Yash Doshi - AU1941097

Q1)

Multi Thread:

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

sem_t semaphore;
int flag = 0;

void *printNumbers(void *arg){
    int i;
    for(i=0;i<26;i++){
        sem_wait(&semaphore);
        if(flag%3 != 1){
            i--;
        }else{
            printf("%d ",1+i);
            flag++;
        }
        sem_post(&semaphore);
    }
    pthread_exit(NULL);
}

void *printCapitalLetters(void *arg){
    int i;
    for(i=0;i<26;i++){
        sem_wait(&semaphore);
        if(flag%3 != 0){
            i--;
        }else{
            printf("%c ",'A'+i);
            flag++;
        }
        sem_post(&semaphore);
    }
    pthread_exit(NULL);
}

void *printSmallLetters(void *arg){
    int i;
    for(i=0;i<26;i++){
```

Yash Doshi - AU1941097

```
        sem_wait(&semaphore);
        if(flag%3 != 2){
            i--;
        }else{
            printf("%c ",'a'+i);
            flag++;
        }
        sem_post(&semaphore);
    }
    pthread_exit(NULL);
}

int main(){
    pthread_t t1, t2, t3;
    sem_init(&semaphore, 0, 1);
    pthread_create(&t1,NULL,printCapitalLetters,NULL);
    pthread_create(&t2,NULL,printNumbers,NULL);
    pthread_create(&t3,NULL,printSmallLetters,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    pthread_join(t3,NULL);
    sem_destroy(&semaphore);
    return 0;
}
```

Q2)

Buddy’s Algorithm:

```
#include<bits/stdc++.h>
using namespace std;
int blockSize;
vector<pair<int, int>> free_list[100000];
map<int, int> m;

void Buddy_init(int s)
{
    int n = ceil(log(s) / log(2));

    blockSize = n + 1;
    for(int i = 0; i <= n; i++){
        free_list[i].clear();
    }
    free_list[n].push_back(make_pair(0, s - 1));
```

```
}

void allocate_memory(int allocation_size)
{
    int x = ceil(log(allocation_size) / log(2));
    if (free_list[x].size() > 0)
    {
        pair<int, int> temp_pair = free_list[x][0];
        free_list[x].erase(free_list[x].begin());

        cout << temp_pair.first<< " to " <<
temp_pair.second<< " allocated" << "\n";

        m[temp_pair.first] = temp_pair.second -
temp_pair.first + 1;
    }
    else
    {
        int i;
        for(i = x + 1; i < blockSize; i++){
            if (free_list[i].size() != 0)
                break;
        }
        if (i == blockSize)
        {
            cout << "Unable to allocate memory\n";
        }
        else
        {
            pair<int, int> temp_pair;
            temp_pair = free_list[i][0];
            free_list[i].erase(free_list[i].begin());
            i--;

            for(;i >= x; i--)
            {
                pair<int, int> pair1, pair2;
                pair1 =
make_pair(temp_pair.first,temp_pair.first +
(temp_pair.second - temp_pair.first) / 2);
                pair2 = make_pair(temp_pair.first +
(temp_pair.second - temp_pair.first + 1) / 2,
temp_pair.second);
```

Yash Doshi - AU1941097

```
                free_list[i].push_back(pair1);
                free_list[i].push_back(pair2);
                temp_pair = free_list[i][0];
                free_list[i].erase(free_list[i].begin()
);
            }

            cout << temp_pair.first<< " to " <<
temp_pair.second<< " allocate" << "\n";

            m[temp_pair.first] = temp_pair.second -
temp_pair.first + 1;
        }
    }
}
void deallocate_memory(int idx){
    if(m.find(idx) == m.end())
    {
        cout << "Invalidx request\n";
        return;
    }
    int n = ceil(log(m[idx]) / log(2));

    int i, buddy_number, buddy_address;
    free_list[n].push_back(make_pair(idx,idx + pow(2,
n) - 1));
    cout << idx<< " to "<< idx + pow(2, n) - 1<< "
freed\n";
    buddy_number = idx / m[idx];

    if (buddy_number % 2 != 0){
        buddy_address = idx - pow(2, n);
    }
    else{
        buddy_address = idx + pow(2, n);
    }
    for(i = 0; i < free_list[n].size(); i++){
        if (free_list[n][i].first == buddy_address)
        {
            if (buddy_number % 2 == 0)
            {
                free_list[n +
1].push_back(make_pair(idx,idx + 2 * (pow(2, n) - 1)));
```

Yash Doshi - AU1941097

```
                    cout << "Coalescing from  "<< idx << "
and " << buddy_address<< " was done" << "\n";
                }
                else
                {
                    free_list[n + 1].push_back(make_pair(
                        buddy_address, buddy_address + 2 *
(pow(2, n))));

                    cout << "Coalescing from "<<
buddy_address << " and "<< idx << " was done" << "\n";
                }
                free_list[n].erase(free_list[n].begin() +
i);
                free_list[n].erase(free_list[n].begin() +
                free_list[n].size() - 1);
                break;
            }
        }
        m.erase(idx);
}

int main(){

    Buddy_init(64);
    allocate_memory(16);
    allocate_memory(7);
    deallocate_memory(0);
    deallocate_memory(7);
    deallocate_memory(16);

    return 0;
}
```

Q3)

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

Yash Doshi - AU1941097

```c
#define MaxItems 5
#define BufferSize 5

sem_t empty_semaphore;
sem_t full_semaphore;
int p = 0;
int v = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;

void *produce_new_item(void *pno)
{
    int newItem;
    for(int i = 0; i < MaxItems; i++) {
        newItem = rand();
        sem_wait(&empty_semaphore);
        pthread_mutex_lock(&mutex);
        buffer[p] = newItem;
        printf("New Item Produced");
        p = (p+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full_semaphore);
    }
}
void *consume_item(void *cno)
{
    for(int i = 0; i < MaxItems; i++) {
        sem_wait(&full_semaphore);
        pthread_mutex_lock(&mutex);
        int newItem = buffer[v];
        printf("Consumer item");
        v = (v+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty_semaphore);
    }
}

int main()
{

    pthread_t t1[5],t2[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty_semaphore,0,BufferSize);
    sem_init(&full_semaphore,0,0);
```

Yash Doshi - AU1941097

```
    int arr[5] = {1,2,3,4,5};

    for(int i = 0; i < 5; i++) {
        pthread_create(&t1[i], NULL, (void
*)produce_new_item, (void *)&arr[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_create(&t2[i], NULL, (void
*)consume_item, (void *)&arr[i]);
    }

    for(int i = 0; i < 5; i++) {
        pthread_join(t1[i], NULL);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(t2[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty_semaphore);
    sem_destroy(&full_semaphore);

    return 0;

}
```