Name: Maulikkumar Bhalani

Enrollment No: AU1940206

2-B.

Buddy’s Algorithm for Memory Alocation:

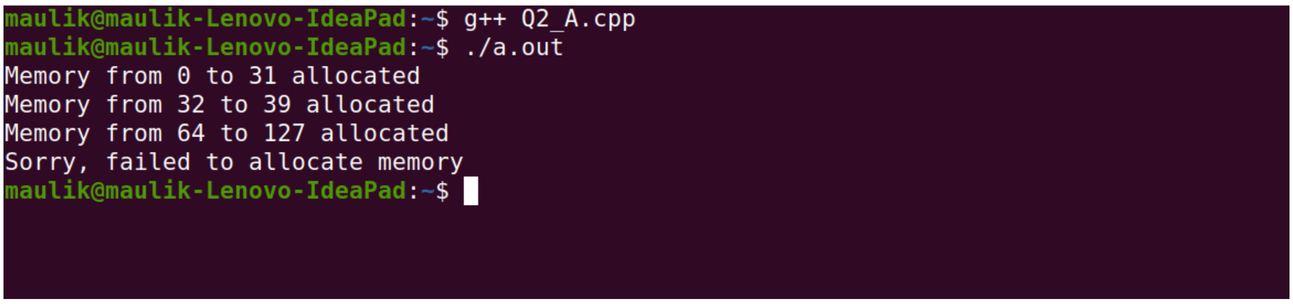
This is the mehtod for alocation in which available spaces is repeatedly split into the halves and it is knows as the buddies.

It keeps spliting buddies until it chooses the smallest possible block that can contain the file. A list of free nodes, of all the different possible powers of two is maintained at all times

When a user is requested for the alocation of memory, we look for the smallest block as compare to bigger than the same. If such a block is found on the free linkedlist, the allocation is done. else we need to traverse the free list *upwards* till the we can find a big enough block for the alocation. Once we find the enough big block, we need to keep splitting it in the two blocks for adding to the next lists. We need to repeat the same process and one to traverse *down* the tree till we reach the target block and return the memory block which is requested by the user and provide the same to the user. If no sucha allocation is found, we simply returns the null value.

Let take one memory size of 1024KB and we take a processor P which requires 90KB swap. 128KB is sufficient, as the hole list is dedicated to powers of 2. Initially (when process start) there are no any 128KB available so firstly 1024KB block is split into two buddies of 512KB each, it is further spilt into two buddies of 256KB each, still it is further split into two buddies of 128KB blocks and now we reach the condition so one of that is allocated to the process. Now if we take processor P2 which requires a 150KB thus we required 256KB of block. Therefore, when 512KB block split into two 256KB buddies then if we take one more process P3 which required 80KB then that will be adjusted in whole 512KB block. So, at the end if any such blocks are free then they block recombined and can make one larger original block and adjust the remaining space.

Screenshot:



```
maulik@maulik-Lenovo-IdeaPad:~$ g++ Q2_A.cpp
maulik@maulik-Lenovo-IdeaPad:~$ ./a.out
Memory from 0 to 31 allocated
Memory from 32 to 39 allocated
Memory from 64 to 127 allocated
Sorry, failed to allocate memory
maulik@maulik-Lenovo-IdeaPad:~$
```

Buddy’s Algorithm for Memory Dealocation:

The allocation is done via the usage of free lists as we discussed. Now, for deallocation, we will maintain an extra data structure-a Map with the starting address of segment as key and size of the segment as value and we will update it whenever an allocation request is done by the user. Now, when a deallocation request comes by the user, we will first check the map to see if it is a valid request or not. If it is, we will then add the block to the free list tracking blocks of their sizes and display it. If not,then we will deny the same.Then, we will search the free list to see if its *buddy* is free-if so, we will merge the blocks and place them on the free list.

