Operating System Project #1 Tutorial

Understanding and Implementing

System Calls

2024. 03. 24

Purpose

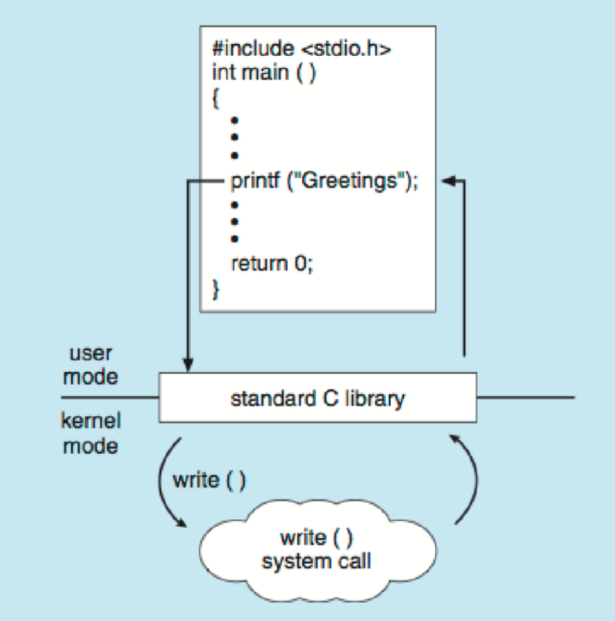
- **Modify and compile source code for Linux to add new system calls**

