

# Linux Fundamental

강사. 유효곤  
(ugonfor@gmail.com)

# Contents

---

## 수업

- OS
  - Process
  - Threading
  - Signal, IPC
- System
  - Unix file system
  - Permission, File/Directory Information
    - `sudo rm -rf /`
  - File Creation & File I/O Function

## 실습

- labyrevnt
- 리눅스 셸 기능 추가하기
  - mv 구현
  - ls 구현
  - sigint 처리
  - sigtstp 와 sigcont로 ST, FG구현

# 시작하기에 앞서서...

---

리눅스를 가르친다고 하면 뭘 가르치죠?

# 리눅스를 배운다면,

## 리눅스 사용법

- 기초 명령어
- 셸 스크립트
- 디렉토리 구조
- 프로세스 실행
- 파일 권한
- ...

리눅스 준비

– Codeonweb

– cloud9

– 맥(osx)

– virtual box(가상머신)

디렉토리과 파일

--help 와 man

필요한 명령을 검색으로 찾는 법

명령의 빈도수

sudo

파일편집 (nano)

패키지 매니저

다운로드 방법 (wget, git)

왜 CUI인가?

IO Redirection

셸과 커널

셸 스크립트

디렉토리의 구조

프로세스

파일을 찾는 법

백그라운드 실행

항상 실행 (daemon, service)

정기적으로 실행 (cron)

셸을 시작할 때 실행

다중 사용자

관리자와 일반 사용자

사용자의 추가

권한 (permission)

|                                     |                            |
|-------------------------------------|----------------------------|
| ^ 섹션 1. 리눅스와 셸                      | 2 강 · 20분                  |
| ① 리눅스와 셸                            | 09:24                      |
| ① 리눅스와 셸(Bash)                      | 10:40                      |
| ^ 섹션 2. 리눅스 명령어                     | 16 강 · 3시간 16분             |
| ① 실습환경                              | <a href="#">미리보기</a> 04:58 |
| ① 실무에서 자주 사용하는 명령어_파일 시스템 관련        | 12:05                      |
| ① 실무에서 자주 사용하는 명령어_디렉토리과 파일 통계      | 11:49                      |
| ① 실무에서 자주 사용하는 명령어_파일관련 명령어         | 09:33                      |
| ① 실무에서 자주 사용하는 명령어_파일관련 명령어(2)      | 14:12                      |
| ① 실무에서 자주 사용하는 명령어_파일관련 명령어(3)      | 09:28                      |
| ① 실무에서 자주 사용하는 명령어_파일관련 명령어(4)      | 15:20                      |
| ① 실무에서 자주 사용하는 명령어_프로세스 관련 명령어      | 05:56                      |
| ① 실무에서 자주 사용하는 명령어_프로세스 관련 명령어(2)   | 12:41                      |
| ① 실무에서 자주 사용하는 명령어_네트워크 관련 명령어      | 11:54                      |
| ① 실무에서 자주 사용하는 명령어_네트워크 관련 명령어(2)   | 13:22                      |
| ① 실무에서 자주 사용하는 명령어_네트워크 관련 명령어(3)   | 11:59                      |
| ① 실무에서 자주 사용하는 명령어_검색/탐색 관련 명령어     | 18:00                      |
| ① 실무에서 자주 사용하는 명령어_I/O 관련 명령어       | 16:15                      |
| ① 실무에서 자주 사용하는 명령어_기타 명령어           | 18:09                      |
| ① 실무에서 자주 사용하는 명령어_실무에 유용한 명령어 옵션 팁 | 11:05                      |
| v 섹션 3. 초간단 셸 스크립트                  | 3 강 · 50분                  |

# 리눅스를 배운다면,

## 리눅스 사용법

- 기초 명령어
- 셸 스크립트
- 디렉토리 구조
- 프로세스 실행
- 파일 권한
- ...

리눅스 준비  
- Codeonweb  
- cloud9  
- 맥(osx)  
- virtual box(가상머신)  
디렉토리와 파일  
--help 와 man  
필요한 명령을 검색으로 찾는 법  
명령의 빈도수  
sudo  
편집기 (nano)  
다운로드  
왜 CUI인가?  
셸 스크립트  
디렉토리의 구조  
프로세스  
파일을 찾는 법  
백그라운드 실행  
항상 실행 (daemon, service)  
정기적으로 실행 (cron)  
셸을 시작할 때 실행  
다중 사용자  
관리자와 일반 사용자  
사용자의 추가  
권한 (permission)

|                                     |           |
|-------------------------------------|-----------|
| ^ 섹션 1. 리눅스와 셸                      | 2 강 · 20분 |
| ① 리눅스와 셸                            | 09:24     |
| ① 리눅스와 셸(Bash)                      |           |
| ^ 섹션 2. 리눅스 명령어                     |           |
| ① 실습환경                              | 04:58     |
| ① 리눅스에서 자주 사용하는 명령어_파일관련 명령어        | 12:05     |
| ① 리눅스에서 자주 사용하는 명령어_디렉토리와 파일 통계     | 11:49     |
| ① 리눅스에서 자주 사용하는 명령어_파일관련 명령어        | 09:33     |
| ① 실무에서 자주 사용하는 명령어_파일관련 명령어(2)      | 14:12     |
| ① 실무에서 자주 사용하는 명령어_파일관련 명령어(3)      | 09:28     |
| ① 실무에서 자주 사용하는 명령어_파일관련 명령어(4)      | 15:20     |
| ① 실무에서 자주 사용하는 명령어_프로세스 관련 명령어      | 05:56     |
| ① 실무에서 자주 사용하는 명령어_프로세스 관련 명령어(2)   | 12:41     |
| ① 리눅스에서 자주 사용하는 명령어_네트워크 관련 명령어     | 11:54     |
| ① 리눅스에서 자주 사용하는 명령어_네트워크 관련 명령어(2)  | 13:22     |
| ① 실무에서 자주 사용하는 명령어_네트워크 관련 명령어(3)   | 11:59     |
| ① 실무에서 자주 사용하는 명령어_검색 관련 명령어        | 18:00     |
| ① 실무에서 자주 사용하는 명령어_I/O 관련 명령어       | 16:15     |
| ① 실무에서 자주 사용하는 명령어_기타 명령어           |           |
| ① 실무에서 자주 사용하는 명령어_실무에 유용한 명령어 옵션 팁 |           |
| v 섹션 3. 초간단 셸 스크립트                  | 3 강 · 50분 |

# 리눅스를 배운다면,

## 시스템 프로그래밍

- 어셈블리 언어
  - 어셈블리 프로그래밍
  - 레지스터, 16bit Dos
- 컴퓨터 구조
  - 하드웨어 연결
  - CPU 프로세스
- 운영체제
  - 프로세스, 쓰레드
  - Signal, IPC
  - Exception Control
- 리눅스 파일 시스템
  - 파일, 디렉토리 구조,
  - 파일 생성, I/O
  - 메모리, 동적할당
- 네트워크
  - 동시성
  - 서버, I/O Multiplexing

# 리눅스를 배운다면,

## 시스템 프로그래밍

- 어셈블리 언어
  - 어셈블리 프로그래밍
  - 레지스터, 16bit Dos
- 컴퓨터 구조
  - 하드웨어 연결
  - CPU 프로세스
- 운영체제
  - 프로세스, 쓰레드
  - Signal, IPC
  - Exception Control
- 리눅스 파일 시스템
  - 파일, 디렉토리 구조,
  - 파일 생성, I/O
  - 메모리, 동적할당
- 네트워크
  - 동시성
  - 서버, I/O Multiplexing

오늘 다룰 내용

6~7주차에 다룰 예정

# 운영체제 관련

---

Process

Threading

Signal, IPC



# Process

---

프로세스가 무엇인가?

