## O_APPEND

- open이 성공하면 파일의 마지막 바이트 바로 뒤에 위치
- 그 이후의 write는 전부 파일의 끝에 자료를 추가하게 됨
- 파일의 끝에 자료를 추가하는 방법

- lseek 사용

```
lseek(fd, 0, SEEK_END)
write(fd, buf, BUFSIZE)
```

-O_APPEND

```
open("filename", O_WRONLY|O_APPEND)
write(fd, buf, BUFSIZE)
```

ls -al > unix.txt
- 기존의 unix.txt 가 있다면, Overwrite 됨.
ls -al >> unix.txt
- 기존의 unix.txt 가 있다면, Append 됨.

# 표준 입력, 표준 출력 및 표준 오류

## 표준 입력(0), 표준 출력(1), 표준 오류(2)

시스템 상에서 사용하는 파일 기술자
사용자가 파일기술자가 이용할 때,
3번 파일기술자부터 사용.

- redirection < >
  - prog_name < infile
    - 파일기술자 0로부터 읽어 들일 때, infile로 부터 자료를 읽음

      입력 단말기로부터 입력을 받는 것이 아닌 infile 로부터 입력을 받음.
  - prog_name > outputfile
    - 출력을 outputfile로 변경

    리다이렉션을 통해 현재 폴더에 있는 파일
    정보들을 파일로 손쉽게 만드는 방법 :
    ex ) ls -al > unix.txt
  - prog_name < infile > outputfile 결합해서 사용 가능
- pipe
  - prog1 | prog2

    prog2 를 실행했을 때, 표준 입력을 통해 입력을 받는 것이 아닌
    prog1 의 출력 내용을 입력으로 받음.

test3.c

## 예제

```
01  #include <fcntl.h>
02  #include <unistd.h>
03  #include <stdlib.h>
04  #include <stdio.h>
05  #define SIZE 512

06  int main(void) {
07      ssize_t nread;
08      char buf[SIZE]
09                                  표준 입력으로 입력을 받을 때는 open 이 필요가 없음.
10      while ((nread = read(0, buf, SIZE)) > 0
11              write(1, buf, nread);
12                          이 또한, 출력에 대한 파일을 open 할 필요가 없음.
13      return 0;
14  }
```

cat 명령어 :
cat 파일이름
파일 내용을 확인할 수 있음.

# 실습

argc > 1 인지 확인

- [문제1]위 프로그램을 수정하여 명령줄 인수가 있는지 조사하고, 만일 인수가 존재하면, 각 인수를 하나의 파일 이름으로 취급하고 각 파일의 내용을 표준 출력으로 복사하는 프로그램을 작성하시오. 만일 명령 줄 인수가 존재하지 않으면, 입력을 표준 입력으로부터 받아야 한다.

argc == 1 이면,

memset(buf, 0, SIZE); : 버퍼의 내용을 클리어시켜주는 명령어

- [문제2][문제1]의 표준출력부분을 outfile에 저장되도록 실행명령을 수정하시오.

./test3 > outfile

# 파일 기술자 제어

## 파일 기술자 제어 : fcntl(2)

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
int fcntl(int fildes, int cmd, /* arg */ ...);
```

GETFL 인 경우에는 인자가 두 개 들어감.

SETFL 인 경우에는 인자가 세 개 들어감.

파일의 속성을 바꿔주는 명령어

- 파일 기술자가 가리키는 파일에 cmd로 지정한 명령을 수행
- cmd의 종류에 따라 인자(arg)를 지정할 수 있음
- 자주 사용하는 cmd

cmd 로 지정한 명령어를 수행.

| | |
|---|---|
| F_GETFL | 상태 플래그 정보를 읽어온다. |
| F_SETFL | 상태 플래그 정보를 설정한다. 설정할 수 있는 플래그는 대부분 open 함수에서 지정하는 플래그다. |

# [예제 2-9] fcntl 함수로 파일 기술자 제어하기

