+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
}
void *consumer(void *cno)                 ……………………………………………………………2
{
    for(int i = 0; i < MaxItems; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),item,
out);
        out = (out+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
}

int main()
{

    pthread_t pro[5],con[5];
    pthread_mutex_init(&mutex, NULL);   ……………………………………………………3
    sem_init(&empty,0,BufferSize);
    sem_init(&full,0,0);
    int a[5] = {1,2,3,4,5}; //Just used for numbering the producer and consumer

    for(int i = 0; i < 5; i++) {
        pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(pro[i], NULL);        ……………………………………………………………4
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(con[i], NULL);
    }
    pthread_mutex_destroy(&mutex);  …………………………………………………………..5
    sem_destroy(&empty);
    sem_destroy(&full);

    return 0;
}
```



```
Consumer 3: Remove Item 2044897763 from 2
Consumer 3: Remove Item 1967513926 from 3
Consumer 3: Remove Item 1365180540 from 4
Producer 5: Insert Item 1649760492 at 0
Producer 5: Insert Item 1540383426 at 1
Producer 5: Insert Item 304089172 at 2
Producer 5: Insert Item 1303455736 at 3
Producer 5: Insert Item 35005211 at 4
Consumer 2: Remove Item 1649760492 from 0
Consumer 2: Remove Item 1540383426 from 1
Consumer 2: Remove Item 304089172 from 2
Consumer 2: Remove Item 1303455736 from 3
Consumer 2: Remove Item 35005211 from 4
Producer 3: Insert Item 596516649 at 0
