Name: Nirav Karavadra

Roll no.: AU1940198

CSE332 – Operating System (Section-2)

End Semester Exam

Que 1.

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 a. Multiprocesses b. Multithreads

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
using namespace std;
void foo(int Z)
{
  for (int i = 0; i < Z; i++) {
    printf("%c" , Alpha);
   Alpha++;
  }
}
class thread_obj {
public:
  void operator()(int x)
  {
    for (int i = 0; i < x; i++)
   }
};
```

```
int Num = 0
int Alpha = 65
int lowerAlpha = 97
int main()
{
    printf("%c" , lowerAlpha);
       lowerAlpha++;
    thread th1(foo, 3);
    thread th2(thread_obj(), 3);
     auto f = [](int x) {
        for (int i = 0; i < x; i++)
        Num ++
        printf("%d" , Num);
    };
    thread th3(f, 3);
    th1.join();
    th2.join();
    th3.join();
  return 0;
}
```

Que 2.

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++.**

```
//Nirav Karavadra
//AU1940198

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

// Size of vector of pairs
```

```
int size;

// Global vector of pairs to track all the free nodes of various sizes
vector<pair <int, int> > arr[100000];

// Map used as hash map to store the starting address as key and size of allocated segment
key as value
map<int, int> mp;

void buddy(int s)
{

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

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

    // Initially whole block of specified size is available
    arr[a].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;
          }

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

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

                    // Remove first block to split 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)
{

    //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 free
        if (arr[n][i].first == buddyAddress)
        {

            // Merge the buddies to make them 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;
        }
    }

    // Remove the key existence from map
    mp.erase(id);
}

int main()
{

    buddy(128);
```

```
        allocate(16);
        allocate(16);
        allocate(16);
        deallocate(0);
        deallocate(32);
        deallocate(16);

        return 0;
    }
```

Que 3.

Describe what is Producer Consumer Problem and its solution in detail using Semaphores and Mutex and implement it in C

```
//Nirav Karavadra

//AU1940198


#include <stdio.h>

#include <stdlib.h>


// Initialize a mutex to 1

int M = 1;


// Number of find slots as 0

int F = 0;

// Number of empty slots as size of buffer

int E = 10, x = 0;

// Function to produce an item and

// add it to the buffer

void producer()

{

    // Decrease M value by 1

    --M;
    // Increase the number of F
    // slots by 1
    ++F;

    // Decrease the number of E
    // slots by 1
    --E;

    // Item produced
    x++;
    printf("\nItems produced by Producer"
        " %d",
        x);

    // Increase M value by 1
    ++M;
}

// Function to consume an item and remove it from buffer
void consumer()
{
    // Decrease M value by 1
    --M;

    // Decrease the number of F
    // slots by 1
    --F;

    // Increase the number of E
    // slots by 1
```

```
        ++E;

        printf("\n Items consumed by Consumer"

                " %d",

                x);

        x--;


        // Increase M value by 1

        ++M;

}




int main()

{

        int n, i;

        printf("\n1. Enter 1 for Producer"

                "\n2. Enter 2 for Consumer"

                "\n3. Enter 3 for Exit");




#pragma omp critical


        for (i = 1; i > 0; i++) {


                printf("\nEnter your choice:");

                scanf("%d", &n);


                // Switch Cases

                switch (n) {

                case 1:
```

```
    // If M is 1 and E is non-zero, then it is possible to produce
    if ((M == 1)
        && (E != 0)) {
        producer();
    }

    // Otherwise, print buffer
    // is F
    else {
        printf("Buffer is F!");
    }
    break;

case 2:

    // If M is 1 and F
    // is non-zero, then it is
    // possible to consume
    if ((M == 1)
        && (F != 0)) {
        consumer();
    }

    // Otherwise, print Buffer
    // is E
    else {
        printf("Buffer is E!");
    }
    break;
```

```
        // Exit Condition
        case 3:
            exit(0);
            break;
        }
    }
}
```