Chapter 10. The UNIX System Interface

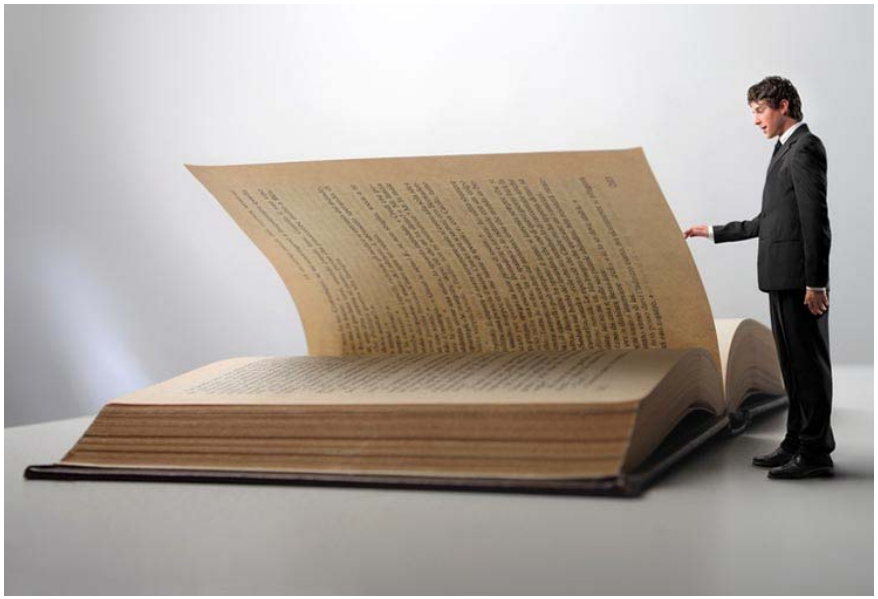
June, 2016

Seungjae Baek

Dept. of software
Dankook University
http://embedded.dankook.ac.kr/~baeksj

파일의 개념 (1/5)

- 파일의 정의
  - ✓ 사용자가 이용할 수 있는 데이터의 실체
  - ✓ 레코드들의 집합이라고 정의
- 파일의 필요성
  - ✓ 데이터의 효율적인 저장 및 검색을 위해 파일 단위 구분



파일의 개념 (2/5)

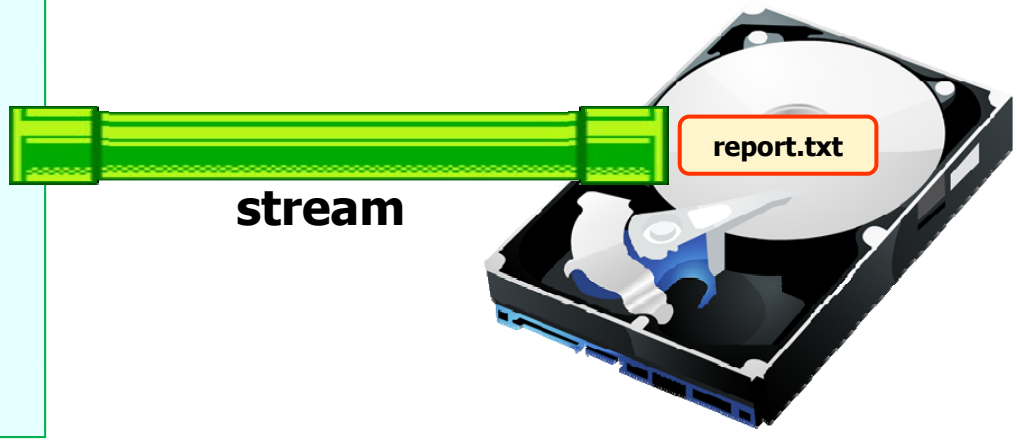
- 파일 스트림
  - ✓ 입 출력 장치와 무관하게 일정한 접속(interface) 방식 제공
  - ✓ 프로그래머와 하드웨어 사이에 추상적인 단계 제공

```
#include <stdio.h>

int main()
{
    int x;
    char buf[32];

    ...
    return 0;
}
```

<소스 코드>



<하드 디스크>

파일의 개념 (3/5)

- fopen() 함수

```
#include <stdio.h>

FILE * fopen(char *fname, char *mode)

Return value: File pointer or NULL
```

mode	Description
"r"	Opens a file for reading. The file must exist.
"w"	Creates an empty file for writing. If a file with the same name already exists, its content is erased and the file is considered as a new empty file.
"a"	Appends to a file. Writing operations, append data at the end of the file. The file is created if it does not exist.
"r+"	Opens a file to update both reading and writing. The file must exist.
"w+"	Creates an empty file for both reading and writing.
"a+"	Opens a file for reading and appending.

