03. 파일 다루기

`

2023. 03. 17
Linux System Programming

01. 시스템 프로그래밍

■ 시스템 호출과 라이브러리 함수

- 시스템 호출과 라이브러리 함수 비교

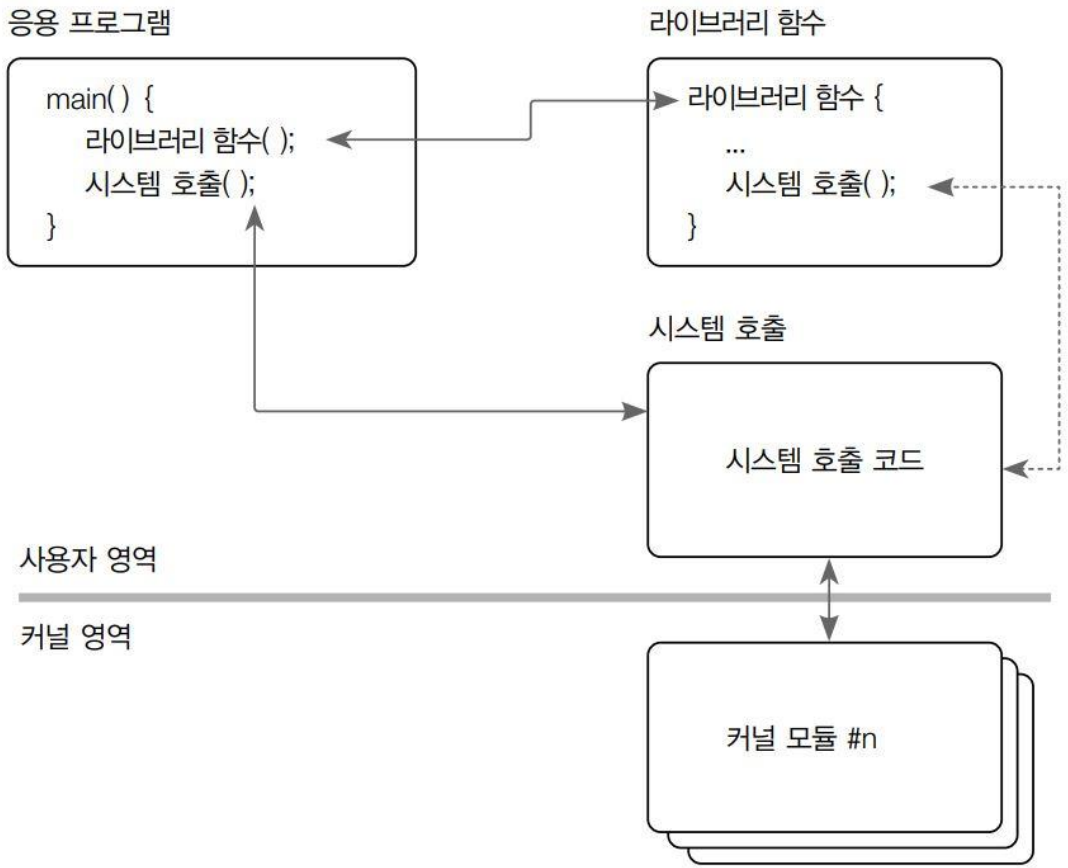

그림 1-2 시스템 호출과 라이브러리 함수의 비교

01. 시스템 프로그래밍

■ 시스템 호출과 라이브러리 함수

▪ man 페이지의 섹션 번호

- 매뉴얼은 항목의 종류에 따라 섹션이 구분되어있음
  - 리눅스에서 사용하는 일반적인 명령에 대한 설명: 섹션 1
  - 시스템 호출: 섹션 2
  - 라이브러리 함수: 섹션 3

- man 명령으로 검색하면 섹션 번호가 가장 낮은 것이 기본으로 출력됨
- man 명령의 결과를 출력하는 형식은 리눅스와 유닉스에서 차이가 있음

01. 개요

■ 파일

▪ 파일

- 관련 있는 데이터의 집합으로, 저장 장치에 일정한 형태로 저장
- 데이터를 저장하는 데는 물론 데이터를 전송하거나 장치에 접근하는 데도 사용
- 리눅스에서 파일은 크게 일반 파일과 특수 파일로 구분
- 특수 파일의 생성과 삭제 및 입출력은 특수 파일별로 약간씩 차이가 있음

표 4-1 파일의 종류

종류	용도
일반 파일	텍스트나 바이너리 형태의 자료를 저장하는 파일
특수 파일	데이터 전송 또는 장치 접근에 사용하는 파일

01. 개요

■ 파일

▪ 파일 읽고 쓰는 방법

- 저수준 파일 입출력
  - 리눅스 커널의 시스템 호출을 이용해 파일 입출력을 수행
  - 시스템 호출을 이용하므로 파일에 좀 더 빠르게 접근할 수 있는 장점
  - 또한 바이트 단위로 파일의 내용을 다루므로 일반 파일뿐만 아니라 특수 파일도 읽고 쓸 수 있음
  - 바이트 단위로만 입출력을 수행 가능 하므로 응용프로그램 작성시 다른 추가기능을 함수로 추가 구현 해야함
  - 열린 파일을 참조할 때 파일 기술자 사용