Producer 3: Insert Item 521595368 at 1
Producer 3: Insert Item 294702567 at 2
Producer 3: Insert Item 1726956429 at 3
Producer 3: Insert Item 336465782 at 4
Consumer 1: Remove Item 596516649 from 0
Consumer 1: Remove Item 521595368 from 1
Consumer 1: Remove Item 294702567 from 2
Consumer 1: Remove Item 1726956429 from 3
Consumer 1: Remove Item 336465782 from 4
```



C.1 1) Producer waits for an empty slot.

Empty semaphore is decremented as one empty slot is less now as producer feeds the data in that.

As it reaches buffer, there's a lock so consumer cannot access buffer until completion

After insertion, lock is open and value of full semaphore is incremenad as producer filled up the space in buffer.

2) The consumer waits until there is atleast one full slot in the buffer

It decrements the full' as slots filled will be decreased by one after completion

Consumer locks the buffer.

Consumer computes to remove the item so that data from full slot is removed.

Consumer relaases the lock. and empty' is incremented by 1 to make it empty

3) mutex is initialised

4) Threads are joined after execution

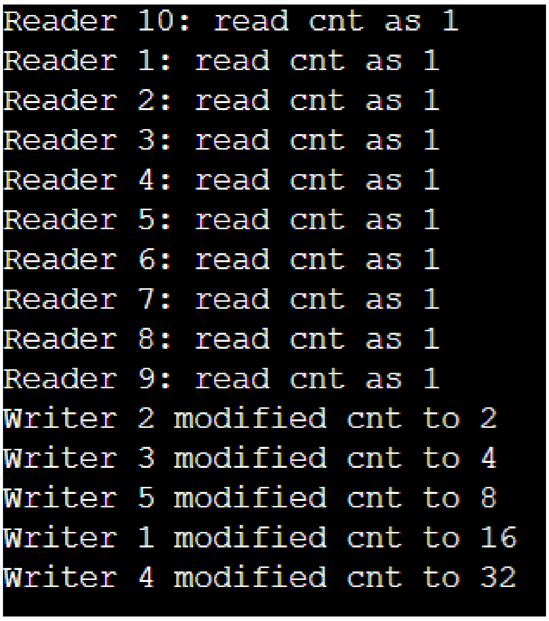
5) 6) empty and full semaphores are destroyed to uninitialise them.

C.2) Readers-Writers Problem

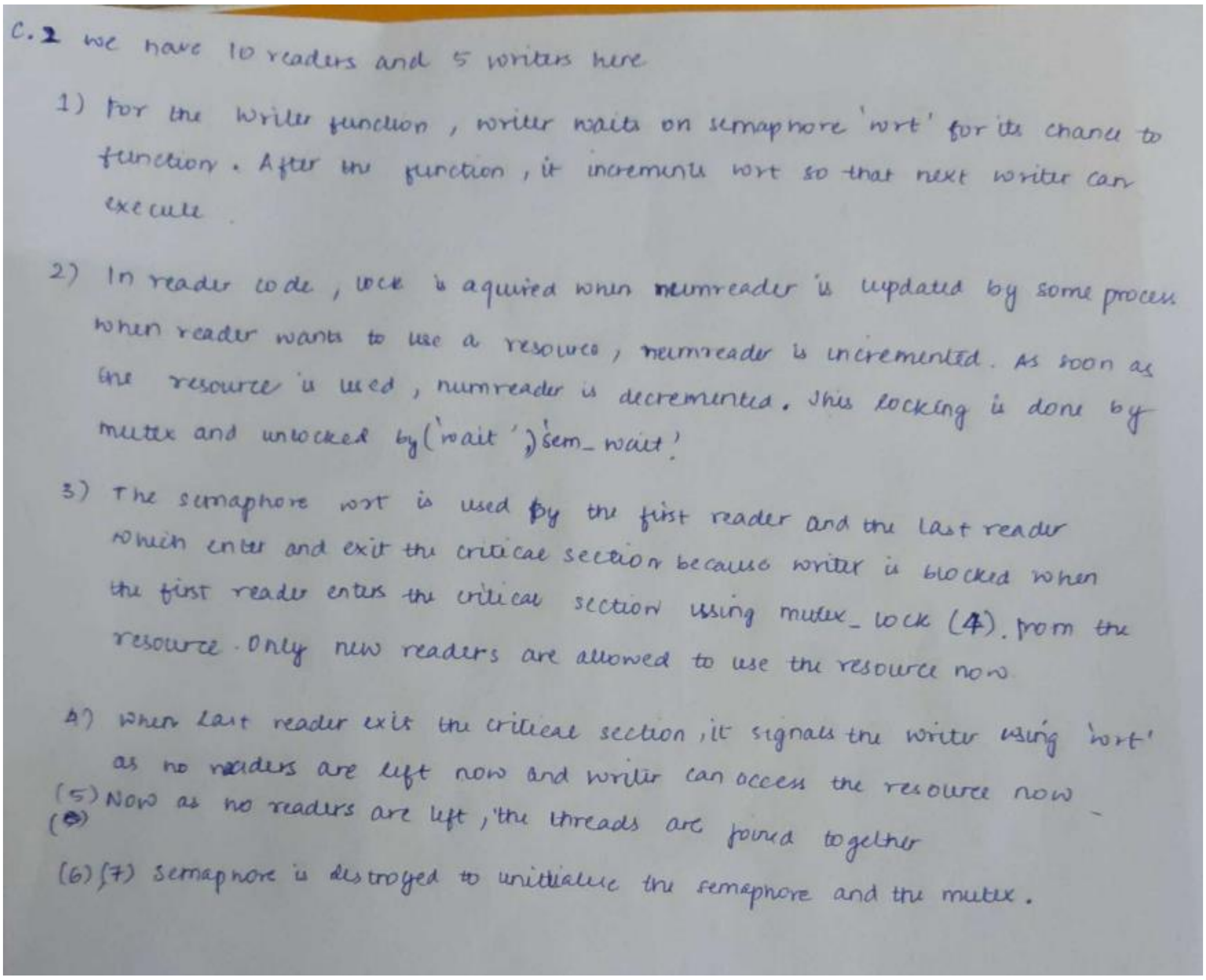
```
#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>


/*

This program provides a possible solution for first readers writers problem using mutex and semaphore.

