<https://www.computerhope.com/jargon/s/special-file.htm>

<https://ehclub.co.kr/1318?category=600567>

1장 베이직

커널 : 운영체제의 심장, OS를 규정짓는 매우 중요한 부분. 하드웨어의 자원을 자원이 필요한 프로세스에 나눠주고 프로세스 제어, 메모리 제어, 프로그램 시스템 콜 등 수행하는 부분으로 운영체제 맨 하부에서 돌아감.

부트 로더 : 컴퓨터가 시동되었을 때 운영 체제에 필요한 조치를 취하고 하드디스크 드라이브에 기록되어 있는 운영체제를 실행시키는 프로그램

시스템 콜 : 커널이 제공하는 서비스에 대해 응용 프로그램의 요청에 따라 커널에 접근하기 위한 인터페이스

디렉토리는 디렉토리 정보를 담고있는 파일이다. (디렉토리 엔트리는 i노드와 파일 이름을 담고있다.)

유닉스 파일시스템은 디렉토리와 파일들의 계층구조이다.

모든 것은 /라는 이름을 가진 루트 디렉토리에서부터 시작된다.

하드링크 : 파일의 아이노드를 똑같이 참조함.

심볼릭링크 : 파일의 바로가기 파일 같은 느낌…

/, null 캐릭터는 파일 이름으로 못쓴다.

새로운 디렉토리가 생성이 되면 . 과 .. 이 자동으로 생성된다.

.은 현재 디렉토리이고 ..은 부모 디렉토리이다.

모든 프로세스는 워킹 디렉토리를 가진다.

로그인하면 워킹 디렉토리가 홈 디렉토리(/etc/passwd)로 바뀐다.

유닉스는 일반 파일들 뿐만 아니라 주변 장치 및 프로세스간 통신 채널도 파일의 개념을 확장한다.

프로세스간 통신 채널 : FIFO, PIPE, SOCKET

일반 파일 : 바이너리 또는 텍스트 파일(유닉스는 차이점을 몰라용 헤헤)

디렉토리 파일 : 다른 파일들의 이름과 위치를 담고있는 파일

캐릭터 스페셜, 블록 스페셜 파일 : 터미널(캐릭터 스페셜) 그리고 디스크(블록 스페셜)

블록 스페셜 파일 : 블록 단위로 I/O를 진행하는 모든 장치에 대한 직접적인 인터페이스를 제공한다. Blockdev –getbsz /dev/sda1 명령어를 통해 1블록 크기 알 수 있음.(바이트)

캐릭터 스페셜 파일 : 블록 스페셜 파일과 비슷하지만 기준이 블록이 아닌 캐릭터이다. Stdin, stdout, stderr등이 이에 속함.

FIFO(named pipe) : 프로세스간 통신에 쓰이는 파일 타입이다.

Socket : 프로세스 사이의 네트워크 통신에 사용되는 파일 타입이다.

이외에도 소켓, 심볼릭 링크, 레귤러파일, 디렉토리, FIFO(PIPE)등이 있다.

Ownership, Permission

각 파일들은 개개인의 유저의 소유이다.

소유자는 파일과 연관된 권한을 고를 수 있다.

권한

어떻게, 그리고 파일에 누가 접근할 수 있는지 조종한다.

소유권은 특정한 권한을 부여하며, 권한이라 불리는 파일의 속성을 변화시킬 수 있다.

d : directory

l : symbolic link

b : block special file

c : character special file

p : FIFO(PIPE)

-: regular file

실행중인 프로그램의 인스턴스 : ls, ps

프로세스간 통신(IPC)

Pipe, FIFO, Signals, Shared memory, Semaphore, Sockets(네트워크를 통한 협력 프로세스에 쓰임)

Ls -l ( long form)

Ls – al (hidden file and long form)

Ls ( 파일의 목록을 보는 것)

Ps (프로세스의 목록을 보는 것)

Ps -ef (시스템 내의 모든 프로세스 목록)

시스템콜 발동 시 유저 프로세스에서 커널모드로 바뀜, 트랩에 걸린 것(인터럽트) 이후 리턴하면 user mode로 되돌아감

fread를 하면 욜로 갔다가 read를 호출함

시스템콜은 read만!

시스템 콜은 두 번의 콘텍스트 스위치를 거친다. (유저에서 커널로 갔다가 다시 돌아오는 것)

시스템 콜을 여러 번 하는 것을 피해야 성능 개선에 유리하다.

모든 시스템 콜은 헤더파일에 정의되어 있다.

------------------------------------------- 2장 파일 (1) ----------------------------

데이터를 담는 콘테이너이다.

파일 : 바이트의 연속적인 시퀀스

운영체제에서 부과하는 방식이 없다.

각각의 바이트는 디스크 파일에서 주소 지정이 가능하다.

파일은 외부장치에 대해서도 균일한 인터페이스이다.

파일 시스템

컴퓨터 파일과 데이터들을 정렬하고 조직하기 위한 방법

파일들을 찾고 접근하기 쉽게 만든다.

