CLIENT-SERVER DOCUMENTATION

MATT MORGIS

OBJECTIVE:

To develop a monitor as a “server” process, and successfully implement it with four clients with shared resources, and synchronize it all within the same process.

IMPLEMENTATION:

I opted to use C++, as I found it easier to build out separate classes and build this in an object-oriented way.

I use five channels in my implementation. All four clients send requests to the server using the same upChannel to request a resource, and each client has their own down channel.

I used a total of ten shared resource blocks in memory. The server keeps monitoring the up channel and responds to keep incoming request. Channels are represented as linked lists. New incoming messages are appended to the end, and the server takes them off of the top.

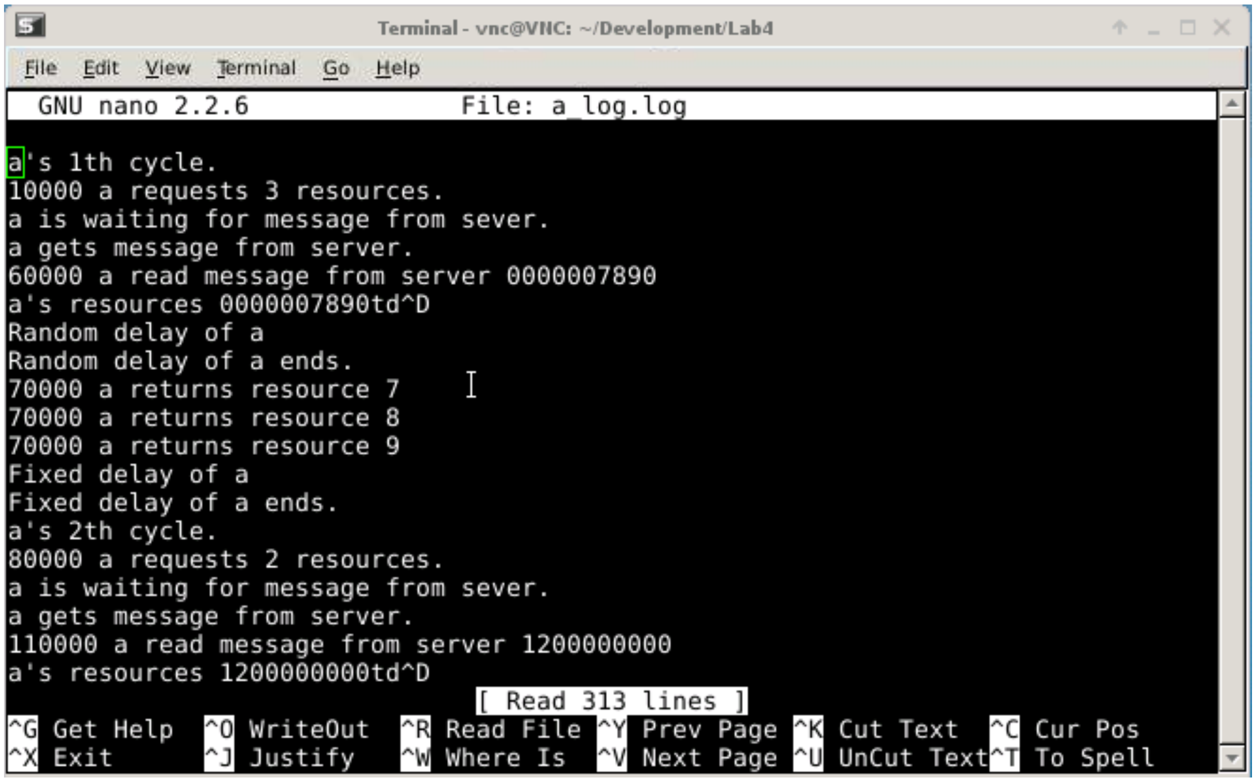
Ch - Defined in DataStructure.h

- Methods:
  - **check()** - checks the channel without modifying it. Returns the top message or NULL if channel is empty.
  - **read() -** takes the top message and removes it from the queue
  - **send() -** sends a message to the end of the channel

Once the server receives the message it will check whether it has enough resources to give. If yes, it will fulfill the request by sending a downMessage to the client using it’s corresponding channel. To simply the implementation, I used a 10-char array as the down message. If client ‘a’ receives a message of “1230000000”, it means it currently has memory blocks 1, 2, and 3 available to it.

