ECS150
WQ2016
Programming Assignment 2
Process management in MINIX
Due February 23, 2016, 4:30 PM, via SmartSite

These problems ask you to make very small modifications to Minix You will modify the kernel, recompile and assemble it using MAKE to link the various executable files, and then produce a new kernel. All of this is quite straightforward, as Yang, Apoorva and Parisa have explained in a posting. Most of the code you will deal with is in proc.c; most of the data structures you need to access or modify are in proc.h.

I want you to work with MINIX 3.0.1, the version described in the Tanenaum textbook

Each of these programs requires only trivial modifications to Minix. The key is locating the source code that should be modified; there are numerous acceptable solutions

To receive credit for you solutions, you will submit a new kernel, a listing of the programs you wish to use to test the new Minix, and a printout of the changes you have made to the source code. All this must be uploaded to SmartSite by the due date. You will make an appointment to demonstrate your new version of Minix with the test programs you have submitted, the demos (interactive) to take place towards the middle or end of the week of Feb. 22.. During the demonstration Parisa, Apoorva, Yang or I are likely to ask you to run the revised kernel with C programs that you have created before the demo. By running these programs, you will exercise the modifications you made. We might ask you questions about your revised kernel.

Please, have your revised MINIX compiled and ready to run prior to the demo. There will not be time for you to edit, compile or reinstall the kernel during the demo.

The key is to come up with interesting and convincing demos.

Note: In some of the problems you are asked to add fields to the process table, do not make your new field the first one. Why?

As important as revising the kernel to include the features requested in these problems is convincing us in the demo that you have made the changes. As I have said on numerous occasions, you will be engaged in demos for much of your career – and you will want them to be convincing. In most cases for this assignment, you should run programs from the MINIX shell that demonstrate the features you have added, likely programs that create a number of processes some of which run in background. You will use the function keys (F1 – F12) to print out kernel data structures, some of which are already there but others of which you will add or modify. We will post advice on how

to use these function keys, including how to modify them to access data structures you add to the kernel. You should try out all of these function keys as they are very helpful in debugging the kernel and demonstrating its actions.

1. Modify the MNIX kernel to keep track of the number of messages a process or server or task *i* sends to a process or server or task *j*. Keep track of these messages in a matrix you will add to the kernel. Use an appropriate "function" key to display this matrix. To keep the matrix of manageable size, only display the rows or columns for active processes or tasks.

   Here is how you might demonstrate your new kernel. Create a few user processes that run programs that cause many messages to be sent, say to a server or cause interrupts that the kernel converts to messages; avoid having the programs you are running do any printing, as this will get in the way. Be prepared to explain your results, e.g., how many messages do you expect to observe being sent between user processes, between user processes and the servers, between the PM and FS, etc.

2. Modify the MINIX scheduler so that newly created user processes are run ahead of older user processes. For your demo, I suggest you run a program that creates a number of processes each executing an infinite loop (with no printing or system calls) but whose creation times are separated by a few seconds. By hitting the appropriate "function" key you can show that the earlier-created processes get more cpu time to begin with but the later processes catch up after a few seconds.

3. Modify the MINIX kernel to keep track of the system calls invoked by each user process, recording for each system call the system call number and the number of times it is invoked. One way to record this data is in the process table slot for each user process, thus adding a field to the struct declared starting on line 5516; you might have the field you add be the last one in proc. Display this data by depressing an appropriate function key. Again, your demo can involve running processes that make many system calls but do no printing. Be prepared to explain your demo.

   Hint: You might hypothesize that the `call_nr` argument to `sys_call` is the system call number but you would be wrong. Look at the `m_type` field of the message; the `m_source` field might also be helpful. See if you can record error returns from system calls.

   As an aside, this exercise is a start towards incorporating into the kernel a capability to monitor the actions of users through monitoring their processes.