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

Advisor	Gyeongsik Yang (g_yang@korea.ac.kr) System Software Lab. **https://ss.korea.ac.kr**
Assistant	Jinwoo Jeong and Jongin Choi (osta@os.korea.ac.kr)

Topic	Understanding and implementing system calls.
Due date (submission)	**2024. 10. 14. (Monday) PM 11:59**
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. Grading rubric

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 specifies 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 "stack" data structure. The stack 1) stores integer values (push) and 2) returns the integer value that is inserted last (pop). That is, the last data on the stack is the first data off the stack. Our new system calls are for 1) pushing and 2) popping an integer value to/from the stack maintained in the kernel space. The number of entries of the stack 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 stack data structure on the kernel.
- Implement two system calls to push and pop 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 push and pop 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.

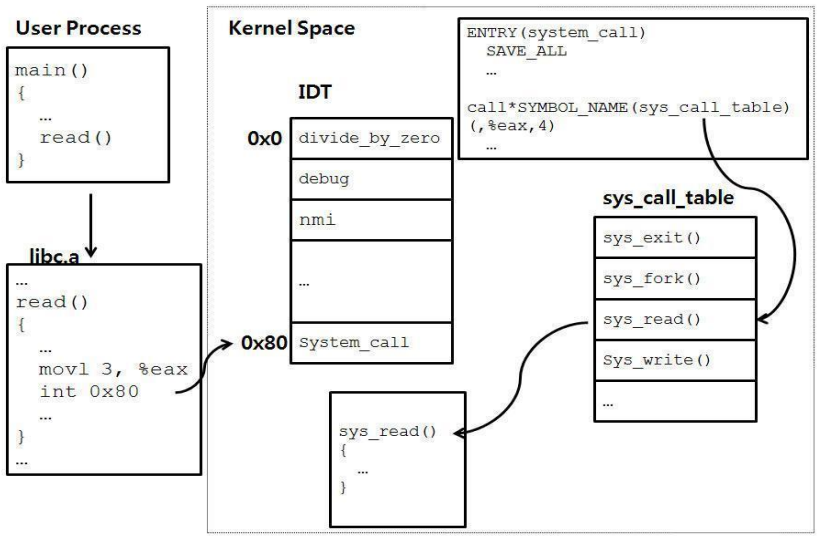


Figure 1. System call example (32-bit machine).

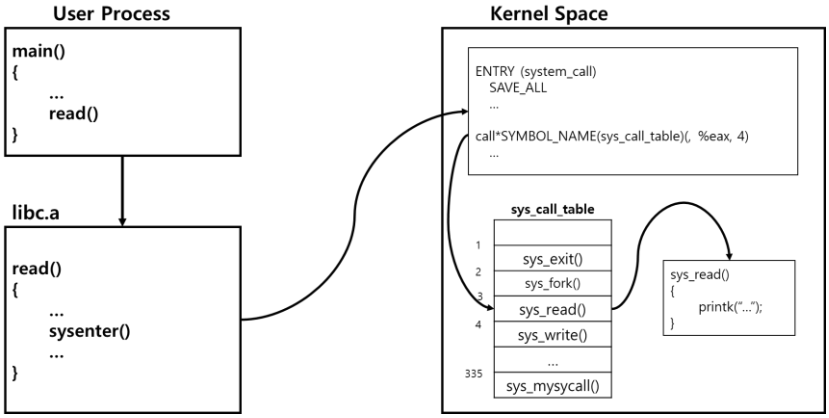


Figure 2. System call example (64-bit machine).

In this project, we will add two new system calls (called **sys_os2024_push** and **sys_os2024_pop**) 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_stack.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_stack.o

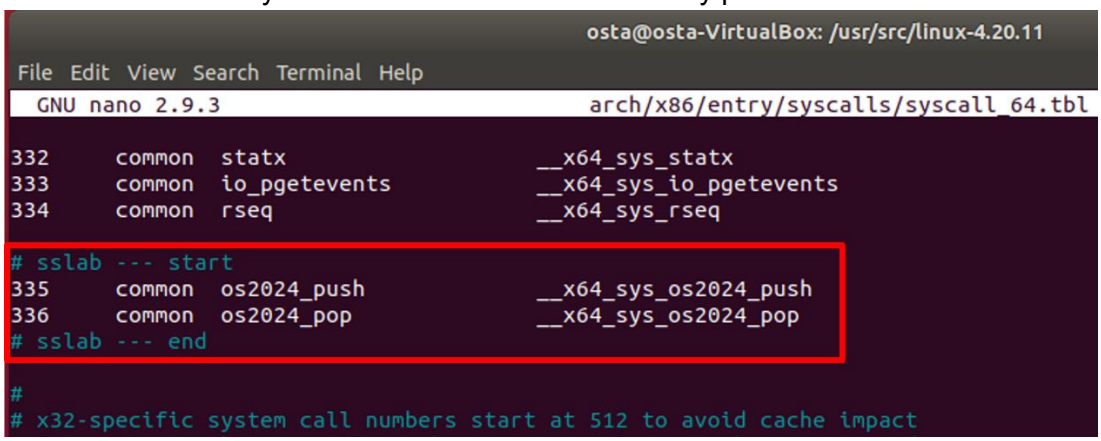
*** (linux) : /usr/src/linux-4.20.11 (The root folder where the kernel's source code is stored)**

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.