I have used 10 readers and 5 producers to demonstrate the solution. You can always play with these values.

*/
```

```
sem_t wrt;
pthread_mutex_t mutex;
int cnt = 1;
int numreader = 0;

void *writer(void *wno)              …………………………………………………..1
{
   sem_wait(&wrt);
   cnt = cnt*2;
   printf("Writer %d modified cnt to %d\n",(*((int *)wno)),cnt);
   sem_post(&wrt);

}
void *reader(void *rno)            ……………………………………………………….2
{
   // Reader acquire the lock before modifying numreader
   pthread_mutex_lock(&mutex);
   numreader++;
   if(numreader == 1) {
      sem_wait(&wrt); // If this id the first reader, then it will block the writer
   }
   pthread_mutex_unlock(&mutex);
   // Reading Section      ………………………………………………………………..3
   printf("Reader %d: read cnt as %d\n",*((int *)rno),cnt);

   // Reader acquire the lock before modifying numreader
```

```
    pthread_mutex_lock(&mutex);              ……………………………………………….4
    numreader--;
    if(numreader == 0) {
        sem_post(&wrt); // If this is the last reader, it will wake up the writer.
    }
    pthread_mutex_unlock(&mutex);
}
int main()
{
    pthread_t read[10],write[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&wrt,0,1);
    int a[10] = {1,2,3,4,5,6,7,8,9,10}; //Just used for numbering the producer and consumer
    for(int i = 0; i < 10; i++) {
        pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);
    }

    for(int i = 0; i < 10; i++) {
        pthread_join(read[i], NULL);   ……………………………………………………….5
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(write[i], NULL);
    }
    pthread_mutex_destroy(&mutex); ……………………………………………………………6
    sem_destroy(&wrt);             ……………………………………………………………….7