- **Create a user application that uses added system calls**

- **Understand how a system call works on Linux**

System call

- **Path to enter kernel mode from user mode**
  – Required to use the protected (privileged) service provided by the kernel
- **User programs usually use the high-level Application Programming Interface(API) rather than using direct system calls**
  – Provides a more convenient interface for users



```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

Results up to the Project #0

- Install Virtual Box
- Install the Linux on Virtual Machine
- Compile Linux-4.20.11 Kernel

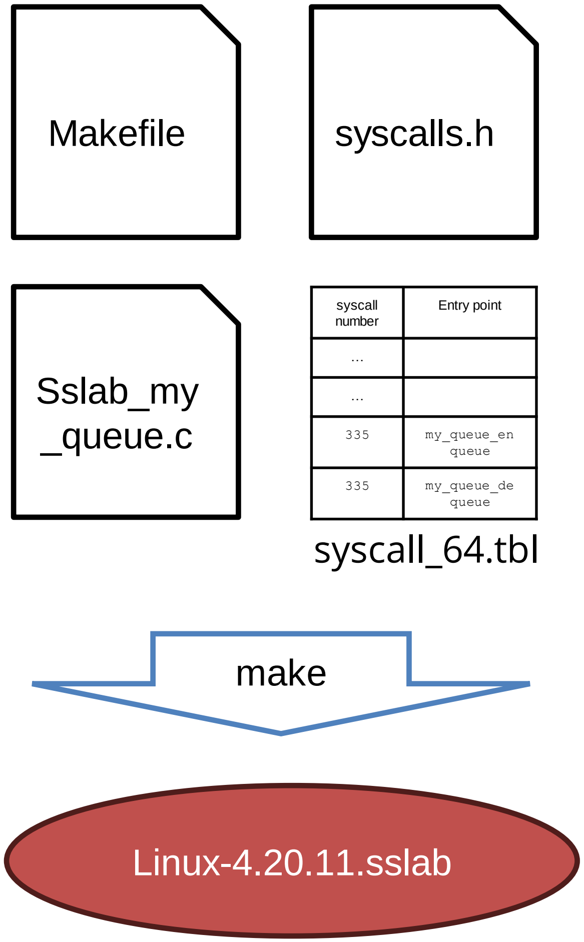
Project #1 is to proceed after the Project #0

Contents of the Project #1

- **Add system call**
  - Define a **queue** structure that 1) enqueues integer values and 2) dequeues the integer **in the order it was enqueued**
  - **Two system calls** to enqueue and dequeue an integer value
- **Condition**
  - Enqueue system call takes an “integer” value as its parameter
