Varun Deliwala
Operating Systems End Semester Lab Examination Section 2
AU1940034

Question 1

```
#include<stdio.h>
#include<pthread.h>
#include<time.h>

pthread_t Thread_id[3];
pthread_mutex_t mutex;
unsigned int rc;
//prototypes for callback functions
int UpperCs = 65;
int LowerCs = 97;
int num = 1;
int fg = 0;
void* PrintCapitals(void*);
void* PrintLowers(void*);
void* PrintNos(void*);

void main(void)
{
   pthread_create(&Thread_id[0],0,&PrintCapitals,0);
   pthread_create(&Thread_id[1],0,&PrintNos,0);

   pthread_create(&Thread_id[2],0,&PrintLowers,0);

 //sleep(3);

   pthread_join(Thread_id[0],NULL);
   pthread_join(Thread_id[1],NULL);
```

```
    pthread_join(Thread_id[2],NULL);


}


void* PrintCapitals(void *ptr)
{
    rc = pthread_mutex_lock(&mutex);
    while(UpperCs<=90)
    {
    if(fg%3==0){


        printf("%c ",UpperCs);
      UpperCs++;
        fg++;


    }
    else{
        rc=pthread_mutex_unlock(&mutex);//if fg modulus is not equal to 0, do not print, release
mutex for num value
    }


    }
}


void* PrintLowers(void* ptr1)
{
   rc = pthread_mutex_lock(&mutex);
   while(LowerCs<=122)
   {
    if(fg%3==2){
    printf("%c ",LowerCs);
      LowerCs++;
      fg++;
```

```
    }
    else{
        rc=pthread_mutex_unlock(&mutex);

    }
  }
}
void* PrintNos(void* ptr1)
{
   rc = pthread_mutex_lock(&mutex);
  while(num <= 26)
   {
   if(fg%3==1){
    printf("%d ",num);
      num++;
      fg++;
   }
   else{
          rc=pthread_mutex_unlock(&mutex);
      }


   }
}
```

Question 2 B

```
#include<bits/stdc++.h>

using namespace std;


int size;


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


map<int, int> mp;


void Buddy(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 Allocate(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 Deallocate(int id)
        {
```

```
if(mp.find(id) == mp.end())
{
        cout << "Sorry, the command cant be executed due to invalid request\n";
        return;
}


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

int i, buddyNum, 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";

buddyNum = id / mp[id];

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


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


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

```
        }


        mp.erase(id);
}
int main()
{
        Buddy(128);
        Allocate(16);
        Allocate(32);
        Allocate(32);
        Deallocate(0);
        Deallocate(9);
        Deallocate(32);
        Deallocate(16);

        return 0;
}
```

Question 3

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

#define MaximumItems 5 // Maximum items a ProducerX can produce or a ConsumerX can consume
#define BuffSize 5 // Size of the buffer

sem_t empty;
sem_t full;
```

```
int in = 0;
int out = 0;
int buffer[BuffSize];
pthread_mutex_t mutex;

void *ProducerX(void *pno)
{
   int item;
   for(int i = 0; i < MaximumItems; i++) {
      item = rand(); // Produce an random item
      sem_wait(&empty);
      pthread_mutex_lock(&mutex);
      buffer[in] = item;
      printf("ProducerX %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);
      in = (in+1)%BuffSize;
      pthread_mutex_unlock(&mutex);
      sem_post(&full);
   }
}
void *ConsumerX(void *cno)
{
   for(int i = 0; i < MaximumItems; i++) {
      sem_wait(&full);
      pthread_mutex_lock(&mutex);
      int item = buffer[out];
      printf("ConsumerX %d: Remove Item %d from %d\n",*((int *)cno),item, out);
      out = (out+1)%BuffSize;
      pthread_mutex_unlock(&mutex);
      sem_post(&empty);
   }
}
```

```
int main()
{

    pthread_t pro[5],con[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty,0,BuffSize);
    sem_init(&full,0,0);

    int a[5] = {1,2,3,4,5}; //Just used for numing the ProducerX and ConsumerX

    for(int i = 0; i < 5; i++) {
        pthread_create(&pro[i], NULL, (void *)ProducerX, (void *)&a[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_create(&con[i], NULL, (void *)ConsumerX, (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;
}
```