# 2주차 2차시 고/저수준 파일 입출력

## 【학습목표】

1. 저수준 파일 입출력을 설명할 수 있다.
2. 고수준 파일 입출력을 설명할 수 있다.

## 학습내용1 : 저수준 파일 입출력 개요

### 1. 특징

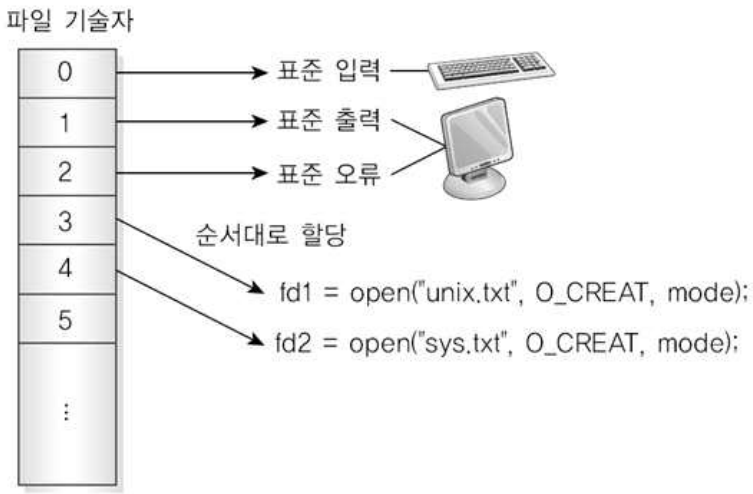
- 파일 기술자(file descritor) 사용
- 바이트 단위로 파일을 다룸
- 시스템 호출을 이용
- 특수, 일반 파일을 읽을 수 있다.

### 2. 주요 함수

- open, close, read, write, dup, dup2, fcntl, lseek, fsync

### 3. 파일 기술자

- 현재 열려 있는 파일을 구분하는 정수값
- open 함수 사용하여 파일 열 때 순차적으로 부여
- 0부터 시작
    - 0 : 표준 입력
    - 1 : 표준 출력
    - 2 : 표준 오류



### 4. 파일 생성과 열고 닫기

과정 : 파일 열기 -> 파일 작업(읽기, 쓰기 등) -> 파일 닫기
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]);
```

path : 열리는 파일이 있는 경로
mode : 접근 권한
oflag : 파일 상태 플래그

주요 oflag

| 종류 | 기능 |
|---|---|
| O_RDONLY | 파일을 읽기 전용으로 연다. |
| O_WRONLY | 파일을 쓰기 전용으로 연다. |
| O_RDWR | 파일을 읽기와 쓰기가 가능하게 연다. |
| O_CREAT | 파일이 없으면 파일을 생성한다 |
| O_EXCL | 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 | 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 |

여러개 플래그 사용 : | 사용

| | |
|---|---|
| 쓰기 전용, 이미 파일 존재시 | O_WRONLY \| OTRUNC |
| 쓰기 전용, 파일 없는 경우 | O_WRONLY \| O_CREAT \| O_TRUNC |
| 읽기/쓰기/추가용 파일 | O_RDWR \| O_APPEND |

mode : O_CREAT 플래그 사용하여 파일을 생성할때만 사용, <sys/stat.h> 파일에 정의된 플래그 사용 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 |

* 파일 닫기 : close(2)

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

fildes : 파일 기술자
성공 시 0 리턴,
실패 시 -1 리턴, 오류 코드를 외부 변수 errno에 저장

```
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
```

* 파일 읽기/쓰기

읽기 : read()

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

fildes : 파일 기술자.

buf : 바이트를 읽어서 저장할 메모리 영역의 시작 주소

nbytes : 읽어올 바이트 수

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.

```
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
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 }
```

파일기술자 2개 선언

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

* 쓰기 : write()

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

fildes : 파일 기술자
buf : 파일에 기록할 데이터를 저장한 메모리 영역
nbytes : buf 크기(기록할 데이터의 크기)
buf가 가리키는 메모리에서 nbytes로 지정한 크기만큼 파일에 기록
실제로 쓰기를 수행한 바이트 수를 리턴

