1st Assignment of Operating System

Major: Computer Science
Student ID: 2021320050
Name: 장오선
Usage of free day:0
Environment: Ubuntu 18.04.02 (64bit),
Linux kernel 4.20.11 (virtual machine environment)

Basic Concepts of System call:

Typically, a set of subroutines (procedures) for implementing various system functions is set up in the core of the OS and provided to application calls. A system call is a programmatic way in which a computer program requests a service from the kernel of the operating system. A computer program makes a system call when it makes a request to the operating system's kernel. System call provides the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and the operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

Kernel mode and User mode:

In computer systems, two types of programs are usually running: system programs and applications. Two modes are set for the computer to ensure that the program is not intentionally or unintentionally destroyed by the application: Kernel mode in which the operating system operates user mode, where applications can only run. During actual operation, the processor switches between the kernel mode and the user state. Accordingly, most modern operating systems divide the CPU's instruction set into privileged and non-privileged instructions.
1) Privileged instructions -- instructions to run while in the kernel mode
There is no restriction on access to memory space. Privileged instructions are allowed only to the operating system and not to the application, otherwise, they can cause system confusion.
2) Unprivileged instructions -- instructions to run in user mode
Most applications use non-privileged instructions, which can only perform general operations and tasks, can not directly access the hardware and software in the system, and its memory access is limited to user space.

System call:

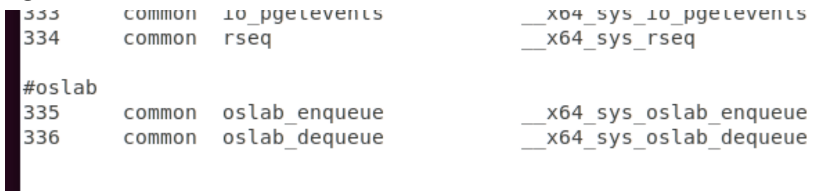
When the CPU calls system calls from the running process, the library stores the number of the system call in the register via the MOVE command and causes an interrupt. (i.e., the kernel uses Wrapper Function to generate a 0x80 Trap Instruction.) When a Trap occurs from User space to Kernel Space, it checks the IDT, and the handler to handle the Trap checks the number of the system call table to find and call the system call that corresponds to the number in the table, and returns a specific value to notify the system call function that the execution has ended. If we perform all of these routines, it will return to User mode from Kernel mode. The application is in the standby/wait state until a specific value is returned.

Description of system calls (including call routines) on Linux:

We can divide our task into two parts by the type of modes:

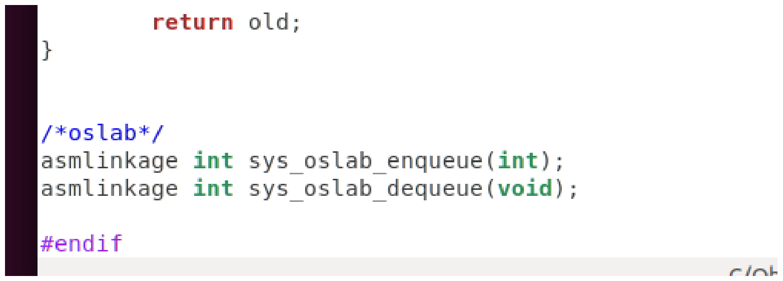
Kernel mode:

1. Adding the new system call to the system call table
   a. Command line:
      sudo gedit /usr/src/linux-4.20.11/arch/x86/entry/syscalls/syscall_64.tbl
   b. Add our system call numbers and system call names after the rule, as shown in the figure below



```
333     common  io_pgetevents           __x64_sys_io_pgetevents
334     common  rseq                    __x64_sys_rseq

#oslab
335     common  oslab_enqueue           __x64_sys_oslab_enqueue
336     common  oslab_dequeue           __x64_sys_oslab_dequeue
```