  - When the value to be enqueued already exists in the queue, the system call handler should not newly store the value in the queue
  - When dequeue system call is called, its handler pulls the oldest integer off the queue.
    - The dequeued (returned) value is the oldest integer, i.e. the integer enqueued first into the queue.
  - User application calls enqueue and dequeue system calls at least three times
    - When calling enqueue system call, it should generate a **random** number and pass it as a parameter

Contents of the Project #1

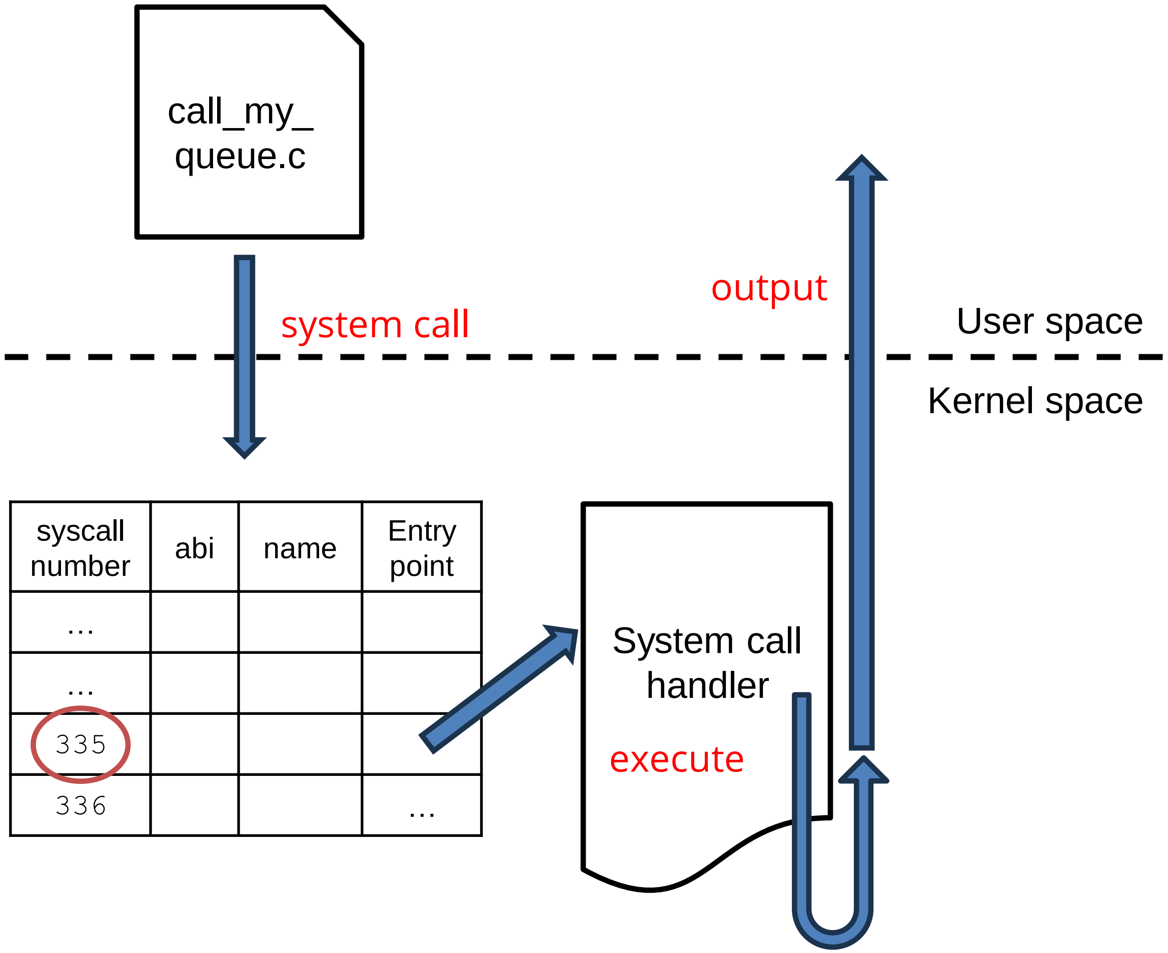
Compile your kernel



syscall number	Entry point
...	
...	
335	my_queue_enqueue
335	my_queue_dequeue

syscall_64.tbl

Workflow

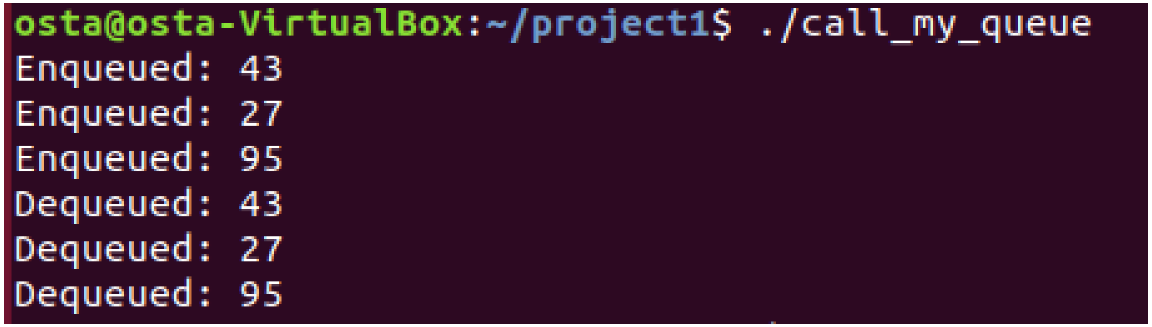


syscall number	abi	name	Entry point
...			
...			
335			
336			...

syscall_64.tbl

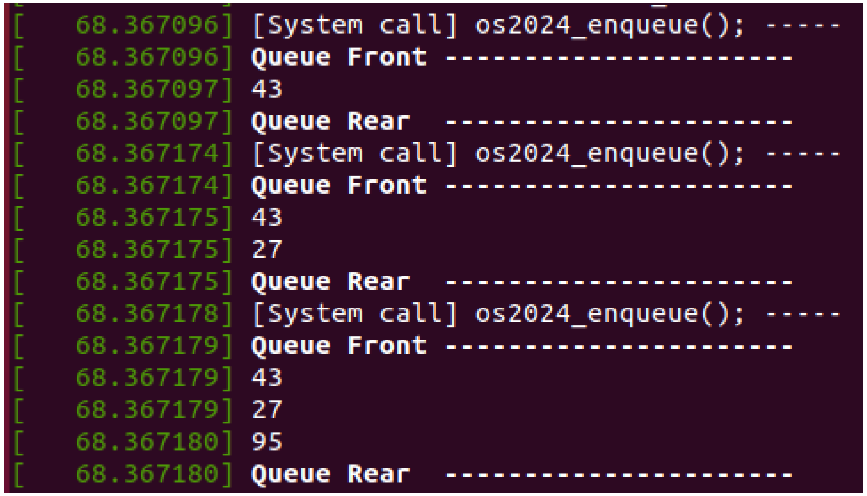
Result example

- **Application output**

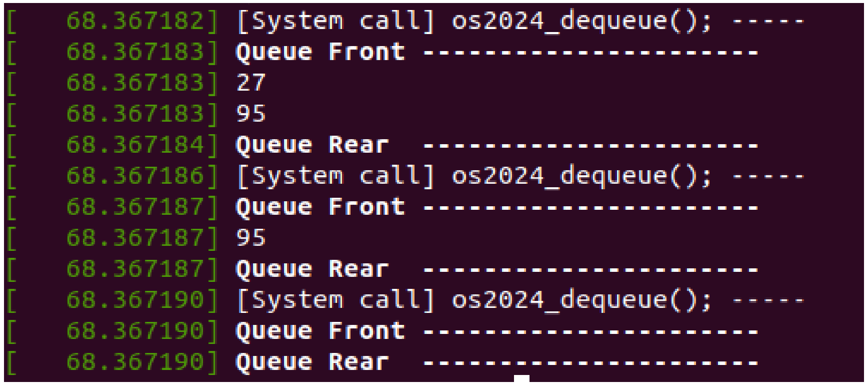


