리눅스 시스템 프로그래밍

담당 강의 내용

- 리눅스 시스템 프로그래밍: 30시간 (6주차)
- 리눅스 프로세스 관리: 20시간 (8주차)
- 리눅스 디바이스 드라이버 구현: 40시간 (10주차)

리눅스 시스템 프로그래밍 수업 목적

- 그 동안 우리는 알고리즘만 학습
- 시스템 프로그래밍을 직접 구현
  - 알고리즘으로는 SW를 완성하지 못함.
  - 이를 통해, OS의 동작 원리를 힐끗 보기
- 방향
  - 임베디드 시스템 - S/W 개발자를 위한 리눅스 운영체제 지식
  - 서버 시스템 - S/W 개발자를 위한 리눅스 운영체제 지식

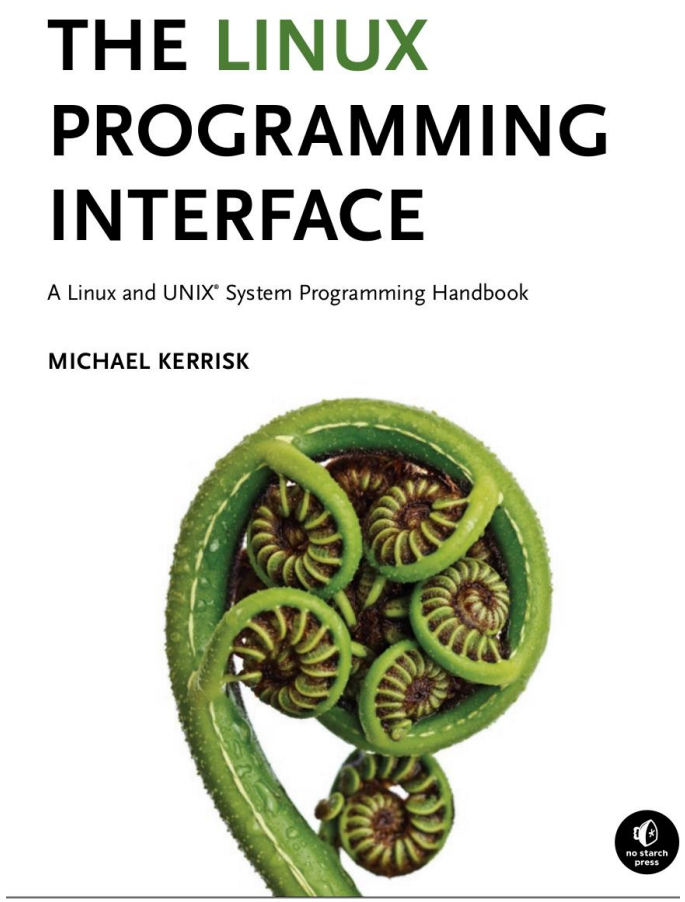
리눅스 시스템 프로그래밍

- 이론
  - 운영체제에서 제공하는 시스템 콜 설명
- 실습
  - 참조 코드 실행 및 분석
  - 토이 프로젝트

리눅스 시스템 프로그래밍 목차

리눅스 시스템 프로그래밍 목차

교재 + 실습 예제


THE LINUX
PROGRAMMING
INTERFACE
A Linux and UNIX® System Programming Handbook
MICHAEL KERRISK
no starch
press

리눅스 시스템 프로그래밍

- 알고리즘만 알고 있다면?
  - 세상의 복잡한 문제를 해결 못함.
- 복잡한 문제에 대한 해결책은?
  - 모듈화, 추상화, 레이어링, 계층화 등등..
  - 대표적인 소프트웨어: 리눅스 OS

운영체제 요약: 리눅스 목적

- 시스템 프로그래밍을 위해서는 리눅스 OS 개념을...
  - 닭이 먼저? 달걀이 먼저?

- 리눅스 운영체제가 제공하는 기능
  - 하드웨어 추상화
  - 다중화(Multiplex)
  - 격리(Isolation)
  - 공유(Sharing)
  - 보안(Security)
  - 성능(Performance)

운영체제 요약: O/S 커널이 제공하는 서비스는?

