Name - Shivam Thakker
Enrollment no - AU1940193

Question 1

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



sem_t sem;
int counter = 0;


void *print_capital(void *arg){
        int i;
        for(i=0;i<26;i++){
        sem_wait(&sem);
        if(counter%3 != 0){
        i--;
        }else{
        printf("%c ",'A'+i);
        counter++;
        }
        sem_post(&sem);
        }
        pthread_exit(NULL);
}


void *print_number(void *arg){
        int i;
        for(i=0;i<26;i++){
        sem_wait(&sem);
        if(counter%3 != 1){
        i--;
        }else{
        printf("%d ",1+i);
        counter++;
        }
        sem_post(&sem);
```

```
        }
        pthread_exit(NULL);
}


void *print_small(void *arg){
        int i;
        for(i=0;i<26;i++){
        sem_wait(&sem);
        if(counter%3 != 2){
        i--;
        }else{
        printf("%c ",'a'+i);
        counter++;
        }
        sem_post(&sem);
        }
        pthread_exit(NULL);
}

int main(){


        pthread_t thread1, thread2, thread3;
        char first = 'A';
        char second = 'a';
        sem_init(&sem, 0, 1);
        pthread_create(&thread1,NULL,print_capital,NULL);
        pthread_create(&thread2,NULL,print_number,NULL);
        pthread_create(&thread3,NULL,print_small,NULL);
        pthread_join(thread1,NULL);
        pthread_join(thread2,NULL);
        pthread_join(thread3,NULL);
        sem_destroy(&sem);
        return 0;
}
```

Question 2

Buddy algorithm code for Memory Allocation

```
#include<stdlib.h>
```

```
#include<stdio.h>
using namespace std;

int size;


vector<pair<int, int>> final[1000000];


map<int, int> store;

void initialize(int temp)
{

   int n = ceil(log(temp) / log(2));
   size = n + 1;

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


   final[n].push_back(make_pair(0, temp - 1));
}

void allocate(int temp)
{


   int n = ceil(log(temp) / log(2));


   if (final[n].size() > 0)
   {
         pair<int, int> temp = final[n][0];


         final[n].erase(final[n].begin());
         cout << "Memory from " << temp.first
               << " to " << temp.second << "has been allocated"
               << "\n";

y
         store[temp.first] = temp.second -
                                       temp.first + 1;
```

```
}
else
{
	int i;
	for(i = n + 1; i < size; i++)
	{


		if(final[i].size() != 0)
			break;
	}


	if (i == size)
	{
		cout << "Error in memory allocation \n";
	}


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


		final[i].erase(final[i].begin());
		i--;

		for(; i >= n; i--)
		{

			pair<int, int> first, second;
			first = make_pair(temp.first,
							temp.first +
							(temp.second -
							temp.first) / 2);
			second = make_pair(temp.first +
							(temp.second -
							temp.first + 1) / 2,
							temp.second);

			final[i].push_back(first);
```

```
                        final[i].push_back(second);
                        temp = final[i][0];


                        final[i].erase(final[i].begin());
                }
                cout << "Memory from " << temp.first
                        << " to " << temp.second
                        << " has been allocated" << "\n";

                store[temp.first] = temp.second -
                                                temp.first + 1;
        }
   }
}


int main()
{
        initialize(2048);
        allocate(22);
        allocate(44);
        allocate(66);
        allocate(88);

        return 0;
}
```

Buddy's algorithm for memory deallocation

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


int size;


vector<pair<int, int>> store[100000];

map<int, int> final;

void Buddy(int s)
{
```

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

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


    store[n].push_back(make_pair(0, s - 1));
}

void allocate(int temp)
{


    int n = ceil(log(temp) / log(2));


    if (final[n].size() > 0)
    {
            pair<int, int> temp = final[n][0];


            final[n].erase(final[n].begin());
            cout << "Memory from " << temp.first
                    << " to " << temp.second << "has been allocated"
                    << "\n";

y
            store[temp.first] = temp.second -
                                            temp.first + 1;
    }
    else
    {
            int i;
            for(i = n + 1; i < size; i++)
            {


