# 서술형 과제 (3주차)

## 1. 리눅스 운영체제는 여러 배포판이 있습니다. 이 중에서 가장 널리 사용되는 Debian GNU, Ubuntu, CentOS, RedHat 4가지의 차이점을 정리하여 작성해주세요.

- 각 배포판별 장점과, 주요 사용처를 포함하여 작성해주세요.

**Debian GNU**

**주요 사용처: 다양한 패키지를 통한 기능 설정 및 의존성 관리가 필요한 서버 구축**

- 무료이며, FSF(Free Software Foundation)의 GNU 프로젝트 지원을 받아 개발
- 2250개의 소프트웨어 패키지를 포함한 거의 모든 기능을 포함하고 있으며, 패키지 및 유틸리티 또한 풍부하다
- 안정적, 탁월한 기능성, 높은 수준의 호환성과 유연성을 특징으로 함
- 유명 소프트웨어와 상용 불가능, 설치는 쉽지만 설정이 까다로움

**Ubuntu**

**주요 사용처: 개인 사용자(가정용 서버), 데스크탑용, 학습 용도 등**

- 데비안 리눅스 기반
- '전 세계 사람 누구나 어렵지 않게 리눅스를 사용할 수 있게 하자'라는 모토 하에, '다른 사람을 위한 인간애'라는 의미를 담은 네이밍 (=우분투)
- '현존 리눅스 배포판 중 가장 유명하고 널리 보급되었음
- GNU/Linux 기반, 사용자 편의성에 초점을 맞춰 개발
    - 누구나 쉽게 설치하여 사용할 수 있다
    - v11.04부터 Ubuntu Unity가 탑재되어 다양한 그래픽 도구를 사용 가능
    - 최상의 범용성을 자랑함
        - 웹서버, 가정용 서버 등 + 해외에서 기업 도입 사례도 증가 추세
        - 최근 모바일/안드로이드용 우분투 버전도 발표되었음

**CentOS**

**주요 사용처: 기업용 서버, 데스크탑용**

- RedHat의 RHEL 버전의 클론 버전으로, 무료 배포본
- 무료지만 RedHat RHEL과 큰 차이가 없음
- 소프트웨어 패키지 관리도구 yem 기능은 오히려 Redhat보다 낫다는 평가

**RedHat**

**주요 사용처: 기업용 서버, 개발**

- 일반적으로 RedHat의 RHEL 버전을 지칭
- 전 세계 (유료) 리눅스 시장의 70-80% 점유
- 기업용 서버 OS로 유명, 현업에서 가장 널리 사용되는 OS

## 2. 리눅스 파일 시스템(ext file system)에 대해서 정리한 내용을 작성해주세요.

운영체제 파일시스템
파일 열기, 읽기, 쓰기, 닫기.

- 네트워크도 유사하게 적용 가능, 쓰기=전송, 읽기=수신
- CD 또한 읽기-쓰기 개념으로 이해할 수 있음
- 마우스는 읽기 전용, 마우스 물리적인 움직임 및 버튼 입력 = 읽기
- 네트워크, 하드웨어 등 모두 열기/읽기/쓰기/닫기라는 파일 인터랙션으로 이해하고 접근

이러한 VFS (가상파일시스템) 개념은 리눅스로부터 탄생하였음

리눅스는 모든 것이 파일이라는 철학을 따름

- 모든 인터랙션은 파일을 읽고, 쓰는 것처럼 구성
- 마우스, 키보드 등 디바이스 관련 기술도 파일처럼 취급함
    - 키보드라는 파일 아래에 입력한 데이터를 작성하고, 그 데이터를 운영체제가 가져간다면, 키보드 입력이 운영체제로 input되는 것처럼 보일 것
- 모든 자원에 대한 **추상화 인터페이스**로 파일 인터페이스를 활용한다.

