fcntl.h: fd=**open**(filename, O_RDWR)/fd=creat(f_name,07644) 모든 장치는 드라이버 필요
/unistd.h : n_char=**read**(fd, &buf, bufsize)/n_char=**write**(fd,buf,size) sys/types.h 옮기기전위치=**lseek**(fd,sizeof(),SEEK_CUR)/
dirent.h,sys/type.h: DIR* dir_ptr=**opendir**(dirname)struct dirent*=**readdir**(dir_ptr)/**stat**(f_name,struct stat* buf) > buf=stat
chmod(path,04644)/**utime**(path,struct utimebuf* newtimes) utimebuf{atime(접근시간), modtime(변경시간)}으로 변경^stat.h
rename(old,new)/<sys/type.h,sys/stat.h> : **mkdir**(pathname,mode)<unistd.h>**rmdir(**path) **chdir**(path) **unlink**(path) **link(**orig, new)
pwd:"."inode(stat) > chdir.. > inode찾고 이름출력
special bit : suid : s root가 실행한것처럼 guid : s group 멤버가 실행한것처럼 sticky : t file :swap공간에 할당하라(파편화X)
directory :만든사람말곤 못건드리게 / 유저는 쉘 통해서 커널이랑, 어플리케이션은 쉘+커널/커널이랑 하드웨어 얘기 쉘:사용자가 입력한 문장을 읽어 요청을 실행하는 명령어 해석기 / 링크파일 : 하드 : "동일한 파일 시스템 내"에서 원본과 완전히 동일한 inode로 생성 소프트(l):원본파일 경로를 가르킴
새로운파일 쓰는법 : 비어있는 inode체크 > 그 inode entry 찾으면 비어있는 디스크에 저장 > 저장한 block number들을 inode entry의 disk allocation entry에 저장 > 이 파일의 정보들을 inode에 저장 > inode entry, link이름이 디렉토리에 저장
kill(pid_t pid(targetid), int sig(보낼sig) : signal을 보내는 프로그램이랑 받는 프로그램이랑 userid 같거나 superuser/같은 PC내에서만 가능, 다른 네트워크 불가 <sys/types.h><signal.h>
처음 signal(sigdemo)의 문제점 : 왜 이 시그널을 보내는지 알 수 없고 block불가 > 개선 (sigaction) : SA_SIGINFO bit 키면 POSIX가 지정한 sigaction >>> struct sigaction{ sighandler, sigaction, sa_flag, sa_mask }로 구성
sigaction(SIGINT, sigaction act(새로운 핸들러), sigaction prev_act) prev는 NULL로 생각
sig_flags : SA_RESETHAND : 핸들링 함수 들어가면 처리 방식 SIG_DFL로 재설정하고 SA_SIGINFO플래그 지움
SA_NODEFER: BLOCK 하지말고 계속 작동해 핸들러 하다가 sig들어오면 recusive핸들러/ SA_RESTART:핸들링하고 있을 때 들어온 시그널 저장해놨다가 끝나면 다시 실행 / SA_SIGINFO
sa_mask: 어떤 시그널 블락할지, 언블락할지 설정
시그널 핸들러 안에서 블락하고싶으면 sa_mask / 프로그램 돌리고 있을 때 시그널 블락하고 싶으면 sigprocmask(how,sigset_t sig, prev)
how: SIG_BLOCK : 기존에다가 sig추가, SIG_UNBLOCK : 기존에서 제거, SIG_SETMASK : 기존꺼는 다 삭제 후 sig추가
sigemptyset(&sigs); sigaddset(&sigs, SIGINT); sigprocmask(SIG_BLOCK, &sigs, &prevsigs);
fcntl.h, unistd.h, sys/types.h // unix vs linux : 1. 유료 vs 무료 2. 독점운영체제 vs 오픈소스 3. 일반적으로 몇가지 버전만
s=fcntl(fd, cmd) cmd:수행해야할 작업(F_GETFL, F_SETFL) /있고 vs 다양한 버전 공통점 : 다중 사용자 가능/서버특화 운영체제
s|=O_SYNC; int result = fcntl(fd, F_SETFL, s) 1: getflag 2: modify flag, 3: setflag
open(WTMP_FILE, O_WRONLY | O_APPEND | O_SYNC);
RACE CONDITION :공유된 어떤 파일에 서로 다른 유저 여러명이 동시 다발적으로 작업을 하면서 outcome이 바뀌는 현상
O_APPEND : write 콜하면 자동으로 파일끝으로 lseek / O_EXCL: 두 프로세스가 동일 파일을 생성하는거 방지
O_SYNC : 커널버퍼 말고 실제 디스크에 기록 / O_TRUNC: 파일이 존재하면 파일 길이 0으로 만듦
디스크파일 vs디바이스 파일. // signal(SIGINT, f) << SIG_IGN, SIG_DFL
차이점 : 디스크파일에 대한 inode는 실제 데이터 영역에 있는 BLOCK정보를 가진다, 버퍼링을 가진다.
디바이스 파일 inode는 커널의 device driver 의 pointer를 가진다.
total time : 전체 시간 usermode(virtual) time : 유저코드가 돌아간 시간 profile mode : 커널시간 + 유저시간
<termios.h><unistd.h>
struct termios { tcflag_t c_iflag(input), c_oflag(output), c_cflag(control), c_lflag(local)}
stty : tcgetattr(fd, struct termios *info) tcsetattr(fd, when, *info)
when : TCSANOW : 즉시 출력 TCSADRAIN : 대기중인 모든 출력이 터미널로 전송될때까지 기다리고 업데이트
TCSAFLUSH : 모든출력이 전송될때까지 기다리고 대기중인 입력데이터 삭제 그 다음 변경
<sys/ioctl.h> ioctl(fd, operation, args) : -> ioctl(0, TIOCGWINSZ, (winfsize) &wbuf) 터미널 설정정보 (가로 세로정도)
wbuf_ws_row, ws_col, ws_xpixel, ws_ypixel
ls | sort : ls 출력을 sort의 input으로,, who>file1 : who의 출력을 file1에 저장
who | tr '[a-z]' '[A-Z]' : who의 아웃풋의 a-z를 A-Z로 변경, /. standout(), standend() : 반짝반짝 키기 / 끄기
&= ~ICANON : icanon 끄기 : no buffering / addstr(" "), move(row, col), initscr() 커서 켜기, endwin() 커서 끄기
<time handling> sleep(n) usleep(u)(ms) /. refresh() : update screen
<unistd.h>second=alarm(N); 0이면 알람 끄기, pause() SIGALRM / which ITMER_REAL, VIRTUAL, PROF, val : 현재 세팅
<sys/time.h> getittimer(which, struct itimerval *val) setittimer(which, itimerval new, old) old NULL