OS prac end-sem

code

AU1940121

Jinil Chandarana

[NOTE: the code is editable and not a screen sort. It appears such because it is 1st written in VSCODE and then pasted in document]

Question 1:

Question 2b:

```
//AU1940121 Jinil
#include<bits/stdc++.h>
using namespace std;

// Size of vector of pairs
int size;

vector<pair<int, int>> free_list[100000];
void initialize(int sz)
{
  int n = ceil(log(sz) / log(2));
  size = n + 1;

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



  free_list[n].push_back(make_pair(0, sz - 1));
}
```

```
void allocate(int sz)
{



    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";



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

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



            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;
        }
    }
}

// Driver code
int main()
{
    initialize(128);
    allocate(32);
    allocate(7);

    allocate(64);

    allocate(56);


    return 0;

}
```

========================

Question 3:

```c
#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* producer(void* args) {
    while (1) {
        // Produce
        int x = rand() % 100;
        sleep(1);

        // Add to the buffer
        sem_wait(&semEmpty);
        pthread_mutex_lock(&mutexBuffer);
        buffer[count] = x;
        count++;
        pthread_mutex_unlock(&mutexBuffer);
        sem_post(&semFull);
    }
}
```

```c
void* consumer(void* args) {
    while (1) {
        int y;

        // Remove from the buffer
        sem_wait(&semFull);
        pthread_mutex_lock(&mutexBuffer);
        y = buffer[count - 1];
        count--;
        pthread_mutex_unlock(&mutexBuffer);
        sem_post(&semEmpty);

        // Consume
        printf("Got %d\n", y);
        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, &producer, NULL) !=
0) {
                perror("Failed to create thread");
            }
        } else {
            if (pthread_create(&th[i], NULL, &consumer, NULL) !=
0) {
                perror("Failed to create thread");
            }
        }
    }
    for (i = 0; i < THREAD_NUM; i++) {
        if (pthread_join(th[i], NULL) != 0) {
            perror("Failed to join thread");
        }
    }
    sem_destroy(&semEmpty);
    sem_destroy(&semFull);
    pthread_mutex_destroy(&mutexBuffer);
    return 0;
}
```