AOS Assignment-2

Creating a custom system call in Linux Kernel

Bratin Mondal (21CS10016)
Datta Ksheeraj (21CS30037)

Structure of the submission

The submission contains the following files and directories:

1. `Readme.md` and `Readme.pdf`: The readme file containing the details of the submission.
2. `Makefile`: The Makefile for running the test programs.
3. `KERNEL_MODIFICATIONS`: The directory containing the modified files of the kernel source.
4. `gettaskinfo`: The directory containing the implementation of the custom system call.
5. `libgettaskinfo`: The directory containing the implementation of the C library wrapper.
6. `test`: The directory containing the test programs.

Steps to insert the custom system call

1. Download the kernel source code from the official website of the Linux Kernel. Unzip the Kernel Code and inside the `linux-5.10.223` directory and enter it.

```
cd linux-5.10.223
```

2. Inside this directory, create a directory `libgettaskinfo` and enter it.

```
mkdir libgettaskinfo
cd libgettaskinfo
```

3. Inside this directory, put the implementation of the code in `gettaskinfo.c` and create a `Makefile` in the same directory. The content of the Makefile:

```
obj-y := gettaskinfo.o
```

Note that both the `gettaskinfo.c` and `Makefile` are included in the submission.

4. Come out of this directory and get back to the `linux-5.10.223` directory. Open the `Makefile` of the kernel. Search for `core-y`. In the second result, add the `libgettaskinfo` directory at the end as shown below.

```
core-y      += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ io_uring/ libgettaskinfo/
```

This file is also included in the submission. The change can be seen in `line number 1144` of the `Makefile`.

5. From the same directory (`linux-5.10.223`), open the `include/linux/syscalls.h` file and add the following line at the end of it before `#endif`:

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

This file is also included in the submission. The change can be seen in `line number 1364` of the `syscalls.h`.

6. Again from the same directory (`linux-5.10.223`), open the `arch/x86/entry/syscalls/syscall_64.tbl` and add the new syscall in it at a free entry:

```
441 common  gettaskinfo     sys_gettaskinfo
```

Currently added at `line number 365` for SYSCALL number 441. This file is also included in the submission. Note that the syscall number 441 is chosen based on the free entries available in the file.

7. Build the kernel and switch to this new version

```
make menuconfig
make -j4
sudo make modules_install -j4
sudo make install -j4
sudo update-grub
```

If not automatically changed to this new version, manually select the new version from the grub menu.

Modified files in Kernel

1. `linux-5.10.223/libgettaskinfo/gettaskinfo.c`
2. `linux-5.10.223/libgettaskinfo/Makefile`
3. `linux-5.10.223/include/linux/syscalls.h`
4. `linux-5.10.223/arch/x86/entry/syscalls/syscall_64.tbl`
5. `linux-5.10.223/Makefile`

The details of the kernel directory is as shown below: The contents of the `linux-5.10.223` directory should be as follows:

```
bratin@Ubuntu:~/KERNEL/linux-5.10.223$ ls -la
total 996
drwxrwxr-x  26 bratin bratin   4096 Sep  3 12:15 .
drwxrwxr-x   5 bratin bratin   4096 Sep  3 12:12 ..
drwxrwxr-x  26 bratin bratin   4096 Sep  3 12:10 arch <---------- (d)
drwxrwxr-x   3 bratin bratin  12288 Sep  3 12:10 block
drwxrwxr-x   2 bratin bratin   4096 Sep  3 12:10 certs
-rw-rw-r--   1 bratin bratin  16673 Jul 27 14:10 .clang-format
-rw-rw-r--   1 bratin bratin     59 Jul 27 14:10 .cocciconfig
-rw-rw-r--   1 bratin bratin    496 Jul 27 14:10 COPYING
-rw-rw-r--   1 bratin bratin 100478 Jul 27 14:10 CREDITS
drwxrwxr-x   4 bratin bratin  36864 Sep  3 12:10 crypto
drwxrwxr-x  81 bratin bratin   4096 Jul 27 14:10 Documentation
drwxrwxr-x 140 bratin bratin   4096 Sep  3 12:10 drivers
drwxrwxr-x  79 bratin bratin  12288 Sep  3 12:10 fs
-rw-rw-r--   1 bratin bratin     71 Jul 27 14:10 .get_maintainer.ignore
drwxrwxr-x   2 bratin bratin   4096 Sep  3 12:10 gettaskinfo  <---------- (a)
-rw-rw-r--   1 bratin bratin     62 Jul 27 14:10 .gitattributes
-rw-rw-r--   1 bratin bratin   1911 Jul 27 14:10 .gitignore
drwxrwxr-x  29 bratin bratin   4096 Sep  3 12:15 include <---------- (c)
drwxrwxr-x   2 bratin bratin   4096 Sep  3 12:10 init
drwxrwxr-x   2 bratin bratin   4096 Sep  3 12:10 io_uring
drwxrwxr-x   2 bratin bratin   4096 Sep  3 12:10 ipc
-rw-rw-r--   1 bratin bratin   1327 Jul 27 14:10 Kbuild
-rw-rw-r--   1 bratin bratin    555 Jul 27 14:10 Kconfig
drwxrwxr-x  21 bratin bratin  12288 Sep  3 12:10 kernel
drwxrwxr-x  21 bratin bratin  28672 Sep  3 12:10 lib
drwxrwxr-x   6 bratin bratin   4096 Jul 27 14:10 LICENSES
-rw-rw-r--   1 bratin bratin  18204 Jul 27 14:10 .mailmap
-rw-rw-r--   1 bratin bratin 576604 Jul 27 14:10 MAINTAINERS
-rw-rw-r--   1 bratin bratin  64983 Sep  3 09:14 Makefile <---------- (b)
drwxrwxr-x   3 bratin bratin  16384 Sep  3 12:10 mm
drwxrwxr-x  72 bratin bratin   4096 Sep  3 12:10 net
-rw-rw-r--   1 bratin bratin    727 Jul 27 14:10 README
drwxrwxr-x  32 bratin bratin   4096 Sep  3 12:10 samples
drwxrwxr-x  17 bratin bratin   4096 Sep  3 12:15 scripts
drwxrwxr-x  13 bratin bratin   4096 Sep  3 12:10 security
drwxrwxr-x  26 bratin bratin   4096 Sep  3 12:10 sound
drwxrwxr-x  36 bratin bratin   4096 Jul 27 14:10 tools
drwxrwxr-x   3 bratin bratin   4096 Sep  3 12:10 usr
drwxrwxr-x   4 bratin bratin   4096 Sep  3 12:10 virt
```