하드디스크, CD-ROM등과 같은 데이터 저장 장치들을 사용한다.

파일 시스템 인터페이스(커널 수준)

open, create, close, read, write, lseek, unlink, remove, fcntl <- 시스템 콜들

<fcntl.h>

/usr/include

커널 수준에서 모든 오픈 파일들은 파일 디스크립터라고 언급된다.

음수가 아닌 정수로 나타내진다.

존재하는 파일을 열거나 새롭게 생성할 때, 커널은 프로세스에게 파일 디스크립터를 리턴

유닉스 셸에 의해 생성된 각각의 프로세스들은 터미널과 관련된 세가지 오픈 파일들과 함께 시작된다.

파일 디스크립터를 0, 1, 2에 이은 lowest 값에 올려놓는다. 이후 open한 파일은 파일 디스크립터저장이 되고 이를 read하여 메모리에 값을 저장한다

int open(const char \*pathname, int flags, [mode\_t mode]);

성공하면 파일 디스크립터를 리턴하고 실패하면 -1리턴

플래그에 들어가는 것은 다음과 같다.

O\_RDONLY #0 읽기 전용

O\_WRONLY #1 쓰기 전용

O\_RDWR #2 읽고 쓰기 가능

이 중 단 하나만 플래그로 사용해야함.

왜냐하면 O\_RDONLY가 0으로 설정되어 있기 때문에 O\_RDONLY || O\_WRONLY = O\_WRONLY이기 때문…

옵션으로 넣을 수 있는 친구들은 다음과 같다.

O\_APPEND 파일의 끝에 추가하기

O\_CREAT 존재하지 않는다면 파일을 생성

O\_EXCL 파일이 이미 존재하면 에러를 내뿜는다(O\_CREAT와 함께 사용)

O\_TRUNC 파일이 존재하면 비워버린다.

O\_NONBLOCK 논 블로킹 파일을 연다.

Mode는 O\_CREAT플래그와 함께 쓰인다. 권한을 나타냄. 0000꼴로 나타낸다. 맨 앞 0은 8진수라는 뜻.

파일 퍼미션은 유저, 그룹, 그 외 순이다. 각각은 rwx순이다. read, write, execute

글로 모드를 쓰자면 S\_I가 공통적으로 붙고 그 이외엔 RUSR, WUSR, XUSR, RWXU, RGRP, WGRP, XGRP, RWXG, ROTH, WOTH, XOTH, RWXO, SUID, SGID 등이 붙는다.

mote\_t fdmode = ( S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH )면 0644를 뜻한다.

create 시스템 콜

int create(const char \*pathname, mode\_t mode);

리턴값은 파일 디스크립터, 실패 시 -1

파일을 생성하기 위한 대안책

만약 파일이 이미 존재한다면 두번째 인자는 무시된다.

open과 같진 않다. creat는 파일 디스크립터를 반환하기 전에 무조건 존재했던 파일 내용을 지워버린다.

creat는 항상 쓰기 전용으로 파일을 연다.

즉 fd = create(“/tmp/newfile”, 0644) == fd = open(“/tmp/newfile”, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644); 와 같다.

System data type 은 \_t로 끝난다.

새로운 파일을 만들면(open, create) 새로운 링크가 거기에 더해지면 부모 디렉토리에서 권한을 써줘야한다.

디렉토리는 디렉토리의 파일의 리스트를 담고있는 파일이다.

그룹 및 소유자는 프로세스의 이펙티브 유저ID로부터 설정된다.

close 시스템 콜

#include <unistd.h>

int close(int filedes);

성공하면 0, 실패하면 -1 리턴

열려있는 파일을 닫는다.

모든 열린 파일들은 자동적으로 실행이 완료되면 끝난다.

read 시스템 콜

#include <unistd.h>

ssize\_t read(int filedes, void \*buffer, size\_t n);

읽은 바이트의 숫자 리턴, 파일의 끝이면 0 리턴, 실패하면 -1 리턴

파일 카피 바이트들을 현재의 파일 포지션에서 메모리까지 읽고 파일 포지션을 업데이트 한다.

파일 디스크립터, 데이터가 복사될 공간, 읽을 바이트 수

파일을 리드하면 오프셋이 읽은 바이트수만큼 변경된다.

write 시스템 콜

#include <unistd.h>

ssize\_t write(int filedes, const void \*buffer, size\_t n);

쓴 바이트만큼 리턴, -1이면 실패

메모리부터 현재 파일 포지션까지 파일에 쓴다. 그리고 파일 포지션을 업데이트한다.

파일 디스크립터, 쓰여질 파일의 데이터, 얼마나 쓸지 바이트

만약 프로그램이 쓰기 위해 이미 존재하는 파일을 연다면 오래된 데이터는 새로운 것에 의해 덮어쓰기된다.

O\_APPEND 옵션이 open될 때 활성화되면 파일의 오프셋은 EOF로 초기화된다. 즉 라이트가 끝부터 시작되는 것.

read, write 효율성

버퍼 사이즈가 클수록 성능 향상이 된다.

