CPE464 Program #2– Chat – Grade Sheet

Grade:______________

Full Name: ________________________________ **Section: 9am noon 3pm**

Date and Time submitted: ___________________________

Normal Date: Monday, April 29th at 11:59 pm
Extra credit (+20%) date: Friday, April 26th
Last late date: Thursday, May 2nd

Code Review: indented, readable, reasonable length functions

____

Program Compiles: __________________________

Correct Executable Names (cclient, server)
and Parameters: __________________________

Responsible use of malloc(), calloc() __________________________
realloc(), and new

Basic testing:

Connect 3 cclients to their server __________________________
(if you cannot connect 3 cclients to their server – stop testing – this is a 0 grade)

Using these three clients and the student's server:

%M command testing allows all 3 clients to talk with each other
(if this fails, stop grading) __________________________

%C command multicast message (2+ destinations) __________________________

%L command (simple test, major testing of this __________________________
feature on the other side of this gradesheet)

%B command (broadcast) __________________________

%E command __________________________

Comments:

____

____

packetTesting: Monitoring via the **packetTesting** (which is a test server) program[1]:

- This program tests that the student’s program conforms to the chat protocol.
- TA: run the packetTesting server program
- TA: connect one cclient to this server with handle: **test**

cclient successfully attaches to packetTesting server: ______________________
(stop testing this section if attaching to the server fails)

1) Verify connection request - use handle: test
   - Recv len: 8 Msg Len: 8 flag: 1 (srcLen: 4 srcHandle: test)

2) Verify command: %M test2 out
   - Recv len = 19, Msg Len = 19, flag = 5 (srcLen: 4 srcHandle: test) numDst: 1 (dst#: 1 dstLen: 5 dstHandle: test2) message strlen len: 4 msg: 'out'

3) Verify command: %L
   - Recv len: 3 Msg Len: 3 flag: 10

4) Verify command: %E
   - Recv len: 3 Msg Len: 3 flag: 8

5) Combined message test (two messages back to back)
   - Two separate messages printed on the client

Did the %M commands have a “NO NULL” (means the message is not null-terminated) error?

____

Did any of the above **commands block** the client (yes/no and if yes which ones):

Comments:

Many handles test

1) TA: Connect their cclient to their server
2) TA Run: manyhandles to create 200 handles on their server
   - E.g. **manyhandles 200 localhost 55555**
3) Perform a %L on the cclient from step 1

Comments:

Used a dynamic data structure (needs to be able to grow) on server for storing the list of handles. What type was used?
malloc/realloc array, link list, tree, hashmap, other:

All code for the handle table is located in a separate .c and .h file

Yes No

(ignore this question if server fails manyHandles test)

[1] None of the commands in this section should cause the client to block. For example, after entering the command: %m 1 test, the cclient should go back to the “$:” prompt immediately.

Other testing (put an x over any that are incorrect and put in a comment somewhere):

A. cclient allows for both upper and lowercase commands (e.g. %m and %M)

B. Verify that the PDU length value is in network order (you can tell this if it prints out correctly in the packetTesting testing.)

C. Broadcast does not come back to self.

D. Allows for %M to send to itself

E. %C with multiple destinations can send to same handle twice (e.g. %C 2 test1 test1 aMessage)

F. Server cleanly handles a ^c being done on the cclient

G. Breaks up text message longer than ~200 bytes into multiple messages

H. Handles tests

   - Does not allow duplicate handles (should not allow 2 cclients with the same handle)
   - Handle removed on server after %E and ^C (so exit client, restart client with same handle then do a %L and a %M)
   - Prints out error message when sending (%M) to a non- existent handle

I. Send an empty message

J. Grep for sleep (grep sleep *.c or .cc – should not find any!)

K. Grep for pollCall(), verify that the timeout value is set to -1

L. Grep for fork(), exec(), pthread – none of these should be found (circle any that are found)

Behavior checks:

Monitor with **top** ________________________________
(e.g. should not have a tight infinite loop, should not use excessive memory)

Unusually delays ________________________________

Number of malloc()s/callac()s (grep) ________________________________

No code in the .h files ________________________________

Lines of code (wc –l *.c) or *.cpp ________________________________

Comments:

____

____

____

____