

Shail Patel

AU1940142

1. Print the following Pattern

A 1 a B 2 b C 3 c ... Y 25 y Z 26 z

Using any one of the following concepts

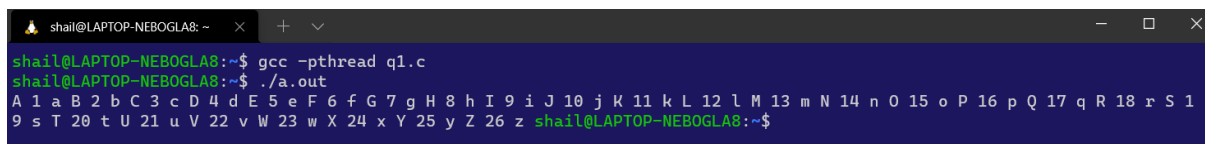
b. Multithreads (Hint: using 3 Threads)

Description:

In this program I have created 3 separate threads.

1 for capital letter display. 2 for lower case display and 3 for displaying numbers. Three different functions for displaying the three different cases. The variable gain lock is used to get lock for different threads so that only after the given thread is completed fully other thread can get a chance to run. At the start of the function the lock is secured by the function and the lock is released when the funtion gets over at the last time. The process keeps on repeating for 26 times for each and every thread to run giving the desired output.

Output:



```
shail@LAPTOP-NEBOGLA8: ~  
shail@LAPTOP-NEBOGLA8:~$ gcc -pthread q1.c  
shail@LAPTOP-NEBOGLA8:~$ ./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 1  
9 s T 20 t U 21 u V 22 v W 23 w X 24 x Y 25 y Z 26 z shail@LAPTOP-NEBOGLA8:~$
```

Q2. Describe and implement any one of the following [20]

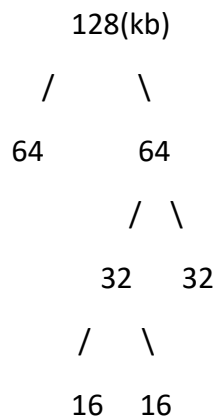
b) Describe the Buddy's Algorithm for Memory Allocation and Deallocation along with an example and implement it in C or C++.

Description

Buddy Algorithm: Static partitioning techniques suffer from having a fixed number of partitions. Due to the fixed partitions the number of processes that can be handled

becomes limited. Buddy algorithm can help in solving this problem to some extent. Buddy algorithms work on dividing the memory space in powers of 2.

Example



if a 16 kb process comes in the process gets allocated to the 16kb space on the above tree. If a 18kb process comes in it searches for the best fit from the bottom of the tree and gets allocated to the 32 kb process. This works on recursively dividing the block into equal parts.

Advantages

1. The left-out blocks can be merged together to create a larger chunk of memory space. (note)
2. Fast to allocate memory and de-allocate memory.
3. Easy to implement.

(Note)

A split block can only be combined with its similar type of buddy block which then reassembles to form the bigger block from which they were split.

Disadvantages

1. Can lead to internal fragmentation as we saw in the above question with the 18kb process.
2. The allocation needs to be in powers of 2.

Output

```

shail@LAPTOP-NEBOGLA8:~$ g++ q2.cpp -o answer -lpthread
shail@LAPTOP-NEBOGLA8:~$ ./answer
Memory from 0 to 31 allocate
Memory from 32 to 47 allocate
Memory from 48 to 63 allocated
Memory from 64 to 79 allocate
Memory block from 0 to 31 freed
Sorry, invalid free request
Sorry, invalid free request
Sorry, invalid free request
shail@LAPTOP-NEBOGLA8:~$

```

3. [Bonus] Describe what is Producer Consumer Problem and its solution in detail using Semaphores and Mutex and implement it in C.

Description

Producer consumer problem: The producer consumer problem is a classic synchronisation problem. The producer is producing something and the consumer is consuming something. The producer and the consumer use the same memory buffer. The producer should only be allowed to produce if the buffer size is not full and the consumer should only consume if the buffer is not empty. Another thing to keep in mind is that the consumer and the producer should not be using the buffer at the same time. We would be needing to achieve this by using mutual exclusion using semaphores.

Pseudo code:

```

Mutex mutex; // Used to provide mutual exclusion for critical section
Semaphore empty = N; // Number of empty slots in buffer
Semaphore full = 0 // Number of slots filled
int in = 0; //index at which producer will put the next data
int out = 0; // index from which the consumer will consume next data
int buffer[N];

