Submitted by: Shah Hiral Rajivkumar AU1940128

1 a

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 (Hint: using 3 child processes)

Code:

```
// 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 (Hint: using 3 child processes)
//Shah Hiral Rajivkumar AU1940128

//include libraries
#include <stdio.h>
#include <stdlib.h>
#include<unistd.h>

int main()
{
        sem S;
   pid_t pid1, pid2, pid3;

   pid1 = fork();

   if (pid1 == 0) {
```

```
    char c;
    for (c = 'A'; c <= 'Z'; ++c){
                        wait(S);
    printf("%c ", c);
    signal(S);
}


}


else {
    pid2 = fork();
    if (pid2 == 0) {
        int i;
        for (i = 1; i <= 26; i++){
                                    wait(S);
         printf("%d ", i);
         signal(S);
     }
    }
    else {
        pid3 = fork();
        if (pid3 == 0) {
            char c;
        for (c = 'a'; c <= 'z'; ++c)
        wait(S);
        printf("%c ", c);
        signal(S);
    }
        }
```

```
        else {
            sleep(3);

        }
      }
   }

   return 0;
}
```

1 b

Code:

```
// 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
//b. Multithreads (Hint: using 3 threads)
//Shah Hiral Rajivkumar AU1940128

//include libraries
//include libraries
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <stdio.h>



#define sz 78

int array[sz];
```

```
pthread_mutex_t mut;


void *thread1(void *one)
{
   for(int z = 0; z < 78; z=z+3) {


      pthread_mutex_lock(&mut); //lock mutex
      char c;
      for (c = 'A'; c <= 'Z'; ++c){


      array[z] = c;
   }


      pthread_mutex_unlock(&mut);// unlock mutes


   }
}
void *thread2(void *two)
{
   for(int z = 1; z < 78; z=z+3) {


      pthread_mutex_lock(&mut); //lock mutex
      int i;
      for (i = 1; i < 26; i++){


      array[z] = i;
   }


      pthread_mutex_unlock(&mut);// unlock mutes
```

```
    }
}
void *thread3(void *three)
{
    for(int z = 2; z < 78; z=z+3) {

        pthread_mutex_lock(&mut); //lock mutex
        char c;
        for (c = 'a'; c <= 'z'; ++c){

        array[z] = c;
    }

        pthread_mutex_unlock(&mut);// unlock mutes

    }
}



//main code
int main()
{

    pthread_t t1, t2, t3;
    pthread_mutex_init(&mut, NULL);
```

```
//create threads
//join threads



   pthread_mutex_destroy(&mut); //destroy mutex



   return 0;

}
```

2 b

Describe the Buddy's Algorithm for Memory Allocation and

Deallocation along with an example and implement it in C or C++.

