Name: Jap Purohit
Enrollment Number: AU1940109

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

a. Multiprocesses (Hint: using 3 child processes)

b. Multithreads (Hint: using 3 Threads)

Ans:

```
#include<stdio.h>
#include<pthread.h>
#include<time.h>

pthread_t tid[3];
pthread_mutex_t mutex;
unsigned int rc;
//prototypes for callback functions
int upperCase = 65;
int lowerCase = 97;
int number = 1;
int flag = 0;
void* PrintCapitalAlpabets(void*);
void* PrintLowerNumber(void*);
void* PrintNumber(void*);


void main(void)
{
   pthread_create(&tid[0],0,&PrintCapitalAlpabets,0);
   pthread_create(&tid[1],0,&PrintNumber,0);

   pthread_create(&tid[2],0,&PrintLowerNumber,0);

  //sleep(3);

   pthread_join(tid[0],NULL);
   pthread_join(tid[1],NULL);
   pthread_join(tid[2],NULL);

}

void* PrintCapitalAlpabets(void *ptr)
{
    rc = pthread_mutex_lock(&mutex);
    while(upperCase<=90)
    {
    if(flag%3==0){

          printf("%c ",upperCase);
                upperCase++;
                flag++;
```

```
    }
    else{
        rc=pthread_mutex_unlock(&mutex);//if flag modulus is not equal to 0, do not print, release mutex
for number value
    }

    }
}

void* PrintLowerNumber(void* ptr1)
{
   rc = pthread_mutex_lock(&mutex);
   while(lowerCase<=122)
   {
     if(flag%3==2){
     printf("%c ",lowerCase);
                    lowerCase++;
                    flag++;

     }
     else{
        rc=pthread_mutex_unlock(&mutex);//if flag modulus is not equal to 0, do not print, release mutex
for upper character value

     }


   }
}

void* PrintNumber(void* ptr1)
{
   rc = pthread_mutex_lock(&mutex);
  while(number <= 26)
   {
   if(flag%3==1){
    printf("%d ",number);
                   number++;
                   flag++;
   }
   else{
          rc=pthread_mutex_unlock(&mutex);//if flag modulus is not equal to 0, do not print, release
mutex for lower character value

          }


   }
}
```

Q2. Describe and implement any one of the following

a. **Describe Buddy's Algorithm for Memory Allocation and Deallocation along with an example and implement it in C or C++.**

Ans:

```
#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 n = ceil(log(s) / log(2));

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

	// Initially whole block of specified
	// size is available
	arr[n].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
        // i.e., no memory block available
        if (i == size)
        {
                cout << "Sorry, failed to allocate memory\n";
        }

        // If found
        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)
{

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

                        // Now merge the buddies to make
                        // them one 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);
}

// Driver code
int main()
{

        // Uncomment following code for interactive IO
        /*
        int total,c,req;
        cout<<"Enter Total Memory Size (in Bytes) => ";
```

```
        cin>>total;
        initialize(total);
        label:
        while(1)
        {
                cout<<"\n1. Add Process into Memory\n
                2. Remove Process \n3. Allocation Map\n4. Exit\n=> ";
                cin>>c;
                switch(c)
                {
                        case 1:
                        cout<<"Enter Process Size (in Bytes) => ";
                        cin>>req;
                        cout<<"\n===>";
                        allocate(req);
                        break;

                        case 2:
                        cout<<"Enter Starting Address => ";
                        cin>>req;
                        cout<<"\n===>";
                        deallocate(req);
                        break;

                        case 3:
                        print();
                        break;

                        case 4:
                        exit(0);
                        break;

                        default:
                        goto label;
                }
        }*/

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

        return 0;
}
```

Q3. Describe what is Producer-Consumer Problem and its solution in detail using Semaphores and Mutex and implement it in C.

Ans:

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

/*
This program provides a possible solution for producer-consumer problem using mutex and semaphore.
I have used 5 producers and 5 consumers to demonstrate the solution. You can always play with these
values.
*/

#define MaxItems 5 // Maximum items a producer can produce or a consumer can consume
#define BufferSize 5 // Size of the buffer

sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;

void *producer(void *pno)
{
   int item;
   for(int i = 0; i < MaxItems; i++) {
      item = rand(); // Produce an random item
      sem_wait(&empty);
      pthread_mutex_lock(&mutex);
      buffer[in] = item;
      printf("Producer %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);
      in = (in+1)%BufferSize;
      pthread_mutex_unlock(&mutex);
      sem_post(&full);
   }
}
void *consumer(void *cno)
{
   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 *)cno),item, out);
      out = (out+1)%BufferSize;
      pthread_mutex_unlock(&mutex);
      sem_post(&empty);
   }
}

int main()
{
```

```
    pthread_t pro[5],con[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty,0,BufferSize);
    sem_init(&full,0,0);

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