리눅스 파일은 inode 기반 알고리즘을 따른다

-

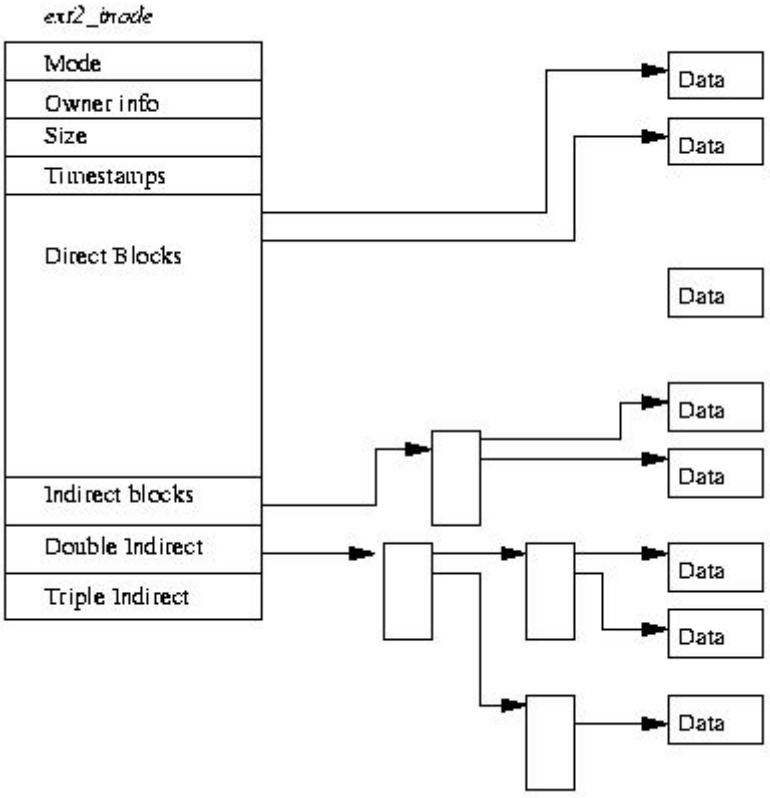
파일 네임스페이스 = 공간에 대해 이름을 붙이는 개념

- A드라이브 (A:/), C드라이브(C:/Windows) 등으로 구분하지 않음
- 전역 네임스페이스 사용
    - /media/floofy/dave.jpg
    - 예 : cat tty
        - 터미널 인터페이스 조차도 VFS 인터페이스를 통해 일종의 파일처럼 취급
    - 플로피 디스켓을 삽입하면, 별도 공간임에도 불구하고, 마치 루트 디렉토리의 하위 디렉토리에 있는 것처럼 취급하여 접근한다.
- 파일은 inode 고유값과 자료구조에 의해 주요 정보 관리
    - 디바이스 포함 모든 것을 파일이라고 취급하고 관리하므로, 모든 것들에 대한 inode 구조체가 존재한다고 말할 수 있다.

슈퍼블록, inode와 파일

- 슈퍼블록
    - 파일 시스템의 정보를 가지고 있다.
    - 전체 사이즈(ex. 10GB)
    - 어떠한 파일시스템 알고리즘을 사용하는지
- 파일(Inode)
    - inode 고유값과 자료구조에 의해 주요 정보를 관리
    - ‘파일이름:inode’로 파일이름은 inode 번호와 match (맵핑됨)
    - 파일 시스템에서는 inode 기반으로 파일 액세스
    - inode 기반 메타 데이터 저장

- 프로세스 내부에 프로세스 상태정보를 담는 PCB가 존재하듯이, inode도 inode 요약정보(메타데이터)를 포함하고 있다




**리눅스 파일 시스템(ext file system)과 inode**

- inode 기반 메타 데이터
  - 파일 권한 (r, w, x)
  - 소유자 정보
  - 파일 사이즈
  - 생성시간 등 시간 관련 정보
  - 데이터 저장 위치
    - Disk Block
    - 파일은 일정 크기의 블럭 단위로 나뉘어진 상태로 관리
      - Direct Blocks - 직접 액세스 가능한 블럭 첫 주소들의 정보
      - Indirect Blocks - 간접 블럭을 경유하여, 여러 Direct Blocks에 접근할 수 있음 (ex. 3X3)
      - Double Indirect - 간접 블럭을 한층 더 경유하도록 계층화한 블럭 (ex. 3X3X3)
      - ...
  - ...

