Question - 1

Code -

```
//AU1940112 kandarp sharda
//calling library
#include <iostream>
#include <thread>
#include<mutex>
#include<semaphore.h>
#include <unistd.h>
//defining number of thread
#define THREAD_NUM 3
using namespace std;

//declaring variable
sem_t smallLetter;
sem_t capitalLetter;
sem_t numerical;


//capital Letter generator
void capitalLetterGenerator(){

//declaring variable
char c;

   //for loop that generate A to Z letter
   for (c = 'A'; c <= 'Z'; ++c){

    //it will wait until that until the siganl value get 1 means until the
running procces get complete
    sem_wait(&capitalLetter);

    //printing output
       std::cout<<c<<"";

   //it will increase the count of numerical variable, in laymans term it
is saying that one process in quess is freed
```

```
   sem_post(&numerical);
   }

}


//numeric value generator
void NumericValueGenerator(){


   //for loop for generating number from 1 to 27
   for(int i = 1;i<27;i++){

       //it will wait until that until the numerical variable get value 1
,means until the running procces get complete
     sem_wait(&numerical);
       std::cout<<" "<<i<<" ";

   //it will increase the count of smallLetter variable, in laymans term
it is saying that one process in quess is freed
   sem_post(&smallLetter);

   }


}

//small letter generator
void SmallLetterGenerator(){

   char c;

  //for loop for generating number from a to z
   for (c = 'a'; c <= 'z'; ++c){

    //it will wait until that until the smallLetter variable get value 1
,means until the running procces get complete
   sem_wait(&smallLetter);
       std::cout<<c<<" ";
```

```
   //it will increase the count of smallLetter variable, in laymans term
it is saying that one process in quess is freed
   sem_post(&capitalLetter);
   }

}

int main(){


//giving semaphores value
sem_init(&smallLetter, 0, 0);
sem_init(&capitalLetter, 0, 1);
sem_init(&numerical, 0, 0);

// declaring the variable of thread
 std::thread small,capital,numeric;


//Passing the function that will get executed by thread
 small =  std::thread(SmallLetterGenerator);
 capital = std::thread(capitalLetterGenerator);
 numeric = std::thread(NumericValueGenerator);

//joining it to main function
capital.join();
numeric.join();
small.join();

}
```

Question2

Code -

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

int size;

// Map used as hash map to store the value in (key -value format)
map<int, int> mapping;
//declaring vecctor globally
vector<pair<int, int>> vect1[10000];


// buddy function
void Buddy(int s)
{

   // Maximum number of powers of 2 possible
   int n = ceil(log(s) / log(2));


   size = n + 1;

   //clearing the value in vector
   for(int i = 0; i <= n; i++)
       vect1[i].clear();
   // intitiall whole vector is available
   vect1[n].push_back(make_pair(0, s - 1));
}
// allocating function
void allocating(int s)
{

