Question 1

Code :-

```
#include <iostream>
#include <thread>
#include<mutex>
#include<semaphore.h>
#include <time.h>
#include <unistd.h>
#define THREAD_NUM 3
using namespace std;

sem_t smallcap;
sem_t capitalcap;
sem_t numb;

void fucntion(){



char temp;

   for (temp = 'A'; temp <= 'Z'; ++temp){
    sem_wait(&capitalcap);
       std::cout<<temp<<"";
   sem_post(&numb);
   }

}

void fucntionA(){
```

```
    for(int a = 1;a<27;a++){
      sem_wait(&numb);
        std::cout<<" "<<a<<" ";
    sem_post(&smallcap);

    }


}

void fucntionB(){

    char temp;
    for (temp = 'a'; temp <= 'z'; ++temp){
    sem_wait(&smallcap);
        std::cout<<temp<<" ";
    sem_post(&capitalcap);
    }

}

int main(){
sem_init(&smallcap, 0, 0);
sem_init(&capitalcap, 0, 1);
sem_init(&numb, 0, 0);
  std::thread small,capital,numeric;

  small =  std::thread(fucntionB);
  capital = std::thread(fucntion);
  numeric = std::thread(fucntionA);
```

```
 capital.join();
 numeric.join();
  small.join();



}
```

Question 2

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


int size;


vector<pair<int, int>> arr[100000];


map<int, int> mp;

void buddy_al(int s)
{

        int n = ceil(log(s) / log(2));
```

```
	size = n + 1;
	for(int i = 0; i <= n; i++)
		arr[i].clear();

	arr[n].push_back(make_pair(0, s - 1));
}

void allo(int s)
{

	int x = ceil(log(s) / log(2));

	if (arr[x].size() > 0)
	{
		pair<int, int> temp = arr[x][0];

		arr[x].erase(arr[x].begin());

		cout << "Memory from " << temp.first
			<< " to " << temp.second
			<< " allocated" << "\n";

		mp[temp.first] = temp.second -
							temp.first + 1;
```

```
}
else
{
	int i;


	for(i = x + 1; i < size; i++)
	{


		if (arr[i].size() != 0)
			break;
	}


	if (i == size)
	{
		cout << "Sorry, failed to allocate memory\n";
	}


	else
	{
		pair<int, int> temp;
		temp = arr[i][0];


		arr[i].erase(arr[i].begin());
		i--;
```

```
for(;i >= x; i--)
{


        pair<int, int> pair1, pair2;
        pair1 = make_pair(temp.first,
                                        temp.first +
                                        (temp.second -
                                        temp.first) / 2);
        pair2 = make_pair(temp.first +
                                        (temp.second -
                                        temp.first + 1) / 2,
                                        temp.second);

        arr[i].push_back(pair1);



        arr[i].push_back(pair2);
        temp = arr[i][0];



        arr[i].erase(arr[i].begin());
}

cout << "Memory from " << temp.first
        << " to " << temp.second
        << " allocate" << "\n";
```

```
                        mp[temp.first] = temp.second -
                                                        temp.first + 1;
                }
        }
}

void dealloc(int id)
{



        if(mp.find(id) == mp.end())
        {
                cout << "Sorry, invalid free request\n";
                return;
        }



        int n = ceil(log(mp[id]) / log(2));


        int i, buddyNumber, buddyAddress;


        arr[n].push_back(make_pair(id,
                                                        id + pow(2, n) - 1));
        cout << "Memory block from " << id
                << " to "<< id + pow(2, n) - 1
                << " freed\n";



        buddyNumber = id / mp[id];
```

```
if (buddyNumber % 2 != 0)
        buddyAddress = id - pow(2, n);
else
        buddyAddress = id + pow(2, n);

for(i = 0; i < arr[n].size(); i++)
{


        if (arr[n][i].first == buddyAddress)
        {


                if (buddyNumber % 2 == 0)
                {
                        arr[n + 1].push_back(make_pair(id,
                        id + 2 * (pow(2, n) - 1)));

                        cout << "Coalescing of blocks starting at "
                                << id << " and " << buddyAddress
                                << " was done" << "\n";
                }
                else
                {
                        arr[n + 1].push_back(make_pair(
                                buddyAddress, buddyAddress +
                                2 * (pow(2, n))));
```

```
                cout << "Coalescing of blocks starting at "
                     << buddyAddress << " and "
                     << id << " was done" << "\n";
            }
            arr[n].erase(arr[n].begin() + i);
            arr[n].erase(arr[n].begin() +
            arr[n].size() - 1);
            break;
        }
    }

    mp.erase(id);
}

    Buddy(128);
    allocate(16);
    allocate(16);
    allocate(16);
    allocate(16);
    dealloc(0);
    dealloc(9);
    dealloc(32);
    dealloc(16);

    return 0;
}
```

Question 3

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

#define THREAD_NUM 8

sem_t semEmpty;
sem_t semFull;

pthread_mutex_t mutexBuffer;

int buffer[10];
int count = 0;

void* prod(void* args) {
   while (1) {

      int a = rand() % 1000;
      sleep(1);
```

```
        sem_wait(&semEmpty);
        pthread_mutex_lock(&mutexBuffer);
        buffer[count] = a;
        count++;
        pthread_mutex_unlock(&mutexBuffer);
        sem_post(&semFull);
    }
}

void* con(void* args) {
    while (1) {
        int b;


        sem_wait(&semFull);
        pthread_mutex_lock(&mutexBuffer);
        b = buffer[count - 1];
        count--;
        pthread_mutex_unlock(&mutexBuffer);
        sem_post(&semEmpty);


        printf("Got %d\n", b);
        sleep(1);
    }
}

int main(int argc, char* argv[]) {
    srand(time(NULL));
```

```
    pthread_t th[THREAD_NUM];
    pthread_mutex_init(&mutexBuffer, NULL);
    sem_init(&semEmpty, 0, 10);
    sem_init(&semFull, 0, 0);
    int i;
    for (i = 0; i < THREAD_NUM; i++) {
        if (i > 0) {
            if (pthread_create(&th[i], NULL,  prod, NULL) != 0) {
                perror("Failing to create threds in this");
            }
        } else {
            if (pthread_create(&th[i], NULL, &con, NULL) != 0) {
                perror("Failing to create threds in this");
            }
        }
    }
    for (i = 0; i < THREAD_NUM; i++) {
        if (pthread_join(th[i], NULL) != 0) {
            perror("Failing to join the thread");
        }
    }
    sem_destroy(&semEmpty);
    sem_destroy(&semFull);
    pthread_mutex_destroy(&mutexBuffer);
    return 0;
}
```