어떻게 생성되고, 실행되고, 종료되는가?

# Process

---

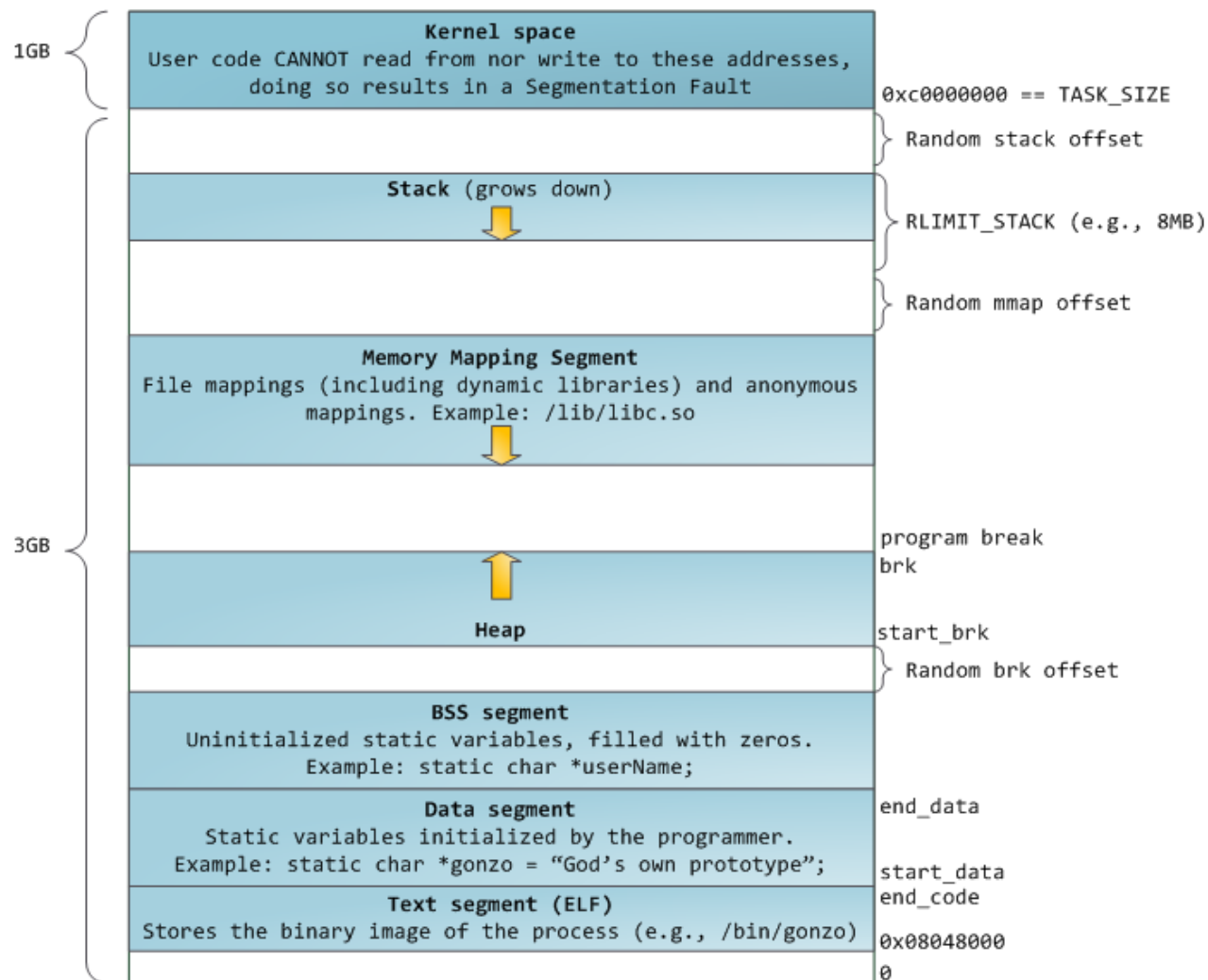
## ■ Process란,

- Processor
  - 중앙 처리 장치
- Process
  - 동작중인 프로그램
  - 커널로 부터 할당받은 자원을 가지고, CPU가 기계어 명령들을 실행함에 따라 끊임없이 변화하는 동적인 객체
  - 즉, 메모리 + 레지스터 + Code Flow 등
- Program vs Process vs Thread

# Process

## 메모리구조

- Text : code, image of process
  - Data : static variable
  - Heap : dynamic memory
  - Stack : local memory
- 
- cat /proc/self/maps를 통해서 확인해보자.



# Process

## Process id

- 프로세스 고유한 ID
- getpid(), getppid()를 통해서 자신의 pid와 부모 프로세스의 pid를 확인할 수 있음.
- 내가 어떤 프로세스를 bash를 통해서 실행시켰을 때, 그 프로세스의 부모 프로세스는?

```
ugonfor@gongon:~/ugonfor/Cyber-Guardians-Lecture-Note/Week2$ ps all
```

| F | UID  | PID  | PPID | PRI | NI | VSZ   | RSS  | WCHAN  | STAT | TTY   | TIME | COMMAND |
|---|------|------|------|-----|----|-------|------|--------|------|-------|------|---------|
| 4 | 1000 | 3388 | 3387 | 20  | 0  | 10184 | 5376 | do_wai | Ss   | pts/0 | 0:00 | -bash   |
| 0 | 1000 | 6160 | 3388 | 20  | 0  | 10536 | 3136 | -      | R+   | pts/0 | 0:00 | ps all  |

# Process

---

## fork & exit & exec

- 프로세스를 구성하는 데 가장 중요한 함수들
  - fork : 자식 프로세스 생성
  - exit : 프로세스를 종료
  - exec : 프로그램을 실행

# Process

---

## 자식프로세스를 어떻게 생성하는 가?

- fork() 함수를 통해 보통 자식 프로세스를 생성한다.
- fork를 하는 순간 자식 프로세스는 부모 프로세스의 모든 메모리를 복사
- 이후, 둘은 독립적인 프로세스가 된다.
- fork의 반환값을 통해 자식/부모 프로세스를 구별함.
  - 반환값은 자식프로세스 : 0
  - 부모프로세스: 자식프로세스의 pid

# Process

---

## 자식프로세스는 어떻게 종료되는 가?

- Exit()
  - File descriptor 종료
  - 메모리, 자원 등 반납
  - 자식 프로세스에 SIGHUP Signal
  - 부모 프로세스에 SIGCHLD Signal
  - exit(0), exit(1) 등 반환값을 부모 프로세스에 전달

# Process

---

## 셸은 어떻게 프로그램을 실행하는 가?

- `exec()` 함수군
  - 명령이나 실행 파일을 실행
  - `exec`함수 실행시 프로세스의 메모리 이미지가 실행파일로 변환됨.



