운영체제 1차 과제 Tutorial

Kernel System Call 이해와 구현

고려대학교 운영체제 연구실

2019년 4월 2일

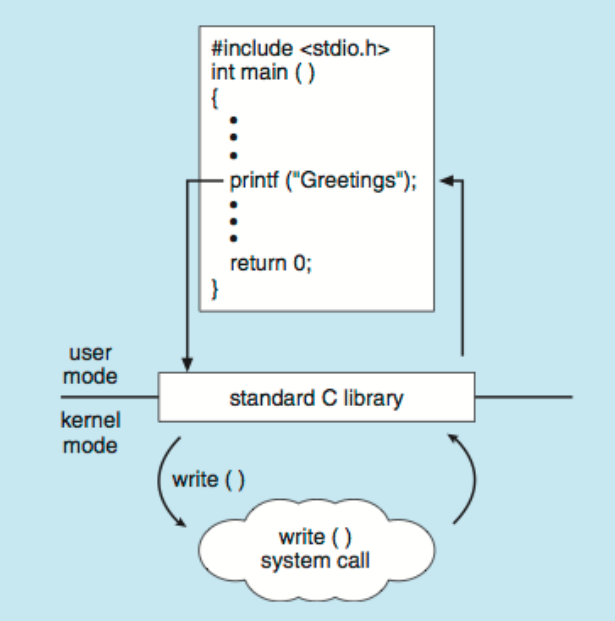
과제의 목적

- 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**
  - 커널에서제공하는 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



0차과제 까지의 결과

- **Virtual Box** 설치
- **Virtual Machine**에 **Linux** 운영체제 설치
- **Linux-4.20.11** 커널 컴파일

1차 과제는 **0**차 과제 과정 이후로 진행하면 됩니다.

1차과제 내용

- 시스템 콜 추가
  - 시스템 콜 코드에 Integer값을 저장하는 **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

```
osta@osta-VirtualBox:~/oslab$ ./call_my_stack
Push: 1
Push: 2
Push: 3
Pop: 3
Pop: 2
Pop: 1
osta@osta-VirtualBox:~/oslab$
```

- 커널 로그 출력 **(dmesg 명령어)**

```
[  337.673626] [System call] oslab_push(): Push 1
[  337.673627] Stack Top ----------
[  337.673627] 1
[  337.673628] Stack Bottom -------
[  337.673629] [System call] oslab_push(): Push 2
[  337.673630] Stack Top ----------
[  337.673630] 2
[  337.673630] 1
[  337.673631] Stack Bottom -------
[  337.673632] [System call] oslab_push(): Push 3
[  337.673632] Stack Top ----------
[  337.673633] 3
[  337.673633] 2
[  337.673633] 1
[  337.673634] Stack Bottom -------
```

```
[  337.673634] [System call] oslab_pop(): Pop 3
[  337.673635] Stack Top ----------
[  337.673635] 2
[  337.673635] 1
[  337.673636] Stack Bottom -------
[  337.673641] [System call] oslab_pop(): Pop 2
[  337.673641] Stack Top ----------
[  337.673641] 1
[  337.673642] Stack Bottom -------
[  337.673643] [System call] oslab_pop(): Pop 1
[  337.673643] Stack Top ----------
[  337.673644] Stack Bottom -------
osta@osta-VirtualBox:~$
```

커널 소스코드의 수정

- 과제 **handout**에 명시한 **4**개의 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
  - my_stack_syscall.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
    - my_stack_syscall.o 오브젝트 추가

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

```
osta@osta-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

335     common  oslab_push              __x64_sys_oslab_push
336     common  oslab_pop               __x64_sys_oslab_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
```

2) syscalls.h

- 시스템 콜 함수들의 **prototype**을 정의
  - (linux)/include/linux/syscalls.h 파일에 등록
  - asmlinkage void sys_oslab_push(int)
  - asmlinkage int sys_oslab_pop(void)