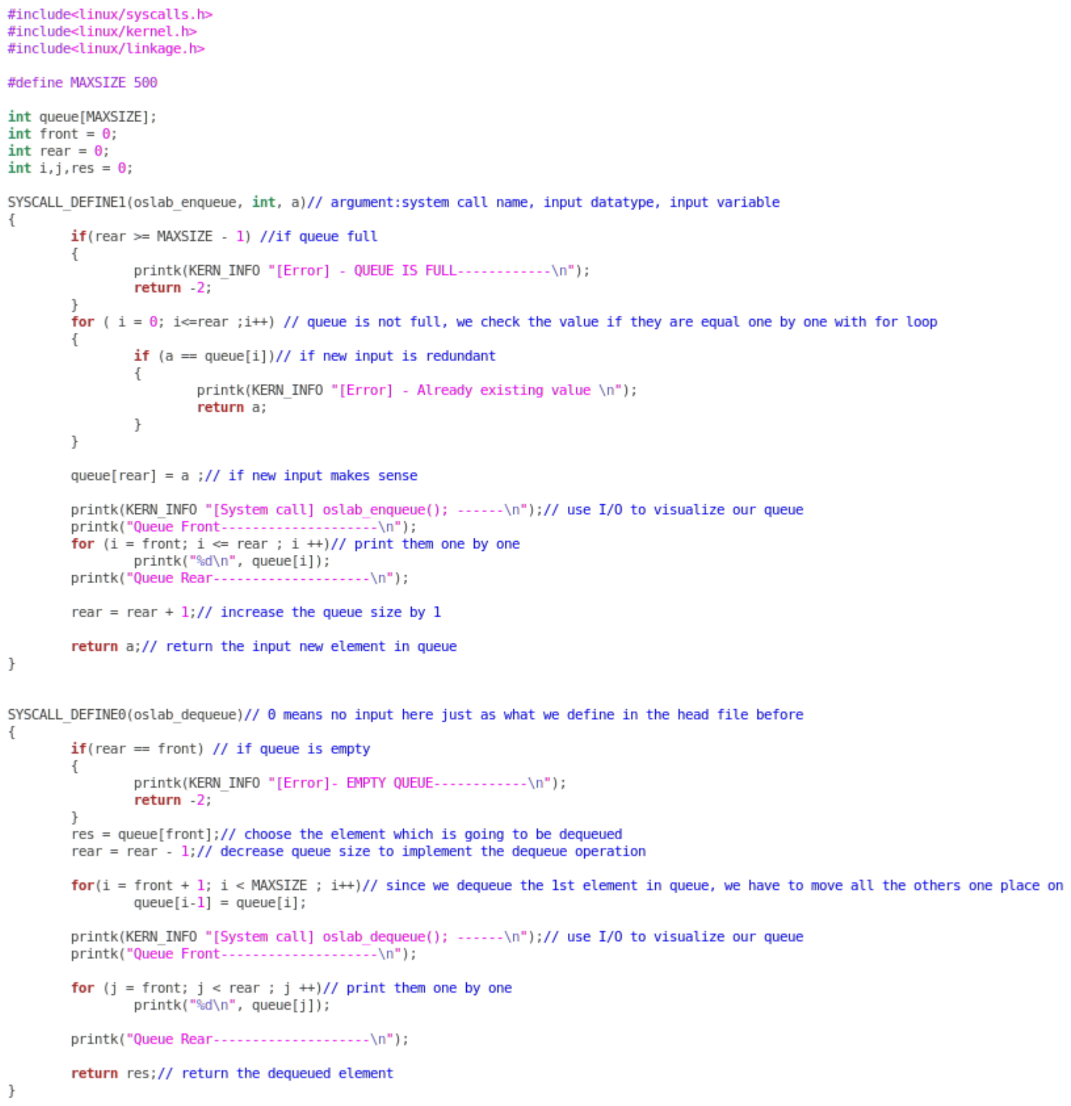
2. Adding the new system call to the system call header file
   a. Command line:
      sudo gedit /usr/src/linux-4.20.11/include/linux/syscalls.h
   b. Add a function declaration corresponding to a system call, as shown below



```
        return old;
}


/*oslab*/
asmlinkage int sys_oslab_enqueue(int);
asmlinkage int sys_oslab_dequeue(void);

#endif
```

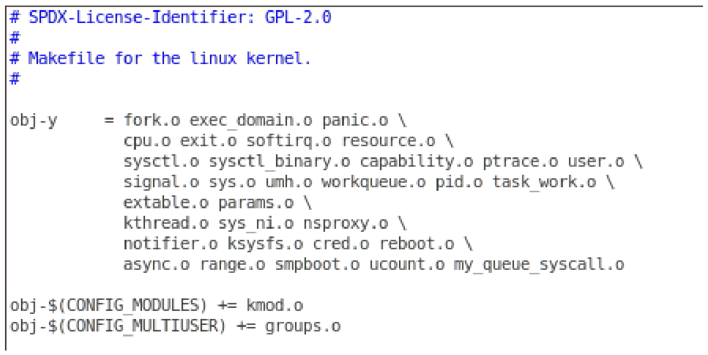
   c. "asmlinkage" is a very important gcc tag in the system call implementation. Asmlinkage tells the compiler that arguments are passed on the stack. When the system call handler calls the corresponding system call routine, it pushes the value of the general-purpose register onto the stack. So the system call routine is going to read the arguments passed by the system call handler from the stack, and that's what the asmlinkage tag is all about.
   d. sys_oslab_enqueue takes one integer as an argument and returns an int value, whereas sys_oslab_dequeue takes no argument and returns an int value.
3. Create a C program for the new system calls
   a. Command line:
      sudo gedit /usr/src/linux-4.20.11/kernel/my_queue_syscall.c

b.



