8. File: Low-level I/O

Hyunchan, Park

http://oslab.jbnu.ac.kr

Division of Computer Science and Engineering

Jeonbuk National University

학습 내용

- File
- Low-level File I/O

개인 과제 7: 실습 (2/2)

- 실습 과제
  - 실습 내용에서 등장한 프로그램들을 작성하고, 그 결과를 확인할 것
    - 결과 성공 후, cat 으로 파일의 내용을 출력할 것
    - 예제 슬라이드에 나온 프로그램들은 모두 포함되어야 함
  - 제출 방법
    - Old LMS, 과제 7
    - Xshell 로그 파일 1개 제출 (앞의 7.System Call 슬라이드에 이어서 작성)
    - 파일 명: 학번.txt
- 제출 기한
  - 10/26 (월) 23:59 (지각 감점: 5%p / 12H, 1주 이후 제출 불가)

File

Volatile and Non-volatile storage devices

- Primary storage: Main memory
  - 주기억장치로 사용하는 DRAM 등의 휘발성 저장 장치
  - 성능이 높지만, 적은 저장 공간 제공
    - 프로그램 내의 변수와 같이 용량은 적지만 자주 접근하는 자료를 저장
- Secondary storage: Storage devices
  - 보조기억장치로 사용하는 HDD, SSD 등은 비휘발성 저장 장치
  - 느리지만, 많은 저장 공간을 제공
    - 시스템 종료 시에도 보관하여야 할 데이터를 적재하고, 시스템 재기동 시 다시 로드
  - 일반적으로 파일(file)의 형태로 데이터를 저장함

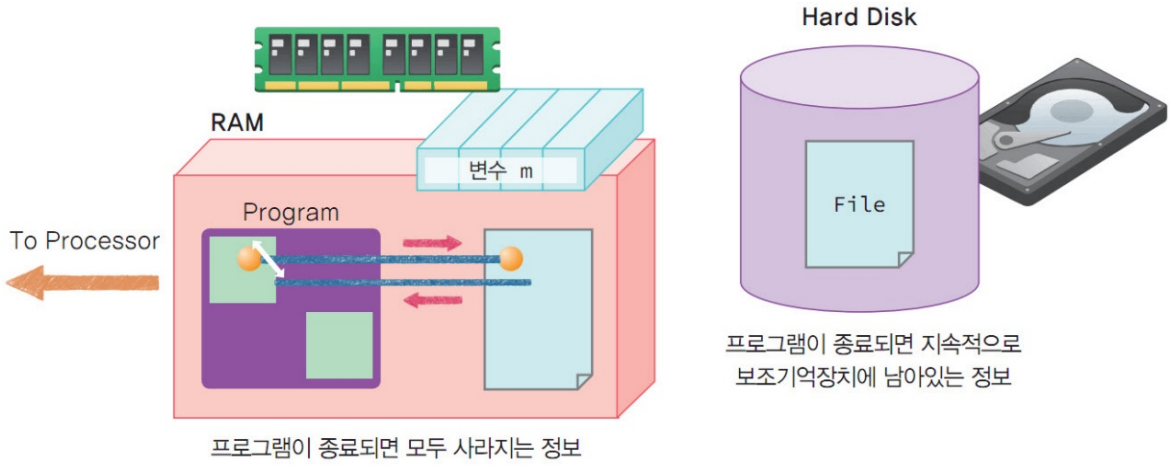


그림 15-2 주기억장치의 저장공간과 파일의 차이

파일과 파일 속성

- 파일: 보조기억장치의 정보저장 단위로 자료의 집합
  - Collection of data
  - 0부터 시작하는 주소 공간에 1B 단위로 데이터를 저장하고, 접근할 수 있음
- 파일의 속성 (attribute)
  - 파일은 Data 와 Metadata 로 구성됨
    - Data: 사용자가 저장하고자 하는 데이터
    - Metadata: 데이터의 속성이나 특징 등, 저장된 데이터를 설명하기 위한 추가적인 데이터
    - Metadata 를 주로 file attribute 라고 지칭함
  - 대표적인 Metadata
    - Name, Size, Creation time, Last modified time, Last access time, access control
  - 추가적인 Metadata
    - 사진: 찍은 날짜, 위치, 포함된 사람들 등등
    - 영화: Runtime, 제목, 첫 상영 날짜, 출연 배우 등등
  - 보다 다양한 정보가 요구됨에 따라 파일의 metadata 저장 방식도 발전하고 있음