The entry at `(a)` is the newly created directory `gettaskinfo` which contains the `gettaskinfo.c` and `Makefile` files. The contents of the `gettaskinfo` directory should be as follows:

```
bratin@Ubuntu:~/KERNEL/linux-5.10.223/gettaskinfo$ ls -la
```

```
total 16
drwxrwxr-x  2 bratin bratin 4096 Sep  3 12:10 .
drwxrwxr-x 26 bratin bratin 4096 Sep  3 12:15 ..
-rw-rw-r--  1 bratin bratin 1220 Sep  3 11:13 gettaskinfo.c
-rw-rw-r--  1 bratin bratin   22 Sep  3 09:13 Makefile
```

The entry at `(b)` is the modified `Makefile` of the kernel.

The entry at `(c)` is `include` directory. Inside this directory, there is another directory `linux` which contains the `syscalls.h` file which is modified.

The entry at `(d)` is the `arch` directory. Inside this directory, there is another directory `x86` which contains the `entry` directory which contains the `syscalls` directory which contains the `syscall_64.tbl` file which is modified.

Implementation of the custom system call

The implementation of the custom system call is done in the `gettaskinfo.c` file.

Input

The system call takes two arguments:

1. `pid_t pid`: The process id of the process for which the information is to be fetched.
2. `char __user *buffer`: The buffer to which the information is to be written.

Output

The system call writes the state, start time and normal priority of the process with the given `pid` to the buffer in space-separated format.

1. It returns `0` if the process with the given `pid` is found and the information is written to the buffer successfully.
2. It returns `-ESRCH` if the process with the given `pid` is not found.
3. It returns `-EFAULT` if the buffer is not accessible.

Implementation

1. The system call first acquires the RCU read lock to safely read the task information.
2. It first finds the `struct pid` of the process with the given `pid` using the `find_vpid` function.
3. It then finds the `task_struct` of the process using the `pid_task` function.
4. If the `task_struct` is `NULL`, it unlocks the RCU read lock and returns `-ESRCH`.
5. If the `task_struct` is found, it formats the state, start time and normal priority of the process into a string and stores it in the `info` buffer.
6. It then unlocks the RCU read lock and checks if the buffer is accessible using the `access_ok` function.
7. If the buffer is not accessible, it returns `-EFAULT`.
8. If the buffer is accessible, it copies the information from the `info` buffer to the user buffer using the `copy_to_user` function.
9. If the copy is successful, it returns `0`, else it returns `-EFAULT`.

C Library Wrapper

Header File

The header file `libgettaskinfo.h` contains the structure `task_info` which is used to store the information of the process. It also contains the function declaration of the `gettaskinfo` function.

Implementation

The implementation of the `gettaskinfo` function is done in the `libgettaskinfo.c` file. The function makes a system call to retrieve the task information for a given PID. It parses the information into a `task_info` structure and returns a pointer to it. If an error occurs, it returns `NULL` and sets `errno` accordingly.

Usage

The `gettaskinfo` function can be used to get the task information of a process with a given PID. The function takes the PID as an argument and returns a pointer to the `task_info` structure containing the state, start time and normal priority of the process.

To use the function, include the `libgettaskinfo.h` header file and link the `libgettaskinfo.c` file with the program while compiling.

Testing the custom system call

Testing the system call

To test the system call, we have created a user-space program `raw_test.c` which takes a PID as a command-line argument and prints the state, start time and normal priority of the process with the given PID. On successful execution, it prints the process state, start time and normal priority. On failure, it prints the error message.

Testing the C Library Wrapper

To test the C library wrapper, we have created a user-space program `test_gettaskinfo.c` which takes a PID as a command-line argument and uses the `gettaskinfo` function to retrieve the task information for the given PID. It prints the process state, start time and normal priority. On failure, it prints the error message.

Running the test programs

First install the newly built kernel and boot into it.

To run the test programs, a Makefile is provided which compiles the test programs and creates the executable files `raw_test` and `test_gettaskinfo`. To run the test programs, use the following commands:

```
make all
```

For running the raw test program:

```
./raw_test <PID>
```

For running the test program using the C library wrapper:

```
./test_gettaskinfo <PID>
```

To clean the executables, use the following command:

```
make clean
```