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

        return old;
}
#endif
asmlinkage void sys_oslab_push(int);
asmlinkage int sys_oslab_pop(void);
```

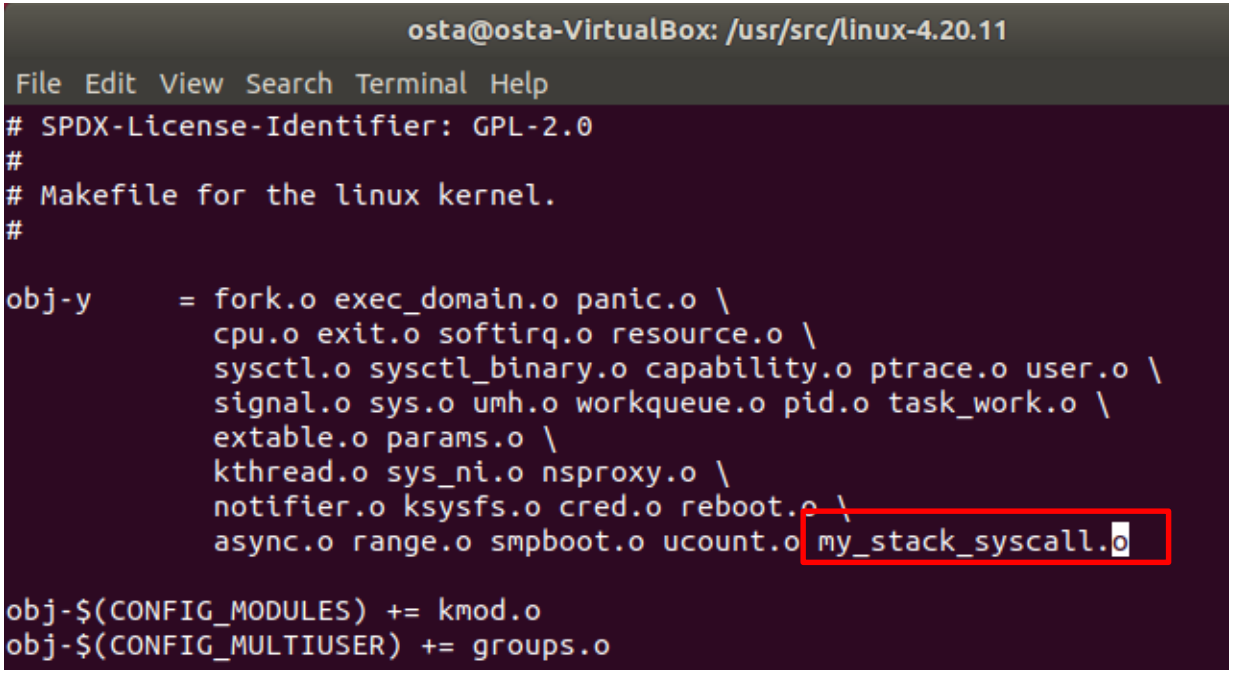
- 왜 **asmlinkage**를 사용하는가?
  - 시스템 콜 호출은 int 80인터럽트 핸들러에서 호출
  - 인터럽트 핸들러는 assembly 코드로 작성됨
  - asmlinkage 를 함수 앞에 선언하면,
    assembly code에서도 C함수 호출이 가능해짐

3) my_stack_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(oslab_push, int, a){
      ...
      }
    - SYSCALL_DEFINE0(oslab_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



- **.o** 오브젝트 파일명은 자신의 **c** 파일이름과 동일
  - 예) my_stack_syscall.c -> my_stack_syscall.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_stack.c라면
    - gcc call_my_stack.c –o call_my_stack
      - "call_my_stack.c를 컴파일해서 call_my_stack라는 이름의 실행 파일을 만들어라"
    - ./call_my_stack로 실행 후 dmesg를 통해서 my_stack_syscall.c의 printk로 원하던 출력이 나왔는지를 확인

Tip

- 대부분의 질문은 웹 검색을 통해 해결 가능
  - Linux 커널에 관련된 웹 문서는 매우 많음
  - 1차적으로 온라인에서 해결책을 찾아보고, 해결할 수 없는 상황에 질문하는 것이 시간을 효과적으로 사용하는 방법!

- 개별 질의응답
  - Email: osta@os.korea.ac.kr
    - 각 조교의 개인 메일보다 osta 메일로 보내면, 두 조교 중 관련 내용 응답이 가능한 조교가 빠르게 회신
  - 연구실 방문 : 우정관 308호 (운영체제연구실)
    연구실 방문 이전에 반드시 이메일로 문의