File operations

- 파일에 대해 수행할 수 있는, OS가 제공하는 기본 동작
  - Basic: open, close, read, write
    - 파일을 사용하기 위해서는 우선 열어야 하고, (open)
    - 사용을 완료한 후에는 닫아야 한다. (close)
    - 이외의 데이터 접근 방법은 없음!
      - Insert? Modification?
      - 모두 Read and Write 로 구현하여야 함
  - Control
    - Seek: 파일의 현재 위치 설정
    - Truncate: 파일의 내용 삭제
    - Delete (or remove, unlink, release): 파일 삭제
    - Flush (or sync): 파일의 내용을 즉각 저장 장치에 기록
    - 기타: mmap, poll, lock 등 공유 및 동기화를 위한 제어 방법들

Low- & High-level File I/O (Input and Output)

- Low-level File I/O
  - 파일을 다루기 위해 OS에서 제공하는 시스템콜들을 직접 사용
  - 특수한 파일 (장치 파일 등)을 제어할 때 주로 사용함
  - Open(), close(), read(), write(), lseek() 등
- High-level File I/O
  - C 라이브러리 등에서 제공하는 보다 편리한 파일 입출력 서비스
  - Low-level file I/O 는 시스템콜 그 자체라면,
  - High-level 은 시스템콜을 다양한 형태로 가공하여 편의성을 높인 것
  - 예) fprintf() : 파일에 바로 원하는 형태의 문자열을 출력할 수 있음
    - Low-level 에서는 우선 문자열을 원하는 형태로 가공한 후, write()를 사용해 문자열의 길이 등을 함께 전달하여 기록해야 함
  - fopen(), fclose() 등, 이름이 유사하나 앞에 f 가 붙는 경우가 많음
  - 라이브러리 내에서 성능 등의 이유로 버퍼를 이용하므로, 버퍼 기반 입출력 이라고 부르기도 함

Low-level File I/O

파일 생성과 열고 닫기[1]

- 파일 열기: open(2)