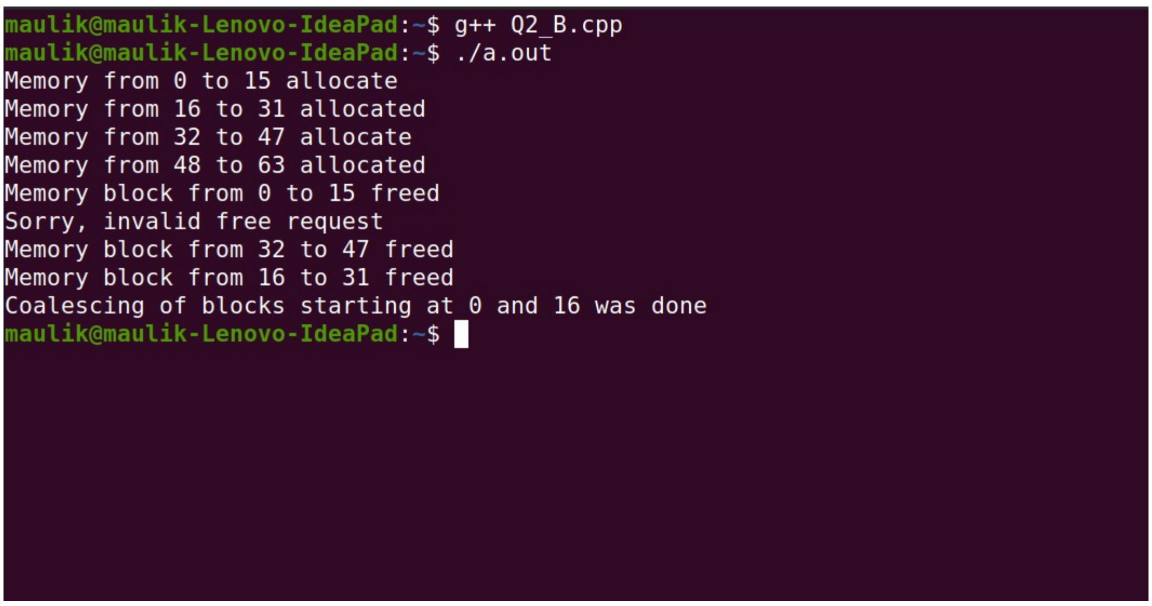
We should note that this is always an integer value, as both numerator and denominator are powers of 2. For example, if 4 consecutive allocation requests of 16 bytes come, we will end up with blocks 0-15, 16-31, 32-47, 48-63 where blocks 0-15 and 16-31 are buddies (as they were formed by splitting block 0-32) but 0-15 and 32-47 aren’t as they aren’t formed from the same block.

==> (base_address-starting_address_of_main_memory)/block_size

If a Allocation Request is 16 bytes then No such block found, so we traverse up and split the 0-127 block into 0-63, 64-127; we add 64-127 to list tracking 64-byte blocks and pass 0-63 downwards; again it is split into 0-31 and 32-63; we add 32-63 to list tracking 32-byte blocks, passing 0-31 downwards; one more splits done and 0-15 is returned to the user while 16-31 is added to free list tracking 16-byte blocks.

Deallocation Request: StartIndex = 16 Deallocation will be done and coalescing of the blocks 0-15 and 16-31 will also be done as the buddyAddress of block 16-31 is 0, which is present in the free list tracking 16-byte blocks.

Screenshot:



```
maulik@maulik-Lenovo-IdeaPad:~$ g++ Q2_B.cpp
maulik@maulik-Lenovo-IdeaPad:~$ ./a.out
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
maulik@maulik-Lenovo-IdeaPad:~$
```

3.