```
osta@osta-VirtualBox:~/project1$ ./call_my_queue
Enqueued: 43
Enqueued: 27
Enqueued: 95
Dequeued: 43
Dequeued: 27
Dequeued: 95
```

- **Kernel log output (check 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  ----------------------
```

1. Implementation on Kernel

- **Modify and create the following files specified in the project instruction**

A. syscall_64.tbl

- A file that collects symbolic information about the names of system call handler (function)
- ***New system call number should be added***

B. syscalls.h

- Define ***prototypes*** and register table for added system call functions

C. sslab_my_queue.c

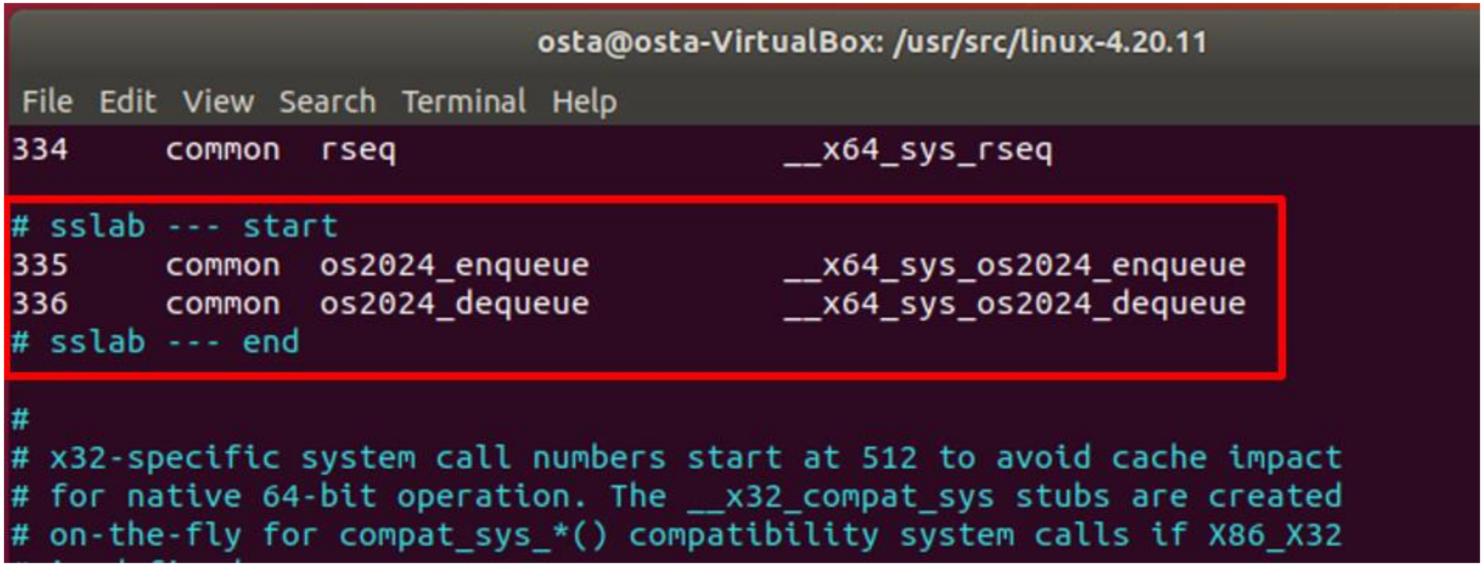
- Create in /usr/src/linux-4.20.11/kernel/
- ***Source of new system call handlers*** to be added

D. Makefile

- Modifying /usr/src/linux-4.20.11/kernel/Makefile
- ***Add object to be compiled with (sslab_my_queue.o)***