# Process

## 부모 프로세스가 자식 프로세스를 관리하는 법

- `pid_t wait(int *stat_loc)` 함수
  - `Stat_loc` : 상태를 저장할 주소

|                    | wait 함수 반환 값 | statloc 값                                                                                                                                                                                                                                                   |
|--------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 자식 프로세스가 정상적으로 종료  | 프로세스 ID      | <ul style="list-style-type: none"><li>- <code>WIFEXITED(statloc)</code> 매크로가 <code>true</code>를 반환</li><li>- 하위 8비트를 참조하여 자식 프로세스가 <code>exit</code>, <code>_exit</code>, <code>_Exit</code>에 넘겨준 인자값을 얻을 수 있음, <code>WEXITSTATUS(statloc)</code></li></ul> |
| 자식 프로세스가 비정상적으로 종료 | 프로세스 ID      | <ul style="list-style-type: none"><li>- <code>WIFSIGNALED(statloc)</code> 매크로가 <code>true</code>를 반환</li><li>- 비정상 종료 이유를 <code>WTERMSIG(statloc)</code> 매크로를 사용하여 구할 수 있음</li></ul>                                                                        |
| wait 함수 오류         | -1           | <ul style="list-style-type: none"><li>- <code>ECHILD</code> : 호출자의 자식 프로세스가 없는 경우</li><li>- <code>EINTR</code> : 시스템 콜이 인터럽트 되었을 때</li></ul>                                                                                                                |

# Process

---

■ 그래서 프로세스가 뭔가요?

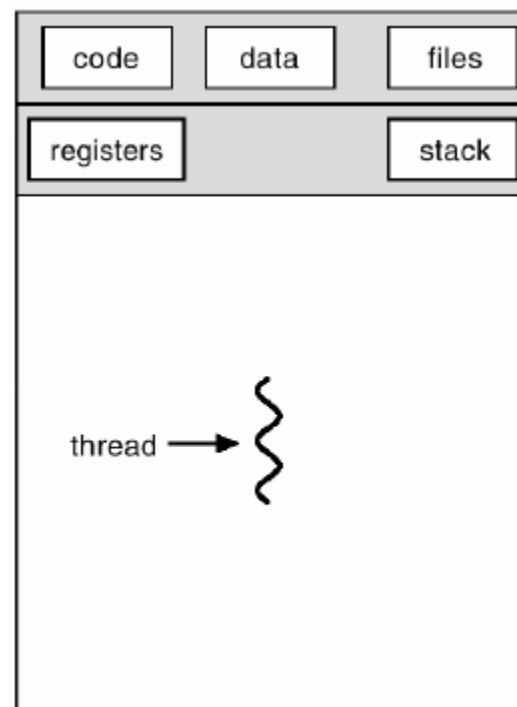
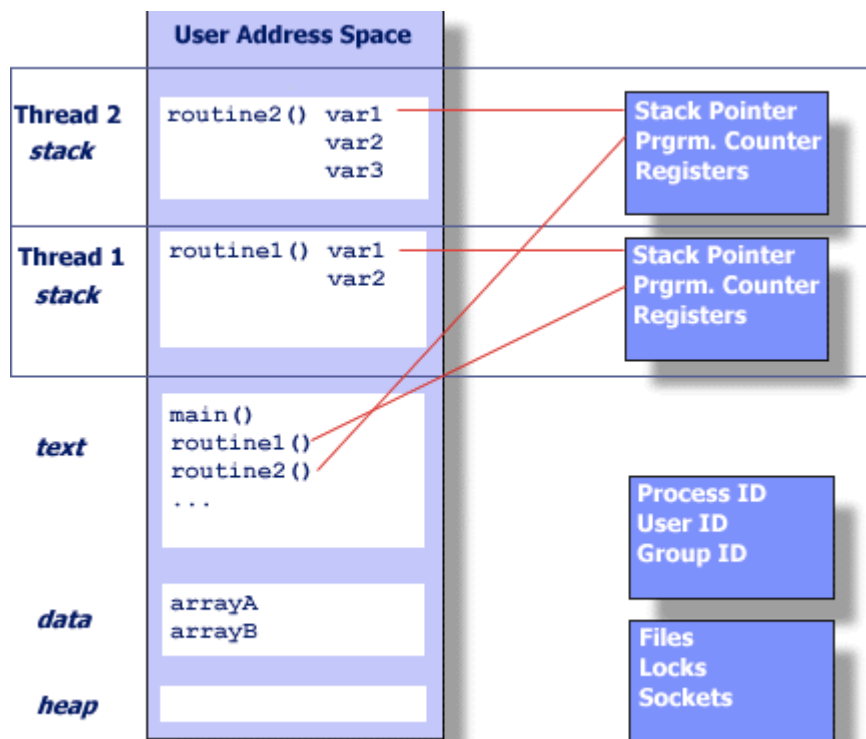
# Thread

---

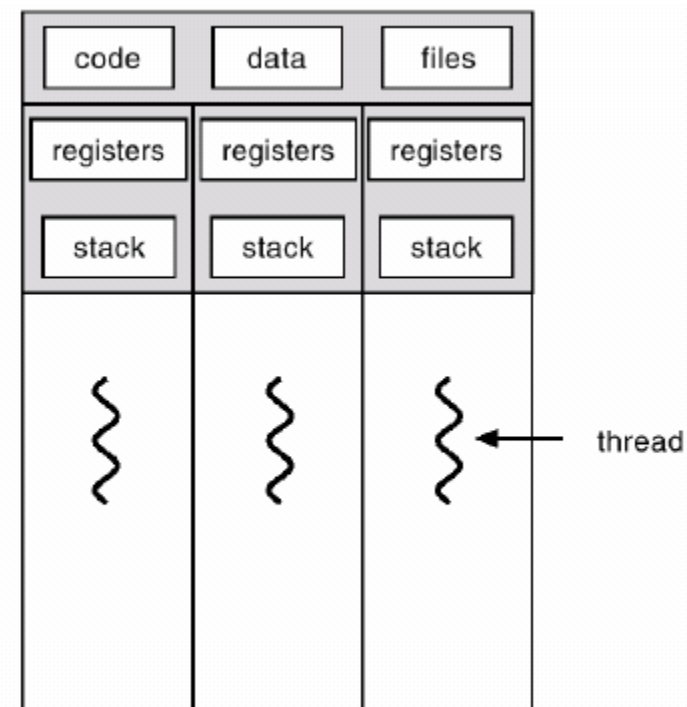
Thread란,  
Process와 차이점은?