파일과 inode
- 리눅스 파일 탐색 : 예 - /home/ubuntu/link.txt

- 디렉토리도 파일로 취급되므로 마찬가지로 inode를 갖고 있으며, inode 기반 mapping을 따른다. 추가적으로 일반 파일과는 다르게 탐색을 위해 dentry를 갖는다.
    1. 각 디렉토리 엔트리(dentry) 탐색
        a. 각 엔트리는 해당 디렉토리 파일/디렉토리 정보를 갖고 있음
        b. root, home, ubuntu 폴더는 각각 dentry를 갖고 있음
        c. dentry는 아래 요소로 구성
            i. 파일 - (inode:파일) mapping
            ii. 서브디렉토리 - (inode:이름) mapping
            iii. ...
    2. inode mapping을 이용하여
        a. '/' dentry에서 'home'을 찾고
        b. 'home'에서 'ubuntu'를 찾고
        c. 'ubuntu'에서 link.txt 파일이름에 해당하는 inode를 얻음

특수 파일
- 디바이스
    - 블록 디바이스 (Block Device)
        - ls -al 출력에서 b로 시작함
        - HDD, CD/DVD와 같이 블록 또는 섹터 등 정해진 단위로 데이터 전송
        - IO 송수신 속도 높음
    - 캐릭터 디바이스 (Character Device)
        - ls -al 출력에서 c로 시작함
        - 키보드, 마우스 등 byte 단위 데이터 전송
        - IO 송수신 속도 낮음

리눅스와 권한
- 운영체제는 사용자/리소스 권한을 관리
- 리눅스는 사용자/그룹으로 권한을 관리
- root = 슈퍼관리자
- 파일마다 소유자, 소유자 그룹, 모든 사용자에 대해
    - 읽고, 쓰고, 실행하는 권한 관리
    - 접근 권한 정보는 inode 자료구조에 저장

리눅스 사용자 (로그인 사용자/ 그룹) 별 권한 관리
리눅스 리소스 권한 관리
- (소유자, 소유자 그룹, 모든 사용자에 대한 읽고, 쓰고, 실행하는 권한 설정)

**3. 다음은 실제로 리눅스 운영체제에서 시스템 콜을 구현한 어셈블리 코드 예제이다.**
**아래 코드를 해석하여, 어떤 프로그램 명령을 수행한 코드인지 서술해주세요.**

```
mov     eax,    0x04
mov     ebx,    0x01
mov     ecx,    $buf
mov     edx,    14

int     0x80
```

- eax, ebx, ecx, edx 레지스터의 역할을 포함합니다.
- 0x04, 0x01이 의미를 포함합니다.
- int 0x80의 의미를 포함합니다.

**mov eax, 0x04**

- eax 레지스터에 **시스템 콜 번호 0x04** 입력
    - eax 레지스터 (Extended Accumulator Register)
    - 사칙연산(+,-,*,/) 및 논리연산 함수의 반환값을 저장하는 레지스터

**mov ebx, 0x01**

- ebx 레지스터에 **시스템 콜에 해당하는 함수 인자값 0x01** 입력
    - ebx 레지스터 (Extended Base Register)
    - 메모리 주소를 저장하는 레지스터

**mov ecx, $buf**

- ecx 레지스터 (Extended Counter Register)
- 반복 명령어 (while, for 등) 사용 시 반복 카운터로 사용되는 레지스터로, 반복 할 횟수를 지정하고 반복 작업을 수행하는 데 사용

**mov edx, 14**

- edx 레지스터 (Extended Data Register)
- 사칙연산과 관련된 eax 레지스터와 같이 사용되며, 부호 확장 명령을 내리고 큰 수의 사칙연산이 발생할 때 사용됨

**int 0x80**

- **소프트웨어 인터럽트 명령을 호출**하면서 0x80값 전달
- int = instruction code to CPU
- 0x80 = 십진수 128