1-A. syscall_64.tbl

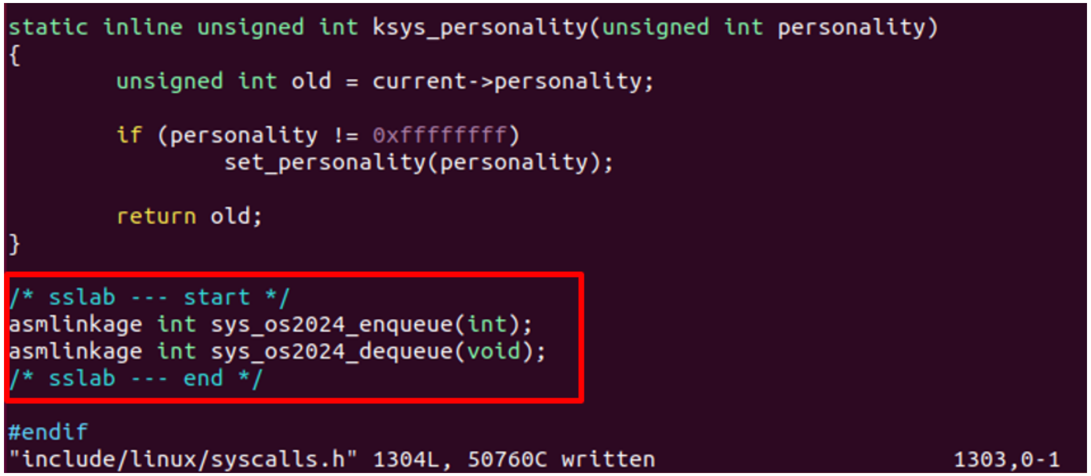
- **Maintain unique numbers for all system calls supported by Linux**
  - (linux)/arch/x86/entry/syscalls/syscall_64.tbl
    - **※ (linux) : /usr/src/linux-4.20.11 (Root folder where the kernel's source code is stored)**
- **Symbol information table for system calls**
  - Information managed by Linker
    - Table that stores addresses of system call functions located in various spots in Linux kernel code directories
    - Linker automatically manages the system call addresses and locations for kernel compile


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
```

1-B. syscalls.h

- **Define prototypes of system call functions**
  - Add them to `(linux)/include/linux/syscalls.h`
  - `asmlinkage int sys_os2024_enqueue(int);`
  - `asmlinkage int sys_os2024_dequeue(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
```

- **Why use asmlinkage?**
  - System call calls from trap instructions
  - Handling routine for the trap instructions – created with assembly code
  - Declaring asmlinkage before function enables calling of C functions within assembly code (trap routines)

1-C. sslab_my_queue.c

- **System call handlers (functions) to add**
  - Implement what system calls will actually task (system call handlers)
    - Create in `/usr/src/linux-4.20.11/kernel/`
  - Declare a queue in the form of an integer array as **global variable**
  - **Skeleton implementation**

    -
      ```
      SYSCALL_DEFINE1(os2024_enqueue, int, a){
                  ...
      }
      ```