# Thread

Thread란,



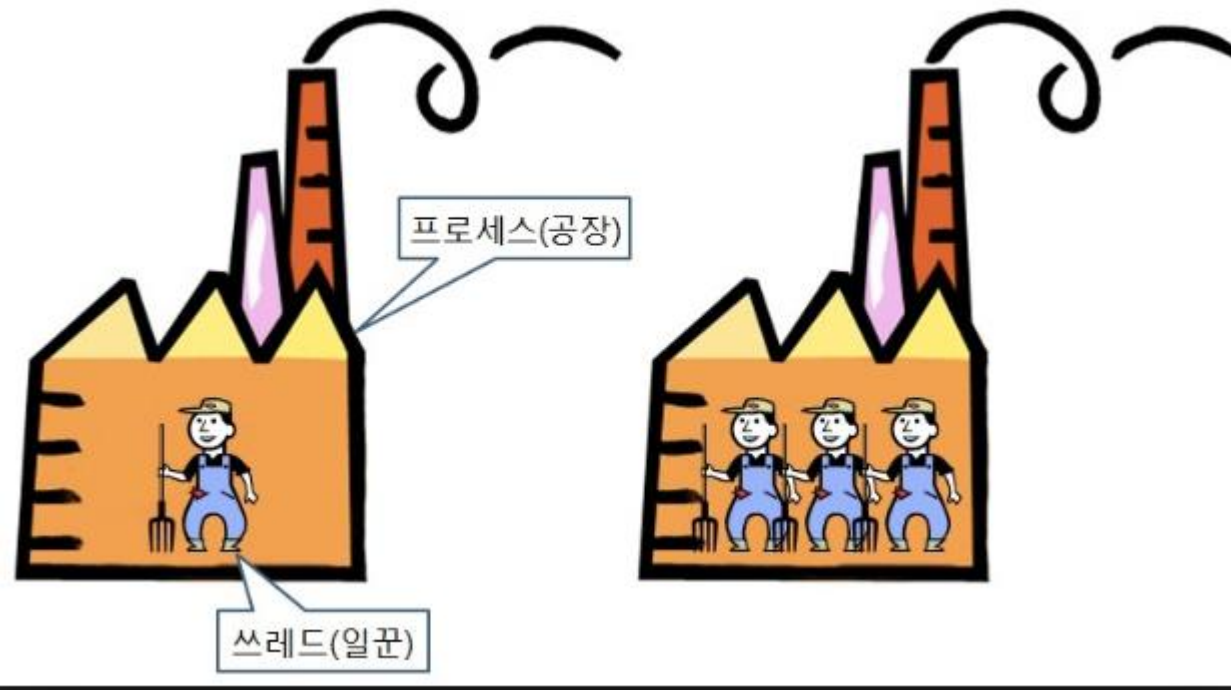
single-threaded



multithreaded

# Thread

Thread란,



# Thread

---

## Thread란,

- 같은 프로세스 내의 쓰레드는 동일한 pid, ppid를 가진다.
- 같은 가상 메모리를 공유
- 자식 쓰레드에서의 문제가 부모 쓰레드에 영향 줄 수 있음.
- Resource
  - 공유 됨: Code, data, heap, fd, ...
  - 분리 됨: register, stack
- Thread는 동시에 일을 처리하는 것인 데 어떻게 그럴 수 있는 가?
- Thread가 많으면 많을 수록 좋을까?

# Thread

---

## ■ Process vs Thread

- Process: 동적으로 실행중인 프로그램으로 CPU 시간, 주소 공간, 독립된 메모리 영역이 주어짐
- Thread : Process 내에서의 서로 다른 실행 흐름으로, 대부분의 자원을 공유함.
- Thread가 가지는 장점은?
  - 프로세스를 생성하는 것보다 Thread를 생성하는 것이 부하가 적음

# Signal, IPC

---

프로세스간 신호, 데이터를 주고받는 방법



# Signal

---

## ■ Signal이란?

- 어떤 이벤트가 발생했다는 것을 프로세스에게 전달하기 위한 일종의 소프트웨어 인터럽트
- 이벤트의 종류
  - Hardware Exception (0으로 나누기)
  - Software Condition (알람시간)
  - 사용자 입력 (CTRL+C)
  - Kill 명령
- 시그널 처리
  - 기본 설정 작업 수행 : 무시, 종료, 종료+core dump
  - Signal handler에서 수행
  - 무시

# Signal

## Signal example

| 번호 | 이름             | 설명                                                                                                              | 기본 처리 |
|----|----------------|-----------------------------------------------------------------------------------------------------------------|-------|
| 1  | SIGHUP (HUP)   | HangUP의 약어로 로그아웃과 같이 터미널에서 접속이 끊겼을 때 보내지는 시그널입니다.<br>데몬 관련 환경 설정 파일을 변경시키고 변화된 내용을 적용하기 위해 재시작할 때 이 시그널이 사용됩니다. | 종료    |
| 2  | SIGINT (INT)   | 키보드로부터 오는 인터럽트 시그널로 실행을 중지.<br>[CTRL]+[c] 입력 시에 보내지는 시그널입니다.                                                    | 종료    |
| 3  | SIGQUIT (QUIT) | 키보드로부터 오는 실행 중지 시그널.<br>[CTRL] + [w] 입력 시에 보내지는 시그널입니다.<br>기본적으로 프로세스를 종료시킨 뒤 코어를 덤프하는 역할을 합니다.                 | 코어 덤프 |
| 4  | SIGILL (ILL)   | illegal instruction의 약자입니다. 잘못된 명령을 사용했을 때 발생합니다.                                                               | 코어 덤프 |
| 5  | SIGTRAP (TRAP) | trace(추적), breakpoint(중지점)에서 TRAP 발생할 때                                                                         | 코어 덤프 |
| 6  | SIGABRT (ABRT) | abort의 약자로 비정상종료 함수에 의해 발생합니다.<br>(즉 abort 시스템 호출을 하였을 때 발생)                                                    | 코어 덤프 |
| 7  | SIGBUS         | 메모리 접근 에러시 발생하는 시그널입니다.                                                                                         | 코어 덤프 |