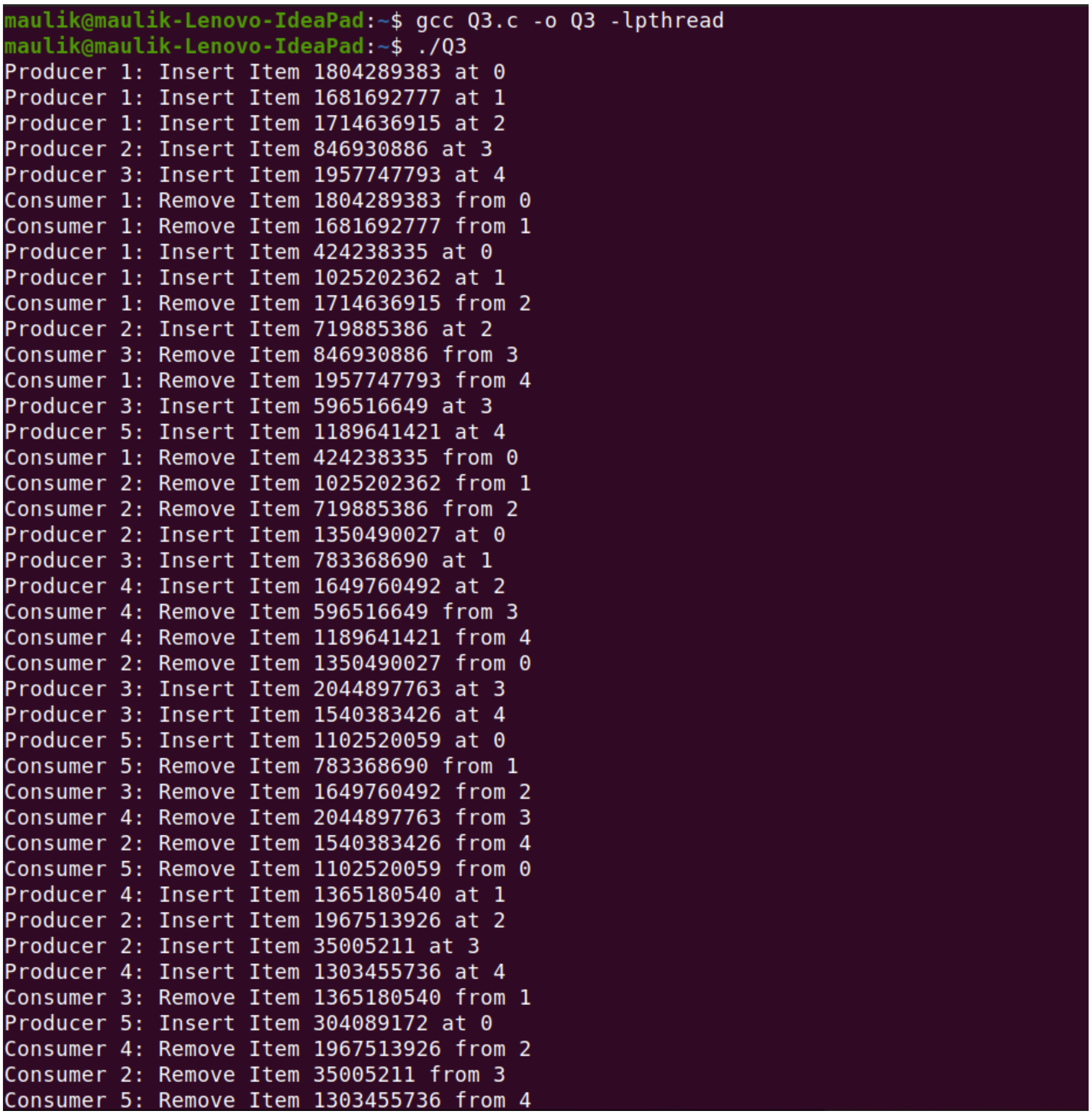
The Producer-Consumer problem is a classic problem this is used for multi-process synchronization. In the producer-consumer problem, there is one Producer that is producing something and there is one Consumer that is consuming the products produced by the Producer. The producers and consumers share the same memory buffer that is of fixed-size. If the consumer is consuming an item from the buffer, the producer should not generate the item in the buffer and vice versa. As the producer or consumer needs access to the buffer.

Producer consumer problem can be solve using Semaphores and Mutex. The main job of the Producer is to generate the data, put it into the buffer, and again start generating data. While the job of the Consumer is to consume the data from the buffer.

General Situation: – One or more producers are generating data and placing these in a buffer – A single consumer is taking items out of the buffer one at time – Only one producer or consumer may access the buffer at any one timeEnsure that the Producer can’t add data into full buffer and consumer can’t remove data from empty buffer.

These are the condition that we need to keep in mind while implementing the producer-consumer problem. The producer should produce data only when the buffer is not full. If the buffer is full, then the producer shouldn't be allowed to put any data into the buffer. The consumer should consume data only when the buffer is not empty. If the buffer is empty, then the consumer shouldn't be allowed to take any data from the buffer. The producer and consumer should not access the buffer at the same time.

