[2024-Spring] Operating System
Project #1 Instruction
Adding a New System Call

Advisor Gyeongsik Yang (g_yang@korea.ac.kr)
System Software Laboratory.
https://ss.korea.ac.kr

Assistant Hyunho Lee and Hwiju Cho (osta@os.korea.ac.kr)

Topic Understanding and implementing system calls.

Due date (submission) **April 11, 2024**

Recommended Environment Ubuntu 18.04.02 (64 bit), Linux kernel 4.20.11 (VM environment)

Contents

1. Purpose
2. Goal
3. Background
4. List of key files to be modified in the kernel
5. User application details
6. Example of project executions
7. Restrictions
8. Submission guidelines
9. Note

1. Purpose

The goal of this project is to **1) compile and modify kernel** of existing open-source Linux and to **2) add a new system call**.

System call is an interface and corresponding handler (function) that specify routines and actions that the OS kernel provides with the underlying hardware. The system call handler is executed in the kernel mode with higher privileges. To switch from user mode (space) to kernel mode, existing OS uses special **trap instruction**. In this project, you are requested to compile your own kernel that includes your new system calls. Also, you are required to implement a user application to call the newly implemented system calls.

Our kernel internally maintains a new "queue" data structure. The queue 1) stores integer values (enqueue) and 2) returns the integer value in the order in which it was enqueued (dequeue). That is, the first data on the queue is the first data off the queue. Our new system calls are for 1) enqueuing and 2) dequeuing an integer value to/from the queue maintained in the kernel space. Note that the two system calls do not store an overlapping integer value that already exist in the queue. The number of entries in the queue can be arbitrary determined (e.g., fixed number of entries).

2. Goal

A. Kernel

- Accomplish this project using a virtual machine environment (VirtualBox) – a recommended environment and basis for TA Q&A.
- Use Linux kernel from the Ubuntu 18.04.2 LTS distribution.
  i. For Apple Silicon Macs (M1, M2 CPUs in MacOS or any other possible situations), the specified version of Ubuntu distribution might not be viable.
  ii. Then, please indicate in the report when using different versions of Ubuntu distribution.
- The newly compiled Kernel version should be 4.20.11.
- Implement a queue data structure on the kernel.
- Implement two system calls to enqueue and dequeue an integer. When each system call is called, the passed parameter, integer, must be printed out using `printk` function.

B. User Process

- Implement a new user program that calls the two newly implemented system calls. Use `syscall` macro when triggering the system calls.
- Check that the enqueue and dequeue operations work properly.
- Print out (log) the returned value from the system call.

3. Background

A. System call

System call is an interface from the user applications to kernel services. For example, if a programmer wants to read a file, he or she should use the corresponding system call (e.g., `read`). When programming in C, most system calls can be accessed through wrapper functions provided by high-level library like `libc`. If `read` function is called by the user application, routines shown in Figure 1 and Figure 2 happen. Figure 1 shows the system call routine in a 32-bit system, while Figure 2 in a 64-bit system.

As shown in Figure 1, when **`read`** is called in the user process, a 0x80 trap instruction occurs through the wrapper function in the library. The 0x80th trap is reserved for system calls, and this trap handler checks the number in the **`sys_call_table`** and calls the corresponding system call handler, **`sys_read`**. When the system call handler finishes the processing, it returns to the user process along with the results.

Figure 2 shows the identical scenario on the 64-bit machine. When **`read`** is called in the user process, a trap instruction, **`sysenter`**, is called by the wrapper function. In comparison to 0x80 trap, **`sysenter`** accesses to the **`sys_call_table`** directly. The remaining steps are similar to the ones of Figure 1.