```
...
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
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
```

## 5. 파일 오프셋

* 오프셋
파일의 읽을 위치/쓰기 위치를 알려줌
하나의 파일에 한 개만 존재

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fildes, off_t offset, int whence);
```

fildes : 파일 기술자
off_t offset : 이동할 오프셋 위치(off_t 데이터형은 long,
whence : 오프셋 기준 위치

* lsweek
파일 기술자가 가리키는 파일에서 offset으로 지정한 크기만큼 오프셋을 이동시킨다.
성공시 새로운 offset 값 리턴
실패 시 -1 리턴
offset 값은 whence값을 기준으로 해석
whence 값

| 값 | 설명 |
|---|---|
| SEEK_SET | 파일의 시작 기준 |
| SEEK_CUR | 현재 위치 기준 |
| SEEK_END | 파일의 끝 기준 |

lseek(fd, 5, SEEK_SET); : 파일의 시작에서 5번째 위치로 이동
lseek(fd, 0, SEEK_END); : 파일의 끝에서 0번째. 즉 끝으로 이동
파일 오프셋 현재 위치
cur_offset = lseek(fd, 0, SEEK_CUR);

## 6. 파일 기술자 복사하기 : dup(), dup2()

* dup()
기존 파일 기술자를 인자로 받아 새로운 기술자 리턴
새로운 기술자는 자동할당.

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

fildes : 파일 기술자

* dup2()
새로운 파일 기술자 지정 가능

```
#include <unistd.h>
int dup2(int fildes, int fildes2);
```

```
01  #include <fcntl.h>
02  #include <unistd.h>
03  #include <stdlib.h>
04  #include <stdio.h>
05
06  int main(void) {
07      int fd;
08
09      fd = open("tmp.bbb", O_CREAT | O_WRONLY | O_TRUNC, 0644);
10      if (fd == -1) {
11          perror("Create tmp.bbb");
12          exit(1);
13      }
14
15      dup2(fd, 1);
16
17      printf("DUP2 : Standard Output Redirection\n");
18      close(fd);
19
20      return 0;
21  }
```

표준출력(1)로 지정하여 복사

표준출력을 출력한 내용이 파일로 저장된다.

```
# ex2_8.out
# cat tmp.bbb
DUP2 : Standard Output Redirection
```

### 7. 파일 기술자 제어 : fcntl(2)

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

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

fildes : 파일 기술자
cmd : 명령
arg : 인자 들

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

### 8. 파일 삭제 : unlink(2), remove(3)

* unlink(2)
path에 지정한 파일의 inode에서 링크 수를 감소시킨다.
링크 수가 0이 되면 path에 지정한 파일이 삭제된다.
파일 뿐만 아니라 디렉터리(빈 디렉터리 아니어도 됨)도 삭제된다

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

path : 삭제할 파일의 경로

* remoe(3)
path에 지정한 파일이나 디렉터리를 삭제한다.
디렉터리인 경우 빈 디렉터리만 삭제한다.

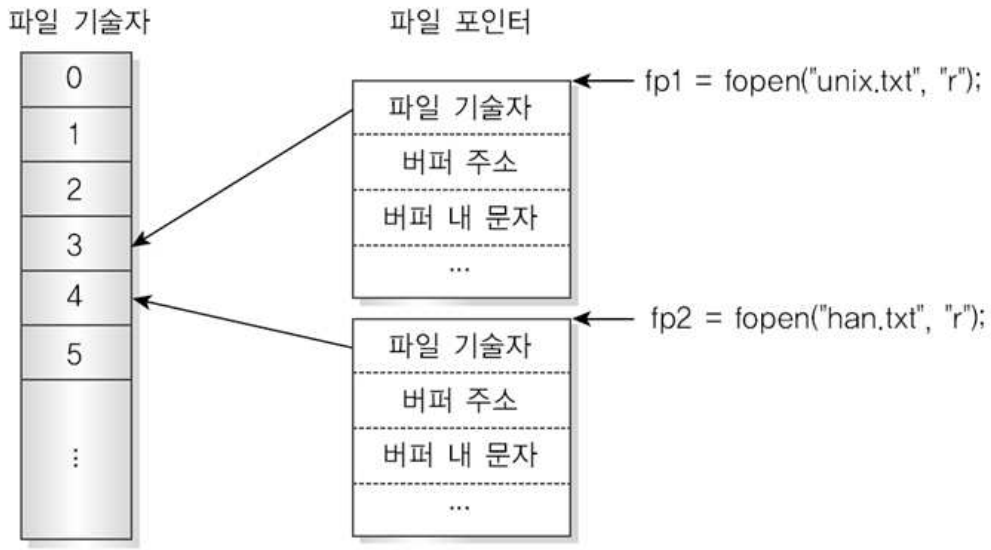
```
#include <stdio.h>
int remove(const char *path);
```

path : 경로

## 학습내용2 : 고수준 파일 입출력 개요

### 1. 특징

파일 포인터(file pointer) 사용
표준 함수 활용
표준 입출력 라이브러리(Standard Input/Output Library)라 함
입출력 : 문자단위, 행 단위, 버퍼 단위, 형식 기반 입출력 등 가능




