리눅스시스템 및 응용

Week 9
2~3

C Programming Basic
in
Linux

리눅스 시스템 프로그래밍

- 추천도서
  - 시스템 프로그래밍 리눅스&유닉스 *
    - 이종원, 한빛아카데미, 2021
  - 리눅스 프로그래밍 기초
    - 최태영 외 4, 한빛아카데미, 2015
  - 리눅스 프로그래밍
    - 창병모, 생능, 2014

학습목표

- 리눅스와 유닉스
- 유닉스
- 시스템 호출 System Calls
- Man Page 활용
- GNU C Compiler
- 프로그램 오류시 에러 출력
- Memory Allocation Programming 연습
- 명령어행 인자
- 리눅스와 유닉스의 파일

- 리눅스와 유닉스
  - 리눅스와 유닉스
    - 서버용 운영체제로 주로 사용
    - 최근엔 원조격이라고 할 수 있는 유닉스를 리눅스가 서버 운영체제 시장에서 밀어냄
    - 금융권에서는 유닉스 시스템을 리눅스 시스템으로 대체하는 U2L이 확산

- 유닉스
  - 유닉스의 개발
    - 1969년에 미국의 통신 회사인 AT&T 산하의 벨 연구소에서 켄 톰슨과 데니스 리치가 개발한 운영체제
    - 처음에는 기존 운영체제처럼 어셈블리어로 개발
    - 데니스 리치가 개발한 C 언어를 사용해 1973년에 다시 만들면서 고급 언어로 작성된 최초의 운영체제
  - 유닉스의 기능
    - 유닉스는 초기에 소스 코드가 공개되어 대학교나 기업에서 쉽게 이용할 수 있었고 이에 따라 다양한 기능이 추가
    - AT&T의 상용 유닉스(시스템 V)와 버클리 대학교의 BSD 계열로 나뉘어 각각 발전
    - BSD 버전은 버클리 대학교 학생들이 많은 기능을 추가했는데 그 중 TCP/IP 기반의 네트워크 기능을 추가

- 유닉스의 발전
  - 1991년에 등장한 리눅스는 오픈 소스로 공개되어 지속적으로 발전
  - 리눅스는 같은 커널을 기반으로 하는 다양한 형태의 배포판을 사용