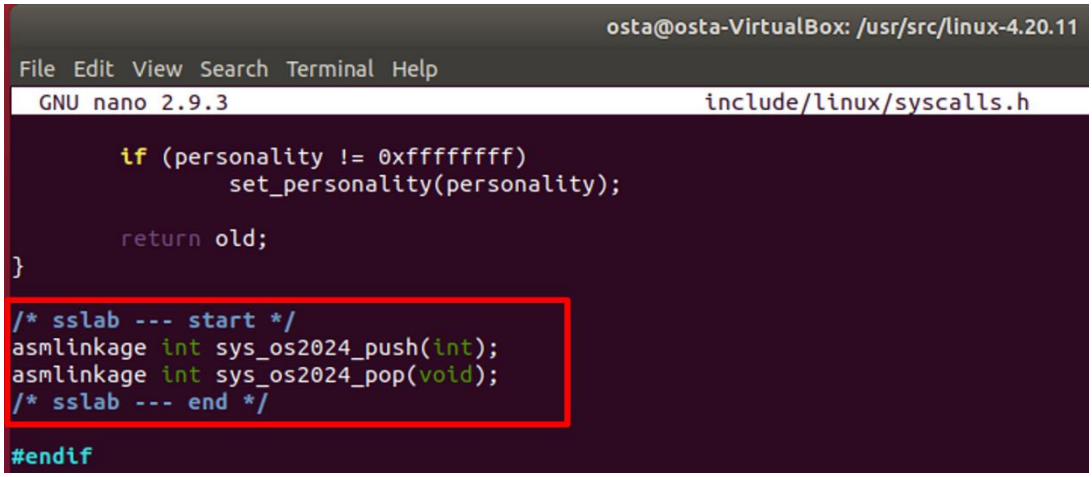
- Format of each entry: <number> <abi> <name> <entry point>



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_os2024_push(int)
- asmlinkage int sys_os2024_pop(void)



C. (linux)/kernel/sslab_my_stack.c

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

Push function should store the passed parameter (integer) from the user application.

Pop function pops (returns) the newest integer value and removes it from the stack, i.e. the last integer pushed shall be the first integer popped.

- Skeleton implementation of push, pop functions

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

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

SYSCALL_DEFINE1(os2024_push, int, a) {
   /* your implementation here */
}

SYSCALL_DEFINE0(os2024_pop) {
   /* your implementation here */
}
```

- SYSCALL_DEFINE1(os2024_push, int, a){
  …
  }
- SYSCALL_DEFINE0(os2024_pop){
  …
  }

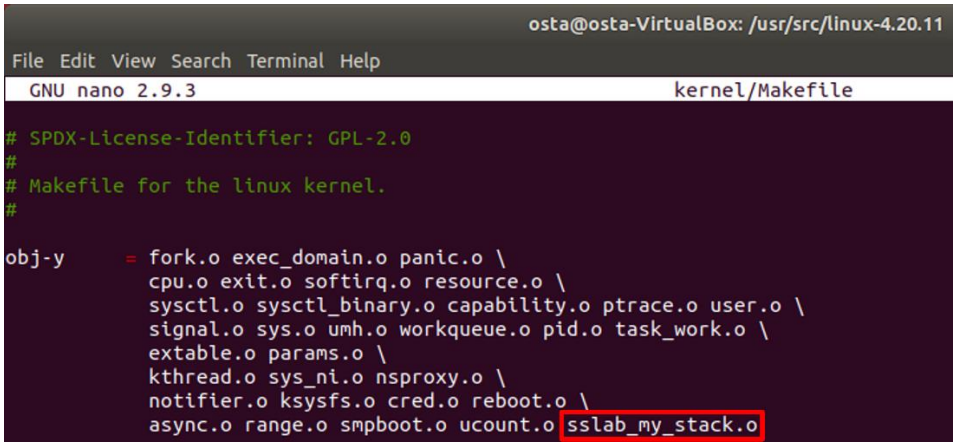
  ✓ SYSCALL_DEFINEx: Macros for implementing system calls requiring “x number of parameters”

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.



  - .o object file name should be the same as its own C file name.
    - e.g., sslab_my_stack.c -> sslab_my_stack.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. **For issues with the kernel version changing after a reboot, see "How to Boot into a Specific Kernel Version".**

- $ sudo make -j 4
- $ sudo make install
- $ sudo reboot

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:

```
#include<unistd.h>

/* ..... */

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

or

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

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

When the user application calls **os2024_push**, it should pass the “randomly” generated integer as its parameter. At least, **os2024_push** and **os2024_pop** should be called **three times**.

