COSE341 Operating Systems

1st Assignment Report

학과: 경영학과

학번: 2017120486

이름: 서천함

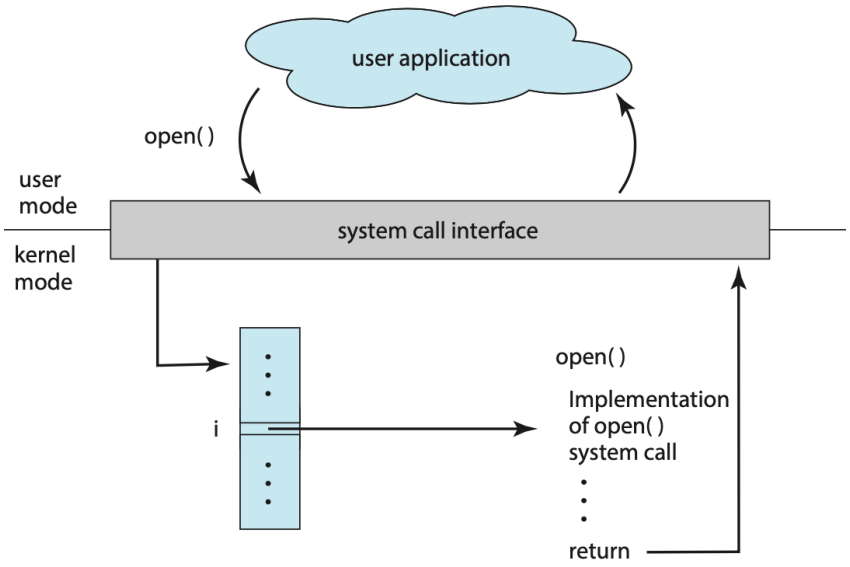
제출 날자: 2021 년 4 월 27 일

프리 데이 사용 일수: 0

1. **Development Environment**
   - Ubuntu 18.04.2 LTS
   - Linux Kernel Ver. 4.20.11

2. **Linux System Call**

System call is the fundamental interface between a user application and the Linux kernel. More specifically, a system call is a C function in kernel space that user space programs call to request for a kernel service. Typically we design programs according to an application programming interface(API). We do not call system calls directly in our code, but instead use wrapper functions from the C standard library. The functions make up an API invoke the actual system calls. There're several reasons for this: firstly, using an API can expect the program to compile and run on any system that support the same API; secondly, actual system calls can often be more difficult than API available. Lastly, the run-time environment(RTE) provides a system-call-interface, which maintains a table indexed to the numbers associated with each system call(sys_call_table). It then intercepts the call and invokes the intended system call in OS kernel and returns the status of the system call.



A system call usually takes the form of a trap to a specific location in the interrupt vector. This trap can be executed by a generic trap instruction(some systems have a specific syscall instruction to invoke a system call). When a system call is executed, it is treated by the hardware as a software interrupt. Control passes to the system-call service routine in OS and mode bit is set to 0 (kernel mode).

The kernel examines the interrupting instruction to determine what system call has occurred(refer to the sys_call_table). The kernel verifies the parameters and execute the request. The caller need only obey the API and details of OS interface are hidden and managed by RTE. The C library takes the value returned by system call interface and passes it back to the user program.

3. Modification of Source Code

a) syscall_64.tbl

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

Every system call has a fixed number. To wire up our new system calls for x86 platforms, we need to update the syscall tables. syscall_64.tbl defines the 64-bit system call numbers and their entry vectors. So we can add the new system calls to the system call table.

b) syscalls.h

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

In order to use system call, we need a corresponding function prototype in syscalls.h, which defines the corresponding function prototype of system calls.
System call handler is the assembly code while the system call routine is the C code. When the system call handler calls the corresponding system call routine, the gcc compiler need to pass parameters through the stack instead of through the register by default. Adding macro “asmlinkage” before the function definition is to indicate that these functions pass parameters through stack, not from the register.

c) my_queue_syscall.c

The main entry point for new system calls will be called sys_oslab_enqueue() and sys_oslab_dequeue but we add this entry point with SYSCALL_DEFINEn() macro rather than explicitly. The ‘n’ indicates the number of arguments to the system call and the macro takes the system call name followed by (type, name)pairs for the parameters as arguments.

```
SYSCALL_DEFINE1(oslab_enqueue, int, a){
SYSCALL_DEFINE0(oslab_dequeue){
```

As for the implementation of enqueue and dequeue functions, initially we set front and rear to zero. If the queue is full, we use printk() function from the kernel interface to print the error messages to the kernel log and return. If the queue is not full, we traverse the queue to check if the value to be inserted is already existed. If there’s no duplicate value, we insert int a into the queue and increase variable rear by one. If the queue is empty, we can use printk() to report error message and end the program. We store the value of the first element of queue in variable res and move all other elements forward to fill the place of the removed element. Lastly, we decrease rear by one and return res.

d) Makefile

```
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
```

Adding the line above is to ensure my_queue_syscall.c file is compiled and included in the kernel source code.