- 프로세스 (a running program)
- 메모리 할당
- 파일 내용
- 파일 이름, 디렉토리
- 엑세스 컨트롤(보안)
- 기타 등등(유저, IPC, 네트워크, 타임, 터미널)

리눅스 시스템 콜 소개

- 응용 프로그램은 시스템 콜을 통해 O/S의 기능을 사용
- 구조

응용프로그램 / 커널 인터페이스는?

- "시스템 콜"
- 예)

```
fd = open("out", 1);
write(fd, "hello₩n", 6);
pid = fork();
```

- 마치 함수처럼 보이지만?

user -> kernel 전환

- 시스템 콜
- 시스템 에러
- 인터럽트

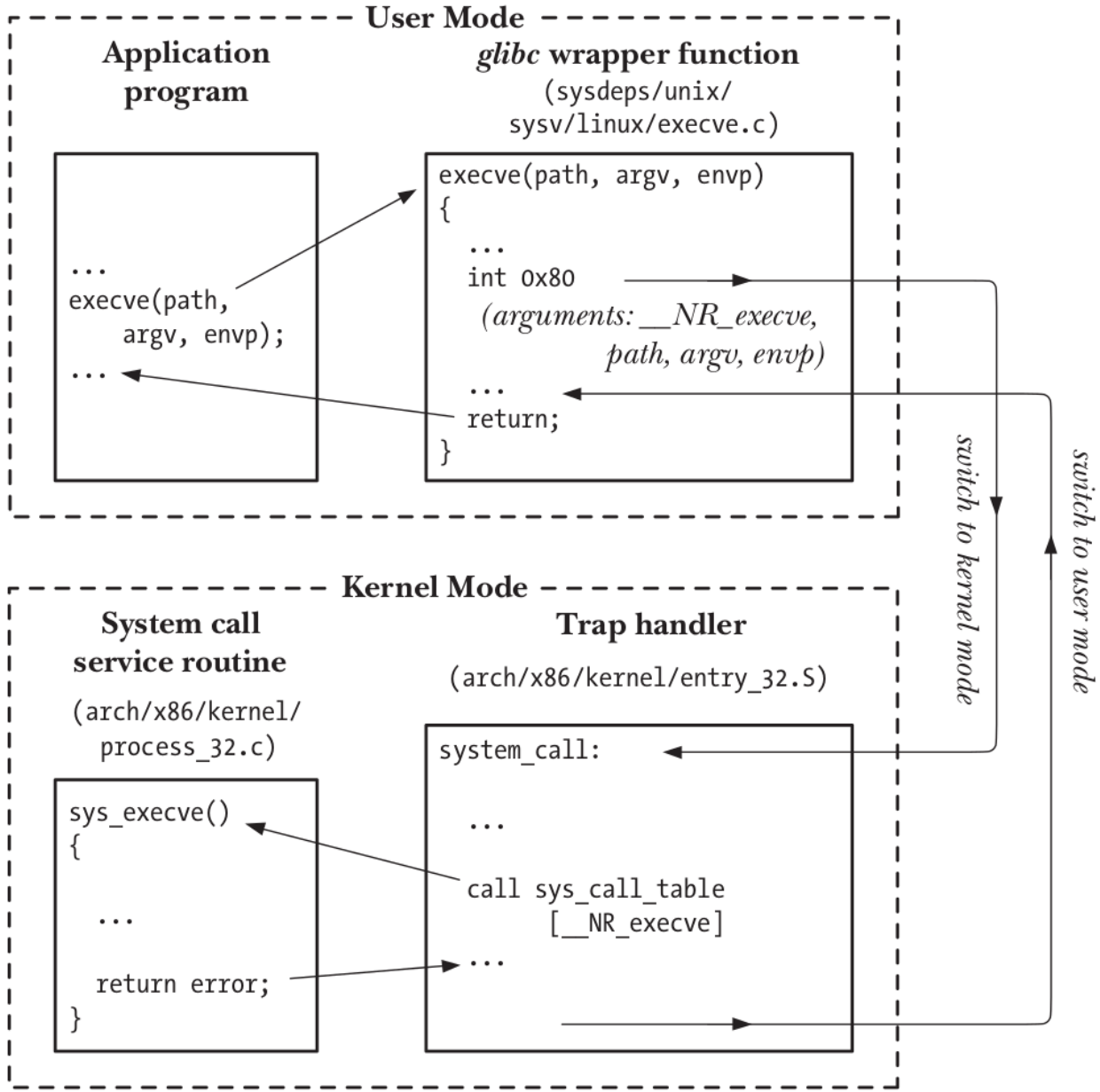


Figure 3-1: Steps in the execution of a system call

시스템 콜이 호출되면 무슨 일이? 예 write()?

