mp1: Understanding Tutor, a stand-alone monitor program

Assigned: 17 September 2020 **Due:** 8 October 2020 class time

I. OBJECTIVES:

The purpose of this assignment is to gain some experience with our LINUX system, the Stand-Alone PCs (SAPCs), and makefiles. At the same time, you will learn how to understand and modify existing C code.

1. Read, compile, run, understand, and modify a C program that runs a monitor program "tutor". We call it tutor because it mimics the Tutor monitor/debugger that we have installed on the SAPCs. You will build a LINUX version as well as a version that runs on SAPC VM.
2. Use your monitor program to explore memory locations on the PC – including the locations in which your program resides.

II. INTRODUCTION:

You should work on this assignment alone and turn in your own individual report and source code including test results. The LINUX dates on your files will determine whether the work was completed on time or not.

You should already have a cs341 logical link in your LINUX home directory. Type in command: cd cs341 to go into your homework directory. Use that subdirectory for all your work in this class. It has the right protection setup so that other students cannot access your files but instructors and graders can via group cs341-1G.

Copy files from /courses/cs341/f20/cheungr/mp1/ to your cs341 folder using:

```
cp –r /courses/cs341/f20/cheungr/mp1/ .
cd mp1
ls
```

Space between / and .

You should see all the files I supplied for this project. I'll call mp1 your project directory in what follows. Use your project directory for all work for this assignment. All files referred to below are in that project directory unless otherwise specified.

NOTE: YOU MUST USE THE DIRECTORIES AND FILE NAMES SPECIFIED SO THAT THE GRADERS AND I CAN FIND YOUR FILES AND TEST YOUR HOMEWORK, IF NEEDED. IF YOU DO NOT DO SO, YOU WILL BE PENALIZED AND YOUR OVERALL PROJECT GRADE WILL BE LESS.

III. DESCRIPTION:

You will write your own "tutor" program, which mimics Tutor. It's a tiny single user terminal monitor system, which we will compile and run both on the SAPC and on the LINUX system. In later assignments you'll add to its capabilities. For now it should accept the commands:

md <hexaddress>
(SAPC and LINUX) Memory Display
Display contents (in hex) of 16 bytes at the specified address.
(Present in the same format as the "real" Tutor program – 16 pairs of hex characters followed by 16 characters interpreting each byte as a printable character for ASCII codes 0x20 through 0x7e or a fixed '.' for all other (non-printable) ASCII codes values.)

ms <hexaddress> <new_val>
(SAPC and LINUX) Memory Set
Stores byte new_val (two hex characters) at specified address.

h <cmd>
(SAPC and LINUX) Help
Help on the specified command, for ex., "h ps", or all commands if no argument.

s
(SAPC and LINUX) Stop
Stop running your tutor and return to the regular SAPC TUTOR (or back to the shell if running on LINUX).

The files you need to build tutor are found in my directory /courses/cs341/f20/cheungr/mp1/. Most of the program has been written for you for two reasons. First, by reading the code provided, you can learn how a standard "table-driven" command processor design works. Second, providing you with a framework allows you to concentrate on the part of the programming that's important in this course.

The main program for tutor is in tutor.c. That driver calls a lexical analyzer (parser) called slex.c. The parser finds the command on the command line and calls the appropriate procedure by consulting the command table. File cmds.c contains that table and the code that implements the various commands. The file makefile builds them into an executable file.

Start by copying all files to your project directory. The makefile will build the executable files from your local directory. Read the makefile and learn how it does that.

a) Build and run the LINUX version:

```
itserver6: make tutor                    # creates an executable tutor to run on LINUX
itserver6: ./tutor                       # runs the program locally under LINUX
itserver6:
```

b) Build the SAPC version:

```
itserver6: make                          # creates a tutor.lnx to download to the SAPC
```

Use the Virtualized SAPC environment to debug the SAPC version. Log in into the tutor-vserver VM using the provided credentials. Transfer the SAPC version to tutor-vserver VM using:

```
vserver$ scp username@users.cs.umb.edu:cs341/mp1/tutor.*  .
vserver$ mtip -f  tutor.lnx              # always use board #1
```

click on the "Send Cntl-Alt-Del" button at the tutor VM to reset the SAPC VM. Hit <CR> at the tutor-vserver VM to get the tutor prompt. Then download the tutor program to the SAPC VM and run it:

```
Tutor> ~d                    # download tutor.lnx
Tutor> go 100100             # start your program on SAPC VM
```

IV. MODIFY PROGRAM:

Read tutor.c, slex.h, slex.c, cmds.c, and the makefile in order to understand the structure of the tutor program. (We will be going over that structure in lecture.) When you're ready,

A. edit cmds.c to implement the required new commands (md, ms and h)

Use sscanf to convert the argument passed md or ms from a character string of hex digits to an integer. Use the return value from sscanf to check if it really worked. Study the makefile -- be sure you understand how it works. Study the output of make as it runs -- what are the program-building steps?

