Nathaniel Fenoglio

1001634050

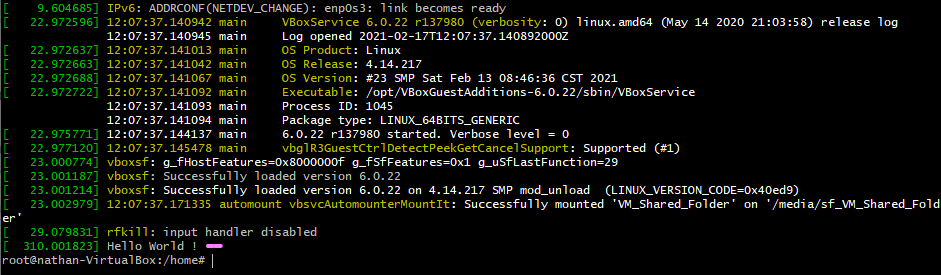
Lemuel Rivera

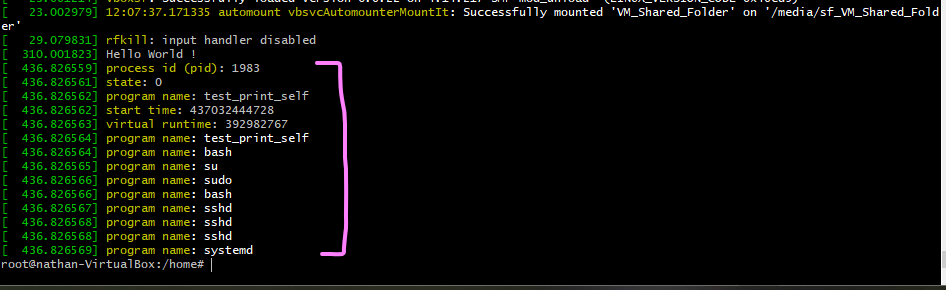
1000842851

**General Process Outline for Project 1 Process and Thread Including Problems and Solutions Encountered Along the Way**

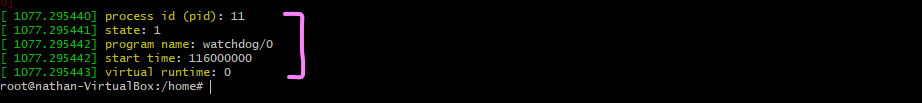
* We used Virtual Box and installed Ubuntu 18.04.5.
* We chose to install linux version 4.14.127 as the second kernel image to boot to. We were not sure if it was a good idea to install release version 4 instead of release version 5, because of the discussion about the greater size of a later release. Ultimately, we went with version 4.
* Went through the make menuconfig process to generate the .config file.
* Executed make -j4 command:
  + The time that it took to recompile the kernel was a challenge in that we were not able to get feedback from changes made for around 3 hours. We took the advice to adjust the settings in Virtual Box to allow 4 cores to be used and this cut down the kernel compiling time.
* Executed the make modules, make modules\_install, and make install commands.
* Created initramdisk successfully and ran update-initramfs:
  + We learned that we needed to have sudo permission when executing some processes to compile the kernel. Otherwise, certain files would not be built, and we would receive errors.
  + Changing to sudo, from the beginning, solved the problem and we had no issue with successfully compiling. In short, we worked as the root users for most of the project.
* Changed the grub configuration file to allow to select which boot image to boot to in order to select the different kernel version.
* Rebooted to have the changes take place and booted into linux version 4.14.127 kernel.
* Created directory my\_source in the new kernel in the top-level directory where we added the 3 C programs to make the system calls.
  + We learned that when writing the system calls, the number at the end of the SYSCALL\_DEFINE# macros indicated the number of parameters that would be passed to the system call and that the syntax is (systemcall\_name, parameter\_1\_type, parameter\_1\_name, parameter\_2\_type, parameter\_2\_name, etc)
  + We learned about the task\_struct structure and the “current” macro to get a pointer to the task\_struct of the currently running process. We learned that the tasks are connected as a double linked list and each process has a pointer to its parent process allowing us to traverse the list to the initial process.
  + We learned the different attributes to find information on the various processes accessed through the task\_struct structure
    - pid - process id
    - state - state
    - comm - program name
    - real\_start\_time - start time
    - virtual runtime - se.vruntime
  + We were able to search for a process id provided as a command line argument by accessing the pid attribute and traversing the linked list back to the initial process and comparing the process ids for a match.
* Added Makefile inside of my\_source directory to create the object files to be built from the 3 C programs.
  + When adding our second system call, we had an error caused by a typo in our Makefile syntax.
  + We fixed the syntax to be obj-y := sys\_helloworld.o sys\_print\_self.o sys\_print\_other.o changing the object files to be built to be all on the same line as the target to fix the error.
* Edited Makefile in the top-level directory to also search in the new my\_source directory when compiling the kernel.
* Added the system call entries to the arch/x86/entry/syscalls/syscall\_64.tbl file to include our new system calls.
  + We initially tried using “common” and the “\_\_x64\_” prefix, but that led to several complications.
  + Instead, we changed “common” to “64” and dropped the “\_\_x64\_” prefix to avoid errors.
  + We think that the need to change the syntax was due to having to install a version 4 kernel.
* Added the asmlinkage lines to the include/linux/syscalls.h file
* Execute make -j4, make modules, make modules\_install, and make install commands and rebooted every time that we added a new system call or made any changes to kernel level code in order to have the changes incorporate into the kernel build and take effect.
* Wrote user-level programs in home directory, outside of kernel, in user space. Used C function syscall and the system call # of the various system calls to trap to the kernel from user space.

Below are screenshots of the output of the 3 system calls:

test\_syscall.c

test\_print\_self.c

test\_print\_other.c

(process id 11 entered as command line argument)

Honor Code

I pledge, on my honor, to uphold UT Arlington’s tradition of academic integrity, a tradition that values hard work and honest.

effort in the pursuit of academic excellence. I promise that I will submit only work that I personally create or contribute to

group collaborations, and I will appropriately reference any work from other sources. I will follow the highest standards of

integrity and uphold the spirit of the Honor Code.

Nathaniel Fenoglio

Lemuel Rivera