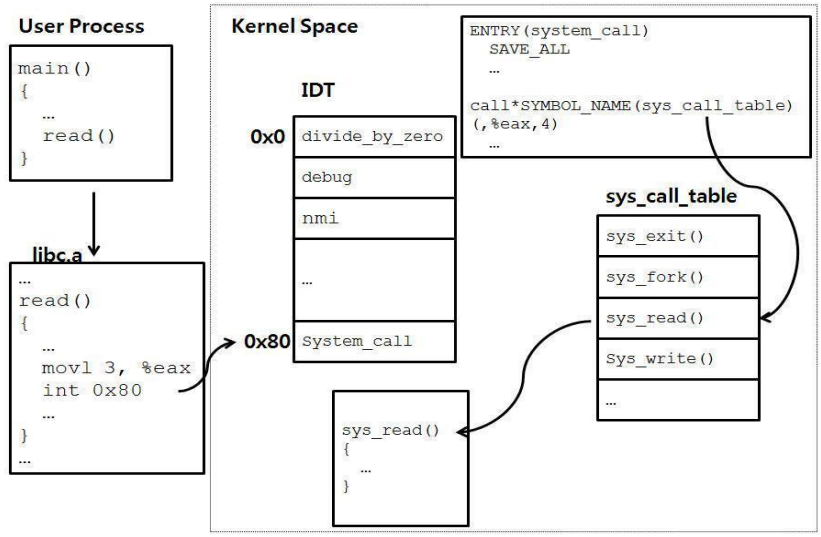

Figure 1. System call example (32-bit machine).

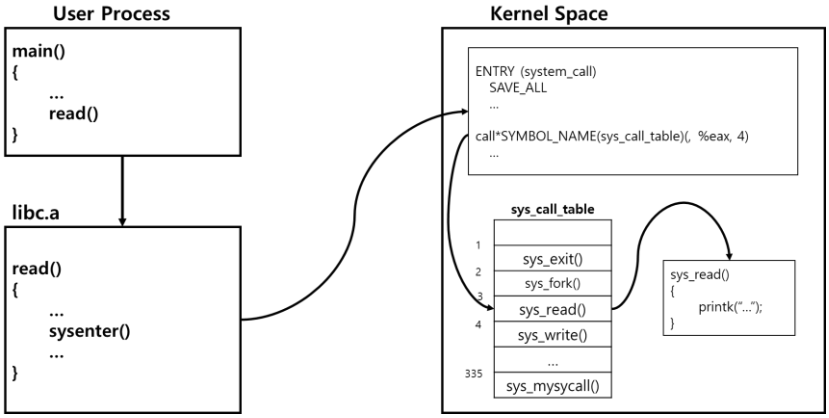

Figure 2. System call example (64-bit machine).

In this project, we will add two new system calls (called **`sys_os2024_enqueue`** and **`sys_os2024_dequeue`**) by 1) adding two more entries to the **`sys_call_table`**, 2) implement system call handlers for the two, and 3) call the two system calls in the user application to identify its operations.

B. Kernel logging (`printk` function)

In the kernel space, the widely used printing function, **`printf`**, cannot be used. Instead, **`printk`** must be used to print out messages. The usage of **`printk`** is essentially the same as that of **`printf`**. While **`printk`** allows messages to be printed with specific levels of importance, we would not use this feature in this project. If a message is printed out through **`printk`** (maybe within a system call), it will not be visible on a monitor or a regular terminal screen but can be checked by typing the **`dmesg`** command that lists the recently generated kernel logs and messages.

4. List of key files to be modified in the kernel

A. (linux)/arch/x86/entry/syscalls/syscall_64.tbl (64bit machine)
→ A file that collects symbol information about the system call handler (function) names

B. (linux)/include/linux/syscalls.h
→ Define prototypes for system call functions

C. (linux)/kernel/sslab_my_queue.c

→ Source of the **new** system call to be implemented

D. (linux)/kernel/Makefile

→ Designate objects to be complied within the kernel

Add objects sslab_my_queue.o

*** (linux) : /usr/src/linux-4.20.11 (The root folder where the kernel's source code is stored)**

4.1 Details.

A. (linux)/arch/x86/entry/syscalls/syscall_64.tbl

It is a file that stores the unique numbers of all system calls provided by Linux and also contains the symbol information for these system calls. The addresses of the system calls, as well as the table that stores these addresses, are scattered throughout the Linux kernel source directories and are managed (collected) by the linker for the kernel compile.

