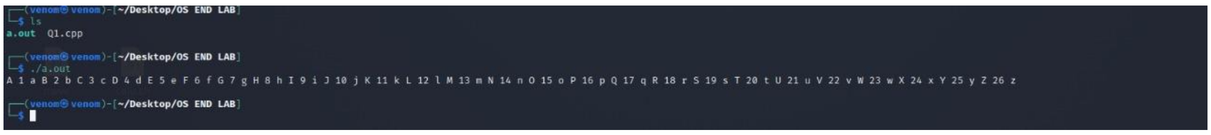
OPERATING SYSTEM PRACTICAL END SEM

Aagam Shah

AU1940148

Question1

So in this question we are suppose to get out put like first capital, second numerical, and third little letter using 3 semphores.

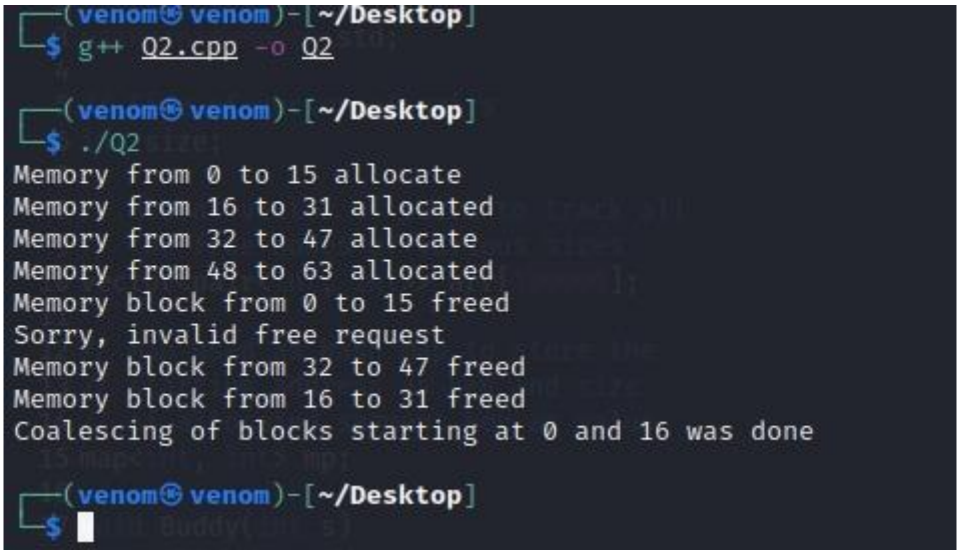


```
┌──(venom㉿venom)-[~/Desktop/OS END LAB]
└─$ ls
a.out  Q1.cpp

┌──(venom㉿venom)-[~/Desktop/OS END LAB]
└─$ ./a.out
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

┌──(venom㉿venom)-[~/Desktop/OS END LAB]
└─$ 
```