|    |                 |                                                                                                          |           |
|----|-----------------|----------------------------------------------------------------------------------------------------------|-----------|
| 9  | SIGKILL (KILL)  | KILL! 무조건 종료, 즉 프로세스를 강제로 종료시키는 시그널!                                                                     | 종료        |
| 11 | SIGSEGV         | invalid memory reference                                                                                 | 종료 + 코어덤프 |
| 15 | SIGTERM (TERM)  | Terminate의 약자로 가능한 정상 종료시키는 시그널로 kill 명령의 기본 시그널입니다.                                                     | 종료        |
| 17 | SIGCHLD (child) | 자식 프로세스가 stop 되거나 종료되었을 때 부모에게 전달되는 신호입니다. (멀티 프로세스 코딩에서 자세한 사용법은 배울 거..)                                | 무시        |
| 18 | SIGCONT (CONT)  | Continue의 약자로 STOP 시그널에 의해 정지된 프로세스를 다시 실행시킬 때 사용됩니다.                                                    | 재시작       |
| 19 | SIGSTOP (STOP)  | 터미널에서 입력된 중지 시그널입니다. SIGCONT로 재실행시킬 수 있습니다.                                                              | 중지        |
| 20 | SIGTSTP (TSTP)  | 실행 정지 후 다시 실행을 계속하기 위해 대기시키는 시그널입니다.<br>[CTRL] + [z]를 입력했을 때 보내지는 시그널입니다.<br>SIGCONT로 역시 다시 실행시킬 수 있습니다. | 중지        |
| 29 | SIGIO           | 비동기 입출력이 발생했을 경우 ! (I/O now possible!)                                                                   | 종료        |

# Signal

---

## ■ Signal 처리

- signal과 sighandler 함수를 정의하여 처리
- `void signal( int, function_pointer )`
  - Int에 해당하는 signal이 발생하면 function\_pointer에 해당하는 함수로 처리하라

# IPC

---

## Inter-Process Communication

- 프로세스들 사이에 데이터를 주고 받는 방식
  - File
  - Signal
  - Socket : 네트워크를 통하여 이동
  - Message Queue : 패킷으로 관리
  - Pipe : 프로세스간 연결되는 file descriptor 생성
  - Shared Memory : 공유 메모리를 할당 가능

# 파일 시스템 관련

---

Unix file system

`sudo rm -rf /`

Permission, File/Directory Information

File Creation & File I/O Function

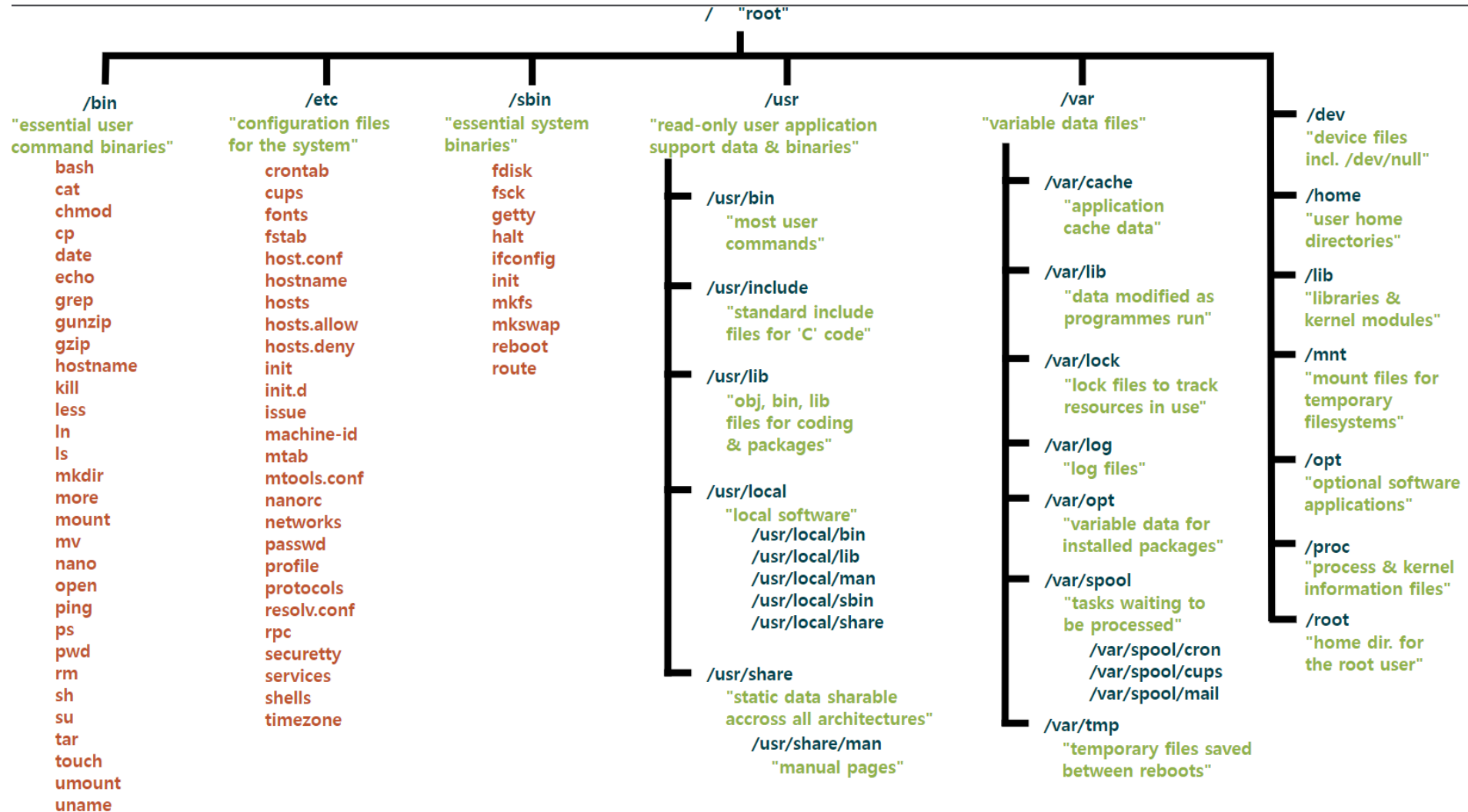
# Unix File System

---

Unix는 어떤 구조로 되어있고, 어떻게 데이터를 저장하는 가?

