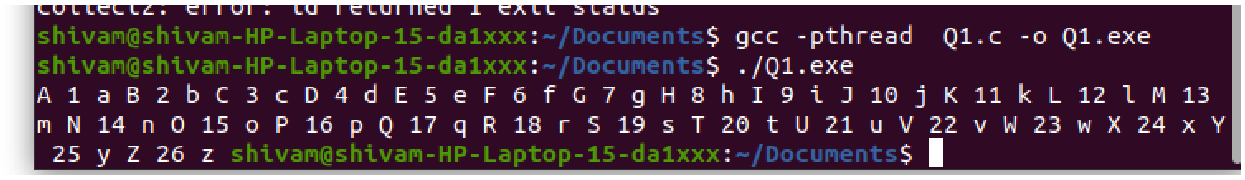
Name - Shivam Thakker
Enrollment no - AU1940193

Question 1(b)



```
collect2: error: ld returned 1 exit status
shivam@shivam-HP-Laptop-15-da1xxx:~/Documents$ gcc -pthread  Q1.c -o Q1.exe
shivam@shivam-HP-Laptop-15-da1xxx:~/Documents$ ./Q1.exe
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 shivam@shivam-HP-Laptop-15-da1xxx:~/Documents$
```

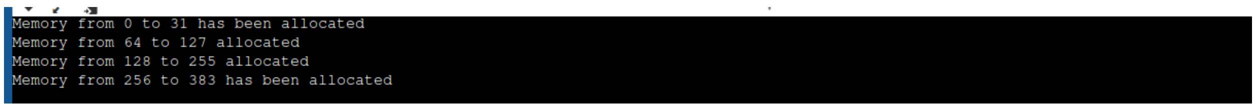
Here in this question to generate the given output we are using threads and semaphores, particularly binary semaphore and a shared variable counter. Then we have created three functions print_capital, print_number and print_small and initialize the threads thread1, thread2 and thread3. In print_capital function, we have used a for loop from 0 to 26 and are waiting for a response from the semaphore. As soon as it gets the response, we check if the shared value of counter modulo 3 is zero then we print “i + A”. Else we decrement i by 1. Similarly we are creating the function of print_number and print_small to print numbers and small letters. Then in the main function we are creating 3 threads using pthread_create and use pthread_join in order to wait for other threads to complete their task.

Question 2(b)

**Buddy’s Algorithm**

Buddy’s algorithm is used as a memory management system to allocate and deallocate the memory when some system faces issues of memory allocation for new processes or the memory is full. The memory in this system is incremented by 2 and it also merges the unallocated block so in order to give memory to some other process which in case require more memory.

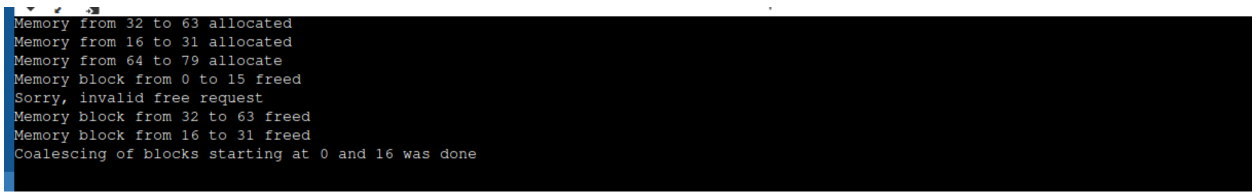
**Allocation of memory using Buddy’s Algorithm**



```
Memory from 0 to 31 has been allocated
Memory from 64 to 127 allocated
Memory from 128 to 255 allocated
Memory from 256 to 383 has been allocated
```

Here in the memory allocation part, we create and maintain a list which includes the powers of 2 like 1,2,4 etc and so on. Now when a user request for some memory, then this algorithm finds the block that is just bigger than the requested block. For eg- If we want a memory of 50kb then the algorithm will give 64kb of memory. If we do not find the block then the algorithm will move further and will find the block with memory greater than the requested memory.

**Deallocation of memory using Buddy's Algorithm**



```
Memory from 32 to 63 allocated
Memory from 16 to 31 allocated
Memory from 64 to 79 allocate
Memory block from 0 to 15 freed
Sorry, invalid free request
Memory block from 32 to 63 freed
Memory block from 16 to 31 freed
Coalescing of blocks starting at 0 and 16 was done
```

Here in the deallocation of the memory, we will create another map during the memory allocation time and store the key and value in the map. We will store the address as a key and the memory requested as value. So when the deallocation of memory is requested, we check the key value pair and if it is valid then we will deallocate the memory and add it into the free list.

**Question 3**

**Producer-Consumer problem**

Here in the producer-consumer problem we are using a fixed size buffer to solve the problem. Here, the producer produces the data and adds into the buffer. Whereas the consumer consumes the data from the buffer. So here the main problem is that we need to make sure that the producer doesn't add the data when the buffer is full and the consumer doesn't take the data when the buffer is empty. Also we need to make sure that the data from the buffer is not lost while doing this process and proper data is added and collected back while requesting from the buffer.

**Producer's Code**

```
while(True) {
  wait(empty);
  wait(mutex);
  buffer[in] = item
  in = (in+1)%buffersize;
  signal(mutex);
  signal(full);
}
```

Here the producer produces the data and first of all checks if the queue or the buffer is empty or not. If it is empty then it will add the data to the index 'in' of the buffer and then increment the value of 'in' for the next data to be added.

**Consumer’s Code**

```
while(True) {
  wait(full);
  wait(mutex);
  item = buffer[out];
  out = (out+1)%buffersize;
  signal(mutex);
  signal(empty);
}
```

Here the consumer first check if there is some data in the buffer. If there exist some data then he will get the data from buffer by using buffer[out] and then increment out in order to get the next data from the buffer and free the space from where it has taken the data so producer can refill the buffer.