최고의 성능은 버퍼사이즈가 시스템 블록 사이즈의 배수일 때이다.

natural disk block : stat’s st\_blksize

시스템 콜 호출을 줄여야하는 것도 잊지말자!

write 시스템 콜이 빨라요. 왜?

Ssize\_t : primptive system data type

O\_CREAT가 없으면 있으면 뜨고 없으면 오류

있으면 뜨고 없으면 만든다

파일이 있으면 fail, 파일이 없을 때만 만든다.

파일이 있으면 3이 리턴되고 없으면 -1이 리턴된다.

User, group, others

R : Readable

W : Writable

X : Executable

Owner : 본 시스템 콜을 호출한 프로세스가 됨

Chmod : 권한을 변경해주는 함수, 기호 모드와 숫자 모드 두 가지로 사용 가능

Creat system call

Close

Read (int filedes, void \*buffer, size\_t n);

File position은 read write하는 위치를 나타낸다.

Read의 리턴값 : n개 만큼 읽었따! 즉 0이면 EOF라는 뜻.

프로세스가 실행이 되면 프로세스 테이블이 만들어지고 파일디스크립터 테이블도 만들어진다. 0부터 2까지는 std in, std out, std err이 된다. 파일 디스크립터번째 인덱스는 파일 테이블을 가리키고 파일 오프셋 다음 내용은 파일을 가리킨다.

시작 위치에서 거꾸로 가는 건 안돼! 마지막 위치에서 더 뒤로 가는 건 가능해…

파일을 오픈할 때마다 파일 디스크립터가 만들어진다.

i-Node : 파일에 1:1로 대응되는 노드(즉 곧 파일이다)

파일테이블에 연결된 파일 디스크립터의 개수

Dub : 파일 디스크립터 복사, 이는 곧 무슨 뜻이냐…

dub보다는 dup2를 더 많이 사용할 것입니다.

Dup2 : filedes1의 파일 디스크립터를 filedes2로 카피

Filedes2 를 가리키는 포인터가 사라지고 1번을 가리키게 됨.

**Dup2가 뭐하는 친구인지 정확하게 기억하자!!**

Redirection

Fopen : system call이 아님다

Restrict가 있는 변수끼리는 메모리를 공유하면 안된다.

어떤 시스템콜을 성공하고 에러넘버를 보는 건 아무 의미가 없다.

실패 하자마자 즉각적으로 에러넘버를 봐야 의미가 있다.

수퍼유저 맘대로

프로세스의 아이디는 리얼 유저와 이펙티브 유저 아이디가 있다.

대부분의 경우 리얼과 이펙트는 같다. 하지만 아주 특별한 경우 다르다.

이펙티브 유저 아이디가 실제론 더 영향을 많이 거의 끼침

유저 이름 같고 퍼미션 있으면 열 수 있다.

유저 이름 다르면서 그룹 같고 그룹 퍼미션 있으면 열 수 있다.

유저 이름 그룹 이름 다 다르고 other퍼미션 있으면 열 수 있다.

다른 사람의 파일 권한을 바꿀 순 없어.

스탯 아주 중요한 시스템 콜

파일과 1:1 대응이 되는 아이노드 자료구조

해당 정보를 가져오는 시스템 콜임.

Syn

디렉토리 read 퍼미션은 ls를 할 수 있는거

Write 퍼미션은 파일 만들거나 지울 수 있는거

Exe 퍼미션은 (search 퍼미션)이라고도 한다. 이게 있어야 cd를 할 수 있다. 어떤 파일을 오픈하려고 하면 이 파일이 위치한 디렉토리에 퍼미션이 있어야 오픈할 수 있다.

Getcwd 리턴값이 문자열로 나온다.

Sync, fsync : 버퍼에 남아있는 친구들을 강제로 디스크에 플러쉬 해주는 친구

메이저넘버 : 디바이스 타입, 같은 드라이버 타입

마이너 넘버 : 여러 개 있으면 그 중에 몇 번째냐?

매크로란? Function 비슷한데 function은 아니고 컴파일 할 때 실제 코드로 확장되는 친구

St\_mode에 따라서 첫 글자가 다르다! Ex) -, d, c, b, p, s, 등등..

프로세스

Getpid : 프로세스 ID 리턴

Getppid : 부모 프로세스 id 리턴

부모는 포크를 실행하고 리턴하지만 차일드는 포크를 실행하지 않고 리턴한다.

Parent fork 리턴값 : 자식의 pid 값 리턴

Child fork 리턴값 : 0 리턴

자식과 부모가 실행하는 부분이 겹치면 골치하프다…

Parent와 child가 실행하는 부분은 겹치지 않도록 강력히 권고하는 바이다도시

Pathname : 경로, finlename : 프로그램 이름

P가 있으면 PATH 환경변수에서 찾는다.

정상 종료면 exit(0)를 넣는다. 1이나 2면 something wrong…

exit 핸들러 : 유저가 클린업 액션을 직접 짜줘서 등록을 해주는 것이다. 대충 atexit이다.

wait = hang = block