Simon Wroblewski, Riley Persily, Andrew Arteaga

Operating Systems Presentation

Dr. Johnson

May 2, 2019

Kernel Modification

Our group’s kernel project has been a long and frustrating journey. From Kernel panics to crashed computers, the obstacles have been anything but in short supply. Although the challenges persisted through the projects entirety, we found it to be a great learning experience, largely due to these challenges. Addressing these problems forced us to research them, giving us a deeper understanding of the Linux operating system and its core components (i.e. the kernel). Through our combined efforts, we were able to overcome these challenges and produce a modified linux module with functionality similar to our original objective.

When the project was in it’s preliminary stages, the instructions on building the kernel were slightly ambiguous and required a large amount of independent research. Our group started the project with these instructions and after a couple hours of research and countless crashes of our virtual machine, we were able to build the new kernel. Much of the process was found on the wiki.ubuntu.com website (url: <https://wiki.ubuntu.com/Kernel/BuildYourOwnKernel>) given to us in Dr. Johnson’s instructions.

Later on in the semester, Dr. Johnson was able to build the kernel correctly. After doing so, he updated the instructions to be more comprehensive and chronicled his process in a step-by-step fashion. Once the new instructions were posted, our group decided to rebuild our kernel using Dr. Johnson’s two for three reasons. One reason was because one of our group members was unable to build the kernel using the previous method and we wanted to keep our process standardized. The other reason was because we wanted to be able to ask Dr. Johnson questions about the process so it would be in our favor to model our process after his.

After troubleshooting the new method, we were able to find the correct way to use “make” to install the kernel and its modules. At this stage, our group was finally ready to attempt the modification of our chosen module. The original idea for our project was to modify the “top” module so that in addition to listing the active processes, you could also reassign any specified process to any desired priority. The “top” module was not native to the Linux kernel we built so we had to search online for it’s source code. We found a tar file (<http://procps.sourceforge.net/download.html>) but ended up choosing not to use it because we feared it would corrupt our new kernel.

At this point in our project we were all feeling pretty lost and defeated but we were determined to not lose hope. Rather than giving up, we put our heads together in hopes of pivoting our project in a new direction. After some brainstorming, we came up with idea for a new command called “randice” which was derived from the renice command. The renice command identifies a process and reassigns its priority to the specified priority in the command. Our new randice command (random + renice) would accomplish the same thing as renice, however, regardless of the priority to specify for the process, the priority will be chosen at random from one through ten.

Luckily, we were able to find the renice module’s source code on our kernel in the linux-utils folder. We looked through the C code and spent some time figuring out how it accomplished it’s functionality. We decided it was best to copy the code from renice into a new file called randice.c and add an option “-r” to the option list which applies a random function to “prio” (the priority the process is set to) and returns the result. We began adding a new syscall by creating a new directory for randice and placing our modified renice file in this new directory. In order for the kernel to know that we had added a syscall and included it in the new build we added a few things. First we added a Makefile in our new directory, then we added randice to the system call table, and then added to the header file. Once this was finished we recompiled and then installed the modules and the modified kernel. Everything appeared to be going well until we restarted the VM to allow the changes to take effect, at which point the VM crashed on startup. It appeared that somewhere in our steps we affected the boot args or removed a module essential to startup. At this point we restored a screenshot from before we began our modification process where the original kernel was built.