### 2. 파일 포인터

파일의 위치에 관한 정보 기억
FILE * 형
FILE 구조체 가르키는 포인터

### 3. 파일 열기와 닫기

* 파일 열기 : fopen(3)
filename으로 지정한 파일을 mode로 지정한 모드에 따라 열고 파일 포인터를 리턴
성공시 : 파일의 주소를 파일 포인터로 리턴
실패시 : 0을 리턴

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
```

filename : 파일의 경로
mode : 파일 열기 모드

- mode 모드

| 모드 | 의미 |
|---|---|
| r | 읽기 전용으로 텍스트 파일을 연다. |
| w | 새로 쓰기용으로 텍스트 파일을 연다. 기존 내용은 삭제된다. |
| a | 추가용으로 텍스트 파일을 연다. |
| rb | 읽기 전용으로 바이너리 파일을 연다. |
| wb | 새로 쓰기용으로 바이너리 파일을 연다. 기존 내용은 삭제된다. |
| ab | 추가용으로 바이너리 파일을 연다. |
| r+ | 읽기와 쓰기용으로 텍스트 파일을 연다. |
| w+ | 쓰기와 읽기용으로 텍스트 파일을 연다. |
| a+ | 추가와 읽기용으로 텍스트 파일을 연다. |
| rb+ | 읽기와 쓰기용으로 바이너리 파일을 연다. |
| wb+ | 쓰기와 읽기용으로 바이너리 파일을 연다. |
| ab+ | 추가와 읽기용으로 바이너리 파일을 연다. |

- 예

```
FILE *fp;
fp = fopen("unix.txt", "r");
```

### 4. 파일 닫기 : fclose(3)

* fclose()
fopen으로 오픈한 파일을 닫는다.
성공시 : 0 리턴
실패시 : EOF(-1)리턴

```
#include <stdio.h>
int fclose(FILE *stream);
```

stream : fopen에서 리턴한 파일 포인터

| 방법 | 함수 | 기능 | |
|---|---|---|---|
| 문자 기반 입력 | fgetc(3) | 문자 한 개를 unsigned char 로 읽어옴 | |
| | getc(3) | 문자 한 개 읽어옴. 매크로 | |
| | getchar(3) | 표준 입력에서 문자 한 개, 매크로 | |
| | getw(3) | 워드 단위로 읽어옴 | |
| 문자 기반 출력 | fputc(3) | 문자 한 개를 unsigned char 출력 | |
| | putc(3) | 문자 한 개 출력. 매크로 | |
| | putchar(3) | 표준 출력에서 문자 한 개 출력, 매크로 | |
| | putw(3) | 워드 단위로 출력 | |
| 문자열 기반 입력 함수 | gets(3) | 표준 입력에서 문자열을 읽어들인다. | 성공 : s의 시작주소 리턴<br>실패 : NULL |
| | fgets(3) | 파일(stream)에서 n보다 하나 적게 문자열을 읽어 s에 저장 | |
| 문자열 기반 출력 함수 | puts(3) | s가 가리키는 문자열을 표준 출력으로 출력(개행문자 포함) | |
| | fputs(3) | s가 가리키는 문자열을 파일 포인터가 가리키는 파일로 출력 | |
| 버퍼 기반 입력 함수 | fread(3) | 항목의 크기가 size인 데이터를 nitems에 지정한 개수만큼 읽어 ptr에 저장<br>성공하면 읽어온 항목 수를 리턴<br>읽을 항목이 없으면 0을 리턴 | |
| 버퍼 기반 출력 함수 | fwrite(3) | 항목의 크기가 size인 데이터를 nitems에서 지정한 개수만큼 ptr에서 읽어서 stream으로 지정한 파일에 출력<br>성공하면 출력한 항목의 수를 리턴<br>오류가 발생하면 EOF를 리턴 | |
| 형식 기반 입력 함수 | scanf(3),<br>fscanf(3) | fscanf도 scanf가 사용하는 형식 지정 방법을 그대로 사용<br>성공하면 읽어온 항목의 개수를 리턴한다. | |
| 형식 기반 출력 함수 | printf(3),<br>fprintf(3) | fprintf는 지정한 파일로 형식 지정 방법을 사용하여 출력. | |

<table>
<tr><th>방법</th><th>형식</th><th>인자</th></tr>
<tr><td>문자 기반 입력</td><td rowspan="4">

