Name: Abdul Rehman
Roll no: 19L-1135
Section: BSCS-4A
Course: Operating System

Question#1

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>
// initialization of sempahores
sem_t semaphore[3];
pthread_t thread[3];

void* thread1(void* arg);
void* thread2(void* arg);
void* thread3(void* arg);

int main() {
   for(int i = 0; i< 3; i++){ //init semaphore
      sem_init(&semaphore[i], 0, 1);
   }

   pthread_create(&thread[0],NULL,thread1,NULL);
   pthread_create(&thread[1],NULL,thread2,NULL);
   pthread_create(&thread[2],NULL,thread3,NULL);

   for(int i = 0; i< 3; i++){ // join thread
      pthread_join(thread[i],NULL);
```

```
    }

    for(int i = 0; i< 3; i++){ // destory semaphore
        sem_destroy(&semaphore[i]);
    }

    return 0;
}



// thread 1
void* thread1(void* arg) {
    while(1) {
        sem_post(&semaphore[2]);
        printf("aaa");
        sem_wait(&semaphore[1]);
    }
}

// thread 2
void* thread2(void* arg) {

    while(1){
        sem_wait(&semaphore[2]);
        printf("c");

        sem_post(&semaphore[0]);
    }
}
void* thread3(void* arg) {
    while(1){
        sem_wait(&semaphore[0]);
        printf("b");
        sem_post(&semaphore[1]);
    }
```

```
}
```

Question#2

Let assume we have one thread so n = 1. So in if condition (n == count) it will return true so barrier will become one but else then one thread we will stuck at line number 17 and code is wait (barrier).

Question#3

```
#include<iostream>
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>

using namespace std;

class Stack {
private:
        int* a;       // array for stack
        int max;      // max size of array
        int top;      // stack top
public:
        Stack(int m) {
                a = new int[m];          max = m;   top = 0;
        }
        void push(int x) {
                while (top == max);    // if stack is full then wait
                a[top] = x;
                ++top;
        }
        int pop() {
                while (top == 0);        // if stack is empty then wait
                int tmp = top - 1;
```

```
                --top;
                return a[tmp];
        }
};

Stack stack(10);
sem_t s;
sem_t s1;



void* writeInStack(void* num){
   while(1){
      sem_wait(&s);
      int number;
      cout<<"\nEnter number that you wanna enter in stack : ";
      cin>> number;
      stack.push(number);


      sem_post(&s1);
   }
}
void* readFromStack(void* num){
   while(1){
      sem_wait(&s1);
      cout<<endl<<"Number pop from stack : "<<stack.pop();
      sem_post(&s);
   }
}

int main(){
   sem_init(&s,0,1); // initialization
   sem_init(&s1,0,0); // initialization
   pthread_t t1[2];
   pthread_create(&t1[0],NULL,& writeInStack,NULL);
   pthread_create(&t1[1],NULL,& readFromStack,NULL);
```

```
    pthread_join(t1[0],NULL);

    pthread_join(t1[1],NULL);

    return 0;

}
```

Question#4

```
                //share data
semaphore sem[4]
//all semaphore (0,1,2,3) initialize with 1
int count[2];
//both count(0,1) initialize with 0, zero index will store the count of those reborts who move upward
//1st index will store the count of those rebort who will move downward

function UpWardMovementThread
        while(1)
                if(count[1]<0) // if rebort at bottom are not present so they will move downward
                        sem_post(sem[0])
                else
                        sem_wait(sem[1])
                        sem_post(sem[1]);
                sem_wait(sem[0]) // sem for moving upward
                count[0]++
                sem_post(sem[0])
                sem_wait(sem[3]) // wait untill operation not done
                //do some task here
                count[0]--;
                sem_post(sem[3]) // operation semaphore end means post here

                if(!count[0])
                        sem_post()sem[1]; // allow rebort to move upward

        return



function downMovementThread
        while(1)
                if(count[0] < 0)
                        sem_post(sem[1]) // for downward movemetn sem
                else
                        sem_wait(sem[1])
                        sem_post(sem[1])

                sem_wait(sem[1])
                count[1]++ //increment downward count
                sem_post(sem[1])
                //write you desire portion here
                sem_wait[sem[2]] // wait for completion of previos thread
                count[1]-- //decrement downward count
                sem_post(sem[1])
                if(count[0] == 0)
                        sem_post(sem[0])


        return
```

Question#5

We will use 4 semaphores for solving this problem.

```
// shared data
semaphore sem[4];
count = 0; // count tottal number of customer
sem[3] = 1;
//sem 0, 1, 2 are all initialize with zero


functionc ustomerThread() // thread function for the customer
        while(1)
                if( count < totalNoOfChair)
                        sem_wait(sem[3]) // for mutex
                        count++ // increment customer count
                        sem_post(sem[3])
                        sem_post(sem[0]) // sem 0 for customer
                        sem_wait(sem[1]) // wait until barber not free
                        //do operation here take time etc depend upon protocols
        return

function barberThread() // thread function for the barber
        while(1)
                sem_wait(sem[0]) // wait untill no customer arrive
                sem_wait(sem[3]) // for mutex we can only access customer if some customer available
                count--;
                sem_post[sem[3]]
                sem_post[sem[1]] // barber will get free after this
                //take a hair cut
        return
```