```
07 int main(void) {
08       int fd, flags;
09
10       fd = open("unix.txt", O_RDWR);
11       if (fd == -1) {
12            perror("open");
13            exit(1);
14       }
15
16       if ((flags = fcntl(fd, F_GETFL)) == -1) {
17            perror("fcntl");
18            exit(1);
19       }
20
21       flags |= O_APPEND;
22
23       if (fcntl(fd, F_SETFL, flags) == -1) {
24            perror("fcntl");
25            exit(1);
26       }
27
28       if (write(fd, "Hanbit Media", 12) != 12) perror("write");
29       close(fd);
30
31       return 0;
32  }
```

O_RDONLY 로 파일을 열고,
flags |= O_APPEND;
로 플래그를 수정하게 되면 에러가 발생.

현재 fd 의 플래그 정보를 받음.

파일을 추가 모드로 수정

현재 플래그의 추가 모드를 더 함.

플래그를 수정

파일에 내용 추가

```
# cat unix.txt
Unix System Programming
# ex2_9.out
# cat unix.txt
Unix System Programming
Hanbit Media
```

# 실습

- 위 프로그램을 수정하여 현재의 파일 플래그를 테스트하여 출력하고, O_APPEND를 추가하는 프로그램을 작성하라

```
arg1 = fcntl(fd, F_GETFL);
switch (현재 파일플래그 & O_ACCMODE) {
    case O_RDWR :
    …
}
```

O_ACCMODE 는 access flag 들의 모든 합임.
O_ACCMODE 를 앤드연산하면,
파일이 가지는 access flag 들만 남음.
access flag는 O_RDONLY, O_WRONLY, O_RDWR 을 말함.

```
arg1 = fcntl(fd, F_GETFL);
switch (arg1 & O_ACCMODE) {
        case O_WRONLY:
            printf("write-only\n");
            break;
        case O_RDWR:
                    …
```

만일 현재 플래그가 O_RDONLY 라면,
O_APPEND 를 추가하고자 하면 fcntl
함수에서 에러가 남.
플래그 값을 모순되게 주면 안됨.

# 파일 삭제

## unlink(2)

unlink 명령어 :
unlink 파일이름

```
#include <unistd.h>
int unlink(const char *path);
```

inode 에는 파일에 있는 링크의 개수가 존재

- path에 지정한 파일의 inode에서 링크 수를 감소시킨다.
- 링크 수가 0이 되면 path에 지정한 파일이 삭제된다.
- 파일 뿐만 아니라 디렉토리(빈 디렉토리 아니어도 됨)도 삭제된다.

파일에 대한 모든 속성을 가지고 있는 inode 라고 하는 데이터는 디스크에 저장되어 있음. inode 와 같은 데이터를 '메타데이터' 라고 함.

링크 연결 명령어 : ln 파일이름
ls -al 파일이름
라는 명령을 입력하여 보게되면
숫자가 보이는데 이 숫자가 현재
이 데이터에 링크하고 있는 파일의 수를 의미

어떤 한 파일에 대해 여러 사람이 공유하고 있다고 할 때, link 를 하고 있다고 함. 링크 수는 공유하고 있는 사람의 수

diff 명령어 :
diff 파일이름1 파일이름2
두 파일이 같은 내용인지 확인해줌.

## remove(3)

```
#include <stdio.h>
int remove(const char *path);
```

- path에 지정한 파일이나 디렉토리를 삭제한다.
- 디렉토리인 경우 빈 디렉토리만 삭제한다.

sync 와 fsync 의 차이점 :
fsync 는 메인 메모리의 내용 중
fildes 의 파일 내용만 디스크에
저정하라는 의미이고,
sync 는 메인 메모리에 있는 모든
내용을 디스크에 업데이트 시킴.

중요한 데이터를 디스크에 저장하기 위해서

fsync – 메모리에 위치하고 있는 파일의 내용을 디스크로 보내 메모리와 디스크의 내용을 동기화한다. 메모리의 내용이 디스크로 모두 기록되기 전에는 리턴하지 않는다

int fsync(int filedes);

fsync 는 write-through 기법

write-through 기법 : 메인메모리에 저장하면서 디스크에도 저장하는 기법
write-back(behind) 기법 : 메인 메모리에 저장하다가 일정 주기마다 디스크에 저장하는 방법.