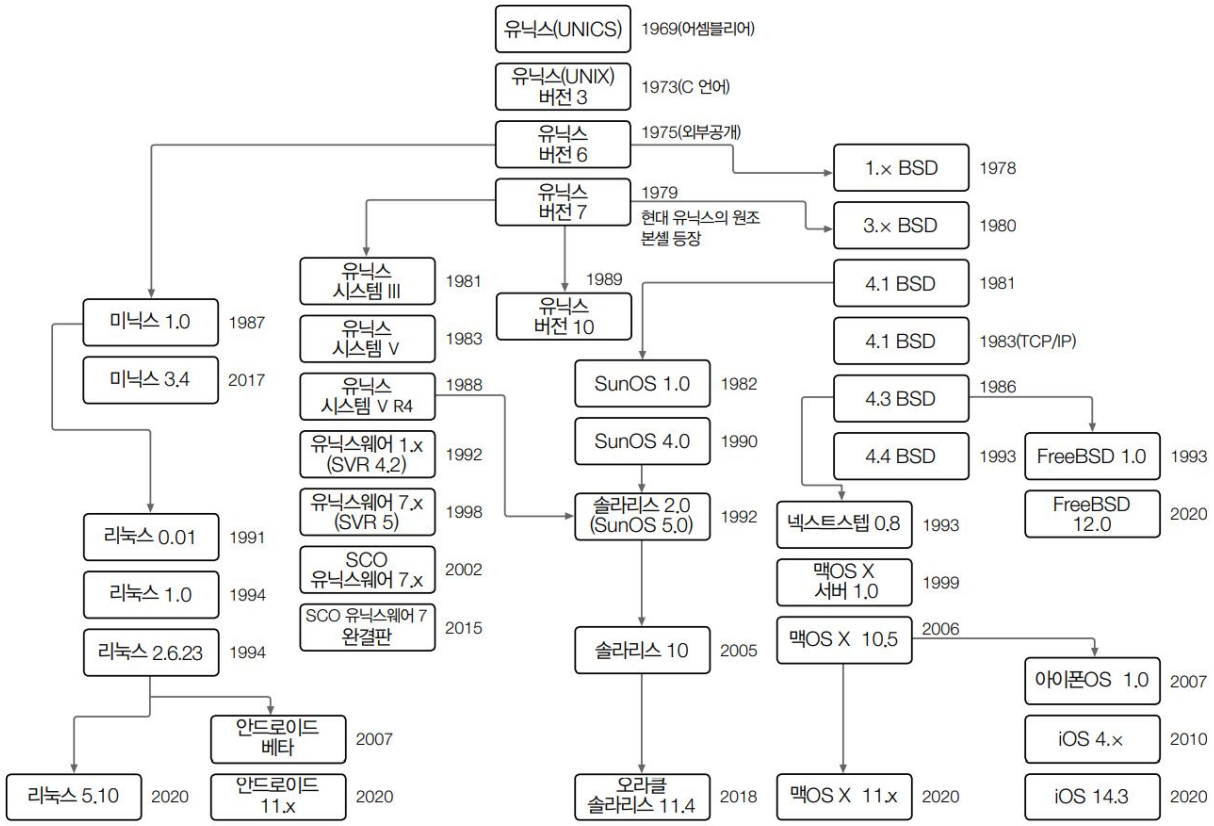

그림 1-1 유닉스와 리눅스의 발전 과정

- 유닉스의 주요 표준
  - ANSIC 표준
    - 미국 표준 협회로, 국제적으로 영향력 있는 표준을 정함
    - ANSI에서 표준화한 C 언어 명세가 ANSI C 표준으로, C 언어 문법과 라이브러리, 헤더 파일 등을 정의
  - POSIX
    - 유닉스에 기반을 두고 있는 표준 운영체제 인터페이스
    - 서로 다른 유닉스 시스템 사이에서 상호 이식이 가능한 응용 프로그램을 개발하기 위해 정해진 표준
    - IEEE에서 정의한 규격으로, 유닉스 시스템의 공통 응용 프로그래밍 인터페이스를 정리

- X/Open 가이드
  - 1984년에 유럽의 유닉스 시스템 제조업체를 중심으로 설립된 단체로, 개방 시스템에 관한 표준 정의와 보급을 목적
  - 다양하게 파생되고 있는 유닉스 시스템 에서 응용 프로그램의 이식성을 높이는 것이 초기 목표
  - 1988년에 발표한 XPG3에서는 POSIX 표준을 통합했고, XPG의 최종 버전인 XPG4는 1992년에 발표

- 유닉스의 주요 표준
  - 단일 유닉스 명세 (SUS)
    - 운영체제가 유닉스라는 이름을 사용하기 위해 지켜야 하는 표준의 총칭
    - IEEE, ISO(JTC 1 SC22), 오픈 그룹의 표준화 작업결과물에 바탕을 두고 있으며 오스틴 그룹이 개발 및 유지·관리를 담당
    - 1980년대 중반부터 시작된 유닉스의 시스템 인터페이스를 표준화하기 위한 프로젝트에서 출발
  - 시스템 V 인터페이스 정의 (SVID)
    - AT&T 유닉스 시스템 V의 인터페이스를 정의
    - 프로그램과 장치에서 이용할 수 있는 시스템 호출과 C 라이브러리에 관한 표준을 포함
    - POSIX나 X/Open 작업은 부분적으로 SVID에 기반
    - 1995년에 발표된 SVID 버전 4는 XPG4 및 POSIX 1003.1-1990과 호환성을 유지
    - SVID는 POSIX와 단일 유닉스 명세에 포함되면서 중요도가 떨어짐