- Format of each entry: `<number> <abi> <name> <entry point>`

```
osta@osta-VirtualBox: /usr/src/linux-4.20.11
File  Edit  View  Search  Terminal  Help
334     common  rseq                    __x64_sys_rseq

# sslab --- start
335     common  os2024_enqueue          __x64_sys_os2024_enqueue
336     common  os2024_dequeue          __x64_sys_os2024_dequeue
# sslab --- end

#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation. The __x32_compat_sys stubs are created
# on-the-fly for compat_sys_*() compatibility system calls if X86_X32
```

B. (linux)/include/linux/syscalls.h

This file defines the prototypes of system call functions and declares the functions using **`asmlinkage`**. System calls are invoked by the interrupt handler, which is written in assembly code. By using **`asmlinkage`**, C function calls can be possible even within assembly code.

- `asmlinkage int sys_os2023_push(int)`
- `asmlinkage int sys_os2023_pop(void)`

```
static inline unsigned int ksys_personality(unsigned int personality)
{
        unsigned int old = current->personality;

        if (personality != 0xffffffff)
                set_personality(personality);

        return old;
}

/* sslab --- start */
asmlinkage int sys_os2024_enqueue(int);
asmlinkage int sys_os2024_dequeue(void);
/* sslab --- end */

#endif
"include/linux/syscalls.h" 1304L, 50760C written                1303,0-1
```

C. (linux)/kernel/sslab_my_queue.c

It implements the source code for the system calls to be added. It defines what the system calls behave. For this project, a **queue** in the form of an int array is declared as a **global variable**, and enqueue and dequeue functions are implemented.

Enqueue function should store the passed parameter (integer) from the user application. Dequeue function pops (returns) the oldest integer value and removes it from the queue, i.e. the first integer enqueued shall be the first integer dequeued.

- Skeleton implementation of enqueue, dequeue functions

```c
/* 2024 Spring COSE341 Operating System */
/* Project 1 */
/* your name here */

#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/linkage.h>

SYSCALL_DEFINE1(os2024_enqueue, int, a) {
    /* your implementation here */
}

SYSCALL_DEFINE0(os2024_dequeue) {
    /* your implementation here */
}
```

  - `SYSCALL_DEFINE1(os2024_enqueue, int, a){ … }`
  - `SYSCALL_DEFINE0(os2024_dequeue){ … }`
    - ✓ `SYSCALL_DEFINEx`: Macros for implementing system calls requiring "`x` number of parameters"
      Such macros are defined in (linux)/include/linux/syscalls.h

- Add the following headers for implementation
  - `<linux/syscalls.h>`
  - `<linux/kernel.h>`
  - `<linux/linkage.h>`

D. (linux)/kernel/Makefile

- Makefile designates the objects to be included during the kernel compile process (`make`)
- Add the file of the above implemented system call functions to the `obj-y` part so that it is compiled into the kernel.

```
# SPDX-License-Identifier: GPL-2.0
#
# Makefile for the linux kernel.
#

obj-y     = fork.o exec_domain.o panic.o \
            cpu.o exit.o softirq.o resource.o \
            sysctl.o sysctl_binary.o capability.o ptrace.o user.o \
            signal.o sys.o umh.o workqueue.o pid.o task_work.o \
            extable.o params.o \
            kthread.o sys_ni.o nsproxy.o \
            notifier.o ksysfs.o cred.o reboot.o \
            async.o range.o smpboot.o ucount.o sslab_my_queue.o
```

  - `.o` object file name should be the same as its own C file name.

- e.g., `sslab_my_queue.c -> sslab_my_queue.o`

E. Kernel compile

If you have completed all of the steps above, run the following commands in the **/usr/src/linux-4.20.11** directory.

