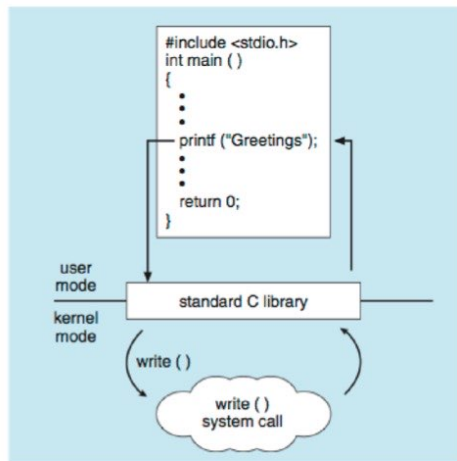


과제의 목적

- 리눅스의 소스 코드를 수정하고 컴파일하여 새로운 시스템 콜 추가
- 추가된 시스템 콜을 사용하는 사용자 응용 프로그램 제작
- 리눅스에서의 시스템 콜 동작 과정 이해

시스템 콜

- **User mode에서 kernel mode로 진입하기 위한 통로**
 - 커널에서 제공하는 protected 서비스를 이용하기 위하여 필요
- **유저 프로그램은 보통 직접 시스템 콜을 이용하기보다, high-level Application Programming Interface (API)를 이용**
 - 유저에겐 보다 편리한 인터페이스의 서비스를 제공함



0차 과제까지의 결과

- Virtual Box 설치
- Virtual Machine에 Linux 운영체제 설치
- Linux-4.20.11 커널 컴파일

1차 과제는 0차 과제 과정 이후로 진행하면 됩니다.

1차 과제 내용

• 시스템 콜 추가

- 시스템 콜 코드에 Integer값을 저장하는 **Last in First Out(LIFO) 형태의 Stack** 선언
- 시스템 콜은 Stack에 Push, Pop하는 역할을 하도록 작성

• 프로그램 조건

- Push 함수는 int 변수를 인자로 갖는다
- **Push 함수를 통해 추가하려는 값이 이미 Stack에 있는 값과 같으면 Stack에 추가하지 않는다**
- Pop 함수는 가장 나중에 들어온 값을 Stack에서 제거하고, 그 값을 return 한다

결과

• 응용 프로그램 출력

```
oslab@oslab-VirtualBox:~/ejko$ ./oslab_call_stack
Push 1
Push 1
Push 2
Push 3
Pop 3
Pop 2
Pop 1
```

• 커널 로그 출력 (dmesg 명령어)

```
[ 45.178975] [System Call] os2023_push :
[ 45.178976] Stack Top -----
[ 45.178977] 1
[ 45.178977] Stack Bottom -----
[ 45.178982] [System Call] os2023_push :
[ 45.178982] Stack Top -----
[ 45.178982] 1
[ 45.178982] Stack Bottom -----
[ 45.178983] [System Call] os2023_push :
[ 45.178983] Stack Top -----
[ 45.178984] 2
[ 45.178984] 1
[ 45.178984] Stack Bottom -----
[ 45.178985] [System Call] os2023_push :
[ 45.178985] Stack Top -----
[ 45.178985] 3
[ 45.178986] 2
[ 45.178986] 1
[ 45.178986] Stack Bottom -----
```

```
[ 45.178987] [System Call] os2023_pop :
[ 45.178987] Stack Top -----
[ 45.178987] 2
[ 45.178988] 1
[ 45.178988] Stack Bottom -----
[ 45.178989] [System Call] os2023_pop :
[ 45.178989] Stack Top -----
[ 45.178989] 1
[ 45.178989] Stack Bottom -----
[ 45.178990] [System Call] os2023_pop :
[ 45.178990] Stack Top -----
[ 45.178990] Stack Bottom -----
```