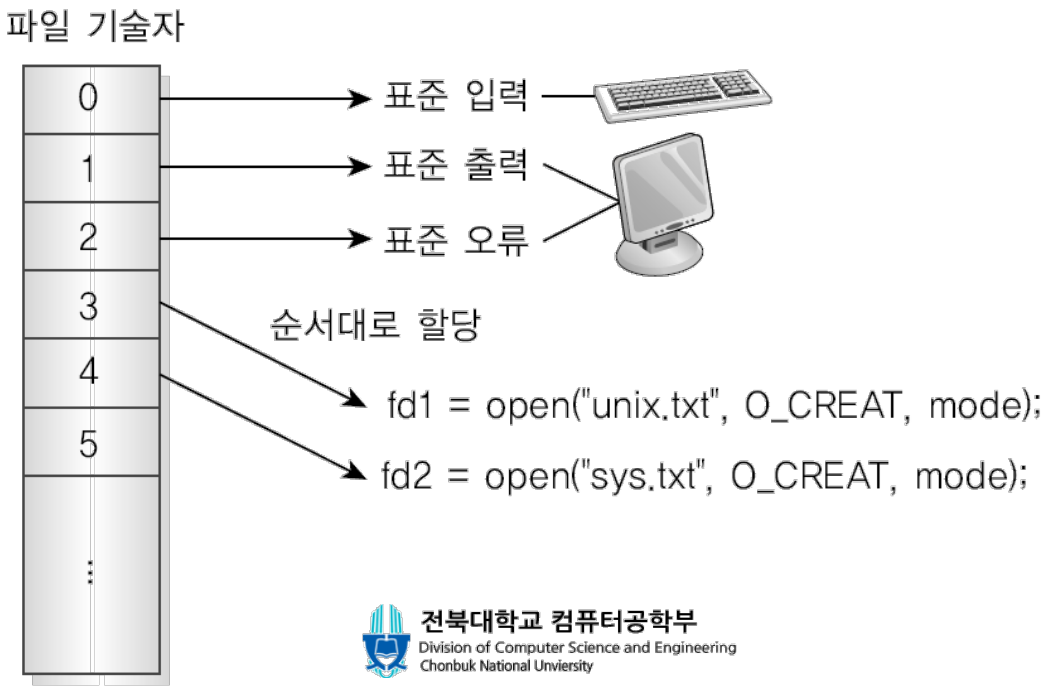
```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

- Pathname 에 지정한 파일을 flags에 지정한 플래그 값에 따라 연다.
  - Pathname 은 상대 주소, 절대 주소 모두 가능
  - 만약 지정한 파일이 없고, flag 에 O_CREAT 가 설정된 경우 새로운 파일 생성
- Return value
  - The new file descriptor (파일 기술자)
  - -1 on error

파일 기술자

- 파일 기술자
  - 현재 프로세스를 통해 열려 있는 파일을 구분하는 정수값.
    - OS가 해당 프로세스의 열린 파일들을 관리하기 위해 사용하는 테이블의 Index
  - 저수준 파일 입출력에서 열린 파일을 참조하는데 사용
  - 기본으로 지정된 파일 기술자들
    - 0번 : 표준 입력, 1번 : 표준 출력, 2번 : 표준 오류



파일 생성과 열고 닫기[2]

- Flags: 파일을 어떤 모드로 열지 선택
  - OR 연산(|)을 이용해 복합하여 사용 가능
    - 예) O_RDWR|O_CREAT : 읽기/쓰기 모두 가능한 모드로 열고, 파일이 없는 경우 새로 생성함

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 추가하는 모드. (position 이 EOF: End-of-File 로 설정됨)
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

파일 생성과 열고 닫기[3]

- mode : 옵션으로, 파일 접근권한을 지정할 수 있음
  - 0644같이 숫자나 플래그 값으로 지정 가능 (앞에 0을 넣어 Octet 임을 표시)

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;
= 0600 소유자 RW

파일 생성과 열고 닫기[4]

- 파일 닫기: close(2)

```
#include <unistd.h>

int close(int fd);
```

  - 파일의 사용이 끝나면 close()를 호출하여 파일을 명시적으로 닫아야 함
    - 프로세스에서 열 수 있는 파일 개수가 제한되어 있으므로 이를 확보하기 위함
    - 파일의 사용이 끝났음을 알려, 다른 프로세스가 파일을 수정하거나 제어할 수 있도록 함
  - 주의! Close()를 수행한다고 해서 모든 데이터가 저장 장치에 기록이 완료된 것은 아님
    - 데이터가 메모리 상에 남아있고, 저장 장치에는 기록이 되지 않은 상태일 수 있음

- Return value
  - Close() returns zero on success.
  - On error, -1 is returned, and errno is set appropriately.
    - 일반적으로 close()는 에러 체크를 하지 않음.
    - Open() 시 에러 체크를 통해, 파일이 열려있다고 확신을 한 상태에서 작업하기 때문
    - 하지만, 드물게 write()가 실패한 경우, close()에서 에러가 발생하여 이를 알릴 수 있으므로 close()에서도 에러 체크를 하는 것이 좋음

파일 생성과 열고 닫기[5]

- 파일 생성 : creat(2)