시스템호출 System Calls

- 시스템 호출
  - 시스템이 제공하 는 서비스를 프로그램에서 이용할 수 있도록 지원하는 프로그래밍 인터페이스를 의미
  - 리눅스/유닉스에서 동작하는 프로그램을 작성하려면 간단한 프로그램을 제외하고 대부분 시스템 호출을 이용
  - 일반적으로 사용자 프로그램은 운영체제 직접 관여 금지
  - Kernel을 직접 통한 접근 금지
  - System call에 의하여 운영체제 자체 인터페이스로 접근
  - 저수준의 함수호출

시스템 호출과 라이브러리 함수

- 시스템 호출과 라이브러리 함수
  - 시스템 호출

```
리턴값 = 시스템 호출명(인자, ...);
```

    - 시스템 호출명은 함수명처럼 사용할 이름이 정의

  - 라이브러리 함수
    - 라이브러리: 미리 컴파일된 함수를 묶어서 제공하는 특수한 형태의 파일
    - 라이브러리 함수: 라이브러리에 포함된 함수를 의미
    - 리눅스 시스템에서 라이브러리는 보통 /usr/lib에 위치

    - 정적 라이브러리는 프로그램을 컴파일할 때 같이 적재되어 실행 파일을 구성
    - 공유 라이브러리는 실행 파일에 포함되지 않아 메모리를 효율적으로 사용하기 위해 사용

