End Sem OS

Nihar Patel

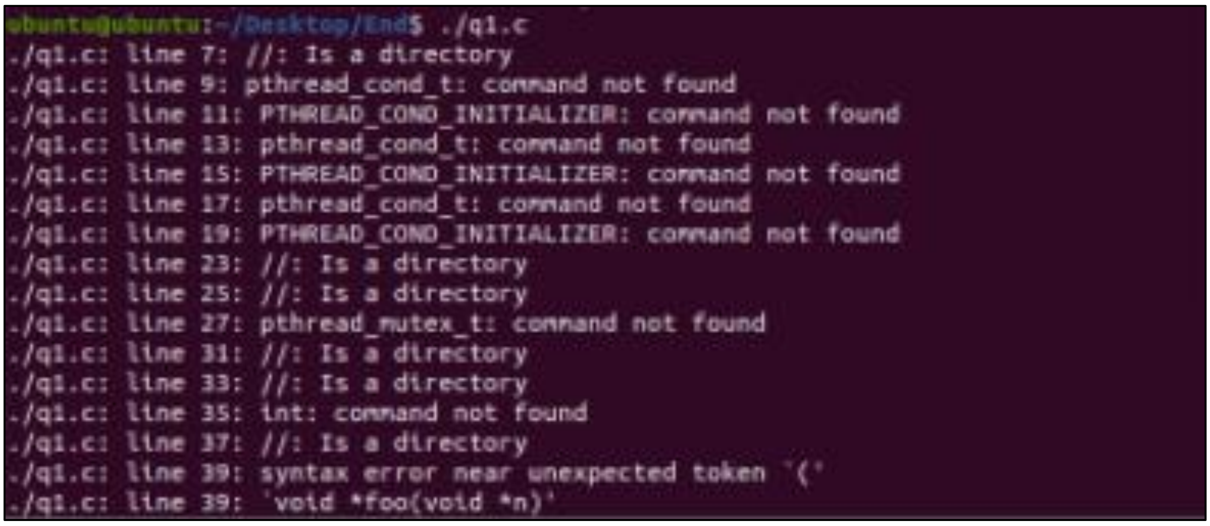
AU1940119

Description File

Q-1.

Print the following Pattern: A 1 a B 2 b C 3 c ... Y 25 y Z 26 z using multi-Threading.

To print the following pattern, we will be using 3 threads in synchronization with each other. One will be printing Capital Letters (like A B C ...) Second one will be printing Digits (like 1 2 3 …) and the Third one will be printing small letters (like a b c …) with all the 3 Threads in synchronization we will be able to generate a pattern (like A 1 a B 2 b C 3 c … Z 26 z).



```
ubuntu@ubuntu:~/Desktop/End$ ./q1.c
./q1.c: line 7: //: Is a directory
./q1.c: line 9: pthread_cond_t: command not found
./q1.c: line 11: PTHREAD_COND_INITIALIZER: command not found
./q1.c: line 13: pthread_cond_t: command not found
./q1.c: line 15: PTHREAD_COND_INITIALIZER: command not found
./q1.c: line 17: pthread_cond_t: command not found
./q1.c: line 19: PTHREAD_COND_INITIALIZER: command not found
./q1.c: line 23: //: Is a directory
./q1.c: line 25: //: Is a directory
./q1.c: line 27: pthread_mutex_t: command not found
./q1.c: line 31: //: Is a directory
./q1.c: line 33: //: Is a directory
./q1.c: line 35: int: command not found
./q1.c: line 37: //: Is a directory
./q1.c: line 39: syntax error near unexpected token `('
./q1.c: line 39: `void *foo(void *n)'
```

Q-2.

Describe the Buddy's Algorithm for Memory Allocation and Deallocation along with an example.

The buddy system is a memory allocation and management algorithm that manages memory in power of two increments.

For Buddy’s algorithm for memory allocation assume that the memory size is 2^U, and suppose a size of S is required.

Memory Allcoation:

If this condition holds true $2^{U-1} < S <= 2^U$ then it will allocate the whole block.

Else it will recursively divide the block equally and test the condition each time, and when it satisfies it will allocate the block and will exit the loop.

Memory Deallocation:

For Deallocation, we will maintain a Map with the starting address of segment as key and size of the segment as value and update it whenever an allocation request comes. Now, when a deallocation request comes, we will first check the map to see if it is a valid request or not. If so, we will then add the block to the free list of tracking blocks of their sizes. Then, we will search the free list to see if its neighbour block is free or not, if so, then we will merge the blocks and place them on the free list above them, else we will not combine and simply return that after.

