클립아트, 만화 영화이(가) 표시된 사진

자동 생성된 설명

COSE341(00)

Operating system

Project #1 Report

Your name : Yeyoung Park

Student ID : 2020160027

Your Department : Mathematics

Submission Date : 2024.10.20

Number of free days used for this project : 6 days

# Development Environment

* 1. Host OS : Windows 10
  2. VirtualBox : version 7.1.2
  3. Ubuntu : 18.04.2 LTS
  4. Linux Kernel : 4.20.11.sslab

# Explanation of system calls on Linux (including call routines) [20 pts]

# System call은 User space에 있는 Application과 Kernel을 이어주는 중요한 Interface이다. System call을 통해 App은 파일 읽기/쓰기, 메모리 할당, 프로세스 관리와 같은 커널 서비스에 접근할 수 있다. User mode, 즉, App이 실행되는 환경은 App이 시스템에 영향을 미치지 않도록 보호하기 위해 제한된 접근 권한을 가지고 있다. 반면 Kernel mode는 모든 접근을 할 수 있다.(즉, 높은 권한을 가지고 있다.) 따라서 App은 직접 하드웨어나 시스템 리소스에 접근할 수 없으므로, System call을 통해 kernel에 필요한 작업을 요청한다.

# 대부분의 system call은 stdlib.c에 정의된 wrapper function을 통해 호출된다. 예를 들자면, read()함수는 파일을 읽는 system call인 sys\_read를 감싸는 wrapper function이다. 이 wrapper function은 CPU에 trap을 발생시켜 kernel mode로 전환한다. 여기서는 syscall명령어를 통해 kernel mode로 전환된다. 이후 CPU는 syscall 명령어를 통해 전달된 system call number를 기반으로 system call handler를 찾는다. Linux kernel은 system call number와 system call hander를 서로 매핑하는 sys\_call\_table을 가지고 있다. (여기에서는 syscall\_64.tbl) 이를 통해 kernel은 요청된 system call handler를 실행하여 필요한 작업을 수행한다. 작업이 완료되면, 그 결과를 App으로 반환한다. 이후 CPU는 user mode로 다시 전환된다.

# Implementation [70 pts]

### (linux)/arch/x86/entry/syscalls/syscall\_64.tbl (4 pts)

# yeyoung --- start

335 common os2024\_push \_\_x64\_sys\_os2024\_push

336 common os2024\_pop \_\_x64\_sys\_os2024\_pop

# yeyoung --- end

### (linux)/include/linux/syscalls.h (4 pts)

/\* yeyoung --- start \*/

asmlinkage int sys\_os2024\_push(int a);

asmlinkage int sys\_os2024\_pop(void);

/\* yeyoung --- end \*/

### (linux)/kernel/sslab\_my\_stack.c (40 pts)

/\* 2024 Fall COSE341 Operating System \*/

/\* Project 1 \*/

/\* Yeyoung Park \*/

#include <linux/syscalls.h>

#include <linux/kernel.h>

#include <linux/linkage.h>

#include <linux/printk.h>

#include <linux/errno.h>

#define STACK\_SIZE 100

static int stack[STACK\_SIZE];

static int top = -1;

void print\_stack(void) {

int i;

printk(KERN\_INFO "Stack Top ---------------------\n");

for (i = top; i >= 0; i--) {

printk(KERN\_INFO "%d\n", stack[i]);

}

printk(KERN\_INFO "Stack Bottom ------------------\n");

}

SYSCALL\_DEFINE1(os2024\_push, int, a) {

// overflow check

if (top >= STACK\_SIZE - 1) {

printk(KERN\_ALERT "Stack Overflow: Cannot push %d\n", a);

return -ENOMEM; // return memory lackage error

}

top++;

stack[top] = a;

printk(KERN\_INFO "[System Call] os2024\_push(); ---\n");

print\_stack();

return 0;

}

SYSCALL\_DEFINE0(os2024\_pop) {

int value;

// underflow check

if (top < 0) {

printk(KERN\_ALERT "Stack Underflow: No values to pop\n");

return -ENODATA; // return no data error

}

value = stack[top];

top--;

printk(KERN\_INFO "[System Call] os2024\_pop(); ---\n");

print\_stack();

return value;

}

### (linux)/kernel/Makefile (2 pts)

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\_stack.o

### call\_my\_stack.c (20 pts)

# #include <stdio.h>

# #include <unistd.h>

# #include <sys/syscall.h>

# #include <errno.h>

# #include <stdlib.h>

# #include <time.h>

# #define my\_stack\_push 335

# #define my\_stack\_pop 336

# int main() {

# int res;

# int random\_value;

# int i;

# srand(time(NULL));

# for (i = 0; i < 10; i++) {

# random\_value = rand();

# res = syscall(my\_stack\_push, random\_value);

# if (res == 0) {

# printf("Push: %d\n", random\_value);

# } else {

# printf("Error %s\n", strerror(errno));

# }

# }

# for (i = 0; i < 10; i++) {

# res = syscall(my\_stack\_pop);

# if (res >= 0) {

# printf("Pop: %d\n", res);

# } else {

# printf("Error %s\n", strerror(errno));

# }

# }

# return 0;

# }