ELEC 424 - Assignment 1: Kernel Hacking

100 points

Overview

You will create your very own system call and add it to the Linux kernel. This is kernel hacking, congratulations! Then you will compile your customized kernel and test it within a Linux environment using CLEAR (not the Raspberry Pi). The workflow will be to edit the kernel to add a system call, develop a C program that calls the new system call, and then use user-mode Linux (UML) to run the modified kernel and C test program.

Rubric

1. **(56 points) In person or submitted recorded video demonstration of functionality. In person demonstrations can be done during instructor or TA office hours.**
    a. Do not implement your counter in user space (e.g. don't have your counter in your C test file - that will result in a serious point reduction (let's say -80 points) because that's not the point of this assignment). The counter **must** be in the syscall function.
    b. Each of the following is worth 7 points. If doing a recorded video, demonstrate items iii to viii sequentially and provide narration stating which item you are testing at each part of the video.
        i. netid_test produces print statement about calling syscall
        ii. syscall prints counter value to terminal/console (this happens by default with printk with the default message priority)
        iii. syscall increments counter value when first input argument is 0 and the count is incremented by the amount specified by the second argument;
        iv. syscall increments counter by when 1 when test script given no input arguments (test script will pass 0 and 1 as first and second arguments, respectively, to syscall in this case)
        v. syscall correctly increments counter if the first argument is 0, the second is a positive number, and the third argument is another positive number
        vi. syscall resets counter value to 0 for first input argument of 1 (regardless of any second and third input arguments)
        vii. syscall prints message about input being invalid if first input argument not 0 or 1
        viii. dmesg shows previous syscall printk() outputs (this verifies that a system call was used)
    c. Recorded video demonstration submission

i. If you prefer to submit a video demo, please upload it to the Box folder that I shared with your Rice email. It is your responsibility to let me know 48 hours before the deadline if you do not have access to the folder, otherwise it is not my responsibility if you have to submit late. Video files submitted must be named as the following (replacing words as appropriate): ELEC424_Assignment1_Firstname_Lastname_netID
    1. Note: I have set the privacy to where I believe no one can view each others' videos, however you will be able to see who has uploaded and who has not. If this is a concern for you, please meet with us to do an in person demonstration instead.

2. **(44 points) Submission of relevant commented code files and report to Canvas**
    a. (5 points) Code attempts to achieve objectives/requirements stated in instructions
    b. (10 points) Code commented in detail
    c. (4 points) Three files must be submitted in their source form (.tbl, .c) - not a PDF
        i. arch/x86/entry/syscalls/**syscall_64.tbl**
        ii. kernel/**sys.c**
        iii. **Netid_test.c**
    d. (25 points) PDF report that includes:
        i. (1 point) Title of project
        ii. (1 point) Name
        iii. (5 points) 1 paragraph (at least 4 sentences) describing the goal of the project, what files you modified, and what your approach was to the problem.
        iv. (5 points) A 2nd paragraph (at least 4 sentences) describing what you found challenging, what you learned, and what you think would be another interesting system call to invent.
        v. (5 points) After completing the coding and demonstration parts of the assignment, I grant permission for you to ask chatGPT to complete an assignment similar to this one. Ask chatGPT how to make a counter system call (no need to give it all the details, you can ask for a simple counter). Write a 3rd paragraph (at least 4 sentences) describing what chatGPT tells you, how accurate you think it is, and how it seems to differ from your solution.
            1. You must include a shared link to your chatGPT chat (instructions) and cite chatGPT according to the standard of this document.
        vi. (4 points) Include a screenshot showing a portion or all of your system call code, and another screenshot showing a portion or all of your test file code.

vii. (2 points) Include two screenshots of terminal output showing your counter working on one test or two.
viii. (2 points) All screenshots in the report must include a figure label with a short description.

Note: Be sure that you are using linux kernel version 6.5.5 (in other words, follow the instructions on the next pages). Otherwise, we may not grant points for the demonstration.

Steps

1. Open up a terminal and log into CLEAR
2. Make a folder for this assignment and “cd” into that directory
3. Download the Linux kernel source code, version 6.5.5 (you *must* use this version for this assignment) using the following command:
    - wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.5.5.tar.xz
4. Try the “ls” command - you should see something like “linux-6.5.5.tar.xz” - this is a compressed folder, we need to uncompress it using tar. Use the following command to uncompress it:
    - tar -xvf linux-6.5.5.tar.xz
    - Note: You can be fancy and lazy instead by running “tar -xvf $(!!)” after running ls, where $(!!) will insert the output of the last command you ran
5. Change your directory to the uncompressed folder (mine was linux-6.5.5)
6. Add your system call to the system call table in the Linux source code. Do so by editing the `arch/x86/entry/syscalls/syscall_64.tbl` file. Look for the end of the first range of system calls - they should end somewhere around 450. Let’s say the final system call was:

    ```
    443   common     joe_cool   sys_joe_cool
    ```

    You would want to add yours on the next row as something like:

    ```
    444   common     netid      sys_netid
    ```

    Use your actual netid. Also be sure to increment whatever the final system call number is by 1 for your own system call. In my example, we went from 443 to 444, but yours will be a different number. NOTE: There are tabs between the items (e.g. between `common` and `netid`), not spaces. So use tabs to make your items line up with the previous system calls.
7. Now we need to write our system call code and include that code in a file that will be included in kernel compilation. I recommend editing `kernel/sys.c` to include your

new syscall function. Go to the absolute end of that file and include your function there. The requirements for this function are that it:

- Takes in 3 input arguments of type: boolean, float, float
- Increments a counter when the first input argument is 0; the second input (float) argument specifies how much the counter should increase by; the third argument (float) specifies what number the counter should start at
    i. This counter should start at zero (before being incremented) unless the third argument specifies something else
    ii. This will require the counter to be a static variable - look around online, talk with a friend, or talk with me if you are unsure of how static variables work
- Resets the counter to 0 when the first input argument (bool) is 1 (second and third input arguments don't matter in this case)
    i. Counter will not be incremented in this instance
- Prints out the current count using printk() [with linefeed, otherwise you might not see it until later - the kernel is fun like that] in both cases of first input 0 and 1
- Has an error condition if the first input is not 0 or 1
    i. Should use printk() to say a string like "input was invalid" [with linefeed, otherwise you might not see it until later - again the kernel is fun like that]
    ii. Should use "return -EINVAL" [EINVAL is a macro defined in the kernel - it means invalid argument]

8. Enter "make ARCH=um menuconfig"
    - This is how we configure the linux kernel before it is compiled
    - ARCH=um is specifying user-mode linux (UML), explained later in this document
    - Use the right arrow key multiple times to select Save on the bottom and then hit enter
    - Hit enter again to save config, then hit enter again to return to the main menu
    - Hit escape twice in a row to exit the configuration
9. Enter "make -j20 ARCH=um linux" [see below if compilation fails]
    - Compilation may fail because of an error mentioning random. If so, you will need to edit the file it mentions (util.c):
        i. Comment out an include line (#include <sys/random.h>)
        ii. Change the os_getrandom() function's return line to be "return 0;"
    - You are compiling the Linux kernel!
    - -j20 has the command run in parallel
10. If it compiled, you shouldn't see any error messages.
    - If it didn't compile, look at your counter function code and syscall table code, then try to figure out what went wrong.
11. Work on a C file to test your system call, which will just be a main function that uses syscall with your syscall number to invoke your custom syscall. Save this c file in the

root of the linux source code that you've compiled. Use the following header files for the file, and see the next point for the requirements for the program.

- `#include <syscall.h>`
- `#include <unistd.h>`
- `#include <stdio.h>`
- `#include <stdlib.h>`
- `#include <sys/types.h>`

Test program requirements: This C test file must:

- Be able to take in three command line arguments and convert them to the right types (bool, float, float) to be passed as the reset flag, the increment value, and the initial value to the counter system call
- Use syscall() to call your custom counter system call
- Include a printf() statement that says "Calling counter syscall…" with a linefeed
- The test file **shouldn't require** command line arguments
    i. If no command line arguments are given, the reset flag should default to 0, the increment value should default to 1 (the counter will be incremented by 1), and the initial value before increment should default to 0
- The test file should support a variable number of command line arguments
    i. If the first argument is 1, no other arguments should be required to be given
    ii. If the first argument is 0, the second argument is an arbitrary value, and the third argument is not provided, the counter should increment the current count value by the second argument
    iii. If the first argument is 0, the second argument is an arbitrary value, and the third argument is provided, then the counter should have its value as the third argument incremented by the second argument

12. Compile your C test file using "gcc -o netid_test netid_test.c", replacing netid in both cases with your own.
13. Now we will run user-mode Linux (UML), which effectively runs an entire Linux instance within a process running on Linux. It's a little meta. This means that you don't have to replace CLEAR's currently running kernel (which is not allowed!), which provides a number of advantages and dramatically improves turnaround time on debugging kernel code.
14. To run UML, we enter "./linux rootfstype=hostfs rw init=/bin/bash"
    - You should see a bash prompt with a blinking cursor
    - If any message pops up like "random: crng init done", just hit enter and the bash prompt will reappear
15. Now change directories to where your linux source and test file are (note: you currently at root /)

- On CLEAR, you want to change to /storage-home/n/netid [except replacing n with the first letter of your netID and replacing netid with your full netid] then change to the specific folder where your test script is

16. Try running your test script via "./netid_test" (with your netid replaced)
17. Do the "Calling counter syscall…" [printf() from your test function] message and count [printk from your syscall code] print to the console?
    - Does the count increment properly when 0 is given as the first input argument and a positive number as the second input argument?
    - Does it reset when 1 is given as the first input argument?
    - Will it start incrementing again after reset with 0 as the first input argument and a positive number as the second input argument?
    - Does the counter increment correctly if the first argument is 0, the second is a positive number, and the third argument is another positive number?
    - Does input besides 0 or 1 (e.g. 10) for the first argument produce your printk() message about invalid input and the count is not incremented?
        i. If you see delays where messages don't appear until another system call, make sure you have linefeeds at the end of every print statement
18. We must also check if the kernel buffer was also getting this output - try "dmesg | tail -n10" to view the 10 most recent messages in the kernel buffer. Your counter printk() statements should appear there!
19. **See the rubric at the beginning of this document for turn in instructions and deliverables**