파일의 개념 (4/5)

- 파일 쓰기 예

```
//fopen.c
#include <stdio.h>

int main()
{
    FILE * fp;
    fp = fopen("test.txt", "w");
    if (fp == NULL)
        return 0;
    fprintf(fp, "%d %d %d\n", 10, 20, 30);
    fclose(fp);
    return 0;
}
```

```
root@localhost:/home/Lecture/C/file
[root@localhost C]# mkdir file
[root@localhost C]# cd file
[root@localhost file]# ls
[root@localhost file]#
[root@localhost file]#
[root@localhost file]# vim fopen.c
[root@localhost file]# gcc -o fopen fopen.c -Wall
[root@localhost file]# ls
fopen  fopen.c
[root@localhost file]# ./fopen
[root@localhost file]# ls
fopen  fopen.c  test.txt
[root@localhost file]# cat test.txt
10 20 30
[root@localhost file]#
```

파일의 개념 (5/5)

- 파일 읽기 예

```
#include <stdio.h>
int main()
{
    FILE * fp;
    int a, b, c;
    fp = fopen("test.txt", "r");
    if (fp == NULL)
        return 0;
    fscanf(fp, "%d %d %d", &a, &b, &c);
    printf("%d %d %d\n", a, b, c);
    fclose(fp);
    return 0;
}
```

```
root@localhost:/home/Lecture/C/file
[root@localhost file]# cat test.txt
10 20 30
[root@localhost file]# vim fscanf.c
[root@localhost file]# gcc -o fscanf fscanf.c -Wall
[root@localhost file]# ./fscanf
10 20 30
[root@localhost file]#
```

파일 예제 (1/2)

- 텍스트 파일 관련 함수

```
#include <stdio.h>

int fgetc(FILE * fp)          // number of bytes read EOF or error
int fputc(int ch, FILE * fp) // number of bytes write or EOF
```

- 텍스트 파일 쓰기

```
#include <stdio.h>
int main()
{
    FILE * fp;
    char str[80] = " It is a period of civil war. Rebel spaceships₩n";
    char * p;

    fp = fopen("star.txt", "w");
    p = str;

    while(*p) {
       if (fputc(*p, fp) == EOF)
           break;
       p++;
    }
    fclose(fp);
    return 0;
}
```

```
root@localhost:/home/Lecture/C/file
[root@localhost file]# vim fputc.c
[root@localhost file]# gcc -o fputc fputc.c -Wall
[root@localhost file]# ./fputc
[root@localhost file]#
[root@localhost file]# ls
fopen  fopen.c  fputc  fputc.c  fscanf  fscanf.c  star.txt  test.txt
[root@localhost file]# cat star.txt
 It is a period of civil war. Rebel spaceships
[root@localhost file]#
```

파일 예제 (2/2)

- 텍스트 파일 읽기

```c
#include <stdio.h>

int main()
{
    FILE * fp;
    int i;

    fp = fopen("star.txt", "r");

    for ( ; ; ) {
       i = fgetc(fp);
       if (i == EOF)
           break;
       putchar(i);
    }
    fclose(fp);
    return 0;
}
```

```
root@localhost:/home/Lecture/C/file
[root@localhost file]# ls
fopen  fopen.c  fputc  fputc.c  fscanf  fscanf.c  star.txt  test.txt
[root@localhost file]# vim fgetc.c
[root@localhost file]# gcc -o fgetc fgetc.c -Wall
[root@localhost file]#
[root@localhost file]# ./fgetc
 It is a period of civil war. Rebel spaceships
[root@localhost file]# cat star.txt
 It is a period of civil war. Rebel spaceships
[root@localhost file]#
```

Low Level I/O

- Open, read, write, close

Synopsis

int open(const char *pathname, int flags, [mode_t mode])

- ✓ **pathname : 절대 경로 or 상대 경로**
- ✓ **flags (참조 : /usr/include/asm/fcntl.h)**
  - **O_RDONLY, O_WRONLY, O_RDWR**
  - **O_CREAT, O_EXCL**
  - **O_TRUNC, O_APPEND**
  - **O_NONBLOCK, O_SYNC**
  - **…**
- ✓ **mode**
  - **O_CREAT 플래그가 있을 때 유의미**
  - **file access mode (S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, …, S_IROTH, …)**
- ✓ **return value**
  - **file descriptor if success**
  - **-1 if fail**

int read(int fd, char *buf, int size) // write도 동일

- ✓ **fd: file descriptor (return value of open())**
- ✓ **buf: memory space for keeping data**
- ✓ **size: request size**
- ✓ **return value**
  - **read size**
  - **-1 if fail**