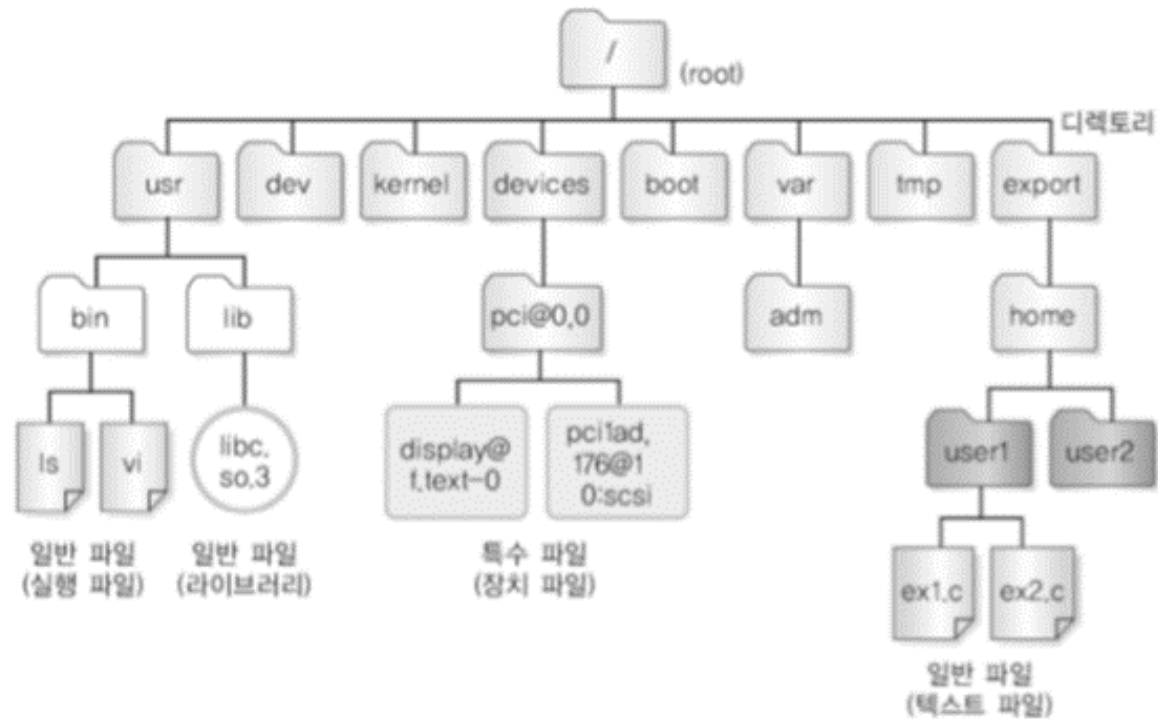
# Unix file system

## File System



# Unix file system

## File System

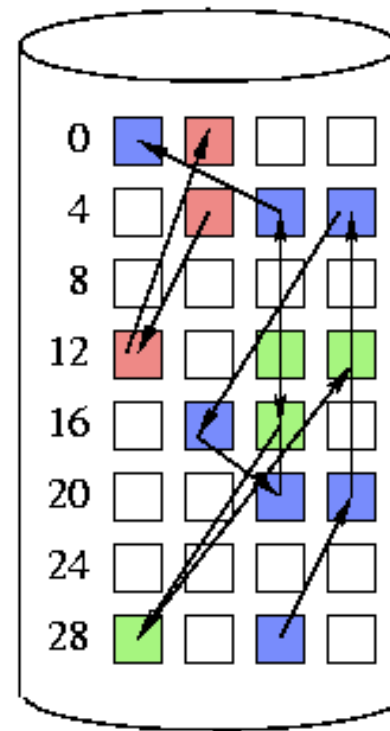




# Unix file system

## 하드웨어에서 파일은 어떻게 관리 되는 가?

- 가상메모리가 아니라, 실제 하드 디스크에서 파일은?
  - 조각나서 데이터를 저장하고 있음.
  - Linked list 형태로 다음 데이터의 위치를 가리켜서 데이터를  
모음
  - 장점 / 단점



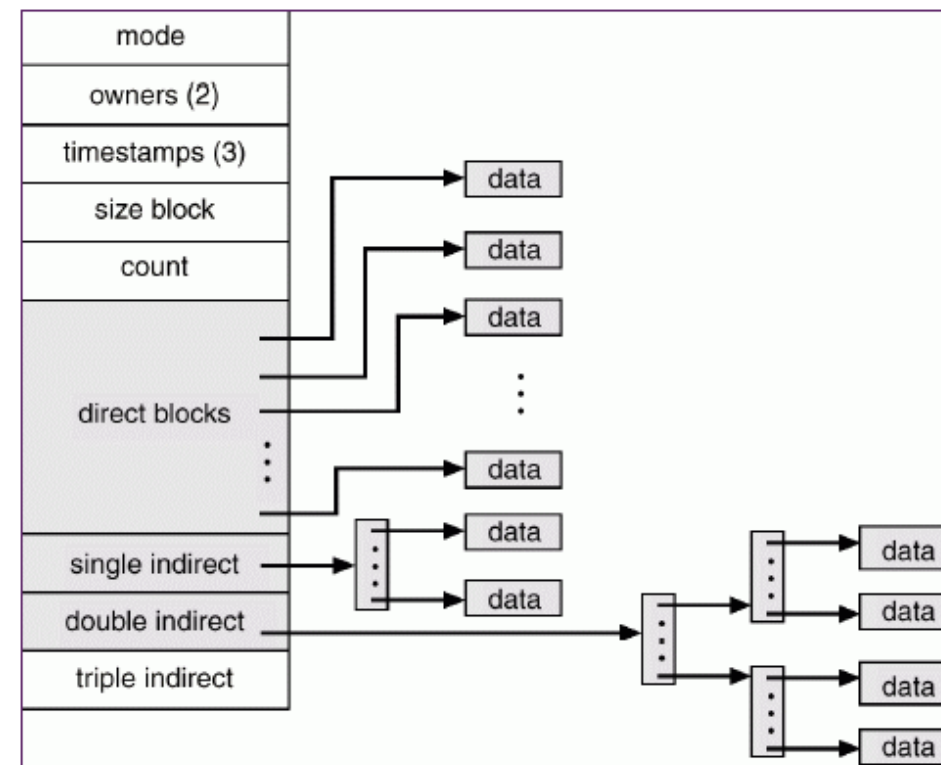
| Directory |       |     |
|-----------|-------|-----|
| File      | Start | End |
| moo       | 5     | 1   |
| snow      | 30    | 0   |
| fall      | 14    | 15  |

# Unix file system

## 하드웨어에서 파일은 어떻게 관리 되는 가?

- Inode란,
  - 조각난 데이터들을 파일, 디렉토리 등으로 인식하게 해주는 것
  - Metadata, 데이터 블록 정보, 블록 사이즈 등을 저장
  - inode 테이블이 따로 존재 => Forensic관점에서 굉장히 중요
- Metadata란,
  - 파일의 소유자 uid, gid, 파일크기, 권한, 변경된 시각 등
- stat()함수를 통해서 관련 정보를 가져올 수 있음.