                    if(final[i].size() != 0)
                            break;
            }
```

```
if (i == size)
{
        cout << "Error in memory allocation \n";
}


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


        final[i].erase(final[i].begin());
        i--;

        for(; i >= n; i--)
        {

                pair<int, int> first, second;
                first = make_pair(temp.first,
                                                temp.first +
                                                (temp.second -
                                                temp.first) / 2);
                second = make_pair(temp.first +
                                                (temp.second -
                                                temp.first + 1) / 2,
                                                temp.second);

                final[i].push_back(first);


                final[i].push_back(second);
                temp = final[i][0];


                final[i].erase(final[i].begin());
        }
        cout << "Memory from " << temp.first
                << " to " << temp.second
                << " has been allocated" << "\n";

        store[temp.first] = temp.second -
                                        temp.first + 1;
}
```

```
    }
}


void deallocate(int id)
{


    if(final.find(id) == final.end())
    {
        cout << "Invalid request\n";
        return;
    }


    int n = ceil(log(final[id]) / log(2));

    int i, b_number, b_address;


    store[n].push_back(make_pair(id,
                                                    id + pow(2, n) - 1));
    cout << "Memory block from " << id
        << " to "<< id + pow(2, n) - 1
        << "has been freed\n";

    b_number = id / final[id];

    if (b_number % 2 != 0)
        b_address = id - pow(2, n);
    else
        b_address = id + pow(2, n);

    for(i = 0; i < store[n].size(); i++)
    {


        if (store[n][i].first == b_address)
        {


            if (b_number % 2 == 0)
            {
                store[n + 1].push_back(make_pair(id,
```

```
                        id + 2 * (pow(2, n) - 1)));

                        cout << "Coalescing of blocks starting from "
                                << id << " and " << b_address
                                << " was done" << "\n";
                }
                else
                {
                        store[n + 1].push_back(make_pair(
                                b_address, b_address +
                                2 * (pow(2, n))));

                        cout << "Coalescing of blocks starting from "
                                << b_address << " and "
                                << id << " was done" << "\n";
                }
                store[n].erase(store[n].begin() + i);
                store[n].erase(store[n].begin() +
                store[n].size() - 1);
                break;
        }
   }

   final.erase(id);
}

int main()
{

        Buddy(2048);
        allocate(16);
        allocate(32);
        allocate(16);
        allocate(9);
        deallocate(0);
        deallocate(9);
        deallocate(32);
        deallocate(16);

   return 0;
}
```

Question 3

Solution for Producer-Consumer problem using semaphore and mutex

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



#define MaxItems 5
#define BufferSize 5

sem_t empty_semaphore;
sem_t full_semaphore;
int up = 0;
int down = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;

void *producer_code(void *prod)
{
        int item;
        for(int i = 0; i < MaxItems; i++) {
        item = rand();
        sem_wait(&empty_semaphore);
        pthread_mutex_lock(&mutex);
        buffer[up] = item;
        printf("Producer %d: Insert Item %d at %d\n", *((int *)prod),buffer[up],up);
        up = (up+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full_semaphore);
        }
}
void *consumer_code(void *cons)
{
        for(int i = 0; i < MaxItems; i++) {
        sem_wait(&full_semaphore);
        pthread_mutex_lock(&mutex);
        int item = buffer[down];
        printf("Consumer %d: Remove Item %d from %d\n",*((int *)cons),item, down);
        down = (down+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
```

```
        sem_post(&empty_semaphore);
        }
}

int main()
{

        pthread_t first[5],second[5];
        pthread_mutex_init(&mutex, NULL);
        sem_init(&empty_semaphore,0,BufferSize);
        sem_init(&full_semaphore,0,0);

        int arr[5] = {1,2,3,4,5};

        for(int i = 0; i < 5; i++) {
        pthread_create(&first[i], NULL, (void *)producer_code, (void *)&arr[i]);
        pthread_create(&second[i], NULL, (void *)consumer_code, (void *)&arr[i]);
        }

        for(int i = 0; i < 5; i++) {
        pthread_join(first[i], NULL);
        pthread_join(second[i], NULL);
        }

        pthread_mutex_destroy(&mutex);
        sem_destroy(&full_semaphore);
        sem_destroy(&empty_semaphore);


        return 0;

}
```