

## Description

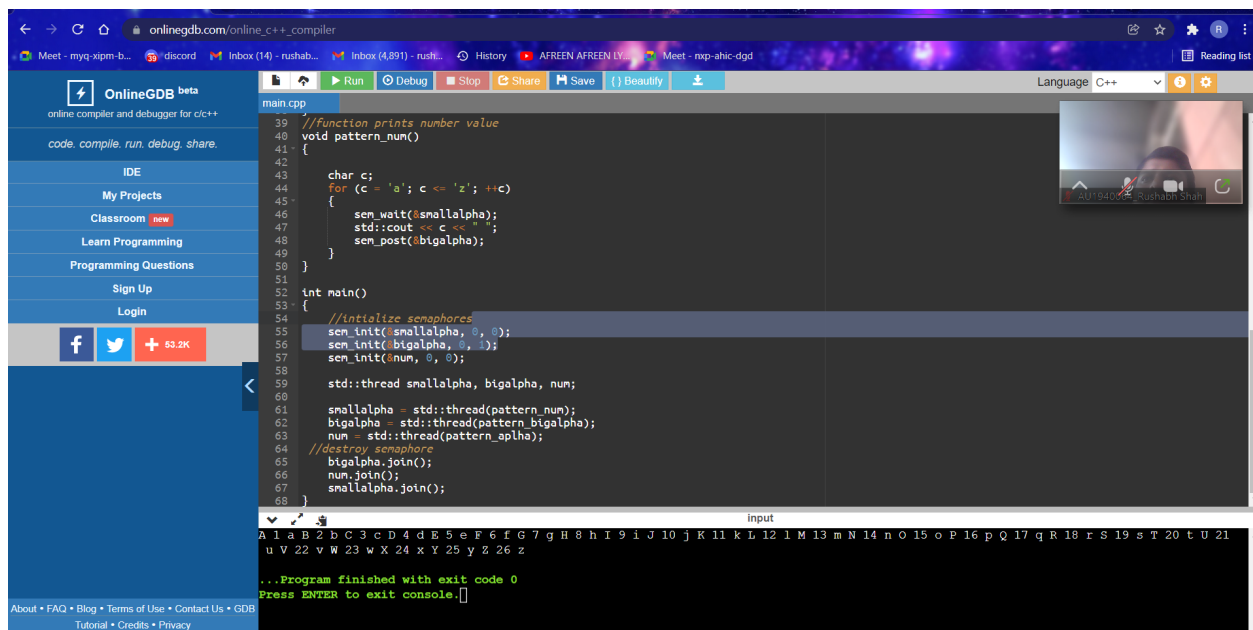
### End-Sem Examination

Name Rushabh Shah  
AU1940064

My virtual box was not working so i have run my codes on online compiler it was crashing.

Q1)

In this most obvious way to generate values is to do by generating 25 threads which can print A1a but it not optimum way we can use only three threads and generate the pattern 1thread responsible for generating capital letter then second for number and atleast third thread for generating small letter hence we can use three semaphore hence the generated output.



```
main.cpp
39 //Function prints number value
40 void pattern_num()
41 {
42
43     char c;
44     for (c = 'a'; c <= 'z'; ++c)
45     {
46         sem_wait(&smallalpha);
47         std::cout << c << " ";
48         sem_post(&bigalpha);
49     }
50 }
51
52 int main()
53 {
54     //initialize semaphores
55     sem_init(&smallalpha, 0, 0);
56     sem_init(&bigalpha, 0, 1);
57     sem_init(&num, 0, 0);
58
59     std::thread smallalpha, bigalpha, num;
60
61     smallalpha = std::thread(pattern_num);
62     bigalpha = std::thread(pattern_num);
63     num = std::thread(pattern_num);
64
65     //destroy semaphore
66     bigalpha.join();
67     num.join();
68     smallalpha.join();
69 }
```

input

A 1 a B 2 b C 3 c D 4 d E 5 e F 6 f G 7 g H 8 h I 9 i J 10 j K 11 k L 12 l M 13 m N 14 n O 15 o P 16 p Q 17 q R 18 r S 19 s T 20 t U 21 u V 22 v W 23 w X 24 x Y 25 y Z 26 z

...Program finished with exit code 0  
Press ENTER to exit console.

