저 수준(low-level) 파일 입출력과 고 수준(high-level) 파일 입출력

저 수준 파일 입출력

- 파일 지시자는 파일 기술자 (file descriptor)
- 유닉스 커널의 시스템 호출을 시용하여 파일 입출력 실행
- 파일에 빠르게 접근 가능하며, 바이트 단위로 파일의 내용을 다루므로 일반 파일 뿐만 아니라 특수 파일도 읽고 쓸 수 있음
- 바이트 단위를 적당한 형태의 데이터로 변환하는 함수 등이 필요
- open, close, read, write, lseek, dup, dup2, fcntl, fsync

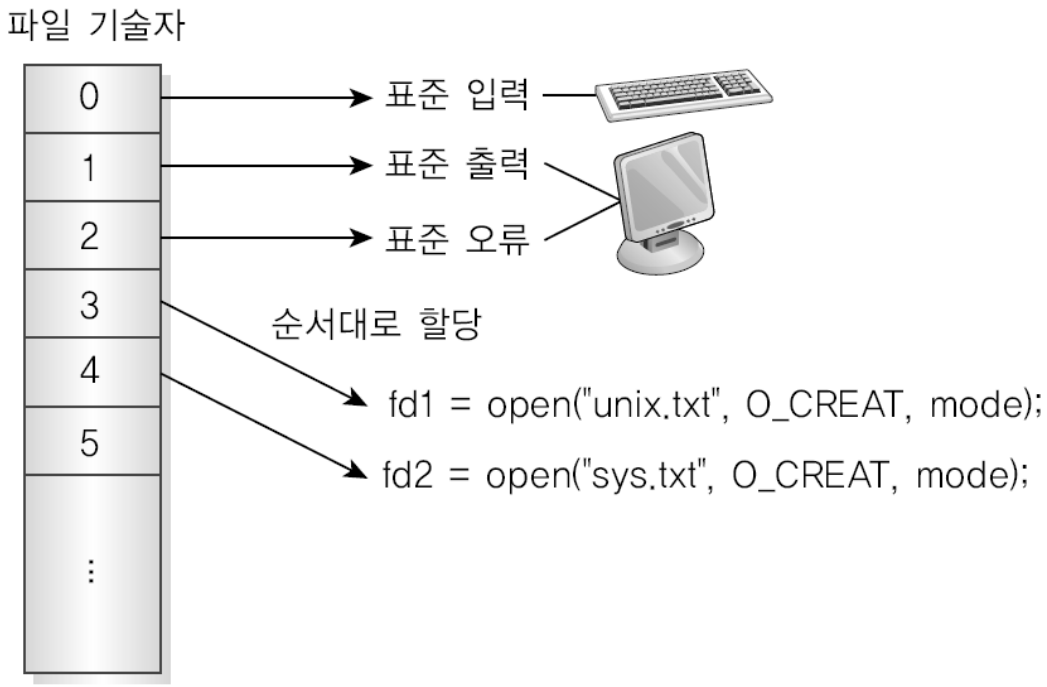
고 수준 파일 입출력

- 파일 지시자는 파일 포인터 (file pointer)
- C 언어의 표준 함수로 제공되며, 여러가지 형태의 데이터 형식을 지원함
- 버퍼 단위로 읽고 쓰기가 가능함
- fopen, fclose, fread, fwrite, fputs, fgets, fprintf, fscanf, fseek

파일 기술자(file descriptor)

파일 기술자

- 현재 열려있는 파일을 구분하는 정수값
- 저수준 파일 입출력에서 열린 파일을 참조하는데 사용
- 0번 : 표준 입력, 1번 : 표준 출력, 2번 : 표준 오류



[그림 2-1] 파일 기술자 할당

파일 생성과 열고 닫기[1]

파일 열기: open(2)