6. Example of project executions

A. When running user applications,

```
osta@osta-VirtualBox:~/project1$ ./call_my_stack
Push: 43
Push: 27
Push: 95
Pop: 95
Pop: 27
Pop: 43
```

B. When checking kernel logs (by dmesg),

```
[   81.973352] [System Call] os2024_push(); ----
[   81.973353] Stack Top ----------------------
[   81.973355] 43
[   81.973355] Stack Bottom -------------------
[   81.973358] [System Call] os2024_push(); ----
[   81.973358] Stack Top ----------------------
[   81.973359] 27
[   81.973359] 43
[   81.973360] Stack Bottom -------------------
[   81.973361] [System Call] os2024_push(); ----
[   81.973361] Stack Top ----------------------
[   81.973361] 95
[   81.973362] 27
[   81.973362] 43
[   81.973362] Stack Bottom -------------------
```

```
[   81.973363] [System Call] os2024_pop(); -----
[   81.973363] Stack Top ----------------------
[   81.973363] 27
[   81.973364] 43
[   81.973364] Stack Bottom -------------------
[   81.973365] [System Call] os2024_pop(); -----
[   81.973365] Stack Top ----------------------
[   81.973366] 43
[   81.973366] Stack Bottom -------------------
[   81.973367] [System Call] os2024_pop(); -----
[   81.973367] Stack Top ----------------------
[   81.973368] Stack Bottom -------------------
```

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 in your report.

8. Submission guidelines

A. Submit all files through Blackboard.

B. 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.

C. How to submit a file

1) Please follow below naming rules including file extensions and directory structure.
2) Compress all the files and directories into a “os1_[your student ID].zip”.

```
os1_[your student ID]/
├── report.docx
├── result.txt
└── source
    ├── Makefile
    ├── call_my_stack.c
    ├── sslab_my_stack.c
    ├── syscall_64.tbl
    └── syscalls.h
```

3) **Never submit the entire Linux kernel source code.**
4) All source code must be able to be compiled and executed in an identical Linux environment.

D. Be sure to include the following sections in the report (The report should not exceed 11 pages, including the cover page):

1) **We provide a skeleton report for your convenience. Be sure to use the skeleton.**
2) **You can use either English or Korean to write the report.**
3) Please list the following on the cover: Name, student ID, department, submission date, number of free days used for Project 1
4) Development environment
5) Explanation of system calls on Linux (including call routines)
   a) **Caution:** Do not copy or paste existing material (including web and GitHub repositories). Although there is abundant data related to system calls, ensure you understand the content and write it in your own words.
6) Implementation

E. Note

1) For late submissions that exceed the “free day” allowance (7 days in total), 1 point will be deducted from the total score (not from project 1’s score) for each day of late submission.
2) **No project results will be accepted after one week (7 days) has passed from the deadline of project #1.**

9. Grading Rubric

A. Submit all files through Blackboard. Other methods will not be accepted.

B. Scoring for each submitted item

- Report [90pts]
  - i. **Limit your writing to 11 pages including a cover page. If the format is incorrect, 1 point will be deducted from the total score. (font size: 13 pt, font type: Times New Roman or Malgun Gothic)**
  - ii. **1 point will be deducted for missing items (name, student ID, Dept, date, free day, Development environment, font size, font) from the total score**
  - iii. **Explanation of system calls on Linux (including call routines) [20 pts]**
  - iv. **Implementation [70 pts]**
    - Zero points will be awarded if you attach screenshots instead of text.
    - If your code does not produce proper results, zero points will be awarded.
    - For A, B, C, D, E
      - Please attach only added or edited part of code in text. Otherwise, zero points will be awarded.

- Source code [6 pts]
  - i. **Please gather source code files (and Makefile) into a directory named "source" and there should be 5 files in the source folder**
    - /usr/src/linux-4.20.11/arch/x86/entry/syscalls/syscall_64.tbl
    - /usr/src/linux-4.20.11/include/linux/syscalls.h
    - /usr/src/linux-4.20.11/kernel/sslab_my_stack.c
    - /usr/src/linux-4.20.11/kernel/Makefile
    - call_my_stack.c

- Text file(result.txt) that contains execution results [2 pts]
  - i. **1 point will be deducted for unsuitable file name.**

- os1_[your student Id].zip [2 pts]
  - i. **Please gather files and folders into a directory named "os1_[your student Id]"**
  - ii. **1 point will be deducted for each incorrect each directory or file name or extension.**

NOTE:

This grading rubric is to ensure fairness among students.
All grading will be conducted based on the criteria outlined in the rubric.
Most importantly, it is intended to help you complete this assignment and achieve a good score.
If you have any questions, please feel free to contact TAs via email.