    return 0;
}
```



```
Reader 10: read cnt as 1
Reader 1: read cnt as 1
Reader 2: read cnt as 1
Reader 3: read cnt as 1
Reader 4: read cnt as 1
Reader 5: read cnt as 1
Reader 6: read cnt as 1
Reader 7: read cnt as 1
Reader 8: read cnt as 1
Reader 9: read cnt as 1
Writer 2 modified cnt to 2
Writer 3 modified cnt to 4
Writer 5 modified cnt to 8
Writer 1 modified cnt to 16
Writer 4 modified cnt to 32
```



C.2 we have 10 readers and 5 writers here

1) For the Writer function, writer waits on semaphore 'wrt' for its chance to function. After the function, it increments wrt so that next writer can execute.

2) In reader code, lock is aquired when numreader is updated by some process when reader wants to use a resource, numreader is incremented. As soon as the resource is used, numreader is decremented. This locking is done by mutex and unlocked by ('wait') 'sem_wait'

3) The semaphore wrt is used by the first reader and the last reader which enter and exit the critical section because writer is blocked when the first reader enters the critical section using mutex_lock (4). from the resource. Only new readers are allowed to use the resource now.

4) When last reader exit the critical section, it signals the writer using 'wrt' as no readers are left now and writer can access the resource now

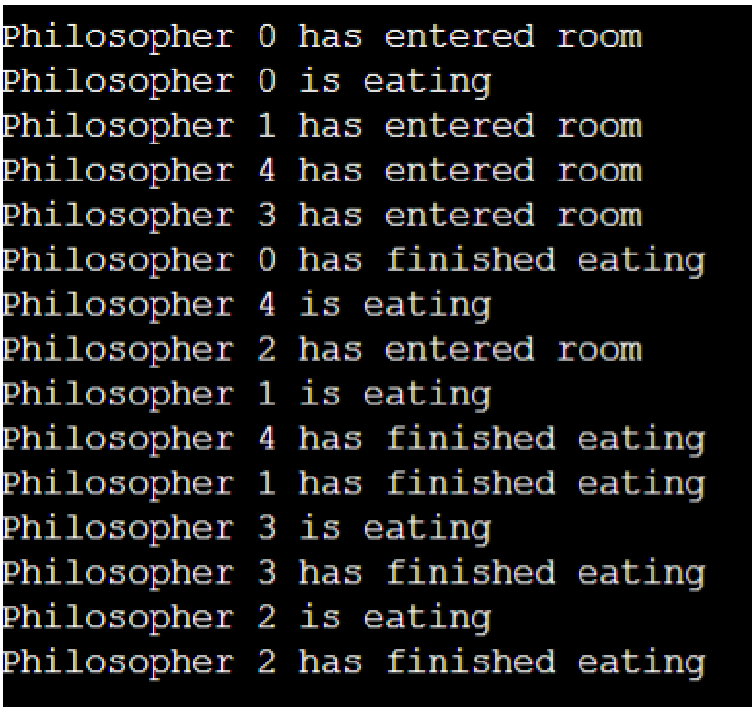
(5) Now as no readers are left, the threads are joined together
(8)

(6)(7) semaphore is destroyed to unitialize the semaphore and the mutex.

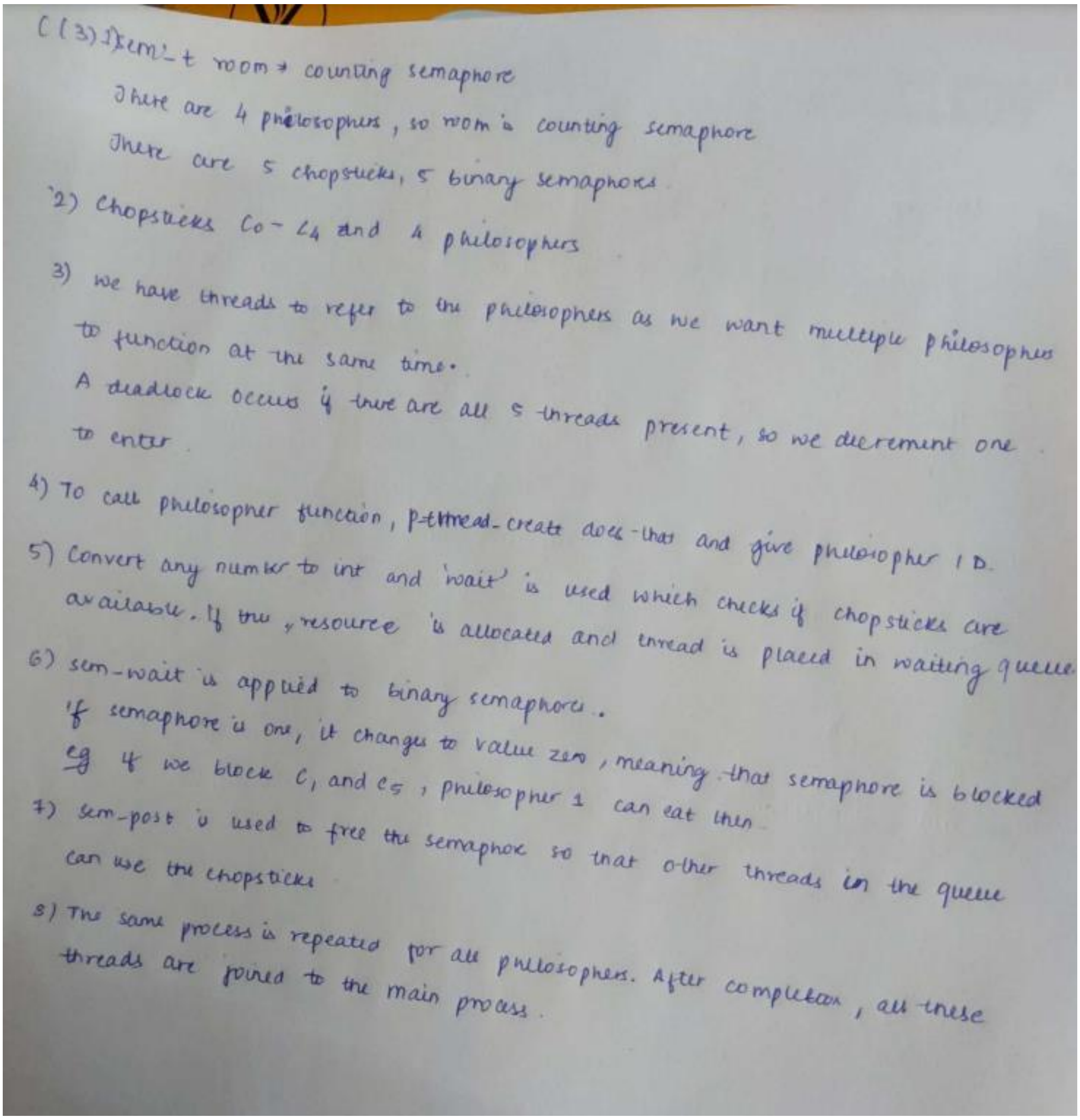
C.3) Dining-Philosopher Problem