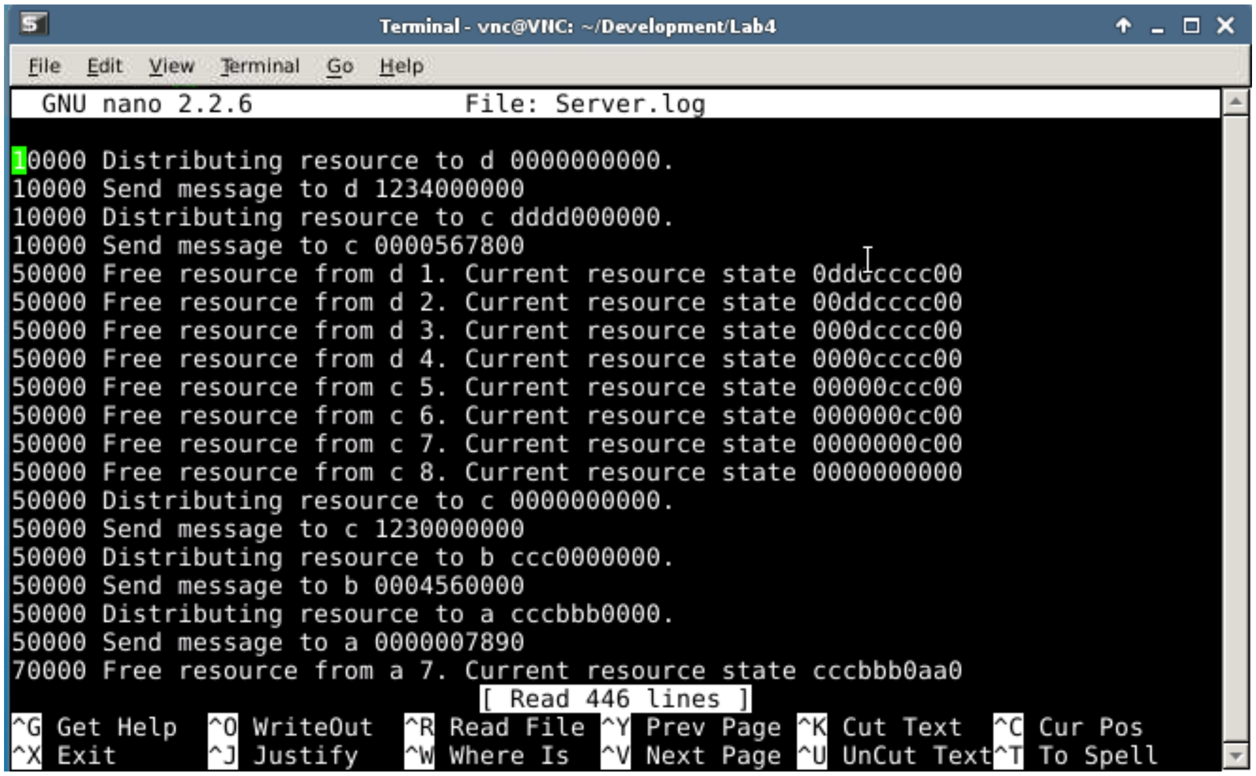
At this stage, the client will make another requesting using the up channel to release the resource after some random waiting. Requests made to the server are a 3-char array. the first char is the sender id (a, b, c, or d). The second char is the request type (Q or L - request or release), and the third char is the resource block.

Logging was the last part of my implementation. I gave the server and each client it’s own output stream to prevent any collision with the file.


Terminal - vnc@VNC: ~/Development/Lab4
File Edit View Terminal Go Help

```
  GNU nano 2.2.6              File: a_log.log

a's 1th cycle.
10000 a requests 3 resources.
a is waiting for message from sever.
a gets message from server.
60000 a read message from server 0000007890
a's resources 0000007890td^D
Random delay of a
Random delay of a ends.
70000 a returns resource 7
70000 a returns resource 8
70000 a returns resource 9
Fixed delay of a
Fixed delay of a ends.
a's 2th cycle.
80000 a requests 2 resources.
a is waiting for message from sever.
a gets message from server.
110000 a read message from server 1200000000
a's resources 1200000000td^D
                          [ Read 313 lines ]
^G Get Help   ^O WriteOut   ^R Read File ^Y Prev Page ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```


Terminal - vnc@VNC: ~/Development/Lab4
File Edit View Terminal Go Help