커널 소스코드의 수정

• 과제 handout에 명시한 4개의 파일 수정 및 작성

- syscall_64.tbl
 - 시스템 콜 함수들의 이름에 대한 심볼정보를 모아 놓은 파일
 - 새로 추가할 시스템 콜 번호
- syscalls.h
 - 추가한 시스템 콜 함수들의 prototype 정의 및 테이블 등록
- oslab_call_stack.c
 - /usr/src/linux-4.20.11/kernel/ 하위에 작성
 - 새로 추가할 시스템 콜의 소스
- Makefile
 - /usr/src/linux-4.20.11/kernel/Makefile 수정
 - oslab_call_stack.o 오브젝트 추가

vi / vim

- 개요

- CLI 환경에서 텍스트 편집(코딩)을 할 수 있게 만들어주는 텍스트 에디터
- vim은 vi improved의 약자로, vi를 기반으로 편의기능이 업그레이드 된 소프트웨어
- gedit과 다르게, GUI 환경이 아니어도 사용이 가능

- 영상 튜토리얼

- vim을 제대로 가르쳐 줍니다 🧐 (개발자라면 한번쯤 꼭 쓴다는 Vim)
- <https://www.youtube.com/watch?v=cY0JxzENBjg>

* vim이 아닌 VS Code, gedit 등 다른 도구로 코드 작성해도 무방



1) syscall_64.tbl

- 리눅스에서 제공하는 모든 시스템 콜의 고유 번호를 저장

- (linux)/arch/x86/entry/syscalls/syscall_64.tbl
 - ※ (linux) 는 /usr/src/linux-4.20.11 (커널의 소스코드가 저장된 루트 폴더)

- 시스템 콜의 symbol 정보 집합

- 링커에 의해 관리되는 정보
 - Linux kernel source tree에 흩어져 있는 시스템 콜 함수의 주소들을 저장하는 테이블
 - 시스템 콜 주소는 링커가 자동으로 관리

```
oslab@oslab-VirtualBox: /usr/src/linux-4.20.11/arch/x86/entry/syscalls
File Edit View Search Terminal Help
331      common  pkey_free          __x64_sys_pkey_free
332      common  statx              __x64_sys_statx
333      common  io_pgetevents     __x64_sys_io_pgetevents
334      common  rseq              __x64_sys_rseq
#oslab
335      common  os2023_push       __x64_sys_os2023_push
336      common  os2023_pop        __x64_sys_os2023_pop
#
# 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
```


2) syscalls.h

• 시스템 콜 함수들의 **prototype**을 정의

- (linux)/include/linux/syscalls.h 파일에 등록
- `asm linkage int sys_os2023_push(int)`
- `asm linkage int sys_os2023_pop(void)`

```
static inline unsigned int ksys_personality(unsigned int personality)
{
    unsigned int old = current->personality;

    if (personality != 0xffffffff)
        set_personality(personality);

    return old;
}

/*oslab*/
asm linkage void sys_os2023_push(int);
asm linkage int sys_os2023_pop(void);
```

• 왜 **asm linkage**를 사용하는가?

- 시스템 콜 호출은 int 80인터럽트 핸들러에서 호출
- 인터럽트 핸들러는 assembly 코드로 작성됨
- `asm linkage` 를 함수 앞에 선언하면,
assembly code에서도 C함수 호출이 가능해짐

