Assignment 1

Creating Custom System Calls

Submission Deadline : September 1, 2025 11:59 pm

Submission Instructions

1. Submit your assignment through the CSE Moodle platform
2. Only one member from each group should submit the assignment as a single zip file.
3. Ensure that the names and roll numbers of all group members are included in the submission.
4. Provide comprehensive documentation of your implementation. Include a README file explaining how to compile and run your code.

Objective

The objective of this assignment is to gain hands-on experience in defining and creating a custom system call to facilitate interaction between user-space programs and the Linux kernel.

Please download the 64-bit Ubuntu 22.04 LTS Desktop image and use it in a Virtual Machine (VM).

For tasks involving kernel configuration and building, use kernel version 5.10.240 or something close (5.10.xxx). (You can download kernel code from kernel.org)

Task Description

In this assignment, you are required to

1. Download the Linux kernel source code.
2. Define and declare 2 custom system calls in the appropriate files.
3. Compile the kernel.
4. Implement a C library wrappers around these system calls.
5. Write a user-space C program to test the system calls.

System Call 1: get_info_for_pid()

This system call should retrieve the following information about a specific process:

- *Parent process pid*
- *State* (as a number)
- *Static priority*
- *Number of child processes it has*
- *Number of sibling processes it has*

Arguments

The system call should accept two arguments:

- **PID**: The PID of the process whose information is required.
- **Buffer**: A char buffer that will store the extracted fields. You have to store all the fields that are extracted in this buffer, and then parse the buffer in the wrapper function. The buffer should contain the fields in the above mentioned order.

Return Values

The system call should return the following error codes:

- -ESRCH: If the PID is invalid.
- -EFAULT: If the user-space buffer location is invalid.

System Call 2: get_info()

This system call should retrieve the following information:

- *Total number of processes in the process list*
- *Number of processes in TASK_RUNNING state*
- *Number of processes in TASK_INTERRUPTIBLE state*
- *Number of processes in TASK_UNINTERRUPTIBLE state*
- *Number of processes in rt class*
- *Number of processes in fair class*
- *Number of processes in CFS runqueue*
- *PID of the process with min vruntime in the CFS runqueue*
- *The corresponding min vruntime*
- *Total load on the CFS runqueue*
- *Current target latency (in ms)*

For CFS runqueue related information, choose the runqueue on which the current process (where this syscall is being made) is present.

The syscall should also print the pids of all the processes in the process list when syscall is being made. This need not be included in buffer.

Arguments

The system call should accept one argument:

- **Buffer**: A char buffer that will store the extracted fields. You have to store all the fields that are extracted in this buffer, and then parse the buffer in the wrapper function. The buffer should contain the fields in the above mentioned order.

Return Values

The system call should return the following error codes:

- -ESRCH: If the PID is invalid.
- -EFAULT: If the user-space buffer location is invalid.

To test the system calls, you can use the **syscall()** function provided by glibc. Refer to syscall manpage for more details.

C Library Wrappers: lib_get_info_for_pid and lib_get_info

Implement wrappers around the custom system call to make it usable by user-space C programs.

Return Values

The wrapper functions should return a pointer to a custom C struct with all the fields populated from the syscall. In case of any error, it should set the appropriate *errno* and return a **NULL** pointer instead.

Steps to add sys call inside kernel:

1. Create a new directory inside the parent directory(which contains the kernel source code).

2. All the code and makefile related to syscalls should be put inside the this directory

3. Open the makefile of the kernel. Search for **core-y**. In the second result, add the directory name at the end. It should look similar to this **core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ io_uring/ yourdirectory/.** It tells the compiler that source code of the syscalls can be found in this directory

4. Open the **include/linux/syscalls.h** file and add a line(an example is shown below) at the end for each syscall before #endif. You need to prefix the syscall name with “sys” as shown.

   asmlinkage long sys_gettaskinfo(pid_t pid, char __user *buffer);

5. Open the **arch/x86/entry/syscalls/syscall_64.tbl** and add the new syscalls in it at a free entry (Use 441 for 1st syscall and 442 for 2nd)

6. Compile the kernel and test the syscalls.

Submissions

Students are required to submit the following:

1. Files modified in the kernel source code to implement the system call
2. The files where syscalls are implemented and C library wrapper files for the system calls.
3. The user-space test program.
4. A README file detailing all modified files and instructions on how to use the library. It should also contain any further design details you made.