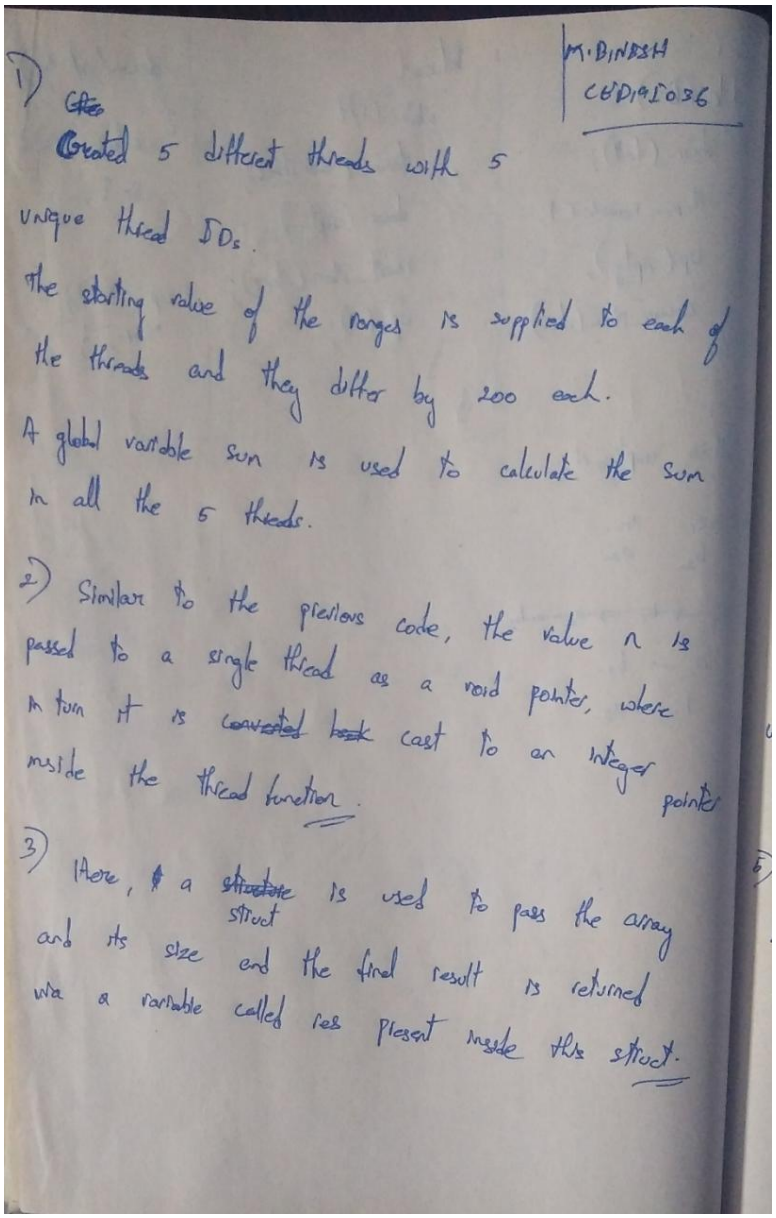
Lab Assignment - 9
(Operating Systems Practice)

Name: - M. Binesh
Roll : - CED19I036



M. BINESH
CED19I036

1) Created 5 different threads with 5 unique thread IDs.
The starting value of the ranges is supplied to each of the threads and they differ by 200 each.
A global variable sum is used to calculate the sum in all the 5 threads.

2) Similar to the previous code, the value n is passed to a single thread as a void pointer, where in turn it is cast to an integer pointer inside the thread function.

3) Here, a struct is used to pass the array and its size and the final result is returned via a variable called res present inside this struct.

1)

```
kuries@Beast:~/.../osp/lab9$ gcc 1.c -lpthread && ./a.out
500500
kuries@Beast:~/.../osp/lab9$
```

2)

```
kuries@Beast:~/.../osp/lab9$ gcc 2.c -lpthread && ./a.out
Enter the value n: 15
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

3)

```
kuries@Beast:~/.../osp/lab9$ gcc 3.c -lpthread && ./a.out
Enter the size: 8
Enter the numbers: 1 2 3 4 5 6 8 7
36
kuries@Beast:~/.../osp/lab9$
```

M. BINESH
CSD19D026

4) Here, we are going to have a buffer of size 5. Let us assume that producer is faster than the consumer, where it produces 9 units in the same time that it takes for the consumer to consume 4 units. To simulate this situation we used for loops.

To keep the buffer from overflowing or from getting emptied completely we use wait and post functions from the semaphore library. We also use two variables full and empty. full → no. of units present in the buffer
Empty → no. of empty slots in the buffer.

We also mutex locks on the buffer which is a shared resource.

5) Two semaphore variables first and second are used. Both are initialised to 0.
first is responsible for b_1 occurring before a_2.
Second is responsible for a_1 occurring before b_2.

4)

```
kuries@Beast:~/.../osp/lab9$ gcc 4.c -lpthread && ./a.out
Producer: 1

Producer: 1 2

Producer: 1 2 3

Producer: 1 2 3 4

Consumer: 1 2 3

Consumer: 1 2

Consumer: 1

Consumer:

Producer: 1

Producer: 1 2

Producer: 1 2 3

Producer: 1 2 3 4

Producer: 1 2 3 4 5

Consumer: 1 2 3 4
```

5)

```
b2
kuries@Beast:~/.../osp/lab9$ gcc 5.c -lpthread && ./a.out
b1
a1
a2
b2
kuries@Beast:~/.../osp/lab9$ gcc 5.c -lpthread && ./a.out
a1
b1
b2
a2
kuries@Beast:~/.../osp/lab9$
```