```
#include <stdio.h>
int fgetc(FILE *stream);
int getc(FILE *stream);
int getchar(void);
int getw(FILE *stream);
```

</td><td>stream : 파일 포인터</td></tr>
<tr><td>문자 기반 출력</td><td>c / w : 출력할 문자<br>stream : 파일 포인터</td></tr>
<tr><td>문자열 기반 입력 함수</td><td>s : 문자열을 저장할 버퍼의 시작주소<br>n : 버퍼의 크기<br>stream : 파일 포인터</td></tr>
<tr><td>문자열 기반 출력 함수</td><td>s : 문자열 주소<br>strem : 파일 포인터</td></tr>
<tr><td>버퍼 기반 입력 함수</td><td colspan="2">

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream);
```

크기가 size인 데이터를 nitems에 지정한 개수만큼 읽어 ptr에 저장<br>성공하면 읽어온 항목 수를 리턴 / 읽을 항목이 없으면 0을 리턴</td></tr>
<tr><td>버퍼 기반 출력 함수</td><td colspan="2">

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size, size_t nitems, FILE *stream);
```

항목의 크기가 size인 데이터를 nitems에서 지정한 개수만큼 ptr에서 읽어서<br>stream으로 지정한 파일에 출력<br>성공하면 출력한 항목의 수를 리턴<br>오류가 발생하면 EOF를 리턴</td></tr>
<tr><td>형식 기반 입력 함수</td><td colspan="2">

```
#include <stdio.h>
int scanf(const char *restrict format, ...);
int fscanf(FILE *restrict stream, const char *restrict format, ..);
```

fscanf도 scanf가 사용하는 형식 지정 방법을 그대로 사용한다.<br>성공하면 읽어온 항목의 개수를 리턴한다.</td></tr>
<tr><td>형식 기반 출력 함수</td><td colspan="2">

```
#include <stdio.h>
int printf(const char *restrict format, /* args */ ...);
int fprintf(FILE *restrict stream, const char *restrict format, /*args */ ..)/
```

fprintf는 지정한 파일로 형식 지정 방법을 사용하여 출력한다.</td></tr>
</table>

## 5. 파일 오프셋

* fssek(3)
stream이 가리키는 파일에서 offset에 지정한 크기만큼 오프셋을 이동
whence는 lseek와 같은 값을 사용
fseek는 성공하면 0을 실패하면 EOF를 리턴

```
#include <stdio.h>
int fseek(FILE *stream, long offset, int whence);
```

stream : 파일 포인터
offset : 이동할 오프셋
whence : 오프셋의 기준 위치

* ftell(3)
현재 오프셋을 리턴.
오프셋은 파일의 시작에서 현재 위치까지의 바이트 수

```
#include <stdio.h>
long ftell(FILE *stream);
```

stream : 파일 포인터

* fsetpos(3)
파일의 현재 오프셋을 pos가 가르키는 영역에 저장
성공시 0 리턴, 실패시 0이 아닌 값

```
#include <stdio.h>
void rewind(FILE *stream);
```

stream : 파일 포인터
pos : 오프셋을 저장하고 있는 영역 주소

* fgetpos(3)
pos가 가르키는 영역의 값으로 파일 오프셋 변경
성공시 0 리턴, 실패시 0이 아닌 값 리턴

```
#include <stdio.h>
int fsetpos(FILE *stream, const fpos_t *pos);
int fgetpos(FILE *stream, fpos_t *pos);
```

stream : 파일 포인터
pos : 오프셋을 저장할 영역 주소

**【학습정리】**

1. 저수준 파일 입출력
- 파일 지시자 : int fd;
- 파일 참조 : 파일 기술자 (file descriptor)
- 특징 : 커널의 시스템 호출 사용, 바이트 단위로 파일 다룸, 처리 속도가 빠름
- 주요함수 : open, close, read, write, dup, dup2

2. 고수준 파일 입출력
- 파일 지시자 : FILE *fp;
- 파일 참조 : 파일 포인터 (file pointer)
- 특징 : C 언어 표준 함수로 제공. 표준 입출력 라이브러리 (Standard input output library), 버퍼 단위로 읽기와 쓰기 수행, 다양한 입출력 데이터 변환 기능 활용 가능, 입출력 동기화가 쉽다.
- 주요함수 : fopen, fclose, fread, fwrite, fputs, fgets, fprintf, fscanf, fseek