Low Level I/O


```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int main(void)
{
        int fd, cnt;
        char buf[4096];
        if ( ( fd = open("star.txt", O_RDWR) ) <= 0 ){
                printf("file open error\n");
                return -1;
        }
        if ( ( cnt = read(fd, buf, 4096) ) <= 0 ){
                printf("file read error\n");
                return -2;
        }
        if ( ( cnt = write(STDOUT_FILENO, buf, cnt) ) <= 0 ){
                printf("file write error\n");
                return -3;
        }
        close(fd);
        return 0;
}
```

```
root@localhost:/home/Lecture/C/file
[root@localhost file]# ls
fgetc  fgetc.c  fopen  fopen.c  fputc  fputc.c  fscanf  fscanf.c  star.txt
[root@localhost file]# vim open.c
[root@localhost file]# gcc -o open open.c -Wall
[root@localhost file]# ./open
 It is a period of civil war. Rebel spaceships
[root@localhost file]#
[root@localhost file]# mv star.txt ../
[root@localhost file]# ./open
file open error
[root@localhost file]#
```

Error Handling

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
int main(void)
{
        int fd, cnt;
        char buf[4096];
        if ( ( fd = open("st...txt", O_RDWR) ) <= 0 ){
                perror("file open error");
                exit(errno);
        }
        if ( ( cnt = read(fd, buf, 4096) ) <= 0 ){
                perror("file read error");
                exit(errno);
        }
        if ( ( cnt = write(STDOUT_FILENO, buf, cnt) ) <= 0 ){
                perror("file write error");
                exit(errno);
        }
        close(fd);
        return 0;
}
```

NAME
perror - print a system error message
SYNOPSIS
#include <stdio.h>
void perror(const char *s);
DESCRIPTION
The routine perror() produces a message on the standard error output, describing the last error encountered during a call to a system or library function.

```
root@localhost:/home/Lecture/C/file
[root@localhost file]# vim error.c
[root@localhost file]# gcc -o error error.c -Wall
[root@localhost file]# ./error
file open error: No such file or directory
[root@localhost file]# echo $?
2
[root@localhost file]#
```

Variable-length Argument Lists

- ...
  - ✓ The number and types of the arguments may vary
  - ✓ Can only appear at the end of an argument list
  - ✓ Example: printf

```
#include <stdio.h>

int printf(const char *format, ...);
```

```
//val.c
#include <stdio.h>

int main(void)
{
        int i = 10;
        char c = 'a';
        char s[] = "string";
        printf("Hi\n");
        printf("%d\n", i);
        printf("%d, %c\n", i, c);
        printf("%d, %c, %s\n", i, c, s);
        return 0;
}
```

Variable-length Argument Lists

```
//val2.c
#include <stdio.h>
#include <stdarg.h>

int sum(int num_args, ...)
{
        int val = 0;
        va_list ap;
        int i;

        va_start(ap, num_args);
        for(i = 0; i < num_args; i++)
        {
                val += va_arg(ap, int);
        }
        va_end(ap);

        return val;
}

int main(void)
{
        printf("Sum of 10 and 20 = %d\n", sum(2, 10, 20));
        printf("Sum of 10, 20 and 30 = %d\n", sum(3, 10, 20, 30));
        printf("Sum of 10, 20, 30, and 40 = %d\n", sum(4, 10, 20, 30, 40));
        return 0;
}
```

- **The called function must declare an object of type va_list which is used by the macros va_start(), va_arg(), and va_end().**
- **void va_start(va_list ap, last) - initializes ap for subsequent use by va_arg() and va_end(), and must be called first.**
- **type va_arg(va_list ap, type) - expands to an expression that has the type and value of the next argument in the call.**
- **va_end() - After the call va_end(ap) the variable ap is undefined, must be matched by a corresponding va_end();**

Command-line Argument


```c
#include <stdio.h>

int main(int argc, char**argv)
{
        int i;
        printf("Number of arguments = %d\n", argc);
        for( i=0 ; i<argc; i++){
                printf("arg[%d] = %s\n", i, argv[i]);
        }
        return 0;
}
```

```
root@localhost:/home/Lecture/C/file
[root@localhost file]# vim cmdarg.c
[root@localhost file]# gcc -o cmdarg cmdarg.c -Wall
[root@localhost file]# ./cmdarg
Number of arguments = 1
arg[0] = ./cmdarg
[root@localhost file]# ./cmdarg Hi How are you
Number of arguments = 5
arg[0] = ./cmdarg
arg[1] = Hi
arg[2] = How
arg[3] = are
arg[4] = you
[root@localhost file]# ./cmdarg 10 20 30
Number of arguments = 4
arg[0] = ./cmdarg
arg[1] = 10
arg[2] = 20
arg[3] = 30
[root@localhost file]#
```

이 장의 결론

- File and file stream
- File I/O functions
- Low-level I/O and error handling
- Argument