- 시스템 호출과 라이브러리 함수 비교

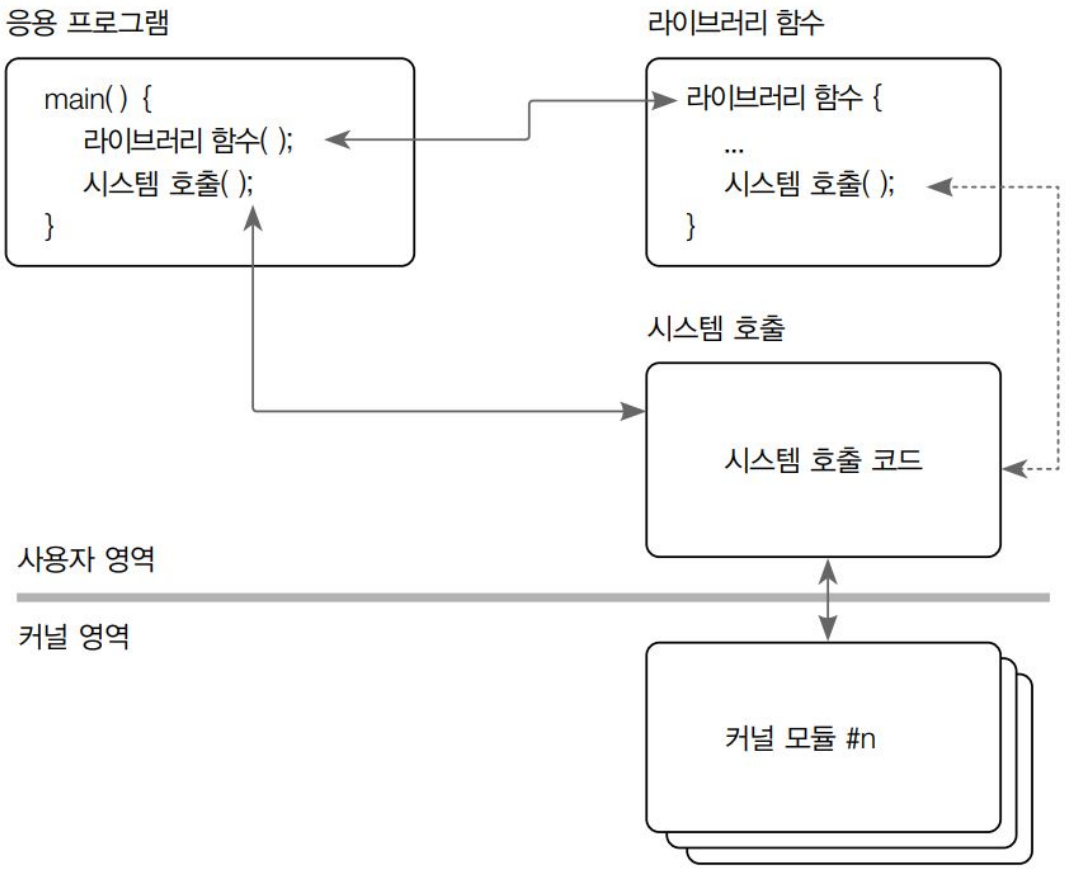

그림 1-2 시스템 호출과 라이브러리 함수의 비교

- man page의 활용
  - man 페이지의 섹션 번호
    - 매뉴얼은 항목의 종류에 따라 섹션이 구분되어있음
      - » 리눅스에서 사용하는 일반적인 명령에 대한 설명: 섹션 1
      - » 시스템 호출: 섹션 2
      - » 라이브러리 함수: 섹션 3
    - man 명령으로 검색하면 섹션 번호가 가장 낮은 것이 기본으로 출력됨
    - man 명령의 결과를 출력하는 형식은 리눅스와 유닉스에서 차이가 있음

- man 페이지의 섹션 번호
  - man 페이지
    - 명령이나 함수 등 시스템의 다양한 서비스에 대한 매뉴얼
    - 매뉴얼은 항목의 종류에 따라 섹션이 구분되어있음
      - » 리눅스에서 사용하는 일반적인 명령에 대한 설명: 섹션 1
      - » 시스템 호출: 섹션 2
      - » 라이브러리 함수: 섹션 3
    - man 명령으로 검색하면 섹션 번호가 가장 낮은 것이 기본으로 출력됨
    - man 명령의 결과를 출력하는 형식은 리눅스와 유닉스에서 차이가 있음

- open() 시스템호출 섹션 번호
  - 리눅스(우분투 리눅스)와 유닉스(솔라리스)에서 open() 함수를 검색하면 open(2)'로 표시 (섹션 2)
  - 리눅스에서는 상단에 'System Calls'라고 명시

[리눅스(우분투 리눅스)]

```
OPEN(2)          Linux Programmer's Manual          OPEN(2)

NAME
       open, openat, creat - open and possibly create a file

SYNOPSIS
     #include <sys/types.h>
```

[유닉스(솔라리스)]

```
OPEN(2)          Linux Programmer's Manual          OPEN(2)

NAME
       open, openat - open a file

SYNOPSIS
     #include <sys/types.h>
```

- fopen() 라이브러리 함수 섹션 번호
  - 리눅스(우분투 리눅스)와 유닉스(솔라리스)에서서 fopen() 함수를 검색
  - 리눅스에서는 'fopen(3)'으로 표시되어 섹션 3에 속한 함수인 것만 알리지만, 유닉스에서는 'fopen(3C)'로 표시

[리눅스(우분투 리눅스)]

```
OPEN(2)          Linux Programmer's Manual          OPEN(3)

NAME
       fopen, fdopen, freopen - stream open functions

SYNOPSIS
     #include <sys/types.h>
```

[유닉스(솔라리스)]

```
OPEN(2)          Linux Programmer's Manual          OPEN(3C)

NAME
       fopen - open a stream
       fopen_s - open a stream with additional safety checks

SYNOPSIS
     #include <sys/types.h>
```

- man 페이지의 섹션 번호
  - 명령과 함수의 이름이 같은 경우: 'man uname'을 입력하면 명령(섹션 1)에 대한 설명만 볼 수 있음
  - 섹션 2의 설명을 보려면 다음과 같이 해당 섹션을 지정해야 함

```
$ man uname

UNAME(1)          User Commands          UNAME(1)

NAME
       uname - print system information

SYNOPSIS
       uname [OPTION] ...
```

```
$ man -s 2 uname

UNAME(2)          Linux Programmer's Manual          UNAME(2)

NAME
       uname - get name and information about current kernel

SYNOPSIS
       #include <sys/utsname.h>
```

