CS3281 Project:

Adding a System Call

Ananth Josyula and Alvin Eizner

Overview

The kernel is a core component of the Linux operating system. It is responsible for managing the system's resources, such as memory and CPU time, and for providing a framework for the system's various components to interact with each other. The kernel is typically built as a monolithic kernel, which means that all of its components are compiled together into a single file that can be loaded into memory and executed.

One of the key features of the Linux kernel is that it is open source, which means that anyone can view, modify, and distribute the source code. This allows developers to add new features to the kernel and customize it for specific purposes.

One way that the Linux kernel can be modified is by adding a new system call. A system call is a request made by a program to the kernel to perform a specific action or service. For example, a program might make a system call to create a new file or to allocate memory.

In this project, we planned to instrument the Linux kernel to track the number of times that system calls related to process creation and process execution are called: fork(), vfork(), execve(), and clone(). In addition to this, we hoped to add in two new system calls to the Linux kernel along with a user-space program to invoke these system calls.

Instructions

Build:

To build our project, save and open the linux folder. Then open terminal in that location and run:

- **make**
- **sudo make modules_install**
- **sudo make install**
- **sudo update-grub**

Run:

To run our project, reboot your Ubuntu and choose “advanced options” from the GRUB menu. Then select the custom kernel out of the list of kernels.

Test:

To test the new system calls, call functions like fork(), clone(), etc. and run our sc_test.c file.

Implementation

To add a new system call to the Linux kernel, we first had to install a new linux kernel from kernel.org onto our already existing Linux system. We extracted the tar, and from there we had to install some additional packages for compiling kernel:

- sudo sed -i "s/# deb-src/deb-src/g" /etc/apt/sources.list
- sudo apt update -y
- sudo apt install -y git fakeroot build-essential libncurses-dev xz-utils libssl-dev libelf-dev bc
- sudo apt install -y bison flex zstd

We then configured the kernel building process by copying the existing current boot configuration file

- cd linux-6.0.11
- cp /boot/config-$(uname-r) .config
- make menuconfig

Once this was complete, we had to make a .config file and configure the system revocation and trusted keys to empty. After this was done, we had to compile the kernel with the *make* command. Then we navigated to *~/linux-6.0.11/arch/x86/entry/syscalls/* and opened *syscall_64.tbl*, and add in our two new system calls. Then in the kernel/sys.c is where we wrote the definitions for both of our system calls.

We also modified fs/exec.c and kernel/fork.c to increment the counts of the necessary system calls. In the init/main.c, we declared a struct variable and initialize all fields to 0. For the struct to be accessible from fork, exec and sys files, we defined struct in include/linux/syscalls.h and declared the global variable as extern.

Finally we tested the syscalls by creating a program test.c that uses syscall(SYS_resetCount_SC) and syscall(SYS_retrieveCount_SC). We compiled it with gcc -o test test.c and executed it as ./test

Challenges

Challenges we faced when installing and building kernel:

- Setting inadequate number of processors
- Not allocating enough memory
- Locating the files we need to modify to trace the system calls

Division of Work

Ananth:

- Gathered sources for the project
- Modified the kernel to track system calls (fork, vfork, execve, clone)
- Kept detailed notes of entire instrumentation process

Alvin:

- Modified the kernel to allow new system calls that retrieved the number of system calls and reset this information
- Formulated Slide Show

Results

- The Linux kernel was instrumented to track the number of times that system calls related to process creation and process execution are called.
  - The specific system calls that were traced were fork(), vfork(), execve(), and clone()

- 2 new system calls to the Linux kernel were added
  - SYS_resetCount_SC and SYS_retrieveCount_SC
  - A user-space program to invoke these system calls:

```c
#define _GNU_SOURCE
#include <unistd.h>
#include <sys/syscall.h>
#include <stdio.h>
#include <string.h>
#include "/proc/self/cwd/syscalls.h"

/*
 * Put your syscall number here.
 */
#define SYS_count_SC 335
#define SYS_retrieveCount_SC 336
#define SYS_resetCount_SC 337

int main(int argc, char **argv)
{
  if (argc <= 1) {
    printf("Must provide a string to choose reset or retrieve.\n");
    return -1;
  }
  char* arg = argv[1];
  long res;
  if (strcmp(arg,"-reset")) {
        printf("Making reset system call.\n");
        res = syscall(SYS_resetCount_SC);
  }
else {
  if (strcmp(arg,"-retrieve")){
    struct count_SC_struct scC;
    printf("Making retrieve system call\n");
    res = syscall(SYS_retrieveCount_SC, &scC);
    printf("System call count values are as follow:\n");
    printf("fork: %d, vfork: %d, clone: %d, execve: %d, clone3: %d\n", scC.
    printf("System call returned %ld.\n", res);
  }
  else{
    printf("Making system call with \"%s\".\n", arg);
    res = syscall(SYS_count_SC, arg);
  }
  return res;
}
```