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *path, int oflag [, mode_t mode]);
```

- **path에 지정한 파일을 oflag에 지정한 플래그 값에 따라 열고 파일기술자를 리턴**
- **oflag 값**

종류	기능
O_RDONLY	파일을 읽기 전용으로 연다.
O_WRONLY	파일을 쓰기 전용으로 연다.
O_RDWR|O_CREAT	파일을 읽기와 쓰기가 가능하게 연다.
O_CREAT	파일이 없으면 파일을 생성한다
O_EXCL|O_CREAT	O_CREAT 옵션과 함께 사용할 경우 기존에 없는 파일이면 파일을 생성하지만, 파일이 이미 있으면 파일을 생성하지 않고 오류 메시지를 출력한다
O_APPEND	파일의 맨 끝에 내용을 추가한다.
O_TRUNC|O_CREAT	파일을 생성할 때 이미 있는 파일이고 쓰기 옵션으로 열었으면 내용을 모두 지우고 파일의 길이를 0으로 변경한다.
O_NONBLOCK/O_NDELAY	비블로킹(Non-blocking) 입출력
O_SYNC/O_DSYNC	저장장치에 쓰기가 끝나야 쓰기 동작을 완료

파일 생성과 열고 닫기[2]

파일 열기: open(2)

- mode : 파일 접근권한 지정, 0644같이 숫자나 플래그 값으로 지정 가능

플래그	모드	설명
S_IRWXU	0700	소유자 읽기/쓰기/실행 권한
S_IRUSR	0400	소유자 읽기 권한
S_IWUSR	0200	소유자 쓰기 권한
S_IXUSR	0100	소유자 실행 권한
S_IRWXG	0070	그룹 읽기/쓰기/실행 권한
S_IRGRP	0040	그룹 읽기 권한
S_IWGRP	0020	그룹 쓰기 권한
S_IXGRP	0010	그룹 실행 권한
S_IRWXO	0007	기타 사용자 읽기/쓰기/실행 권한
S_IROTH	0004	기타 사용자 읽기 권한
S_IWOTH	0002	기타 사용자 쓰기 권한
S_IXOTH	0001	기타 사용자 실행 권한

```
mode=S_IRUSR | S_IWUSR;
```

파일 생성과 열고 닫기[3]

파일 생성 : creat(2)

```
#include <sys/stat.h>
#include <fcntl.h>
int creat(const char *path, mode_t mode);
```

- 파일 생성 함수, open 함수에 파일 생성 기능이 없던 구버전 유닉스에서 사용
- open 함수와 달리 옵션을 지정하는 부분이 없다.
- creat 함수로 파일을 생성하면 파일 기술자를 리턴하므로 별도로 open할 필요 없음
- open with open(path, O_CREAT|O_WRONLY|O_TRUNC, mode)와 동일

파일 닫기: close(2)

```
#include <unistd.h>
int close(int fildes);
```

- 프로세스에서 열 수 있는 파일 개수가 제한되어 있으므로 파일의 사용이 끝나면 닫아야 한다.

[예제 2-1] 새 파일 열고 닫기

ex2_1.c

```
01  #include <sys/types.h>
02  #include <sys/stat.h>
03  #include <fcntl.h>
04  #include <unistd.h>
05  #include <stdlib.h>
06  #include <stdio.h>
07
08  int main(void) {
09      int fd;
10      mode_t mode;
11
12      mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
13
14      fd = open("unix.txt", O_CREAT, mode);
15      if (fd == -1) {
16          perror("Creat");
17          exit(1);
18      }
19      close(fd);
20
21      return 0;
22  }
```

```
# ls unix.txt
unix.txt: 해당 파일이나 디렉토리가 없음
# gcc -o ex2_1.out ex2_1.c
# ex2_1.out# ls -l unix.txt
-rw-r--r--   1 root   other   0   1월   6일   13:10 unix.txt
```

접근권한:644

[예제 2-2] O_EXCL 플래그 사용하기

ex2_2.c

```
01  #include <sys/types.h>
02  #include <sys/stat.h>
03  #include <fcntl.h>
04  #include <unistd.h>
05  #include <stdlib.h>
06  #include <stdio.h>
07
08  int main(void) {
09      int fd;
10
11      fd = open("unix.txt", O_CREAT | O_EXCL, 0644);
12      if (fd == -1) {
13          perror("Excl");
14          exit(1);
15      }
16      close(fd);
17
18      return 0;
19  }
```