```

Producer Code

```

while(True) {
    produce(); // produce an item
    wait(empty); // wait till there is an empty slot in the buffer
    wait(mutex); // wait for mutual exclusion
    buffer[in] = item
    in = (in+1)%buffersize;
    signal(mutex);
    signal(full); // signal the consumer that the buffer has data now and can start consuming
}

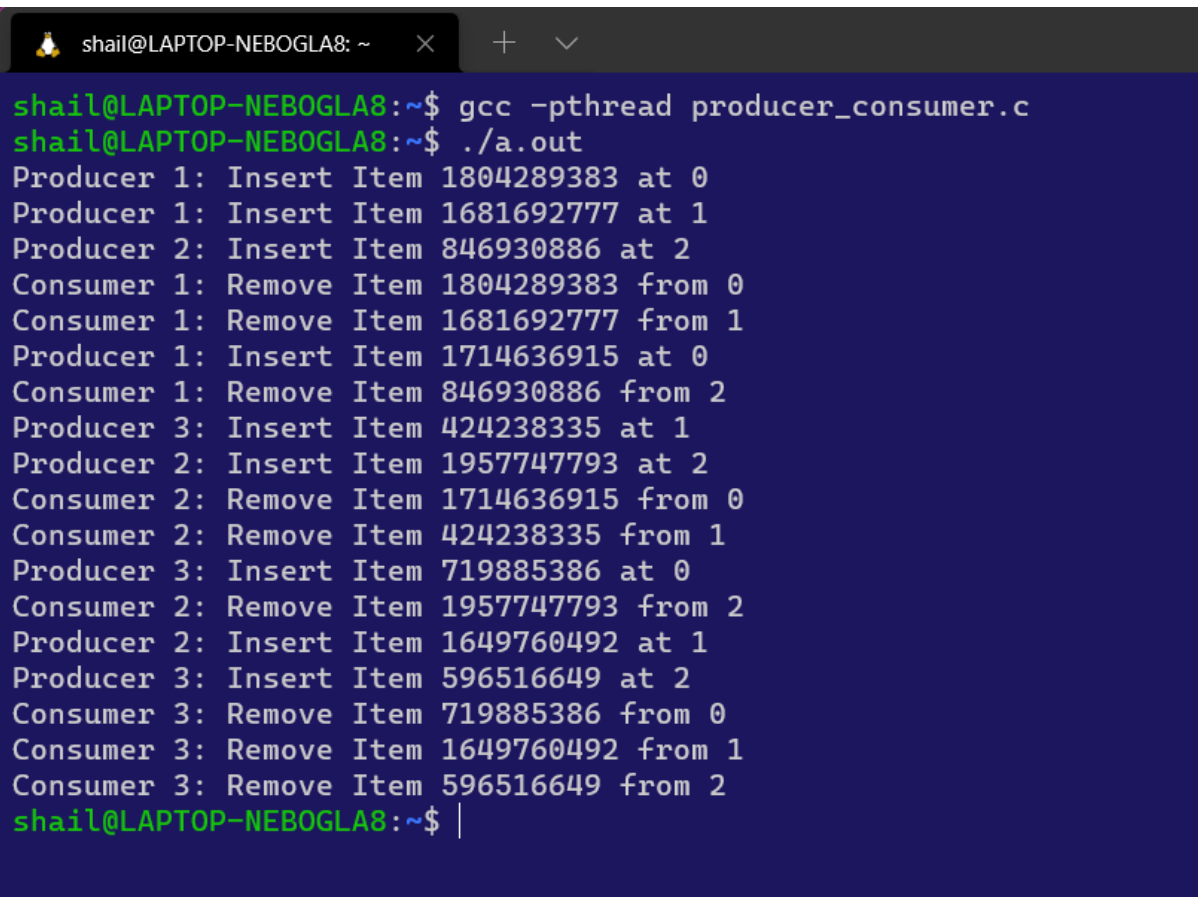
```

```
}
```

Consumer Code

```
while(True) {  
    wait(full); // the consumer waits until the producer sends the signal that the buffer has data  
    wait(mutex); // wait for mutual exclusion to ensure that buffer is not jointly used by the  
                //producer and consumer  
    item = buffer[out]; // take item out of the buffer  
    consume(); // consume the item  
    out = (out+1)%buffersize;  
    signal(mutex); // signal that the consumer has finished consuming  
    signal(empty); // signal producer that the buffer slots are now available for filling  
}
```

Output



```
shail@LAPTOP-NEBOGLA8: ~  
shail@LAPTOP-NEBOGLA8:~$ gcc -pthread producer_consumer.c  
shail@LAPTOP-NEBOGLA8:~$ ./a.out  
Producer 1: Insert Item 1804289383 at 0  
Producer 1: Insert Item 1681692777 at 1  
Producer 2: Insert Item 846930886 at 2  
Consumer 1: Remove Item 1804289383 from 0  
Consumer 1: Remove Item 1681692777 from 1  
Producer 1: Insert Item 1714636915 at 0  
Consumer 1: Remove Item 846930886 from 2  
Producer 3: Insert Item 424238335 at 1  
Producer 2: Insert Item 1957747793 at 2  
Consumer 2: Remove Item 1714636915 from 0  
Consumer 2: Remove Item 424238335 from 1  
Producer 3: Insert Item 719885386 at 0  
Consumer 2: Remove Item 1957747793 from 2  
Producer 2: Insert Item 1649760492 at 1  
Producer 3: Insert Item 596516649 at 2  
Consumer 3: Remove Item 719885386 from 0  
Consumer 3: Remove Item 1649760492 from 1  
Consumer 3: Remove Item 596516649 from 2  
shail@LAPTOP-NEBOGLA8:~$ |
```