Q2)

Buddy memory allocation is a memory allocation approach in which a memory allocation algorithm divides memory into segments and tries to satisfy a memory request as appropriately and quickly as feasible. Thus what the buddy memory allocator does is give the allocation in the form of a power of two, so for example, if there is a request of 25 KB, the block of 32 KB will be chosen. This implies that if there is a

request, it will choose the closest larger block, such as 54 KB or 64 KB. There is memory waste in this memory allocation since a block of 54 is put in a block of 64. Now is the time to find a deal. When a deallocation request comes in, we'll check at the map to see if it's a valid request. If that's the case, the block will be added to the free list, which tracks blocks of varying sizes. Then we'll test if its companion is available on the free list, and if so, we'll combine the blocks and add them to the free list.

```

1
2
3
4 //Rushabh Shah
5 #include<bits/stdc++.h>
6 using namespace std;
7
8 // Size of vector of pairs
9 int size;
10
11
12 vector<pair<int, int>> arr[100000];
13
14 // Map used as hash map to store the
15 // starting address as key and size

```

Memory from 0 to 15 allocate  
Memory from 16 to 31 allocated  
Memory from 32 to 47 allocated  
Memory from 48 to 63 allocated  
Memory block from 0 to 15 freed  
Sorry, invalid free request  
Memory block from 32 to 47 freed  
Memory block from 16 to 31 freed  
Coalescing of blocks starting at 0 and 16 was done

...Program finished with exit code 0  
Press ENTER to exit console.

Q3)

## Description

Producer consumer problem is a problem on synchronization which can be solved by using semaphores and mutex.

## Statement:

A producer can produce an item and can place it in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time the consumer should not consume any item. In this problem, the buffer is the critical section.

## Solution:

### Key problems:

- Manage shared memory access
- Check if buffer is full or empty.

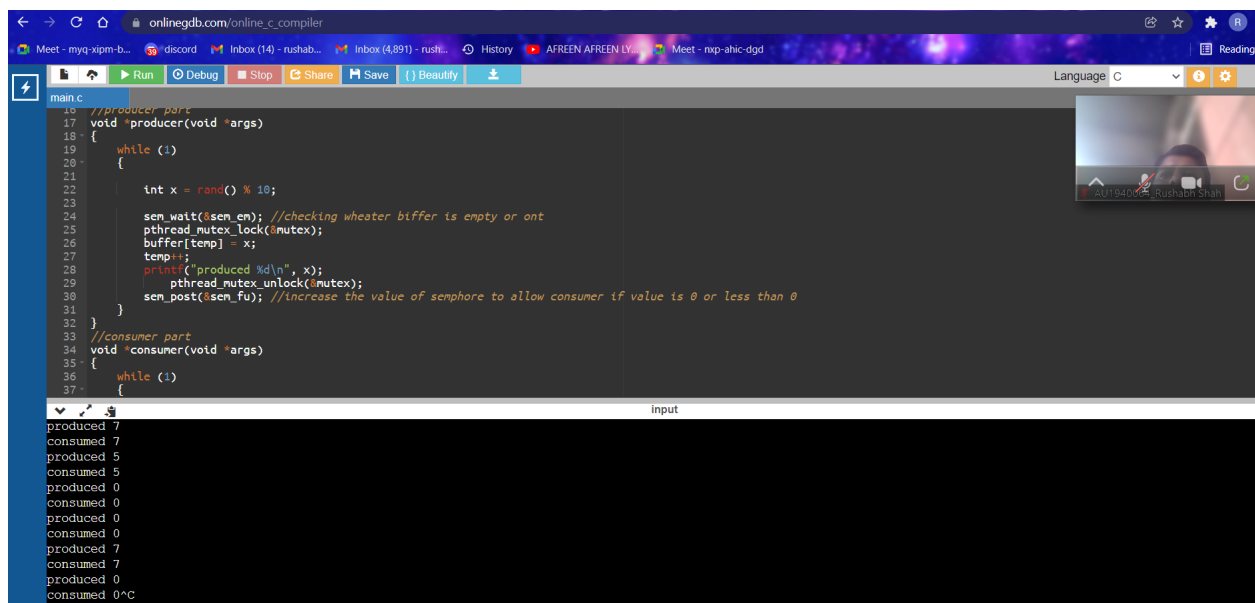
### Basic idea

Managing shared access can be solved with help of the mutex before and after accessing the buffer from the main memory. To solve the issues whether the buffer is full or empty can be implemented with help of sleep function unit the buffer is full either empty but this method is not that efficient so we are using two counting semaphore

Basic info about counting semaphore

When value of semaphore is one 0 thread has to wait until the value of semaphore becomes more than 0

We are using two semaphores. One is initialized at the max size of the buffer will other is initialized at 0 because for the consumer part the initial buffer is empty so it has to wait until producer produces something and adds it to the buffer.



```
16 //producer part
17 void *producer(void *args)
18 {
19     while (!)
20     {
21         int x = rand() % 10;
22
23         sem_wait(&sem_en); //checking wheater biffer is empty or ont
24         pthread_mutex_lock(&mutex);
25         buffer[temp] = x;
26         temp++;
27         printf("produced %d\n", x);
28         pthread_mutex_unlock(&mutex);
29         sem_post(&sem_fu); //Increase the value of semaphore to allow consumer if value is 0 or less than 0
30     }
31 }
32
33 //consumer part
34 void *consumer(void *args)
35 {
36     while (!)
37     {
```

input

```
produced 7
consumed 7
produced 5
consumed 5
produced 0
consumed 0
produced 0
consumed 0
produced 7
consumed 7
produced 0
consumed 0
```