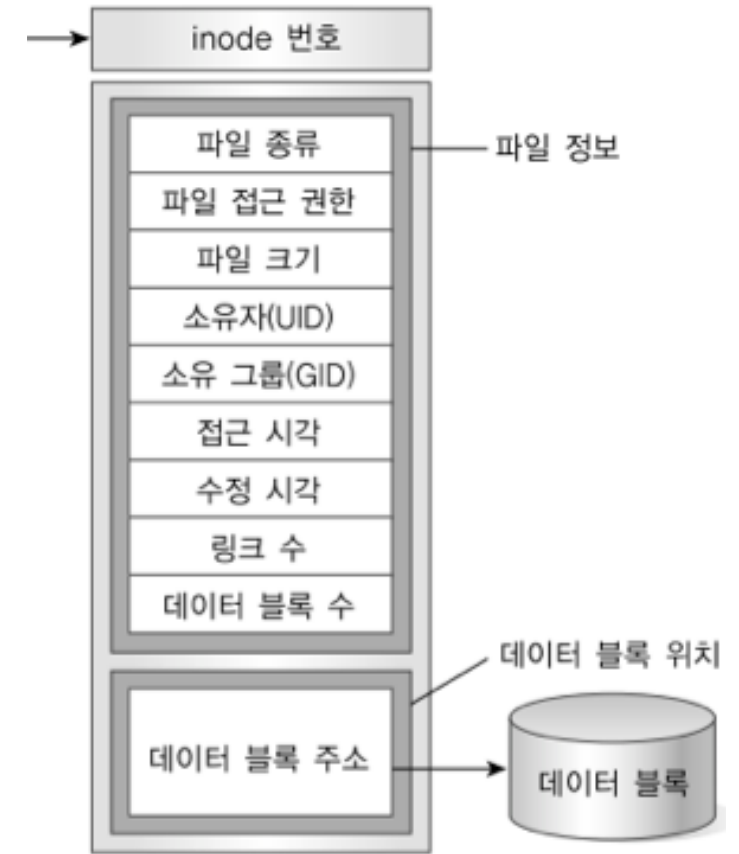
```
ugonfor@gongon:~/ugonfor/Cyber-Guardians-Lecture-Note/Week2$ stat babyVM/Makefile
File: babyVM/Makefile
Size: 151      Blocks: 8      IO Block: 4096   regular file
Device: 810h/2064d Inode: 220378  Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/  ugonfor)   Gid: ( 1000/  ugonfor)
Access: 2021-10-14 22:09:54.840000000 +0900
Modify: 2021-10-14 22:08:28.850000000 +0900
Change: 2021-10-14 22:08:28.850000000 +0900
Birth: -
```



# Unix file system

## 하드웨어에서 파일은 어떻게 관리 되는 가?

- 파일 하나에 Inode 하나가 무조건 매칭 되어야 함.
- Inode 정보가 없다면, 파일로 인식을 못한다.



# Unix file system

---

```
sudo rm -rf /
```

- <https://drive.google.com/drive/folders/18pWBnR6Nk742pBFaM0eTW7q-jCHFM2-G?usp=sharing>

# Permission File/Directory Information

---

# Permission

---

drwxrwxrwx

Q. 파일 접근 권한(Permission) 은 몇 바이트 인가?

# Permission

drwxrwxrwx

Q. 파일 접근 권한(Permission) 은 몇 바이트 인가?

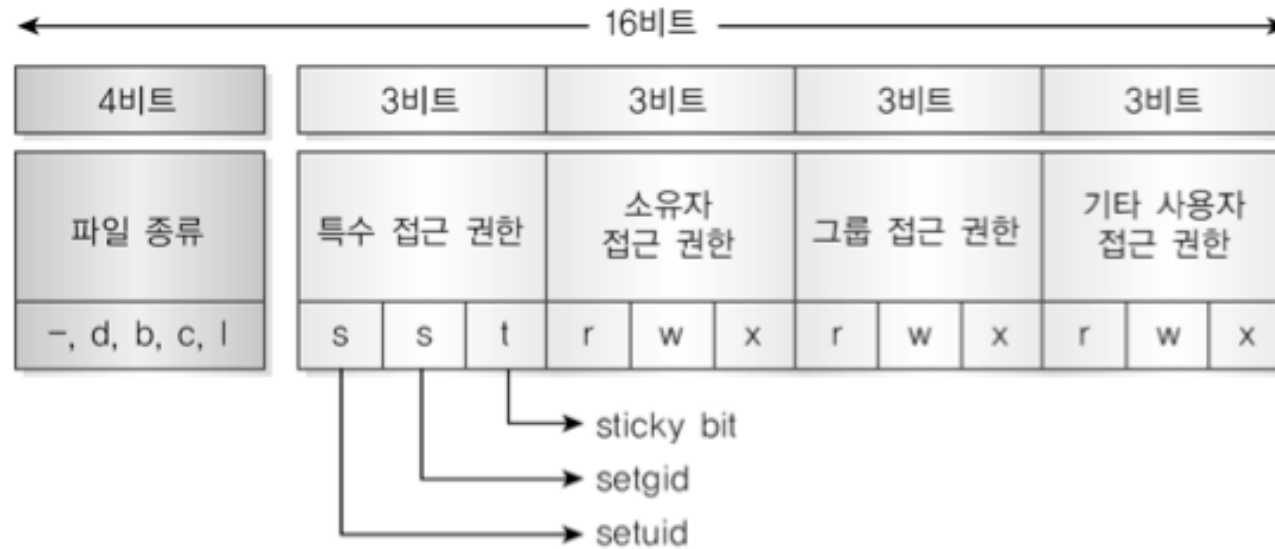
```
1 2 3 4
ugonfor@gongon:~/ugonfor/tmp$ ll
total 8
drwxr-xr-x 2 ugonfor ugonfor 4096 Oct 15 12:15 ./
drwxr-xr-x 9 ugonfor ugonfor 4096 Oct 12 01:08 ../
-rw-r--r-- 2 ugonfor ugonfor    0 Oct 15 12:13 1
-rw-r--r-- 2 ugonfor ugonfor    0 Oct 15 12:13 2
lrwxrwxrwx 1 ugonfor ugonfor    1 Oct 15 12:13 3 -> 2
prw-r--r-- 1 ugonfor ugonfor    0 Oct 15 12:15 4|
```

| 상수명      | 상수값(16진수) | 기능                              |
|----------|-----------|---------------------------------|
| S_IFMT   | 0xF000    | st_mode 값에서 파일의 종류를 정의한 부분을 가져옴 |
| S_IFIFO  | 0x1000    | FIFO 파일                         |
| S_IFCHR  | 0x2000    | 문자 장치 특수 파일                     |
| S_IFDIR  | 0x4000    | 디렉토리                            |
| S_IFBLK  | 0x6000    | 블록 장치 특수 파일                     |
| S_IFREG  | 0x8000    | 일반 파일                           |
| S_IFLNK  | 0xA000    | 심볼릭 링크 파일                       |
| S_IFSOCK | 0xC000    | 소켓 파일                           |

