CSC501 Spring 2009

Lab 0: Getting Acquainted with XINU

Due: Jan. 20th 2009

1. Objectives

The objectives of this introductory lab are to familiarize you with the process of compiling and running XINU, the tools involved, and the run-time environment and segment layout.

2. Readings

1. *Lab setup guide*
2. The Intel manuals which are available on the main lab web page, as well as on the Intel web site.
3. AT&T assembly information specific to the gnu assembler is available here as a wikibook.
4. Any man pages/manuals you discover that you need.

Note: The Intel document is hundreds of pages long and there is no need to print it. Please do not print it!

3. What to do

You will be using the lab0-spring09.tgz you have downloaded and compiled by following the lab setup guide. And you are asked to write several XINU functions that perform the following:

1. ***long net2hostl_asm (long param)***
   Convert the parameter *param* from network byte order (always Big Endian) to host byte order. The code for this function should be entirely written in x86 assembly. You should ***not use*** in-line assembly (i.e., do not use asm("...")). You can assume that the size of long is 4 bytes and the byte order of the host machine is Little Endian. To investigate the assembly code generated by the compiler, you can use the tool ***objdump -d <___.o>*** to disassemble an object file. The object files reside in the /compile directory within the main Xinu directory. You can also see some of the *.S files in the /sys directory for reference.

2. ***void printsegaddress()***
   Print the address of the end of the text, data, and bss segments of the Xinu OS. Also print the 4 bytes (in hexadecimal) preceding the end of the three segment boundaries, and similarly for the 4 bytes following the segment boundaries. This function can be written in C.

3. ***void printtos()***
   Print the address of the top of the run-time stack for whichever process you are currently in. In addition, print the contents of up to five stack locations at and below the top of the stack (the five or fewer items that have been the most recently pushed, if any). Remember that stack elements are 32 bits wide, and be careful to perform pointer arithmetic correctly. Also note that there are local variables and arguments on the stack, among other things. See the hints given for #4 below, especially on stacktrace.c and proc.h. Your function can be written entirely in C, or you can use in-line assembly if you prefer.

4. ***void printprocstks()***
   For each existing process, print the *stack base*, *stack size*,*stack limit*, and *stack pointer*. Also, for each process, include the *process name* and the *process id*. An example output might look something like:

```
Proc [prnull]. Pid = 0.
   Stack: Base  = 4095996
          Len   = 0
          Limit = 4091904
          StkPtr  = 4095820

Proc [main]. Pid = 49.
   Stack: Base  = 4091896
          Len   = 4096
          Limit = 4087804
          StkPtr  = 4091872
```

To help you do this, please look into **proc.h** in the **h/** directory. Note the *proctab[]* array that holds all processes. Also, note that the *pesp* member of the *pentry* structure holds the *saved* stack pointer. Therefore, the *currently* executing process has a stack pointer that is different from the value of this variable. In order to help you get the stack pointer of the currently executing process, carefully study the **stacktrace.c** file in the **sys/**directory. The register *%esp* holds the current stack pointer. You can use in-line assembly (i.e., asm("...")) to do this part.

Implement this lab as a set of functions that can be called from main(). Each function should reside in a separate file in the *sys* directory, and should be incorporated into the Makefile. The files should be named after the functions they are implementing with C files having the**.c** extension and the assembly files having the**.S** extension. So, for example, the file that will hold ***void printsegaddress()*** should be named ***printsegaddress.c***; and the file that will hold ***long net2hostl_asm(long param)*** should be named ***net2hostl_asm.S***. If you require a header file, please name it **lab0.h**. Note: as you create new files, you may need to update the Makefile (located in the compile/ directory) to configure it to compile your files correctly. Just look at what is done for the existing files (e.g., main.c) to see what you have to do.

---

4. Additional Questions

Write your answers to the following questions in a file named **Lab0Answers.txt**(in simple text). Please place this file in the **sys/** directory and turn it in, along with the above programming assignment.

1. Assuming the XINU text begins at address 0x0, draw a rough diagram of XINU's memory layout with addresses derived from your experimental measurements. Include the information you uncovered from running your version of ***printsegaddress()*** and ***printprocstks().***
2. Draw a rough layout of the stack for a user process based on your measurements. Show the relative locations of local variables and arguments. If you found other information about the stack, include it on the diagram. What other items would be found on a stack that are not included in your diagram? Include the information you uncovered from running your version of ***printtos().***
3. Briefly describe the mov instruction in the x86.
4. Briefly describe the push, pusha, pop, and popa instructions in the x86.

Turn-in Instructions

Electronic turn-in instructions:

i) go to the csc501-lab0/compile directory and do "make clean".

ii) go to the directory of which your csc501-lab0 directory is a subdirectory (NOTE: please do not rename csc501-lab0, or any of its subdirectories.)

e.g., if /home/csc501/csc501-lab0 is your directory structure, goto /homes/csc501/

iii) create a subdirectory TMP (under the directory csc501-lab0) and copy all the files you have modified/written, both .c files and .h files into the directory.

iv) compress the csc501-lab0 directory into a tgz file and use Wolfware's Submit Assignment facility. Please only upload one tgz file.

```
tar czf csc501-lab0.tgz csc501-lab0
```

You can write code in main.c to test your procedures, but please note that when we test your programs we will replace the main.c file! Therefore, do not put any functionality in the main.c file.

Also, ALL debugging output should be turned off before you submit your code.

Back to the CSC501 web page