```
# ls unix.txt
unix.txt
# ex2_2.out
Excl: File exists
```

```
# rm unix.txt
# ex2_2.out
# ls unix.txt
unix.txt
```

[예제 2-3] 파일 기술자 할당

ex2_3.c

```
01  #include <sys/types.h>
02  #include <sys/stat.h>
03  #include <fcntl.h>
04  #include <unistd.h>
05  #include <stdlib.h>
06  #include <stdio.h>
07
08  int main(void) {
09      int fd;
10
11      close(0);
12
13      fd = open("unix.txt", O_RDWR|O_CREAT);
14      if (fd == -1) {
15          perror("Excl");
16          exit(1);
17      }
18
19      printf("unix.txt : fd = %d\n", fd);
20      close(fd);
21
22      return 0;
23  }
```

```
# ex2_3.out
unix.txt : fd = 0
```

11행에서 0번을 닫았으므로
새로 생성한 파일은 가장 작은 번호인 0번이 할당된다.

파일 읽기와 쓰기

파일 읽기 : read(2)

```
#include <unistd.h>
ssize_t read(int fildes, void *buf, size_t nbytes);
```

- 파일에서 nbytes로 지정한 크기만큼 바이트를 읽어서 buf에 저장
- 실제로 읽어온 바이트 개수를 리턴
- 리턴값이 0이면 파일의 끝에 도달했음을 의미
- 파일의 종류에 상관없이 무조건 바이트 단위로 읽어온다.

파일 쓰기 : write(2)

```
#include <unistd.h>
ssize_t write(int fildes, const void *buf, size_t nbytes);
```

- buf가 가리키는 메모리에서 nbytes로 지정한 크기만큼 파일에 기록
- 실제로 쓰기를 수행한 바이트 수를 리턴

[예제 2-4] 파일 읽기

ex2_4.c

```
01 #include <fcntl.h>
02 #include <unistd.h>
03 #include <stdlib.h>
04 #include <stdio.h>
05
06 int main(void) {
07 int rfd, wfd, n;
08 char buf[10];
09
10 rfd = open("unix.txt", O_RDONLY);
11 if(rfd == -1) {
12 perror("Open unix.txt");
13 exit(1);
14 }
15
16 wfd = open("unix.bak", O_CREAT | O_WRONLY | O_TRUNC, 0644);
17 if (wfd == -1) {
18 perror("Open unix.bak");
19 exit(1);
20 }
21
```

파일기술자 2개 선언

[예제 2-4] 파일 읽기

```
22 while ((n = read(rfd, buf, 6)) > 0)
23 if (write(wfd, buf, n) != n) perror("Write");
24
25 if (n == -1) perror("Read");
26
27 close(rfd);
28 close(wfd);
29
30 return 0;
31 }
```

6바이트씩 읽어온다

```
# ls unix.bak
unix.bak: 해당 파일이나 디렉토리가 없음
# ex2_5.out
# cat unix.bak
Unix System Programming
```

[예제 2-5] 파일 읽고 쓰기

ex2_5.c

```
...
06  int main(void) {
07      int rfd, wfd, n;
08      char buf[10];
09
10      rfd = open("unix.txt", O_RDONLY);
11      if(rfd == -1) {
        perror("Open unix.txt");
13          exit(1);
14      }
15
16      wfd = open("unix.bak", O_CREAT | O_WRONLY | O_TRUNC, 0644);
17      if (wfd == -1) {
18          perror("Open unix.bak");
19          exit(1);
20      }
21
22      while ((n = read(rfd, buf, 6)) > 0)
23          if (write(wfd, buf, n) != n) perror("Write");
24
```

파일기술자 2개 선언

6바이트씩 읽어온다

[예제 2-5] 파일 읽고 쓰기

```
25      if (n == -1) perror("Read");
26
27      close(rfd);
28      close(wfd);
29
30      return 0;
31  }
```

```
# ls unix.bak
unix.bak: 해당 파일이나 디렉토리가 없음
# ex2_5.out
# cat unix.bak
Unix System Programming
```