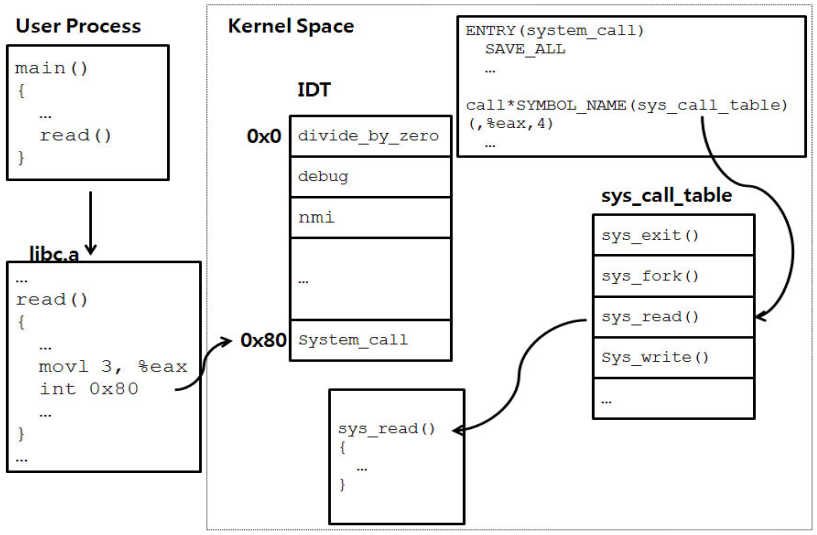
e) call_my_queue.c

```
#define my_queue_enqueue 335
#define my_queue_dequeue 336

a = syscall(my_queue_enqueue, 1);
a = syscall(my_queue_dequeue);
```

This program is a user application which invokes the newly added system calls and used to test them. Here we use syscall macro by invoking syscall(system call number, arguments…). The number of each system call(335 and 336) is predefined and designated in the kernel header file. We can check the message of the kernel by running dmesg command.

f) Overall Flow



Based on the graph above, we can summarize the overall flow of calling a system call. System call is a kernel-level service provided by Linux kernel. Initially, the system call is invoked in a user application via wrapper functions in libc. libc contains a function and when it is called in the user process, 0x80 trap instructions in IDT are executed. 0x80 trap instruction reserves the system call and the trap handler checks the system call

number and calls the corresponding system call stored in sys_call_table(syscall_64.tbl), which are sys_my_enqueue or sys_my_dequeue in this assignment. After the kernel service is finished, the result value will be returned back to the user process.

4. Execution Result

```
osta@osta-VirtualBox:~/oslab$ ./call_my_queue
Enqueue : 1
Enqueue : 2
Enqueue : 3
Enqueue : 3
Dequeue : 1
Dequeue : 2
Dequeue : 3
osta@osta-VirtualBox:~/oslab$ dmesg
```

```
[  111.378102] [System call] oslab_enqueue(); -------
[  111.378103] Queue Front ------------------
[  111.378104] 1
[  111.378104] Queue Rear ------------------
[  111.378275] [System call] oslab_enqueue(); -------
[  111.378275] Queue Front ------------------
[  111.378276] 1
[  111.378276] 2
[  111.378277] Queue Rear ------------------
[  111.378280] [System call] oslab_enqueue(); -------
[  111.378280] Queue Front ------------------
[  111.378281] 1
[  111.378281] 2
[  111.378281] 3
[  111.378282] Queue Rear ------------------
[  111.378284] [Error] - EXISTED VALUE --------------
[  111.378286] [System call] oslab_dequeue(); -------
[  111.378286] Queue Front ------------------
[  111.378287] 2
[  111.378287] 3
[  111.378287] Queue Rear ------------------
[  111.378289] [System call] oslab_dequeue(); -------
[  111.378289] Queue Front ------------------
[  111.378290] 3
[  111.378290] Queue Rear ------------------
[  111.378292] [System call] oslab_dequeue(); -------
[  111.378292] Queue Front ------------------
[  111.378293] Queue Rear ------------------
osta@osta-VirtualBox:~/oslab$ sudo
```

5. Problems and Solutions

The first few times when I executed call_my_queue, I always got the same wrong result that both newly added system calls returned -1. I thought that maybe call_my_queue.c went wrong, so I reprogrammed this file and ensured the program itself to be correct. However, I still couldn't get the correct returned value. I started to check each file I modified but they turned out to be all correct. I thought that probably something went wrong with the installation of development environment. So I uninstalled the Ubuntu from VirtualBox and restarted from the very beginning.

Unfortunately, I still got the same wrong result. When I was about to give up, I checked printed error message `[drm:vmw_host_log [vmwgfx]] *ERROR* Failed to send host log message.` and searched the solution to fix this bug on Stack Overflow. They suggested me to set `Display-Graphics Controller: VBoxVGA` and finally I got the expected correct result. The lesson I learned from this problem is: Always check and see the error message first.

[References]

https://www.kernel.org/doc/html/v4.10/process/adding-syscalls.html
https://www.programmersought.com/article/90795521947/
SILBERSCHATZ, A., 2018. OPERATING SYSTEM CONCEPTS. [S.l.]: JOHN WILEY.