## 4. 리눅스 운영체제에서 사용자가 CLI를 통해서 “gcc -o test test.c” 명령어를 실행했을 때 실제 내부에서 동작하는 방식을 상세하게 서술해주세요.
## <참고 test.c의 실제 코드 내용>


- 사용자가 커멘드창을 오픈하면, 자동으로 쉘프로그램이 실행됨
- gcc 명령은 주어진 C 파일을 컴파일 하는 명령어임
- 해당 쉘에서는 키보드를 입력하면, 해당 키보드 문자가 화면에 표시하며, 엔터를 누르면 그동안 입력받은 문자열을 명령으로 인식하고, 해당 명령을 실행함
- 스케쥴링 방식은 선점형을 지원하며, 기본적으로 멀티 태스킹을 지원함
- 인터럽트가 발생하며 내부에서는 gcc 컴파일러가 수행됨
- gcc 컴파일러는 주어진 옵션으로 test.c에 해당하는 test.o를 생성함
- test.o 파일을 기준으로 실행파일인 test 파일을 생성함

➡ 위에서 기술한 사항 외에 추가 가정이 꼭 필요한 경우에는 각자 가정한 상황을 기술하고, 동작 방식을 기술하세요.

1. 사용자 커맨드창 오픈
2. **gcc -o test test.c** 키보드를 사용하여 글자 1개 입력할 때마다 아래 내용이 실행
    a. 키보드(하드웨어) 인터럽트 발생
    b. eax 레지스터에 시스템 콜 번호 input
    c. ebx 레지스터에 시스템 콜에 해당하는 인자값 input
    d. 소프트웨어 인터럽트 명령 호출하면서 CPU opcode 0x80(인터럽트 번호) 넘겨줌
    e. CPU는 사용자 모드에서 커널 모드로 전환
    f. IDT(Interrupt Descriptor Table)에서 0x80에 해당하는 주소(함수)를 찾아 실행
        i. 컴퓨터 부팅시 운영체제가 인터럽트 각각의 번호와 실행 코드를 가리키는 주소를 IDT에 기록
        ii. 운영체제 내부 코드이므로 현재 사용자 모드라면 커널 모드로 전환 필요함
    g. system_call() 함수의 eax로부터 시스템 콜 번호를 찾아, 해당 번호에 맞는 시스템 콜 함수로 이동
    h. 해당 시스템 콜 함수를 실행
    i. CPU는 커널 모드에서 사용자 모드로 복귀
    j. 해당 키보드 문자를 화면에 표시
3. 사용자 키보드 enter 입력
    a. 키보드(하드웨어) 인터럽트 발생
    b. eax 레지스터에 시스템 콜 번호 input
    c. ebx 레지스터에 시스템 콜에 해당하는 인자값 input

d. 소프트웨어 인터럽트 명령 호출하면서 CPU opcode 0x80(인터럽트 번호) 넘겨줌
e. CPU는 사용자 모드에서 커널 모드로 전환
f. IDT(Interrupt Descriptor Table)에서 0x80에 해당하는 주소(함수)를 찾아 실행
    i. 컴퓨터 부팅시 운영체제가 인터럽트 각각의 번호와 실행 코드를 가리키는 주소를 IDT에 기록
    ii. 운영체제 내부 코드이므로 현재 사용자 모드라면 커널 모드로 전환 필요함
g. system_call() 함수의 eax로부터 시스템 콜 번호를 찾아, 해당 번호에 맞는 시스템 콜 함수로 이동
h. 해당 시스템 콜 함수를 실행
i. CPU는 커널 모드에서 사용자 모드로 복귀
j. 현재까지 입력된 문자열을 명령으로 인식

4. **gcc -o test test.c 명령 실행**
    a. 인터럽트 발생
    b. gcc 컴파일러 실행
        i. 현재 디렉토리의 dentry로부터 디렉토리 파일 및 정보 탐색
        ii. test.c 파일을 inode 기반 mapping 정보를 바탕으로 검색
        iii. 현재 디렉토리에서 test.c의 존재를 확인함
        iv. 주어진 명령어 옵션에 따라 test.c에 해당하는 test.o 생성
        v. test.o 파일을 기준으로 실행파일인 test 파일 생성
    c. 인터럽트 종료
    d. 이전 프로세스로 복귀