# Permission

drwxrwxrwx

Q. 파일 접근 권한(Permission) 은 몇 바이트 인가?





# File/Directory Information

---

## 파일, 디렉토리 관련 함수

- `int stat(const char *pathname, struct stat *statbuf);`
- `int access(const char* path, int amode);`
- `int chmod(const char* path, mode_t mode);`
- `int fchmod(int fd, mode_t mode)`
- `int symlink(const char *target, const char *linkpath);`
- `int lstat(const char *pathname, struct stat *statbuf);`
- `ssize_t readlink(const char *pathname, char *buf, size_t bufsiz);`
- `char *realpath(const char *path, char *resolved_path);`

# File/Directory Information

---

## 파일, 디렉토리 관련 함수

- 디렉토리 생성 : mkdir (2)
  - `int mkdir(const char *path, mode_t mode);`
- 디렉토리 삭제 : rmdir (2)
  - `int rmdir(const char *path);`
- 디렉토리 이름 변경 : rename (2)
  - `int rename(const char *old, const char *new);`
- 현재 작업 디렉토리 위치 : getcwd (3)
  - `char *getcwd(char *buf, size_t size);`
- 디렉토리 이동 : chdir (2)
  - `int chdir(const char *path);`
- 오류 발생 시 리턴 값이 -1

# File/Directory Information

---

## 파일, 디렉토리 관련 함수

- 디렉토리 열기 : opendir (3)
  - include <dirent.h>
  - DIR \*opendir(const char \*dirname);
- 디렉토리 닫기 : closedir (3)
  - int closedir(DIR \*dirp);
- 디렉토리 정보 읽기 : readdir (3)
  - struct dirent \*readdir(DIR \*dirp)

# File Creation & File I/O Function

---

# File Creation & File I/O Function

## open vs fopen

|        | 저수준 파일 입출력                                                                                                    | 고수준 파일 입출력                                                                                                                               |
|--------|---------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| 파일 지시자 | int fd                                                                                                        | FILE *fp;                                                                                                                                |
| 특징     | <ul style="list-style-type: none"><li>- 훨씬 빠르다</li><li>- 바이트 단위로 읽고 쓴다</li><li>- 특수 파일에 대한 접근이 가능하다</li></ul> | <ul style="list-style-type: none"><li>- 사용하기 쉽다.</li><li>- 버퍼 단위로 읽고 쓴다.</li><li>- 데이터의 입출력 동기화가 쉽다.</li><li>- 여러 가지 형식을 지원한다.</li></ul> |
| 주요 함수  | open, close, read, write, dup, dup2, fcntl, lseek, fsync                                                      | fopen, fclose, fread, fwrite, fputs, fgets, fprintf, fscanf, freopen, fseek                                                              |

# File Creation & File I/O Function

---

## Low level functions

- `int open(const char *pathname, int flags, mode_t mode);`
- `int close(int fd);`
- `ssize_t read(int fd, void *buf, size_t count);`
- `ssize_t write(int fd, const void *buf, size_t count);`
- `off_t lseek(int fd, off_t offset, int whence);`
- `int dup(int oldfd);`
- `int dup2(int oldfd, int newfd);`
- `int fcntl(int fd, int cmd, ... /* arg */);`
- `int unlink(const char *pathname);`
- `int remove(const char *pathname);`

# File Creation & File I/O Function

## High level functions

- `FILE *fopen(const char *filename, const char *mode);`
- `int fclose(FILE *stream);`
- `int fgetc(FILE *stream);` // 문자 한 개를 unsigned char형
- `int getc(FILE *stream);` // 매크로
- `int getchar(void);` // 매크로
- `int getw(FILE *stream);` // 워드 단위로 읽기
- `int fputc(int c, *stream);`
- `int putc(int c, *stream);`
- `int putchar(int c);`
- `int putw(int w, FILE *stream);`
- 문자열 기반 입력 함수
  - `char *gets(char *s);` // 표준입력에서 문자열 읽기
  - `char *fgets(char *s, int n, FILE *stream);` // 파일(stream)에서 n 보다 하나 적은 문자열을 읽기 (한줄씩)
- 문자열 기반 출력 함수
  - `char *puts(const char *s);`
  - `char *fputs(const char *s, FILE *stream);`
- Buffer 기반 함수
  - `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`
  - `size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);`
- 형식 기반 함수
  - `int scanf(const char *restrict format, ...);`
  - `int fscanf(FILE *restrict stream, const char *restrict format, ...);`
  - `int printf(const char *restrict format, /* args */ ...);`
  - `int fprintf(FILE *restrict stream, const char *restrict format, /*args */ ..)/`
- `int fseek(FILE *stream, long offset, int whence);`
- `long ftell(FILE *stream);`
- `void rewind(FILE *stream);`
- `int fsetpos(FILE *stream, const fpos_t *pos);`
- `int fgetpos(FILE *stream, fpos_t *pos);`

# 실습

---

1. 저번 과제
2. 쉼 기능 구현 (과제)



# labyrevnt

---

■ 플이

# Shell 기능 구현

---

## Simple Shell

- 주어진 Simple Shell 스켈레톤 코드에 다음 기능들을 구현할 것
- mv 구현
- ls 구현
  - ls
  - ls -l
  - ls -la
  - ls --help (help는 자신이 작성한 ls에 대한 설명)
  - 위 기능 모두 구현할 것, 정렬은 안해도 됨. 데이터만 제대로 출력 되게.
- sigint(ctrl+c) 처리
  - signal, sighandler로 처리
- sigtstp(ctrl+z) 와 sigcont(fg)로 process stop, foreground구현 (Challenge, 선택)
  - jobs, fg, bg, kill 검색해볼 것

# Shell 기능 구현

---

## ■ github repository 설명

- repository name : SimpleShell
- public으로 할 것
- Makefile, lsh.c, lsh.h, main.c, README.md
  - 위 파일들이 repository에 포함되어 있어야 함.

### ! Caution

- 베끼지 마세요
- 본인의 실력향상에 굉장히 도움되는 과제이기에 꼭 스스로 해볼 것
- 친구들과 상의하는 것 O
- 코드 복붙 X

# 과제

---

- 메일 제목: [SimpleShell][디미고]본인이름
  - ex) [SimpleShell][디미고]유효곤
- 내용: 본인 repository url을 포함하여! (파일, 코드 제출 x)

제출기한: 10/29 수업시작 전까지

제출: ugonfor@gmail.com

# reference

---

- CS230(KAIST) <https://cp-git.kaist.ac.kr/cs230/cs230/-/tree/main/labs/lab5>
- <https://github.com/brenns10/lsh> (Simple Shell Skeleton Code)
- Computer Systems: A Programmer's Perspective by Randall Bryant and David O'Halloran
- CYDF311(KoreaUniv, CYDF)
- CYDF211(KoreaUniv, CYDF)