- `$ sudo make`
- `$ sudo make install`

5. User application details

System calls are APIs, so they should be called by user applications. There are several ways to invoke a system call, one of which is to use the **`syscall`** macro. To use **`syscall`**, the syntax is **`syscall(system_call_number, args...)`**. For example, if you define a newly added system call as the 335th system call, the corresponding user application would be implemented as follows:

```c
#include<unistd.h>

/* ..... */

        int r;
        r = syscall( 335, 123 );             // System Call
```

or

```c
#include<unistd.h>
#define my_queue_syscall 335   // specifies the system call defined by the same number
                               // as the header file of the kernel
/* ..... */

        int r;
        r = syscall(my_queue_syscall, 123);  // System Call
```

When the user application calls **`os2024_enqueue`**, it should pass the "randomly" generated integer as its parameter. At least, **`os2024_enqueue`** and **`os2024_dequeue`** should be called **three times**.

6. Example of project executions

A. When running user applications,

```
osta@osta-VirtualBox:~/project1$ ./call_my_queue
Enqueued: 43
Enqueued: 27
Enqueued: 95
Dequeued: 43
Dequeued: 27
Dequeued: 95
```

B. When checking kernel logs (by `dmesg`),

```
[   68.367096] [System call] os2024_enqueue(); -----
[   68.367096] Queue Front ----------------------
[   68.367097] 43
[   68.367097] Queue Rear  ----------------------
[   68.367174] [System call] os2024_enqueue(); -----
[   68.367174] Queue Front ----------------------
[   68.367175] 43
[   68.367175] 27
[   68.367175] Queue Rear  ----------------------
[   68.367178] [System call] os2024_enqueue(); -----
[   68.367179] Queue Front ----------------------
[   68.367179] 43
[   68.367179] 27
[   68.367180] 95
[   68.367180] Queue Rear  ----------------------
```

```
[   68.367182] [System call] os2024_dequeue(); -----
[   68.367183] Queue Front ----------------------
[   68.367183] 27
[   68.367183] 95
[   68.367184] Queue Rear  ----------------------
[   68.367186] [System call] os2024_dequeue(); -----
[   68.367187] Queue Front ----------------------
[   68.367187] 95
[   68.367187] Queue Rear  ----------------------
[   68.367190] [System call] os2024_dequeue(); -----
[   68.367190] Queue Front ----------------------
[   68.367190] Queue Rear  ----------------------
```

7. Restrictions

A. Please use Linux distribution and kernel versions specified in this instruction.

- You can use other versions, but Q&A with TAs are based on the specified version.
- If other versions are used, please specify the versions on your report.

8. Submission guidelines

A. Submit all files through Blackboard.

B. Be sure to include the following sections in the report (The report should not exceed 5 pages, excluding the cover page):

- Please list the following on the cover: Department, Student number (학번), Name, Submission date, Number of "Freedays" used for Project 1
- Development environment
- Explanation of system calls on Linux (including call routines)
  i. **Caution:** Do not copy or paste existing material (including web).
  Although there is abundant data related to system calls, ensure you understand the content and write it in your own words.
- Implementation: modified and written codes with descriptions
  i. Do not attach the source code verbatim; instead, describe the overall workflow with important code-pieces.
- Snapshot of project execution (both from the user application and the kernel)
- Difficulties and your efforts (maybe solutions) encountered during this project

C. List of files to submit

- Report
- All source code files you modified or newly implemented.
  i. Comments explaining the roles of important variables and functions should be included in the source code.
- Execution result file (result.txt)
  i. Submit the execution results in the format specified in Section 6 ("example of project executions") as a text file.

D. How to submit a file

1) Gather source codes that you modified into a folder named "**source**". Compress the source folder with the **report** and **result.txt** together. Submit the compressed file (named **"os1_[student number]_[name].zip"**).
2) **Never submit the entire Linux Kernel Source as part of the source files.**

9. Note

A. For late submissions that exceed the "Freeday" allowance (7 days in total), 1 point will be deducted from the total score of the project for each day of late submission.

B. No projects will be accepted after 15 days has passed from the deadline of the project #1.