Sample Output

```
itserver6:  ./tutor
   command        help message
    md            Memory display: MD <addr>
    ms            Memory set: MS <addr> <value>
    h             Help: H <command>
    s             Stop

LINUX-tutor> md 10000
00010000 7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00 .ELF.....
LINUX-tutor>
```

B. play with the tutor programs that you've built

This is a central part of the assignment. If your code is elegant and perfect but you do not test it, exercise it, and think about the results it generates you will have learned less than half of what you should - so will get less than half the credit.

You must create a file discussion.txt. Write clear, grammatically correct English. Consider showing drafts to your friends (or your English teacher) to find out if your writing is clear.

V. DISCUSSIONS:

In your discussion.txt file, answer all of the following questions:

(1) Describe how you tested your code. Try the following experiments. Write several sentences about what you found or learned working on each, and on any other similar kinds of questions that occur to you.

(2) What happens if you call md for an address that does not correspond to a physical memory address? What if you write to an address that's part of your tutor code or an address in ROM area of memory? Do these questions have the same answers on LINUX and the SAPC?

(3) Read the makefile to see where it puts the symbol table (nm output) for your tutor code. Use that symbol table to figure out:

(a) the address for test global variable xyz, which has value 6. Use tutor with that address to verify the value in memory.

(b) the address of the pointer variable pxyz. This address should be close to the one you determined in a, but not equal to it, since pxyz is the next variable in memory after xyz. Find the value of pxyz in memory. This should be equal to the address you found in (a) because of the initialization of this variable to &xyz. Note that you need to get 4 bytes of data for the value here. See IMPORTANT NOTE just below.

(c) the address of the cmds array. Use this address to determine the very first pointer in the array, the string pointer to "md". Then find the 'm' and 'd' and terminating null of this string.

(d) change the stop command from 's' to 'x' while the tutor program is running. Can you change the tutor prompt the same way?

IMPORTANT NOTE: When you try to access a 32-bit value (a pointer for example) in SAPC memory via Tutor, you need to reverse the displayed byte order, because of the little-endian representation of numbers in the Intel architecture.

Example: Suppose a pointer (or any 32-bit numeric quantity) is at address 0x100200 with value 0x00100abc. It would show up as:

```
Tutor> md 100200
100200   bc 0a 10 00
```

because of "little-endian" storage of numbers in memory in the Intel architecture. See the lecture notes.

To help with displays, there is an "mdd" command (memory-display-doubleword) in the “real” Tutor monitor. This command reorders the bytes for you and displays four bytes at a time as a doubleword value:

```
Tutor> mdd 100200      (for same memory contents as with md command above)
100200   00100abc ...
```

(4) Read the nm output to determine where in memory the code resides, on SAPC and LINUX. Hint: code symbols are marked t or T. Similarly determine where the data (set of variables) resides.

(5) Try to change the code itself so that tutor crashes (any random change that actually takes effect should do this). What happens on SAPC? on LINUX?

(6) You can't find the program stack using the nm output, but you can find it by looking at the stack pointer, called ESP on the SAPC and LINUX. Record your observations. Use "i reg" (info on registers) to see sp in gdb and "rd" to see registers in Tutor.

(7) What other interesting things have you tried?

(8) (optional) More questions you should consider answering. What did you learn from this project? Was it worth the time it took? What parts were the hardest, what parts the easiest, what parts most surprising, most interesting?

What idiosyncrasies of C or LINUX or the SAPC or our installation slowed you down or helped you out? How might the assignment be improved?

VI. TURN-IN FOR GRADING:

You should turn in a hard copy of the typescript file at the beginning of the class when it is due. (Use this procedure for all future mp assignments.)

To generate a typescript file, execute the LINUX command "script". This will start recording all terminal commands and responses into a file "typescript". At a minimum, you must show all of the following in your typescript file:

- ls –al to show your user name
- cat your discussion.txt file
- cat the sources of all of your .c files
- execution of make clean
- execution of make to create all program executable files
- a sample run of each test case that is required in the assignment

Leave your discussion.txt file, typescript file, .c files, and a working version of tutor and tutor.lnx in your mp1 directory. Do not modify any of these files after printing the typescript file that you turn in. Please remember to put your name on the first page. The graders or I may login to this directory to check the LINUX dates on these files and to run them or test them ourselves. We may also recompile your cmds.c with another main program as an extra test case. Please remember to close your typescript file by issuing the command "exit".

A rubric sheet for grading mp1 is included. In the event that you are unable to correctly complete the entire assignment by the due date, do not remove the work you were able to accomplish. Submit what you have completed - partial credit is always better than none.