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

#define MAXSIZE 500

int queue[MAXSIZE];
int front = 0;
int rear = 0;
int i,j,res = 0;

SYSCALL_DEFINE1(oslab_enqueue, int, a)// argument:system call name, input datatype, input variable
{
        if(rear >= MAXSIZE - 1) //if queue full
        {
                printk(KERN_INFO "[Error] - QUEUE IS FULL------------\n");
                return -2;
        }
        for ( i = 0; i<=rear ;i++) // queue is not full, we check the value if they are equal one by one with for loop
        {
                if (a == queue[i])// if new input is redundant
                {
                        printk(KERN_INFO "[Error] - Already existing value \n");
                        return a;
                }
        }

        queue[rear] = a ;// if new input makes sense

        printk(KERN_INFO "[System call] oslab_enqueue(); ------\n");// use I/O to visualize our queue
        printk("Queue Front--------------------\n");
        for (i = front; i <= rear ; i ++)// print them one by one
                printk("%d\n", queue[i]);
        printk("Queue Rear--------------------\n");

        rear = rear + 1;// increase the queue size by 1

        return a;// return the input new element in queue
}


SYSCALL_DEFINE0(oslab_dequeue)// 0 means no input here just as what we define in the head file before
{
        if(rear == front) // if queue is empty
        {
                printk(KERN_INFO "[Error]- EMPTY QUEUE------------\n");
                return -2;
        }
        res = queue[front];// choose the element which is going to be dequeued
        rear = rear - 1;// decrease queue size to implement the dequeue operation

        for(i = front + 1; i < MAXSIZE ; i++)// since we dequeue the 1st element in queue, we have to move all the others one place on
                queue[i-1] = queue[i];

        printk(KERN_INFO "[System call] oslab_dequeue(); ------\n");// use I/O to visualize our queue
        printk("Queue Front--------------------\n");

        for (j = front; j < rear ; j ++)// print them one by one
                printk("%d\n", queue[j]);

        printk("Queue Rear--------------------\n");

        return res;// return the dequeued element
}
```

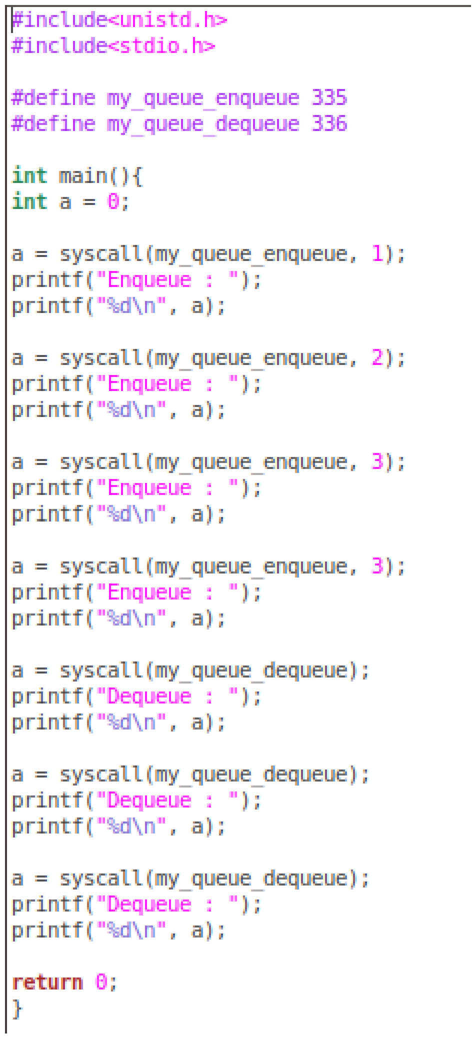
4. Create a makefile for your system call
    a. Create the Makefile with the following command:
       sudo gedit /usr/src/linux-4.20.11/kernel/Makefile
    b. Add to obj-y part to be included in kernel make.o object file, and file name is the same as our c file name before:



```makefile
# 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 my_queue_syscall.o

obj-$(CONFIG_MODULES) += kmod.o
obj-$(CONFIG_MULTIUSER) += groups.o
```

5. Compile the kernel:
    a. Change the directory to the kernel source folder:
       Command line: cd /usr/src/linux-4.20.11
    b. Compiled the kernel:
       Command line: sudo make
    c. Install the kernel:
       Command line: sudo make install

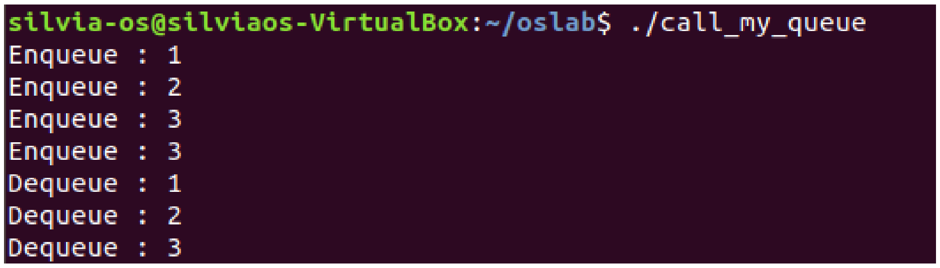
User mode:

1. Change the directory back to the user mode field at any place:
   For me: cd oslab
2. Write a test c program:
    a. Command line: sudo gedit call_my_queue.c
    b. Make a system call using a macro function called syscall()
    c. 