```
  GNU nano 2.2.6              File: Server.log

10000 Distributing resource to d 0000000000.
10000 Send message to d 1234000000
10000 Distributing resource to c dddd000000.
10000 Send message to c 0000567800
50000 Free resource from d 1. Current resource state 0dddcccc00
50000 Free resource from d 2. Current resource state 00ddcccc00
50000 Free resource from d 3. Current resource state 000dcccc00
50000 Free resource from d 4. Current resource state 0000cccc00
50000 Free resource from c 5. Current resource state 00000ccc00
50000 Free resource from c 6. Current resource state 000000cc00
50000 Free resource from c 7. Current resource state 0000000c00
50000 Free resource from c 8. Current resource state 0000000000
50000 Distributing resource to c 0000000000.
50000 Send message to c 1230000000
50000 Distributing resource to b ccc0000000.
50000 Send message to b 0004560000
50000 Distributing resource to a cccbbb0000.
50000 Send message to a 0000007890
70000 Free resource from a 7. Current resource state cccbbb0aa0
                          [ Read 446 lines ]
^G Get Help   ^O WriteOut   ^R Read File ^Y Prev Page ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

About are client ‘a’s log and the server log file. As you can see, client ‘a’ makes a request to have 3 resources. The server returns back and says ‘0000007890’. This means that buffer blocks 7, 8, and 9 are now available to client ‘a’. It waits a bit, and sends a request back to the server to to release each memory resource. Once it is free, it waits a fixed time, and makes another request.

The server log file does just the opposite. It constantly handles requests from the clients, and has all of the control to free and distribute the shared memory cells.

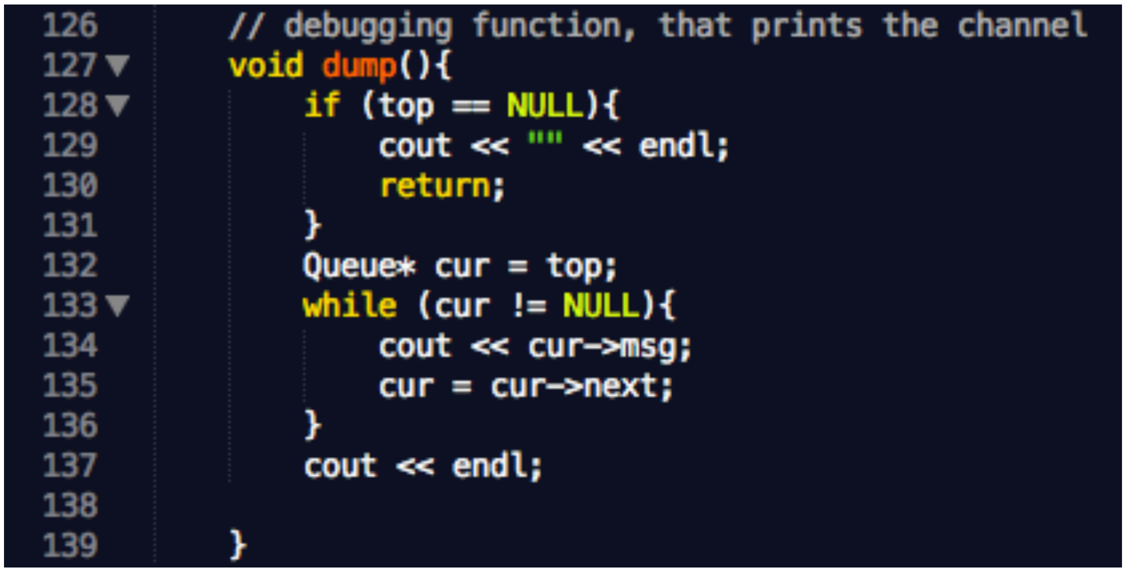
TESTING:

Testing was the hardest task for this lab.

To keep track of the shared 10-slot buffer, and I decided to keep a resource array of 10-chars. If the corresponding block of memory is free, a 0 will appear in that slot, otherwise the client with that resource was there. For example: ‘0a0a0bcccc’ means resources 1, 3, and 5 are free.

Another issue with this was controlling who had access to this data. I used the system’s <mutex> library and standard conventional mutex locks here.

Finally, I wrote a few debugging methods like the one below to constantly dump what is in the channels and print to the screen. Controlling channels shared between threads was a tricky task, and this function was used many times to step through and debug.



```
    // debugging function, that prints the channel
    void dump(){
        if (top == NULL){
            cout << "" << endl;
            return;
        }
        Queue* cur = top;
        while (cur != NULL){
            cout << cur->msg;
            cur = cur->next;
        }
        cout << endl;

    }
```