Part 02. 리눅스 시스템 프로그래밍

Chapter 04. 표준 I/O 라이브러리(1)

진행 순서

Chapter 04_01 표준 I/O 라이브러리 개요

< 파일 열고 닫기 >
Chapter 04_02 fopen
Chapter 04_03 fclose
Chapter 04_04 fdopen

< 파일 읽고 쓰기 >
Chapter 04_05 fgetc / fputc
Chapter 04_06 fgetc / fputc 실습
Chapter 04_07 fgets / fputs
Chapter 04_08 fread / fwrite
Chapter 04_09 fread / fwrite 실습

Chapter 04_01 표준 I/O 라이브러리 개요

표준 I/O 라이브러리란?

I/O 시스템 콜	표준 I/O 라이브러리
저수준 (커널 수준)	고수준 (어플리케이션 수준)
리눅스 커널 지원 시스템 콜	ANSI C 표준 지원 라이브러리
	기본 버퍼링
	Linux의 경우 open(), read(), write() 등 시스템 콜 이용 구현
open()	fopen()
close()	fclose()
read()	fread()
write()	fwrite()

Chapter 04_01 표준 I/O 라이브러리 개요

사용자 버퍼 입출력

사용자 버퍼 입출력은 블록 단위로 동작하는 파일 시스템과 추상 개념을 갖고 있는 애플리케이션 간의 간극을 좁혀준다.

데이터가 쓰여지면 프로그램 주소 공간 내 버퍼에 저장된다.
버퍼가 특정 크기(버퍼 크기)에 도달하면 전체 버퍼는 한번의 쓰기 연산을 통해 실제로 기록이 된다.
읽기 역시 마찬가지로 버퍼 크기에 맞춰 블록에 정렬된 데이터를 읽는다.

그 결과 데이터가 많더라도 모두 블록 크기에 맞춰 적은 횟수의 시스템 콜만 사용하게 되며
이를 통해 성능 향상을 얻을 수 있다.

표준 C 라이브러리가 제공하는 stdio 라이브러리는 플랫폼 독립적인 사용자 버퍼링 해법을 제공한다.

Chapter 04_02 fopen

```
#include <stdio.h>

FILE * fopen (const char *path, const char *mode);
```

표준 입출력 함수들은 파일 디스크립터를 직접 다루지 않는 대신 파일 포인터라는 식별자를 사용한다.
파일 포인터는 C 라이브러리 내부에서 파일 디스크립터로 매핑된다.
파일 포인터는 <stdio.h>에 정의된 FILE typedef 를 가리키는 포인터이다.

path 인자는 파일 경로를 의미하여, mode 인자는 주어진 파일을 어떻게 열지 기술한다.

r / r+	r(읽기) / r+(읽기, 쓰기) 목적으로 파일을 연다.
w / w+	w(쓰기) / w+(읽기, 쓰기) 목적으로 파일을 연다. 파일이 이미 존재하면 길이를 0으로 잘라버린다. 파일이 존재하지 않으면 새로 만든다.
a / a+	덧붙이기 상태에서 a(쓰기) / a+(읽기, 쓰기) 목적으로 파일을 연다. 파일이 존재하지 않으면 새로 만든다. 스트림(위치)은 파일 끝 지점에 위치한다. 쓰기는 파일 끝에서부터 진행된다.

성공 시 유효한 FILE 포인터를 반환한다.
실패 시 NULL을 반환하고 errno를 적절한 값으로 설정한다.

Chapter 04_03 fclose

```
#include <stdio.h>

int fclose (FILE *stream);
```

fclose()는 fopen 등을 통해 열린 파일 포인터(스트림)을 닫는다.
버퍼에 쌓여 있지만 아직 스트림에 쓰지 않은 데이터는 먼저 처리한 후 닫는다.

성공하면 0 리턴, 실패 시 EOF 리턴하고 errno 설정

Chapter 04_04 fdopen

```
#include <stdio.h>

FILE * fdopen (int fd, const char *mode);
```

fdopen() 함수는 이미 열린 파일 디스크립터를 통해 파일 포인터(스트림)를 생성한다.

mode 인자는 fopen()과 동일하며, 원래 파일 디스크립터를 열 때 사용했던 모드와 호환성을 유지해야 한다.
(w/w+ 모드일 경우 이미 파일이 존재하더라도 0으로 초기화 되지 않는다.)
스트림은 파일 디스크립터가 가리키는 위치에서 시작한다.

스트림 닫을 경우(fclose) 파일 디스크립터도 닫힌다.(close)

성공 시 유효한 FILE 포인터를 반환한다.
실패 시 NULL을 반환하고 errno를 적절한 값으로 설정한다.