3) my_queue_syscall.c

• 추가할 시스템 콜 소스

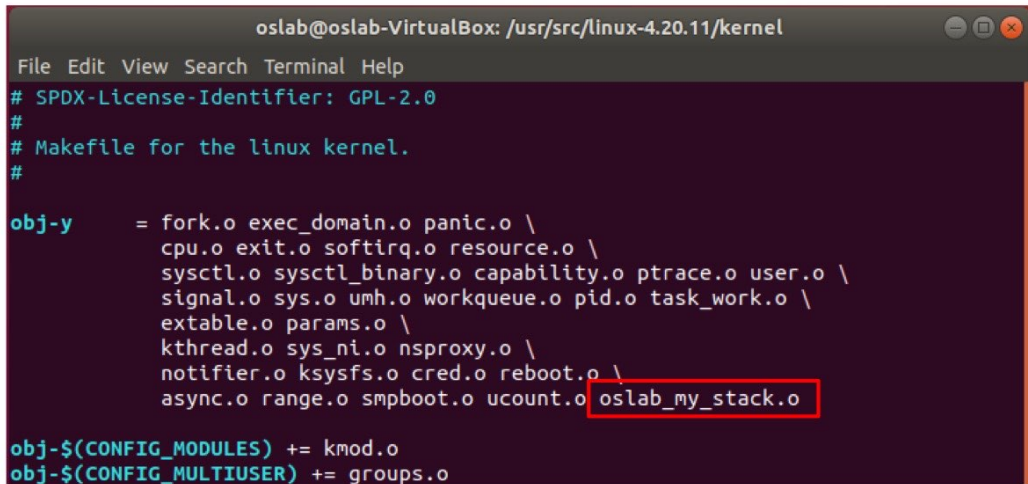
- 시스템 콜이 실제로 할 일을 구현
 - /usr/src/linux-4.20.11/kernel/ 하위에 작성
- int 배열 형태의 stack을 전역 변수로 선언
- **push, pop 함수 구현**
 - SYSCALL_DEFINE1(os2023_push, int, a){
 ...
}
 - SYSCALL_DEFINE0(os2023_pop){
 ...
}
- SYSCALL_DEFINEx: “파라미터의 개수가 x개” 인 시스템콜 구현을 위한 매크로
 - linux/include/linux/syscalls.h 에 정의되어 있음

• 헤더 추가

- <linux/syscalls.h>
- <linux/kernel.h>
- <linux/linkage.h>

4) Makefile

- /usr/src/linux-4.20.11/kernel/Makefile
- kernel make 시에 포함되도록 obj-y 부분에 추가



```
oslab@oslab-VirtualBox: /usr/src/linux-4.20.11/kernel
File Edit View Search Terminal Help
# 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 oslab_my_stack.o

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

- .o 오브젝트 파일명은 자신의 c 파일이름과 동일
 - 예) oslab_my_stack.c -> oslab_my_stack.o

커널 컴파일

- 위 내용들을 모두 완료하였으면,
`/usr/src/linux-4.20.11` 에서 (커널 소스코드 폴더) 다음 명령어 실행
 `sudo make`
 `sudo make install`

유저 Application 작성

• 추가한 시스템 콜을 사용하는 application 작성

- syscall() 이라는 매크로 함수를 이용하여 시스템 콜 호출
 - 사용법 : 시스템 콜 번호와 인자를 넣어서 사용
 - syscall(335, ...);
 - » <unistd.h> 헤더 추가
 - #define my_stack_push 335 // 시스템 콜 번호 선언 후에
syscall(my_stack_push, ...); 으로 사용하면 더 편리

• Application 컴파일

- Application 소스 파일이 call_my_queue.c라면
 - gcc oslab_call_stack.c -o oslab_call_stack
 - “oslab_call_stack.c 를 컴파일해서 oslab_call_stack 라는 이름의 실행 파일을 만들어라”
 - ./oslab_call_stack 로 실행 후 dmesg를 통해서
oslab_call_stack.c의 printk로 원하던 출력이 나왔는지를 확인

이때 Linux 4.20.11 커널이 아닌 경우 변경 내용이 적용되지 않을 수도 있음.
추가한 시스템콜이 불러와지지 않는다면

`uname -r` 명령어를 통해 커널 버전을 확인 후 재부팅 후 “왼쪽 쉬프트키”를 누르고
Advanced options for Ubuntu 에서 해당 커널 버전 선택

과제 제출

• 제출할 파일 목록 (1차과제_instruction.pdf 파일 참고)

- 보고서
 - 리눅스의 시스템 콜에 대한 설명, 수정 및 작성한 부분과 설명, 실행 결과 스냅샷 등
 - 자세한 사항은 **1차과제_instruction.pdf** 참고
- 직접 작성한 소스 파일 전체
 - 소스 코드에는 중요 변수와 함수의 역할에 대한 주석 작성
- 실행 결과 파일
 - 실행한 결과를 result.txt로 만들어서 제출