Question2

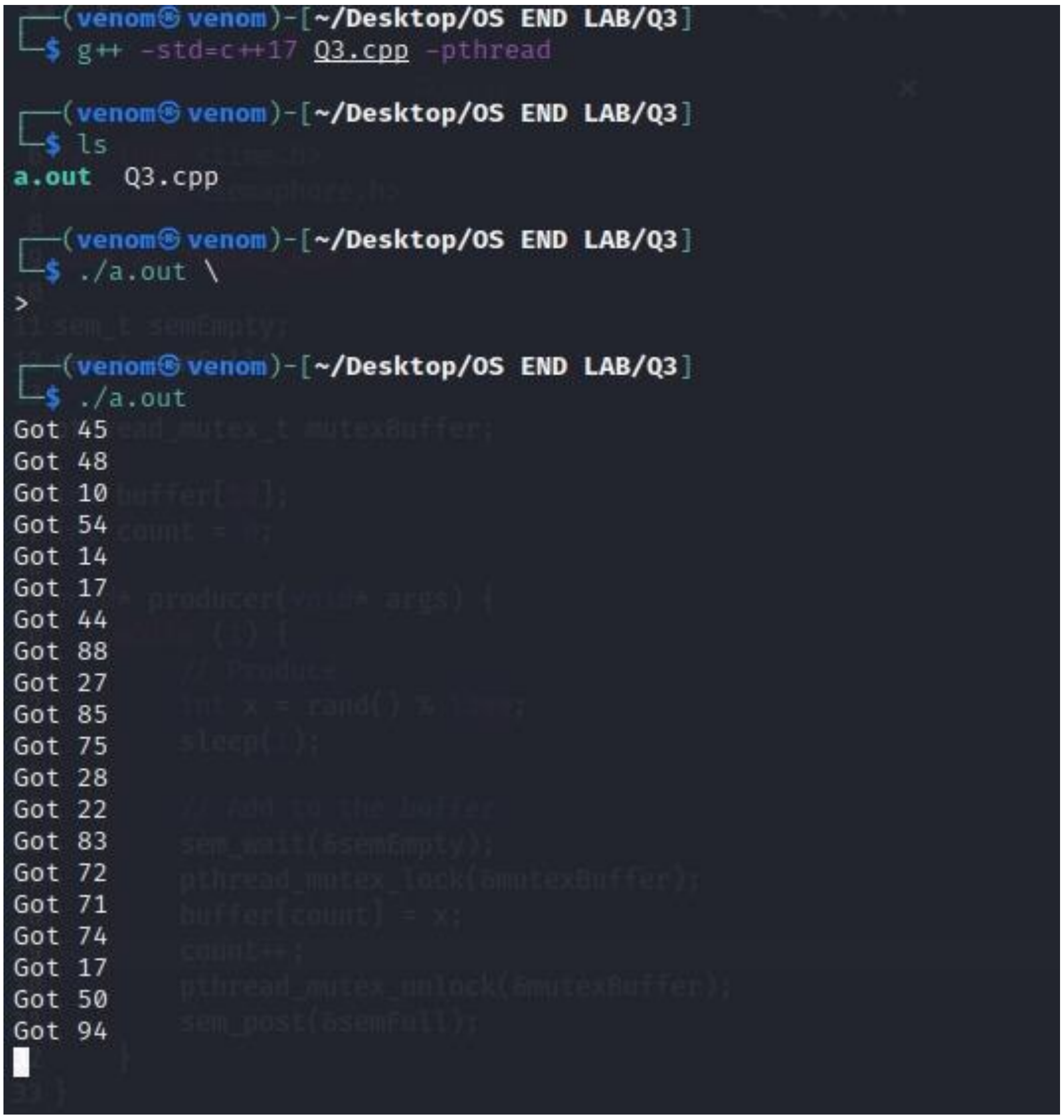
Description :- The buddy memory allocation is a technique in which memory allocation algorithm that divides memory into partitions and will try to satisfy a memory requestion as suitably and as soon possible. So what buddy memory allocator does is that it gives the allocation in the form of power of 2 so for example when there Is a request of 28 KB so the block of 32 KB will be block so that means that when ever there is a request it will choose the closet bigger block than it means of 44kb than it will chose block of 64kb. In this memory allocation there Is a waste-age of memory as if block is of 44 and it is stored in block of 64. So the above was a explanation of allocation using free lists. Now in deallocation

When a deallocation request arrives, we'll first look at the map to determine if it's a legitimate request. If that's the case, we'll add the block to the free list that keeps track of blocks of various sizes. Then we'll check the free list to see whether its mate is available; if so, we'll combine the blocks and add them to the free list.



```
┌──(venom㉿venom)-[~/Desktop]
└─$ g++ Q2.cpp -o Q2

┌──(venom㉿venom)-[~/Desktop]
└─$ ./Q2
Memory from 0 to 15 allocate
Memory from 16 to 31 allocated
Memory from 32 to 47 allocate
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

┌──(venom㉿venom)-[~/Desktop]
└─$ 
```

Question 3



```
┌──(venom㉿venom)-[~/Desktop/OS END LAB/Q3]
└─$ g++ -std=c++17 Q3.cpp -pthread

┌──(venom㉿venom)-[~/Desktop/OS END LAB/Q3]
└─$ ls
a.out  Q3.cpp

┌──(venom㉿venom)-[~/Desktop/OS END LAB/Q3]
└─$ ./a.out \
>

┌──(venom㉿venom)-[~/Desktop/OS END LAB/Q3]
└─$ ./a.out
Got 45
Got 48
Got 10
Got 54
Got 14
Got 17
Got 44
Got 88
Got 27
Got 85
Got 75
Got 28
Got 22
Got 83
Got 72
Got 71
Got 74
Got 17
Got 50
Got 94
```

The producer-consumer problem is a synchronisation issue that can be handled with semaphores and mutexes.

A producer has the ability to create an item and store it in the buffer. A consumer has the ability to choose and consume objects. We must ensure that when a producer places an item in the buffer, the consumer does not eat any of the items at the same time. The buffer is the critical portion in this situation.

Problems must be solved:

- Control access to shared memory.
- Determine whether the buffer is full or empty.

Before and after requesting the buffer from main memory, the mutex can be used to manage shared access. The buffer is full or empty may be implemented with the assistance of the sleep function unit, but this way is inefficient, thus we are employing two counting semaphores. The fundamentals of semaphore counting .When the value of the semaphore is one zero, the thread must wait until the value of the semaphore is greater than zero.