```
 #include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
#include<unistd.h>

sem_t room;         …………………………………………………………………………………….1
sem_t chopstick[5];  ……………………………………………………………………………..2

void * philosopher(void *);
void eat(int);
```

```
int main()
{
        int i,a[5];
        pthread_t tid[5];       ………………………………………………………………….3

        sem_init(&room,0,4);

        for(i=0;i<5;i++)
                sem_init(&chopstick[i],0,1);

        for(i=0;i<5;i++){
                a[i]=i;
                pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);
        }
        for(i=0;i<5;i++)
                pthread_join(tid[i],NULL);
}

void * philosopher(void * num)      ………………………………………………………………..4
{
        int phil=*(int *)num;                           ……………………………………………5

        sem_wait(&room);
        printf("\nPhilosopher %d has entered room",phil);
        sem_wait(&chopstick[phil]);
        sem_wait(&chopstick[(phil+1)%5]);
        eat(phil);
        sleep(2);
        printf("\nPhilosopher %d has finished eating",phil);

        sem_post(&chopstick[(phil+1)%5]);
        sem_post(&chopstick[phil]);
        sem_post(&room);
}

void eat(int phil)
{
        printf("\nPhilosopher %d is eating",phil);
}
```



```
Philosopher 0 has entered room
Philosopher 0 is eating
Philosopher 1 has entered room
Philosopher 4 has entered room
Philosopher 3 has entered room
Philosopher 0 has finished eating
Philosopher 4 is eating
Philosopher 2 has entered room
Philosopher 1 is eating
Philosopher 4 has finished eating
Philosopher 1 has finished eating
Philosopher 3 is eating
Philosopher 3 has finished eating
Philosopher 2 is eating
Philosopher 2 has finished eating
```



C(3) 1) sem_t room → counting semaphore

There are 4 philosophers, so room is counting semaphore

There are 5 chopsticks, 5 binary semaphores

2) Chopsticks $C_0 - C_4$ and 4 philosophers

3) We have threads to refer to the philosophers as we want multiple philosophers to function at the same time.

A deadlock occurs if there are all 5 threads present, so we decrement one to enter

4) To call philosopher function, pthread_create does that and give philosopher ID.

5) Convert any number to int and 'wait' is used which checks if chopsticks are available. If true, resource is allocated and thread is placed in waiting queue

6) sem_wait is applied to binary semaphore.

if semaphore is one, it changes to value zero, meaning that semaphore is blocked

eg if we block C_1 and C_5, philosopher 1 can eat then

7) sem_post is used to free the semaphore so that other threads in the queue can use the chopsticks

8) The same process is repeated for all philosophers. After completion, all these threads are joined to the main process.