1. User 레지스터와 PC를 메모리에 저장
2. CPU의 supervisor mode(kernel mode)로 전환
3. 메모리 관리를 kernel page table로 변경
4. 스택을 kernel stack으로 변경
5. kernel C code 점프

- 시스템 콜의 가장 중요한 부분은?
-

  유저의 코드를 절대로 supservisor mode에서 실행시켜주면 안됨!!!
  다시 유저 코드로 전환은 자연스럽게 복귀.

copy

- tlpi-dist/fileio/copy.c
- 입력 파일에서 바이트를 읽어서, 출력 파일에 쓰기
- open()
  - open() 은 파일을 생성하고 파일 디스크립터를 반환(또는 에러(-1))
  - fd는 작은 정수
  - 리눅스 커널은 프로세스 마다 fd 인덱스를 테이블로 관리
    - 다른 프로세스는 다른 fd 네임스페이스를 가짐
- 내부적으로 무슨일이 벌어질까?
  - 마치 함수 호출처럼 보임. 하지만, 실제적으로는 특별한 명령어
  - 유저 레지스터 저장 -> supervisor 모드로 전환 -> 커널 entry point로 진입 -> sys_open() -> 파일 시스템에서 이름 찾고 커널 자료 구조 수정 -> 유저 레지스터 복구 -> 유저 모드로 전환

copy

- 여기서 read()/write()는 시스템 콜

- 첫번째 Read()/ write() input 변수는
  - "파일 디스크립터" (fd)
  - 리눅스 파일 디스크립터 규칙
    0: standard input
    1: standard output
    2: standard error

- 두번째는 메모리 버퍼

- 세번째는 읽을 바이트 수

fork

- tlpi-dist/procexec/fork_whos_on_first.c
- 새로운 프로세스를 생성하는 시스템 콜
- Shell: 커맨드로 새로운 프로세스를 생성함.
  - 예: $echo hello
- fork(): 시스템 콜은 새로운 프로세스를 생성
- 리눅스 커널은 호출 한 프로세스의 자원을 복사하여 생성함
  - Code, Data, Registers, File descriptors, current directory
  - 차이점? 반환하는 PID
    - 부모: pid, 자식: 0

exec

- tlpi-dist/procexec/t_execl.c
- 실행 파일로 호출한 프로세스를 변경
  - 실행 파일(프로그램) -> 호출한 프로세스
- $echo a b c
- 프로그램은 컴파일러에 의해 명령어와 메모리 초기값을 저장함.
- exec()은 호출한 현재 프로세스를 교체
  - 호출한 프로세스의 명령어와 메모리 값은 버림
  - 파일로 부터 새로운 명령어와 메모리 값을 읽음
  - 열린 파일 디스크립터는 계속 유지
- exec(파일 이름, 인자)

pipe

- tlpi-dist/pipes/simple.pipe.c
- 프로세스간 통신을 위한 기법
  - ls | grep –nr simple*

- pipe() 시스템 콜은 두 개의 FD를 생성함
  - 첫번째는 read를 위한 FD
  - 두 번째는 write를 위한 FD

시스템 프로그래밍 프로젝트 실습

- 예제 소스 코드

  https://man7.org/tlpi/code/download/tlpi-221220-dist.tar.gz

- 라이브러리 설치

  sudo apt install libcap-dev libacl1-dev libselinux1-dev libseccomp-dev gcc-multilib

- 빌드

  make

- 분석

시스템 프로그래밍 실습 맛보기

- tlpi-dist/fileio/copy.c
  - 코드 분석 및 실행
- tlpi-dist/procexec/fork_whos_on_first.c
  - 코드 분석 및 실행
- tlpi-dist/procexec/f_exec.c
  - 코드 분석 및 실행
- tlpi-dist/pipes/simple_pipe.c
  - 코드 분석 및 실행

TOY 프로젝트

- 직접 만드는 작은 프로젝트
- 매시간 샘플 코드를 기반으로 직접 내용 구현
- 소프트웨어 구조

TOY 프로젝트

소스 구조

main.c
ui/ui.c
ui/input.c
web/webserver.c
system/system_server.c