   // Calculate index in free list
   // to search for block if available
```

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

// Block available
if (vect1[x].size() > 0)
{
    pair<int, int> temporary = vect1[x][0];
    // Remove block from free list that alread y come
    vect1[x].erase(vect1[x].begin());

    cout << "Memory from" << temporary.first
         << " to " << temporary.second
         << " allocating" << "\n";
    // Map starting address with
    // size to make deallocating easy
    mapping[temporary.first] = temporary.second -
                    temporary.first + 1;
}
else
{
    int i;

    // If not, search for a larger block
    for(i = x + 1; i < size; i++)
    {

        // Find block size greater
        // than request
        if (vect1[i].size() != 0)
            break;
    }
    // If no such block is found
    // i.e., no memory block available
    if (i == size)
    {
        cout << "Sorry, failed to allocating memory\n";
    }

    // If found
    else
    {
```

```
            pair<int, int> temporary;
            temporary = vect1[i][0];
            // Remove first block to split
            // it into halves
            vect1[i].erase(vect1[i].begin());
            i--;

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

                // Divide block into two halves
                pair<int, int> pairing1, pairing2;
                pairing1 = make_pair(temporary.first,
                                     temporary.first +
                                    (temporary.second -
                                     temporary.first) / 2);
                pairing2 = make_pair(temporary.first +
                                    (temporary.second -
                                     temporary.first + 1) / 2,
                                     temporary.second);

                vect1[i].push_back(pairing1);
                // Push them in free list
                vect1[i].push_back(pairing2);
                temporary = vect1[i][0];
                // Remove first free block to
                // further split
                vect1[i].erase(vect1[i].begin());
            }

            cout << "Memory from " << temporary.first<< " to " <<
temporary.second<< " allocating" << "\n";

            mapping[temporary.first] = temporary.second -
                                temporary.first + 1;
        }
    }
}
void deallocating(int id)
{
```

```
    // If no address availble of that type
    if(mapping.find(id) == mapping.end())
    {
        cout << "Invalid request\n";
        return;
    }

    // Size of block to be searched
    int n = ceil(log(mapping[id]) / log(2));

    int i, bdnumber, bdAddress;
    // Add the block in free list
    vect1[n].push_back(make_pair(id,
                               id + pow(2, n) - 1));
    cout << "Block available for memory use" << id<< " to "<< id + pow(2,
n) - 1 << " freed\n";
    // Calculate buddy number
    bdnumber = id / mapping[id];
    if (bdnumber % 2 != 0)
        bdAddress = id - pow(2, n);
    else
        bdAddress = id + pow(2, n);

    // Search in free list to find it's buddy
    for(i = 0; i < vect1[n].size(); i++)
    {

        // If buddy found and is also free
        if (vect1[n][i].first == bdAddress)
        {

            // Now merge the buddies to make
            // them one large free memory block
            if (bdnumber % 2 == 0)
            {
                vect1[n + 1].push_back(make_pair(id,
                   id + 2 * (pow(2, n) - 1)));

                cout << "Coalescing of blocks starting at "
```

```
                    << id << " and " << bdAddress
                    << " was done" << "\n";
            }
            else
            {
                vect1[n + 1].push_back(make_pair(
                    bdAddress, bdAddress +
                    2 * (pow(2, n))));

                cout << "Coalescing of blocks starting at "
                     << bdAddress << " and "
                     << id << " was done" << "\n";
            }
            vect1[n].erase(vect1[n].begin() + i);
            vect1[n].erase(vect1[n].begin() +
            vect1[n].size() - 1);
            break;
        }
    }
    // Remove the key existence from map
    mapping.erase(id);
}
// main component
int main()
{



    Buddy(128);
    allocating(16);
    allocating(16);
    allocating(16);
    allocating(16);
    deallocating(0);
    deallocating(9);
    deallocating(32);
    deallocating(16);
    return 0;
}
```

Question3

Code -

```
//AU1940112 kandarp sharda

// deffining library
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <semaphore.h>
#include <string.h>
#include <pthread.h>



pthread_mutex_t mux;
sem_t semEmpty;
sem_t semFull;



int buffer[5];  // declaing buffer
int counter = 0; //increasing counter value

//producer it wil generate random numer and that generated number will be
stored in buffer
void* producer(void* args) {
   while (1) {


       // generating random number
       int number = rand() % 1000;
```

```
        //using sleep to produce the number
        sleep(1);

        //If the buffer is full it will wait until buffer gets space
        sem_wait(&semEmpty);

        //taking process in critical section using mutext lock
        pthread_mutex_lock(&mux);

        // storing the value in buffer
        buffer[counter] = number;

        //increasing counter for adding number
        counter++;


        //taking process out of critical section so that another process
can come in critical section
        pthread_mutex_unlock(&mux);

        //This is saying that buffer has added some value
        sem_post(&semFull);
    }
}


//consumer used for consuming the number that has been produced by
producer
void* consumer(void* args) {

    //infinite while loop
    while (1) {
        int number;

        // Waiting till the buffer get added by some value .
        sem_wait(&semFull);

         //taking process in critical section using mutext lock
        pthread_mutex_lock(&mux);
        number = buffer[counter - 1];
```

```
        // decreasing counter value
        counter--;


        //taking process out of critical section so that another process
can come in critical section
        pthread_mutex_unlock(&mux);
        sem_post(&semEmpty);

        // Consuming  = printing the value that has been taken from buffer
        printf("number %d\n", number);

        //giving time to thread to consume data
        sleep(1);
    }
}

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


    //declaring variable for thread
    pthread_t consump,produc;

    //declaring variale for mutext
    pthread_mutex_init(&mux, NULL);

    //declaring semaphores
    sem_init(&semEmpty, 0, 10);
    sem_init(&semFull, 0, 0);


   //creating thread
    if (pthread_create(&produc, NULL, &producer, NULL) != 0) {
                perror("Failed to create thread");
            }
    if (pthread_create(&consump, NULL, &consumer, NULL) != 0) {
                perror("Failed to create thread");
            }
```

```c
  //joining it to main thread
   if (pthread_join(produc, NULL) != 0) {
            perror("Failed to join thread");
        }
   if (pthread_join(consump, NULL) != 0) {
            perror("Failed to join thread");
        }

   //destroying the semaphores
   sem_destroy(&semEmpty);
   sem_destroy(&semFull);
   pthread_mutex_destroy(&mux);

   return EXIT_SUCCESS;
}
```