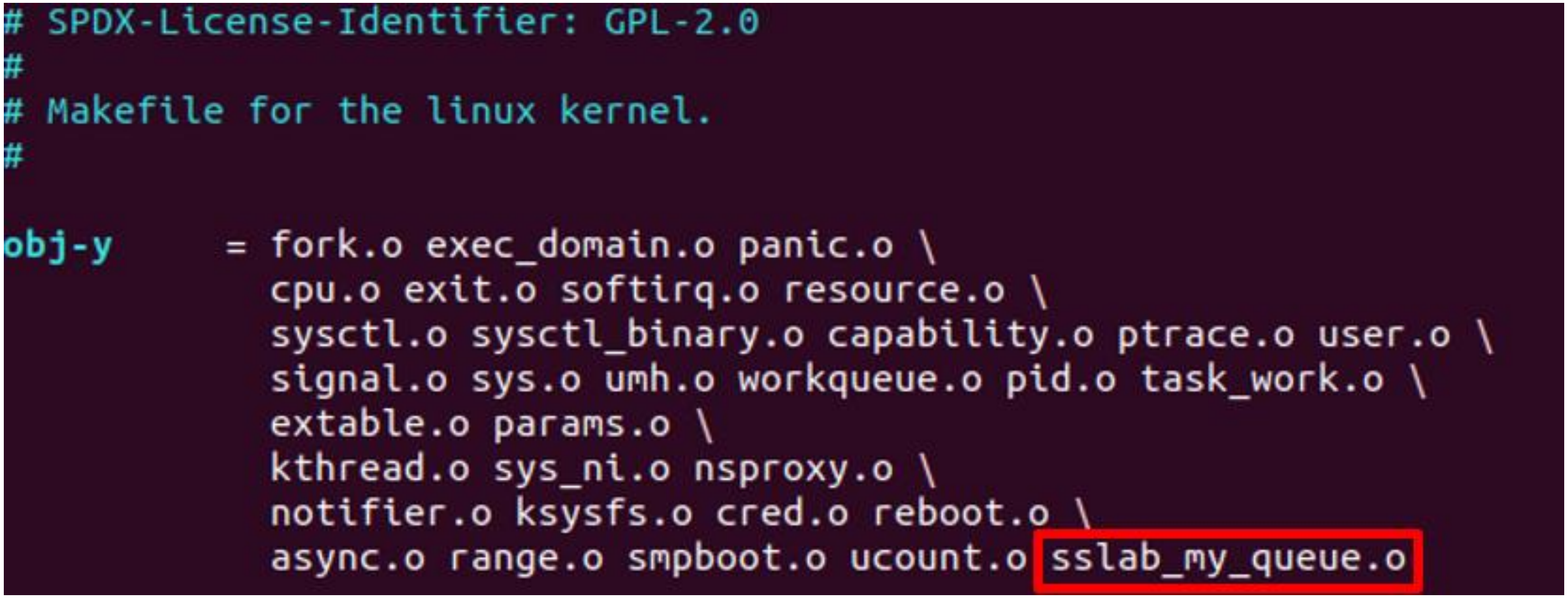
    -
      ```
      SYSCALL_DEFINE0(os2024_dequeue){
                  ...
      }
      ```
  - `SYSCALL_DEFINEx`: Macros for implementing system calls with "x number of parameters"
    - Defined in (linux)/include/linux/syscalls.h
- **Use the following headers**
  - `<linux/syscalls.h>`
  - `<linux/kernel.h>`
  - `<linux/linkage.h>`

1-C. sslab_my_queue.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 */
}
```

1-D. Makefile

- /usr/src/linux-4.20.11/kernel/Makefile
- **Add things to be compiled within kernel (obj-y part)**



```
# 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 is the same as its own C file name**
  - E.g., sslab_my_queue.c -> sslab_my_queue.o

Kernel Compile

- **If you have completed all of the above, run the following commands in the `/usr/src/linux-4.20.11` (kernel source code folder)**

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

2. Implmenetation of user application

- **Create an application that uses the added system calls**
  - Use a macro function called **syscall()** to call the system
    - How to use it: pass the system call number and parameters
      - `syscall(335, …);`
        - » Add header `<unistd.h>`
      - `#define my_queue_enqueue 335` // After declaring the system call number
        `syscall(my_queue_enqueue, …);` more convenient to use as

- **Application compile**
  - If the application source file is saved as **call_my_queue.c**
    - `gcc call_my_queue.c -o call_my_queue`
      - Means that "compile `call_my_queue.c` and create an executable (e.g., `call_my_queue`)"
    - Run to the `./call_my_queue` and check the `dmesg` to see if the desired output has come from by `printk` as well

> Note that code changes may not apply if they are not Linux 4.20.11 kernels.
> If the system call you added is not loaded, check the kernel version by the command '**`uname -r`**'.
> Then, reboot, press "left shift key", and select the proper kernel version (the one you implemented your system calls) in Advanced options for Ubuntu

Project submission

- **List of files to submit (Check the Project #1_instruction.pdf)**
  - Report
    - See **Project #1_instruction.pdf** for the details to explain in the report
  - All source code files you modified or newly implemented
    - Comments explaining the roles of important variables and functions should be included in the source code
  - Execution result file
    - Make the execution results into the text log (result.txt) and submit it

Tip

- **Most questions can be solved through web searching**
  - There are a lot of web documents related to the Linux kernel
  - To use your time ***effectively***, first, look for solutions online.
    If you encounter specific issues you can't resolve after searching,
    do not hesitate to ask questions by contacting the TAs.

- **Individual Q&A**
  - Blackboard Q&A Bulletin
  - Email: osta@os.korea.ac.kr