Example:

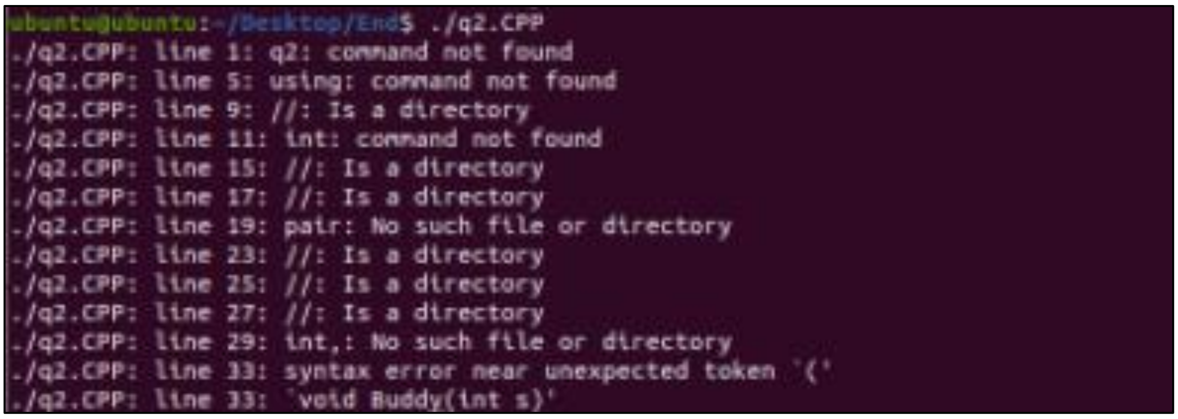
Consider a system having buddy system with physical address space 64 KB. Calculate the size of partition for 20 KB process.

64

/ \

32 32

So, it will create a partition of 32KB to store 20KB.



```
ubuntu@ubuntu:~/Desktop/End$ ./q2.CPP
./q2.CPP: line 1: q2: command not found
./q2.CPP: line 5: using: command not found
./q2.CPP: line 9: //: Is a directory
./q2.CPP: line 11: int: command not found
./q2.CPP: line 15: //: Is a directory
./q2.CPP: line 17: //: Is a directory
./q2.CPP: line 19: pair: No such file or directory
./q2.CPP: line 23: //: Is a directory
./q2.CPP: line 25: //: Is a directory
./q2.CPP: line 27: //: Is a directory
./q2.CPP: line 29: int,: No such file or directory
./q2.CPP: line 33: syntax error near unexpected token `('
./q2.CPP: line 33: `void Buddy(int s)'
```

Q-3.

Describe what is Producer Consumer Problem and its solution in detail using Semaphores and Mutex.

A semaphore (S) is an integer variable that can be accessed only through two standard operations: wait() and signal().

The wait() operation reduces the value of semaphore by 1 and the signal() operation increases its value by 1.

Problem Statement:

We have a buffer of fixed size. A producer can produce an item and can place in the buffer. While 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 consumer should not consume any item.

Solution:

We need two counting semaphores (1) Full and (2) Empty.

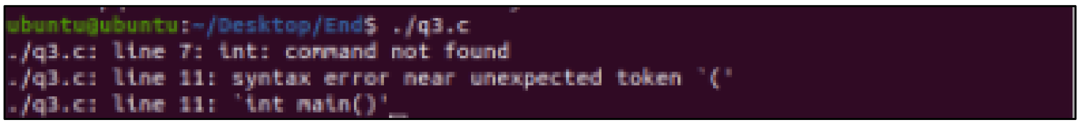
“Full” semaphore will keep track of number of items in the buffer at any given time and “Empty” semaphore will keep track of number of unoccupied slots.

Producer:

When producer produces an item then the value of “empty” is reduced by 1 because one slot will be filled now. So, the value of mutex is also reduced to prevent consumer to access the buffer. Now, the producer has placed the item and thus the value of “full” is increased by 1. So, the value of mutex is also increased by 1 because the task of producer has been completed and consumer can access the buffer.

Consumer

When consumer is removing an item from buffer, therefore the value of “full” is reduced by 1 and so, the value is mutex is also reduced so that the producer cannot access the buffer at this moment. Now, the consumer has consumed the item, thus increasing the value of “empty” by 1 and so, the value of mutex is also increased so that producer can access the buffer now.



```
ubuntu@ubuntu:~/Desktop/End$ ./q3.c
./q3.c: line 7: int: command not found
./q3.c: line 11: syntax error near unexpected token `('
./q3.c: line 11: `int main()'
```

----X----