Name: Maulikkumar Bhalani

Enrollment No: AU1940206

1.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/syscall.h>

pthread_mutex_t mymutex=PTHREAD_MUTEX_INITIALIZER;
int common_data = 0;

void function1()
{
   int j;
    for ( j = 0; j < 27; j++ ) {



       pthread_mutex_lock(&mymutex);
      common_data++;
     printf( "%d\n", j );
      pthread_mutex_unlock(&mymutex);
    }
   }

void function2()
{
   int i;
   for (i = 97; i <= 122; i++)
   {
      pthread_mutex_lock(&mymutex);
      common_data++;
      printf("%c\n", i);
      pthread_mutex_unlock(&mymutex);
   }

   }

int main()
{

   pthread_t thread1, thread2;
   pthread_create(&thread1, NULL, (void *)&function1, NULL);
   pthread_create(&thread2, NULL, (void *)&function2, NULL);

   pthread_join(thread1, NULL);
   pthread_join(thread2, NULL);

   return 0;
}
```

2.

Allocation:

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

// Size of vector of pairs
int size;

// Global vector of pairs to store
// address ranges available in free list
vector<pair<int, int>> free_list[100000];

// key as value
map<int, int> mp;

void initialize(int sz)
{

    // Maximum number of powers of 2 possible
    int n = ceil(log(sz) / log(2));
    size = n + 1;

    for (int i = 0; i <= n; i++)
        free_list[i].clear();

    // size is available
    free_list[n].push_back(make_pair(0, sz - 1));
}

void allocate(int sz)
{
    // to search for block if available
    int n = ceil(log(sz) / log(2));

    // Block available
    if (free_list[n].size() > 0)
    {
        pair<int, int> temp = free_list[n][0];

        // Remove block from free list
        free_list[n].erase(free_list[n].begin());
        cout << "Memory from " << temp.first
             << " to " << temp.second << " allocated"
             << "\n";

        // size to make deallocating easy
        mp[temp.first] = temp.second -
                         temp.first + 1;
    }
    else
    {
        int i;
        for (i = n + 1; i < size; i++)
        {

            // Find block size greater than request
            if (free_list[i].size() != 0)
                break;
        }

        // If no such block is found
        if (i == size)
        {
            cout << "Sorry, failed to allocate memory \n";
        }

        // If condition satisfy,
        else
        {
            pair<int, int> temp;
            temp = free_list[i][0];
```

```
            // Remove first block to split it into halves
            free_list[i].erase(free_list[i].begin());
            i--;

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

                // Divide block into twwo halves
                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);

                free_list[i].push_back(pair1);

                // Push them in free list
                free_list[i].push_back(pair2);
                temp = free_list[i][0];

                // Remove first free block to
                // further split
                free_list[i].erase(free_list[i].begin());
            }
            cout << "Memory from " << temp.first
                 << " to " << temp.second
                 << " allocated"
                 << "\n";

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

//main code
int main()
{

    initialize(128);
    allocate(32);
    allocate(7);
    allocate(64);
    allocate(56);

    return 0;
}
```

B. Dealocation:

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

// Size of vector of pairs
int size;

// the free nodes of various sizes
vector<pair<int, int>> arr[100000];

// of allocated segment key as value
map<int, int> mp;

void Buddy(int s)
{

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

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

    // Initially whole block of specified
    // size is available
    arr[n].push_back(make_pair(0, s - 1));
}

void allocate(int s)
{

    // Calculate index in free list
    // to search for block if available
    int x = ceil(log(s) / log(2));

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

        // Remove block from free list
        arr[x].erase(arr[x].begin());

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

        // Map starting address with
        // size to make deallocating easy
        mp[temp.first] = temp.second -
                         temp.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 (arr[i].size() != 0)
                break;
        }

        // i.e., no memory block available
        if (i == size)
        {
            cout << "Sorry, failed to allocate memory\n";
        }

        // If found
```

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

            // it into halves
            arr[i].erase(arr[i].begin());
            i--;

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

                // Divide block into two halves
                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);

                // Push them in free list
                arr[i].push_back(pair2);
                temp = arr[i][0];

                // Remove first free block to
                // further split
                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 deallocate(int id)
{

    // If no such starting address available
    if (mp.find(id) == mp.end())
    {
        cout << "Sorry, invalid free request\n";
        return;
    }

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

    int i, buddyNumber, buddyAddress;

    // Add the block in free list
    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";

    // Calculate buddy number
    buddyNumber = id / mp[id];

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

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

```
    {

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

            // them one large free memory block
            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);
}

// main code
int main()
{

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

    return 0;
}
```

3.

```
//AU1940206 Maulikkumar Bhalani

#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <stdio.h>
#define MaxItems 5
#define BufferSize 5

sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;
void *producer(void *pno)
{
   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)
{
   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);
   sem_init(&empty, 0, BufferSize);
   sem_init(&full, 0, 0);
   int a[5] = {1, 2, 3, 4, 5};
   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);
   }
   for (int i = 0; i < 5; i++)
   {
      pthread_join(con[i], NULL);
   }
   pthread_mutex_destroy(&mutex);
   sem_destroy(&empty);
   sem_destroy(&full);
   return 0;
}
```