시스템호출 오류 처리

- 시스템 호출의 오류 처리하기

```
01 #include <stdio.h>
02 #include <unistd.h>
03 #include <errno.h>
04
05 extern int errno;
06
07 int main() {
08     if(access("test.txt", F_OK) == -1) {
09         printf("errno=%d₩n", errno);
10     }
11 }
```

실행

```
$ ./ch1_1.out
errno=2
```

- **08~10행** access( ) 함수의 리턴값 검사, –1이면 오류 발생이므로 전역 변수 errno 값 검사
- **실행 결과** errno 변수 값은 2. 해당 시스템 호출에서 발생한 오류가 무엇인지 알려줌

- errno 정의
  - errno에 저장된 값 2가 의미하는 바를 해석하려면 헤더 파일을 참조
  - 리눅스는 asm-generic/errno-base.h 파일에 정의되어 있고 유닉스는 sys/errno.h 파일에 정의

```
$ vi /usr/include/asm-generic/errno-base.h
/* SPDX-License-Identifier : GPL-2.0 WITH Linux-syscall-note */
#ifndef _ASM_GENERIC_ERRNO_BASE_H
#define _ASM_GENERIC_ERRNO_BASE_H

#define EPERM           1               /* Operation not permitted */
#define ENOENT          2              /* No such file or directory */
(생략)
```

- man 페이지의 error number 확인
  - access() 함수에서 발생하는 오류 코드로는 EACCES, ELOOP, ENAMETOOLONG, ENODIR, EROFS 등이 있음
  - man access 명령으로 access() 함수에서 발생하는 오류 코드와 해당 설명을 확인

```
$ man access
ACCESS(2)              Linux Programmer's Manual              ACCESS(2)

NAME
       access, faccessat - check user's permissions for a file
 (생략)
 ERRORS
       access() and faccessat() shall fail if:
       EACCES The requested access would be denied to the file, or search permission is denied
              for one of the directories in the path prefix of pathname. (See also path_
              resolution(7).)

       ELOOP Too many symbolic links were encountered in resolving pathname.

       ENAMETOOLONG
              pathname is too long.
       ENOENT A component of pathname does not exist or is a dangling symbolic link.
       (생략)
```

라이브러리 함수 오류 처리

- 라이브러리 함수 오류 처리

```
01 #include <stdlib.h>
02 #include <stdio.h>
03 #include <errno.h>
04
05 extern int errno;
06
07 int main() {
08     FILE *fp;
09
10     if((fp=fopen("test.txt", "r")) == NULL) {
11         printf("errno=%d₩n", errno);
12         exit(1);
13     }
14     fclose(fp);
15 }
```

실행

```
$ ./ch1_2.out
errno=2
```

- **10행** 라이브러리 함수인 fopen()을 사용해 test.txt 파일을 실행
  여기서는 파일이 존재하지 않으므로 오류가 발생해 NULL을 리턴
- **실행 결과** errno에 저장된 값이 2임을 알 수 있음
  fopen() 함수에서 발생할 수 있는 오류 코드는 man 페이지에서 찾아볼 수 있음

GNU C 컴파일러 : gcc

- GNU C 컴파일러
  - gcc
  - 기능 : C 프로그램을 컴파일해 실행 파일을 생성
  - 형식 : gcc [옵션][파일명]
  - 옵션 -c : 오브젝트 파일(.o)만 생성
  - -o 실행 파일명 : 지정한 이름으로 실행 파일을 생성, 기본 실행 파일명은 a.out
  - 사용 예:
    - **$ gcc test.c**
    - **$ gcc -c test.c**
    - **$ gcc -o test test.c**

Question?