- 고수준 파일 입출력
  - 저수준 파일 입출력의 불편함을 해결하기 위해 제공
  - C 언어의 표준 함수로 제공
  - 데이터를 바이트 단위로 한정하지 않고 버퍼를 이용해 한꺼번에 읽기와 쓰기를 수행
  - 다양한 입출력 데이터 변환 기능도 이미 구현되어 있어 자료형에 따라 편리하게 이용할 수 있음
  - 열린 파일을 참조할 때 파일 포인터 사용

01. 개요

■ 파일

▪ 파일 읽고 쓰는 방법

표 4-2 저수준 파일 입출력과 고수준 파일 입출력 비교

	저수준 파일 입출력	고수준 파일 입출력
파일 지시자	int fd	FILE *fp;
특징	– 훨씬 빠름 – 바이트 단위로 읽고 쓰기 – 특수 파일에 대한 접근 가능	– 사용하기 쉬움 – 버퍼 단위로 읽고 쓰기 – 데이터의 입출력 동기화가 쉬움 – 여러 가지 형식을 지원
주요 함수	open(), close(), read(), write(), dup(), dup2(), fcntl(), lseek(), fsync()	fopen(), fclose(), fread(), fwrite(), fputs(), fgets(), fprintf(), fscanf(), freopen(), fseek()

- **Example 1**

```
#include <stdio.h>

void writeFile(void);
void readFile(void);

int main()
{
        writeFile();
        readFile();
}
```

```
void writeFile()
{
        FILE *fp;
        fp = fopen("testFile", "w");
        fprintf(fp, "File Write Test₩n");
        char testStr[1024] = "Test String₩n";
        fwrite(testStr, 100, 1, fp);
        fclose(fp);
}

void readFile()
{
        FILE *fp;
        char str[1024];
        fp = fopen("testFile", "r");
        fgets(str, 1024, fp);          printf("%s", str);
        fread(str, 1024, 1, fp);       printf("%s₩n", str);
        fclose(fp);
}
```

- **Example 2**

```
#include <stdlib.h>

void file1(char *);

int main(int argc, char **argv)
{
        if(argc!=2) exit(0);

        file1(argv[1]);
}
```

```
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

void file1(char *filename)
{
        int fd = open(filename, O_RDWR|O_CREAT, 0644);
        char a[1024] = "01234567890123";
        write(fd, a, 10);
        close(fd);

        fd = open(filename, O_RDWR, 0644);
        char b[1024]={};
        read(fd, b, 4);
        close(fd);
        printf("%s₩n", b);
}
```

- **errno**
  - 마지막 error에 해당하는 숫자 저장
- **strerror**
  - errno를 문자열로 표현

#include <errno.h> // errno	#include <string.h> char *strerror(int errnum);

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>

int main()
{
   int file;
   if( ( file = open("testFile", O_RDONLY) ) < 0 ) // 존재하지 않는 파일
   {
      printf("%d₩n", errno);
      printf("%s₩n", strerror(errno));
   }
}
```

- **Example 3**

```
#include <stdlib.h>

void file1(char *);
void file2(char *);

int main(int argc, char **argv)
{
        if(argc!=3) exit(0);

        file1(argv[1]);
        file2(argv[2]);
}

#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
```

```
void file1(char *filename)
{
        int fd = open(filename, O_RDWR, 0644)
        if (fd < 0)
        {
                printf("%s(%d)₩n", strerror(errno), errno);
        }
}


void file2(char *filename)
{
        int fd = open(filename, O_RDWR|O_CREAT, 0644);
        char a[1024] = "01234567890123";
        printf("%d₩n", (int)write(fd, a, 10));
        printf("%s(%d)₩n", strerror(errno), errno);
        close(fd);
}
```

- **File 관리 명령**
  - unlink, remove : 삭제
  - rename : 변경
  - cp/copy : 복사

`

- **rename**

  - 파일 명 또는 위치를 변경한다

```
#include <stdio.h>

int rename(const char *oldpath, const char *newpath);
```

oldpath	Old Path
Newpath	New Path
반환값	작업이 성공할 경우 0이 반환되며, 실패할 경우 -1이 반환된다.

- **cp or copy**
  - 파일을 복사한다.

```
int cp (const char *oldpath, const char *newpath);   /* ?? */

/* linux에서 제공하지 않음 */
/* System 함수 사용 ??     */
/* cp 함수 제작 ??         */
```

oldpath	Old Path
Newpath	New Path
반환값	작업이 성공할 경우 0이 반환되며, 실패할 경우 -1이 반환된다.