Chapter 04_05 fgetc / fputc

```
#include <stdio.h>
```

```
int fgetc (FILE *stream);
```

stream에서 다음 문자를 읽고 int 타입으로 변환해서 반환한다.
에러 발생 시 EOF 리턴 (errno)

```
int fputc (int c, FILE *stream);
```

int 타입으로 변환한 한 문자를 stream에 쓴다.
쓰기 성공하면 c를 반환하고, 에러 발생 시 EOF 리턴 (errno)

```
FILE *stream = NULL;
int c;

stream = fopen("./test_file.txt", "r+");
if (stream == NULL)
        /* Error */

c = fgetc(stream);
if (c == EOF)
        /* Error */
else
        printf("c = %c\n", (char) c);

if (fputc('p', stream) == EOF)
        /* Error */

if (fclose(stream) == EOF)
        /* Error */
```

Chapter 04_06 fgetc / fputc 실습

```
#include <stdio.h>

int main(int argc, char *argv[])
{
        FILE *fp;
        int c;

        if (argc < 2) {
                printf("Usage: %s <file>\n", argv[0]);
                return -1;
        }

        fp = fopen(argv[1], "r+");
        if (!fp) {
                perror("fopen error: ");
                return -1;
        }

        while ((c = fgetc(fp)) != EOF) {
                if (c == '!' || c == '@' || c == '#' || c == '$' ||
                  c == '%' || c == '^' || c == '&' || c == '*' ||
                  c == '-' || c == '_') {
                        if (ungetc(c, fp) == EOF) {
                                perror("ungetc error: ");
                                return -1;
                        }
                        if (fputc(' ', fp) == EOF) {
                                perror("fputc error: ");
                                return -1;
                        }
                }
        }

        if (fclose(fp) == EOF) {
                perror("fclose error: ");
                return -1;
        }

        return 0;
}
```

Chapter 04_06 fgetc / fputc 실습 (결과)

```
[root@localhost ch11]# gcc -g stdio_example2.c -o stdio_example2

[root@localhost ch11]# ./stdio_example2
Usage: ./stdio_example2 <file>

[root@localhost ch11]# cat data2.file
Hello World!
This is test-file.
test@email.com

[root@localhost ch11]# ./stdio_example2 data2.file

[root@localhost ch11]# cat data2.file
Hello World
This is test file.
test email.com
```

Chapter 04_07 fgets / fputs

```
#include <stdio.h>

char * fgets (char *str, int size, FILE *stream);
```

fgets()는 스트림에서 한 문자열을 읽는다.
stream에서 size보다 한 문자가 적은 내용을 읽어서 str에 저장하고,
버퍼 마지막에 문자열의 끝을 나타내는 null 문자(\0)을 저장한다.
EOF나 개행문자를 만나면 읽기를 중단한다.
성공하면 str를 반환하고 실패하면 NULL을 리턴

```
int fputs (const char *str, FILE *stream);
```

fputs()는 스트림에 문자열을 기록한다.
str이 가리키는 NULL로 끝나는 문자열 전부를 stream이 가리키는 스트림에 기록한다.
성공하면 쓴 크기를 반환하고, 실패하면 EOF를 반환한다.

```
FILE *stream = NULL;
char buf[MAX];

stream = fopen("./test_file.txt", r+);
if (stream == NULL)
        /* Error */

if (!fgets(buf, MAX, stream))
        /* Error */

if (fputs("Hello World.\n", stream) == EOF)
        /* Error */

if (fclose(stream) == EOF)
        /* Error */
```

Chapter 04_08 fread / fwrite

```
#include <stdio.h>

size_t fread (void *buf, size_t size, size_t nr, FILE *stream);
```

fread()는 stream에서 크기가 size 바이트인 요소를 nr개 읽어서 buf가 가리키는 버퍼에 저장한다.
즉 원하는 총 읽는 바이트 수는 size * nr 바이트가 된다.
주의할 점은 읽은 바이트 수가 리턴되는 것이 아니라 읽어 들인 요수의 수(nr)가 리턴된다.
즉 nr 보다 적은 값을 반환하여 실패나 EOF를 알려준다.

```
size_t fwrite (void *buf, size_t size, size_t nr, FILE *stream);
```

fwrite()는 buf가 가리키는 데이터에서 size 크기의 요소 nr 개를 stream에 쓴다.
즉 쓰기 원하는 총 바이트 수는 size * nr 바이트가 된다.
fread와 마찬가지로 쓴 요소의 수 nr가 리턴된다.
nr 보다 작은 리턴값은 실패를 나타낸다.

Chapter 04_09 fread / fwrite 실습

```
#include <stdio.h>

#define FILENAME "./data.file"

int main(void)
{
        FILE *readfile, *writefile;

        struct address_info {
                unsigned int index;
                char name[32];
                char phone_number[16];
                char address[100];
        };

        struct address_info kim, who;

        kim.index = 1;
        sprintf(kim.name, "Kim");
        sprintf(kim.phone_number, "010-1234-5678");
        sprintf(kim.address, "Seoul");

        writefile = fopen(FILENAME, "w");
        if (!writefile) {
                perror("fopen error: ");
                return -1;
        }
        if (!fwrite(&kim, sizeof(struct address_info), 1, writefile)) {
                perror("fwrite error: ");
                return -1;
        }

        if (fclose(writefile)) {
                perror("fclose error: ");
                return -1;
        }

        readfile = fopen(FILENAME, "r");
        if (!readfile) {
                perror("fopen error: ");
                return -1;
        }

        if (!fread(&who, sizeof(struct address_info), 1, readfile)) {
                perror("fread error: ");
                return -1;
        }

        if (fclose(readfile)) {
                perror("fclose error: ");
                return -1;
        }
```

Chapter 04_09 fread / fwirte 실습 (결과)

```
        printf("index: %d\n", who.index);
        printf("name: %s\n", who.name);
        printf("phone number: %s\n", who.phone_number);
        printf("address: %s\n", who.address);

        return 0;
}
```

```
[root@localhost ch11]# gcc -g stdio_example.c -o stdio_example
[root@localhost ch11]# ./stdio_example
index: 1
name: Kim
phone number: 010-1234-5678
address: Seoul
[root@localhost ch11]# ls
data.file  stdio_example  stdio_example.c
```