```
#include<unistd.h>
#include<stdio.h>

#define my_queue_enqueue 335
#define my_queue_dequeue 336

int main(){
int a = 0;

a = syscall(my_queue_enqueue, 1);
printf("Enqueue : ");
printf("%d\n", a);

a = syscall(my_queue_enqueue, 2);
printf("Enqueue : ");
printf("%d\n", a);

a = syscall(my_queue_enqueue, 3);
printf("Enqueue : ");
printf("%d\n", a);

a = syscall(my_queue_enqueue, 3);
printf("Enqueue : ");
printf("%d\n", a);

a = syscall(my_queue_dequeue);
printf("Dequeue : ");
printf("%d\n", a);

a = syscall(my_queue_dequeue);
printf("Dequeue : ");
printf("%d\n", a);

a = syscall(my_queue_dequeue);
printf("Dequeue : ");
printf("%d\n", a);

return 0;
}
```

3. Compile and run the test program:
    a. Compile: Command line: gcc call_my_queue.c –o call_my_queue

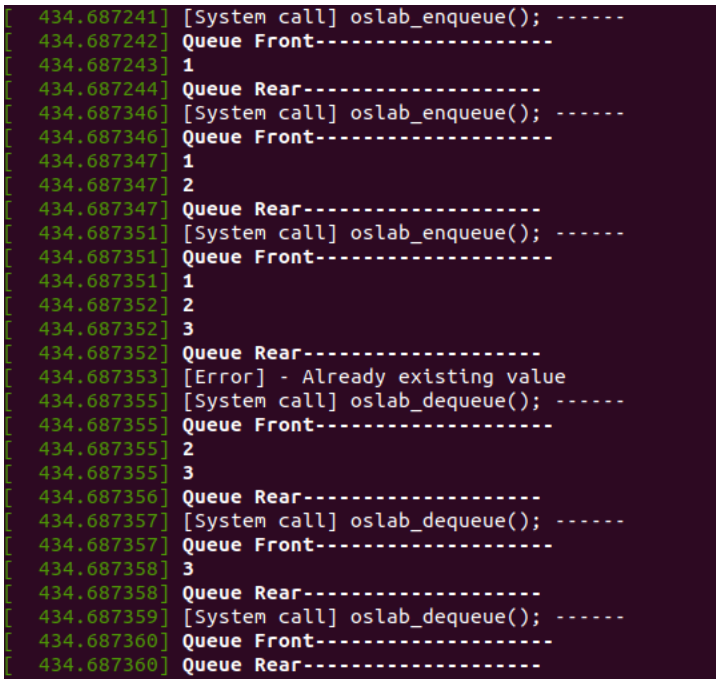
b. Run: Command line: ./call_my_queue

c. 

```
silvia-os@silviaos-VirtualBox:~/oslab$ ./call_my_queue
Enqueue : 1
Enqueue : 2
Enqueue : 3
Enqueue : 3
Dequeue : 1
Dequeue : 2
Dequeue : 3
```

4. Check the message of your kernel:

   a. Command line: dmesg

   b. 

```
[  434.687241] [System call] oslab_enqueue(); ------
[  434.687242] Queue Front-------------------
[  434.687243] 1
[  434.687244] Queue Rear-------------------
[  434.687346] [System call] oslab_enqueue(); ------
[  434.687346] Queue Front-------------------
[  434.687347] 1
[  434.687347] 2
[  434.687347] Queue Rear-------------------
[  434.687351] [System call] oslab_enqueue(); ------
[  434.687351] Queue Front-------------------
[  434.687351] 1
[  434.687352] 2
[  434.687352] 3
[  434.687352] Queue Rear-------------------
[  434.687353] [Error] - Already existing value
[  434.687355] [System call] oslab_dequeue(); ------
[  434.687355] Queue Front-------------------
[  434.687355] 2
[  434.687355] 3
[  434.687356] Queue Rear-------------------
[  434.687357] [System call] oslab_dequeue(); ------
[  434.687357] Queue Front-------------------
[  434.687358] 3
[  434.687358] Queue Rear-------------------
[  434.687359] [System call] oslab_dequeue(); ------
[  434.687360] Queue Front-------------------
[  434.687360] Queue Rear-------------------
```