CS341 Bob Wilson

MP1: Tutor, a stand-alone debug monitor
Due: Start of Class 6

The purpose of this assignment is to gain some experience with our ulab system, VMWare Tutor, and our makefiles for building on various systems.

o Read, understand, compile, run, and modify a C program that runs a monitor program "PC-Tutor". We call it tutor because it mimics the Tutor monitor/debugger that we have installed on the SAPCs.
o Use your monitor program to explore memory locations on the PC - including the locations in which your program resides.

To get started, always login to users@cs.umb.edu and rlogin to ulab. You need to enter your password for both logins. The first time you run on ulab, edit your .cshrc file using whatever UNIX editor you prefer. I prefer pico. Find the line that begins with "module", i.e.

```
module load standard
```

Add " ulab" (w/o quotes) to the end of that line.
Save and close the editor.
Logout of ulab and rlogin to ulab again to make the change take effect.

You should already have a cs341 subdirectory in your UNIX home directory. It is a soft link to your subdirectory in the course 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 the instructor and graders can via group cs341-1G.

Change directory (cd) to your cs341 directory.
Make a subdirectory named "test".
Copy ~bobw/cs341/examples/lecture02/* to your cs341/test directory.
Compile and run the sample test program on ulab to confirm the tools work.

```
gcc -o test test.c
./test
(Follow the instructions)
```

Then, you can start on your mp1 assignment.
You should work on this assignment alone and turn in your own individual report and source code including a typescript file of test results. The dates on your UNIX files will determine whether the work was completed on time or not.

Create an mp1 subdirectory on your cs341 directory. Note that case matters to UNIX so don't make your subdirectory Mp1 or MP1 since scripts won't go looking for those variations. 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.

The files you need to build tutor are found in my directory ~bobw/cs341/mp1.

Copy the contents of that directory to your mp1 directory. 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.

Read the makefile in your mp1 directory to understand how it works. Execute "make clean" to start your builds. To build an executable to run on ulab itself, use "make tutor". To build an executable (.lnx) to run on our virtual embedded systems, use "make".

You will write your own additions to PC-Tutor, which mimics the real Tutor on our virtual embedded systems. It's a tiny single user command line debug monitor, which we will compile and run both on the ulab UNIX system and on a virtual embedded system. Study Tutor on the virtual embedded system to duplicate its functionality.

In later assignments you'll add to its capabilities. For mp1, it should accept the commands:

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

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

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

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.

From the ulab console:

```
ulab% make tutor   # creates an executable tutor to run on UNIX
ulab% tutor        # runs the program locally under UNIX
ulab%              # UNIX-Tutor exits when you enter stop (s).
```

From the ulab console:

```
ulab% make          # creates a tutor.lnx to download to the VM
```

Start your VMWare virtual machine and power up both the tutor-vserver and tutor VMs. Select tutor-vserver tab. Now, you will be working under the Linux OS in tutor-vserver.

After Linux boot, login to Linux with user name "tuser" and password "cs444". When you get the Linux prompt, use scp to transfer the .lnx file that you built on ulab:

```
Prompt: scp username@users.cs.umb.edu:cs341/mp1/*.lnx .
      Enter your UNIX password when prompted
      You should see the status of the downloaded files.

Prompt: mtip -f tutor.lnx
<CR>                                (to see if Tutor VM condition looks ok)
or if Tutor VM doesn't respond properly: Switch to tutor tab and use
tutor reset command
Switch back to tutor-vserver tab
<CR>
Tutor> ~d                           (download PC-Tutor to tutor VM)
Tutor> go 100100                    (starts your program on Tutor VM)
```

UNIX-Tutor and PC-Tutor should run on their respective machines, but won't do much except a stub for memory display (md) and stop (s) until you add your code.

Read tutor.c, slex.h, slex.c, cmds.c, and the makefile in order to understand the structure of the tutor program.
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

```
blade57(2)% tutor      cmd    help message
md    Memory display: MD <addr>       ms
Memory set: MS <addr> <value>          h
Help: H <command>        s    Stop
UNIX-tutor> md 10000
00010000    7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00 .ELF............
UNIX-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, grammatical English. Consider showing drafts to your friends (or your English teacher) to find out if your writing is clear. Run discussion.txt through the unix spell utility.

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 UNIX 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 embedded system VM 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 0x100200with 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 UNIX. 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 UNIX?

(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 sp on the Sparc (ulab's CPU). 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 hardest, what parts easiest, what parts most surprising, most interesting? What idiosyncrasies of C or unix or the VM or our installation slowed you down or helped you out? How might the assignment be improved?

TURN-IN FOR GRADING:

You should turn in a hard copy of a 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 UNIX command “script”. This will start recording all terminal commands and responses into a file “typescript”. At minimum, you must show all of the following in your typescript file:

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. The graders or I may login to this directory to check the unix 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.

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.