CSE332 - Operating Systems

End Semester Exam

Submitted by, Axay Ghoghari

Enrollment Number : AU1940269

Code File

Q1 : Print the following Pattern A 1 a B 2 b C 3 c … Y 25 y Z 26 z Using any one of the following concepts
b. Multithreads (Hint: using 3 Threads) (PS: Process Synchronization or Thread Synchronization is key thing for pattern printing).

Code :

```
#include <iostream>
#include <thread>
#include <mutex>
#include <semaphore.h>
#include <unistd.h>
#define THREAD_NUM 3
using namespace std;
sem_t smallLetter;
sem_t capitalLetter;
sem_t numerical;
void capitalFunc()
{
    char c;

    for (c = 'A'; c <= 'Z'; ++c)
    {
        sem_wait(&capitalLetter);
        std::cout << c << "";
        sem_post(&numerical);
    }
}
void smallFunc()
{

    for (int i = 1; i < 27; i++)
    {
        sem_wait(&numerical);
        std::cout << " " << i << " ";
        sem_post(&smallLetter);
    }
}

void func()
{

    char c;
    for (c = 'a'; c <= 'z'; ++c)
    {
        sem_wait(&smallLetter);
        std::cout << c << " ";
        sem_post(&capitalLetter);
    }
}
int main()
{
    sem_init(&smallLetter, 0, 0);
    sem_init(&capitalLetter, 0, 1);
    sem_init(&numerical, 0, 0);
    std::thread small, capital, numeric;
    small = std::thread(func);
    capital = std::thread(capitalFunc);
    numeric = std::thread(smallFunc);
    capital.join();
    numeric.join();
    small.join();
}
```

Q2 : Describe and implement any one of the following.
b. Describe the Buddy’s Algorithm for Memory Allocation and Deallocation along with an example and implement it in C or C++.

Code:

```
#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 << "Ooops !!!, Memory  allocation is failed \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)
{
    {
        cout << "Oops!!!, invalid free request\n";
        return;
    }
    int n = ceil(log(mp[id]) / log(2));
    int i, buddyNumber, 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";
    buddyNumber = id / mp[id];

    if (buddyNumber % 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 (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);
}
int main()
{



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

    return 0;
}
```

Q3 : [Bonus] Describe what is Producer Consumer Problem and its solution in detail using Semaphores and Mutex and implement it in C.

Code :

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

#define MaxItems 5
#define BuffSize 5
sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BuffSize];
pthread_mutex_t mutex;

void *producer(void *producerNo)
{
   int item;
   for (int i = 0; i < MaxItems; i++)
   {
      item = rand();
      sem_wait(&empty);
      pthread_mutex_lock(&mutex);
      buffer[in] = item;
      printf("Producer %d: Insert Item %d at %d\n", *((int *)producerNo), buffer[in], in);
      in = (in + 1) % BuffSize;
      pthread_mutex_unlock(&mutex);
      sem_post(&full);
   }
}
void *consumer(void *consumerNo)
{
   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 *)consumerNo), 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 n[5] = {1, 2, 3, 4, 5}; //Just used for numbering the producer and consumer

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