Code:

```
//2 b Describe the Buddy's Algorithm for Memory Allocation and Deallocation along with an example and implement it in C or C++.
//Buddy's Algorithm for Memory Allocation and De-Allocation in C++
//Shah Hiral Rajivkumar AU1940128

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

int s; // declaring the variable for the size of vector pairs

//f_lst has the available address ranges
vector<pair<int, int>> f_lst[100000]; //vector pair is of intergers
map<int, int> m;
```

```
void function_initialize(int size)
{           //Maximum possible powers of 2
        int z = ceil(log(size)/log(2));
        s = z+1;
        for(int j =0;j<=z;j++)
                f_lst[j].clear();
//The whole block will have free space initially
        f_lst[z].push_back(make_pair(0,size-1));
}

void mem_allocation(int size )
{
                //search for available block
        int z = ceil(log(size)/log(2));

        //Allocate if block is available
        if(f_lst[z].size()>0)
        {
                pair<int,int> t = f_lst[z][0];
                //Removing the allocated block from the free list
                f_lst[z].erase(f_lst[z].begin());
                cout << "The memory space from " <<t.first<<"- "<< t.second << " has been allocated.
"<< "\n" << "\n";

                m[t.first]=t.second-t.first+1;
        }
        else
                {
```

```
                        pair<int, int> t;
                        t =f_lst[j][0];

                        f_lst[j].erase(f_lst[j].begin());
                        j--; //decrement variable

                        for(; j>= z;j--)
                        {
                                //Division of blocks into half
                                pair<int,int> p1, p2;
                                p1 = make_pair(t.first,t.first + (t.second - t.first) / 2);
                                p2 = make_pair(t.first +(t.second-t.first+1) / 2,t.second);

                                f_lst[j].push_back(p1);

                                // Add them in the list
                                f_lst[j].push_back(p2);
                                t=f_lst[j][0];

                                f_lst[j].erase(f_lst[j].begin());
                        }
                        cout << "The memory space from " << t.first << " - " << t.second << " has been
allocated" << "\n \n";
                        m[t.first] = t.second -     t.first+1;
                }
        }
}

void mem_deallocation(int i)
```

```
{
        //If the inputted starting address is not available
        if(m.find(i)==m.end())
        {
                cout << "Please enter the correct address \n \n";
                return;
        }

        //size of the memory block to be released
        int z = ceil(log(m[i])/log(2));

        int j, budnum, budadd;

        //Add the realeased block in the free list
        f_lst[z].push_back(make_pair(i,i + pow(2,z)-1));
        cout << "Memory requested from " << i  << " - "<< i + pow(2, z) -1          << " is released \n \n";

        budnum = i / m[i];

        if (budnum % 2!= 0)
                budadd = i - pow(2, z);
        else
                budadd =i + pow(2, z);

        //Finding the buddy
        for(j =0; j < f_lst[z].size(); j++)
        {

                //To do when buddy is found and free
```

```
                if (f_lst[z][j].first == budadd)
                {
                //merging the buddies together
                        if (budnum % 2 == 0)
                        {
                                f_lst[z+1].push_back(make_pair(i,i + 2 * (pow(2, z)-1)));
                                cout << "Merging of the from "<< i << " and " << budadd << " is
completed" << "\n \n";
                        }
                        else
                        {
                                f_lst[z + 1].push_back(make_pair(budadd, budadd + 2 * (pow(2, z))));
                                cout << "Merging of the from "<< budadd << " and " << i << " is
completed" << "\n \n";
                        }
                        f_lst[z].erase(f_lst[z].begin() + z);
                        f_lst[z].erase(f_lst[z].begin() +
                        f_lst[z].size() - 1);
                        break;
                }
        }

        m.erase(i);
}

// main code
int main()
{
```

```
int sum,a,request;
cout<<"Input the total memory size: ";
cin>>sum;
function_initialize(sum);
label:
while(1)
{
        cout<<"\n 1. Add Process into Memory\n          2. Remove Process \n3. Exit\n=> ";
        cin>>a;
        switch(a)
        {
                case 1: //allocate memory
                cout<<"Request size ";
                cin>>request;
                cout<<"\n \n";
                mem_allocation(request);
                break;

                case 2: //deallocate memory
                cout<<"Starting address of memory to deallocate";
                cin>>request;
                cout<<"\n \n";
                mem_deallocation(request);
                break;

                case 3: //exit
                exit(0);
                break;
```

```
                default: //default
                goto label;
            }
        }
        return 0;
}
```

3. Describe what is Producer Consumer Problem and its solution in

detail using Semaphores and Mutex and implement it in C.

Code:

```
// 3. Describe what is Producer Consumer Problem and its solution in detail using Semaphores and Mutex and implement it in C.
// Shah Hiral Rajivkumar AU1940128

//include libraries
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <stdio.h>

#define max 5
#define sz 5
sem_t E;
sem_t F;
int i = 0;
int j = 0;
int array[sz];
pthread_mutex_t mut;

void *producer(void *prono)
```

```
{   //producer
    int data;
    for(int z = 0; z < max; z++) {
        data = rand(); //produce data
        sem_wait(&E);
        pthread_mutex_lock(&mut); //lock mutex
        array[i] = data;
        i = (i+1)%sz;
        pthread_mutex_unlock(&mut);// unlock mutes
        sem_post(&F);
    }
}
void *consumer(void *conno)
{   //consumer
    for(int z = 0; z < max; z++) {
        sem_wait(&F);
        pthread_mutex_lock(&mut); //lock mutex
        int data = array[j];
        j = (j+1)%sz;
        pthread_mutex_unlock(&mut);//unlock mutex
        sem_post(&E);
    }
}

//main code
int main()
{

    pthread_t p[5],c[5];
```

```
    pthread_mutex_init(&mut, NULL);
    sem_init(&E,0,sz);
    sem_init(&F,0,0);

    int arr[5] = {1,2,3,4,5};
//create threads
    for(int z = 0; z < 5; z++) {
        pthread_create(&p[z], NULL, (void *)producer, (void *)&arr[z]);
    }
    for(int z = 0; z < 5; z++) {
        pthread_create(&c[z], NULL, (void *)consumer, (void *)&arr[z]);
    }
//join threads
    for(int z = 0; z < 5; z++) {
        pthread_join(p[z], NULL);
    }
    for(int z = 0; z < 5; z++) {
        pthread_join(c[z], NULL);
    }

    pthread_mutex_destroy(&mut); //destroy mutex
    sem_destroy(&E);
    sem_destroy(&F);

    return 0;

}
```