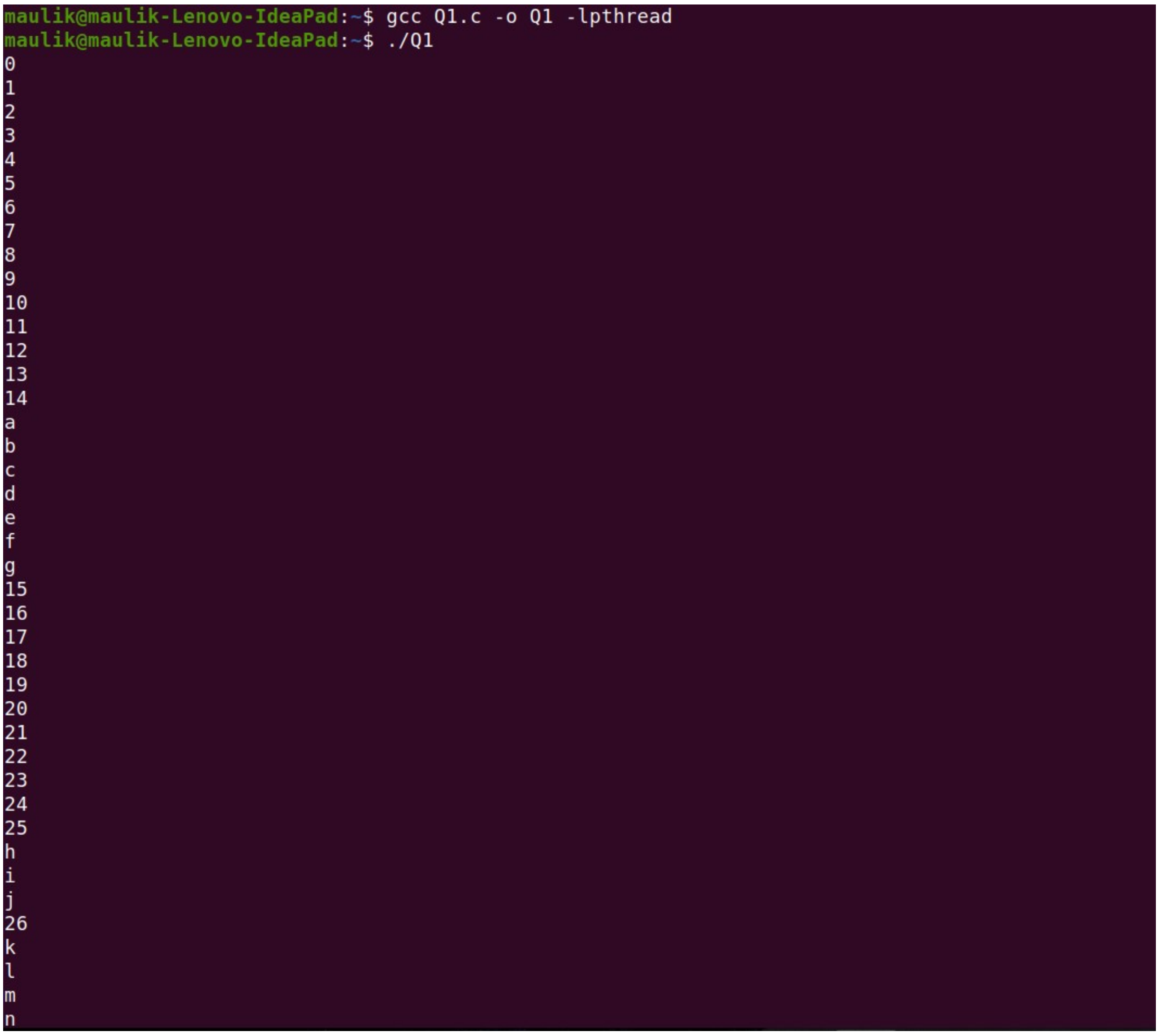
Screenshot:



```
maulik@maulik-Lenovo-IdeaPad:~$ gcc Q3.c -o Q3 -lpthread
maulik@maulik-Lenovo-IdeaPad:~$ ./Q3
Producer 1: Insert Item 1804289383 at 0
Producer 1: Insert Item 1681692777 at 1
Producer 1: Insert Item 1714636915 at 2
Producer 2: Insert Item 846930886 at 3
Producer 3: Insert Item 1957747793 at 4
Consumer 1: Remove Item 1804289383 from 0
Consumer 1: Remove Item 1681692777 from 1
Producer 1: Insert Item 424238335 at 0
Producer 1: Insert Item 1025202362 at 1
Consumer 1: Remove Item 1714636915 from 2
Producer 2: Insert Item 719885386 at 2
Consumer 3: Remove Item 846930886 from 3
Consumer 1: Remove Item 1957747793 from 4
Producer 3: Insert Item 596516649 at 3
Producer 5: Insert Item 1189641421 at 4
Consumer 1: Remove Item 424238335 from 0
Consumer 2: Remove Item 1025202362 from 1
Consumer 2: Remove Item 719885386 from 2
Producer 2: Insert Item 1350490027 at 0
Producer 3: Insert Item 783368690 at 1
Producer 4: Insert Item 1649760492 at 2
Consumer 4: Remove Item 596516649 from 3
Consumer 4: Remove Item 1189641421 from 4
Consumer 2: Remove Item 1350490027 from 0
Producer 3: Insert Item 2044897763 at 3
Producer 3: Insert Item 1540383426 at 4
Producer 5: Insert Item 1102520059 at 0
Consumer 5: Remove Item 783368690 from 1
Consumer 3: Remove Item 1649760492 from 2
Consumer 4: Remove Item 2044897763 from 3
Consumer 2: Remove Item 1540383426 from 4
Consumer 5: Remove Item 1102520059 from 0
Producer 4: Insert Item 1365180540 at 1
Producer 2: Insert Item 1967513926 at 2
Producer 2: Insert Item 35005211 at 3
Producer 4: Insert Item 1303455736 at 4
Consumer 3: Remove Item 1365180540 from 1
Producer 5: Insert Item 304089172 at 0
Consumer 4: Remove Item 1967513926 from 2
Consumer 2: Remove Item 35005211 from 3
Consumer 5: Remove Item 1303455736 from 4
```

1.

Screenshot:



```
maulik@maulik-Lenovo-IdeaPad:~$ gcc Q1.c -o Q1 -lpthread
maulik@maulik-Lenovo-IdeaPad:~$ ./Q1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
a
b
c
d
e
f
g
15
16
17
18
19
20
21
22
23
24
25
h
i
j
26
k
l
m
n
```