```
#include <sys/stat.h>
#include <fcntl.h>

int creat(const char *path, mode_t mode);
```

  - A call to creat() is equivalent to calling open() with flags equal to O_CREAT|O_WRONLY|O_TRUNC
  - open 함수와 달리 옵션을 지정하는 부분이 없다.
  - Creat() 함수로 파일을 생성하면 파일 기술자를 리턴하므로 별도로 open할 필요 없음

  - 이런건 배우지 말자. Open() 만 알면 된다!
    - Open() 에 파일 생성 flag이 없던 구버전 유닉스에서 사용

[예제 1] 새 파일 열고 닫기

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main(void) {
    int fd;
    char name[] = "unix.txt";
    mode_t mode;

    mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;

    fd = open(name, O_CREAT, mode);
    if (fd == -1) {
        perror("Creat");
        exit(1);
    }

    printf("%s is opened! fd = %d\n", name, fd);
    close(fd);

    return 0;
}
```

```
ubuntu@41983:~/hw2$ ls -al unix.txt
-rw-r--r-- 1 ubuntu ubuntu 0 Oct 14 14:01 unix.txt
ubuntu@41983:~/hw2$ rm unix.txt
ubuntu@41983:~/hw2$ gcc -o file1 file1.c && ./file1
unix.txt is opened! fd = 3
ubuntu@41983:~/hw2$ ls -al unix.txt
-rw-r--r-- 1 ubuntu ubuntu 0 Oct 14 14:01 unix.txt
ubuntu@41983:~/hw2$ echo hello > unix.txt
ubuntu@41983:~/hw2$ cat unix.txt
hello
ubuntu@41983:~/hw2$ ./file1
unix.txt is opened! fd = 3
ubuntu@41983:~/hw2$ cat unix.txt
hello
ubuntu@41983:~/hw2$
```

[예제 2] O_EXCL 플래그 사용하기

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main(void) {
    int fd;
    char name[] = "unix.txt";
    mode_t mode;

    mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;

    fd = open(name, O_CREAT|O_EXCL, mode);
    if (fd == -1) {
        perror("Creat");
        exit(1);
    }

    printf("%s is opened! fd = %d\n", name, fd);
    close(fd);

    return 0;
}
```

```
ubuntu@41983:~/hw2$ cp file1.c file2.c
ubuntu@41983:~/hw2$ vi file2.c
ubuntu@41983:~/hw2$ rm unix.txt
ubuntu@41983:~/hw2$ gcc -o file2 file2.c && ./file2
unix.txt is opened! fd = 3
ubuntu@41983:~/hw2$ ls -al unix.txt
-rw-r--r-- 1 ubuntu ubuntu 0 Oct 14 13:58 unix.txt
ubuntu@41983:~/hw2$ ./file2
Creat: File exists
ubuntu@41983:~/hw2$
```

File Read and Write: Sequential Access

- 파일은 기본적으로 데이터를 sequential 하게 접근하는 것을 가정
  - Sequential access: 처음부터 순서대로 데이터를 읽음
    - 예) 음악이나 영화 파일의 재생
  - Random access: 임의의 위치에 저장된 데이터를 순서에 무관하게 접근
    - 예) 성적 리스트에서 학번을 찾아 해당하는 성적을 확인하는 것
- File access position
  - OS는 파일에 대한 현재 읽고 쓰기 위한 위치 (position or offset)을 관리함
    - 위치는 Byte 단위로 표현
    - 이 정보는 OS가 관리하므로, 프로그램은 직접 수정할 수 없음 (system call 사용)
  - 파일을 열면 (open), position 은 0으로 설정됨
  - 이때, 파일을 읽거나 쓰면, 0번 주소부터 데이터를 읽거나 씀
  - 그리고 이때 읽거나 쓴 데이터의 양만큼 position 은 자동으로 이동함
    - 예) 만약 80 B의 데이터를 읽은 다음에는 position 은 80 으로 변경됨
  - 이후 읽거나 쓰게 되면, 이동한 위치에서부터 다시 데이터를 접근함

파일 읽기와 쓰기

- 파일 읽기 : read(2)

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t nbytes);
```

  - fd가 가리키는 파일에서,
  - nbytes로 지정한 크기만큼 바이트를 읽어서,
  - buf에 저장

  - Return value
    - 실제로 읽어온 바이트 개수를 리턴
    - 리턴값이 0이면 파일의 끝에 도달했음을 의미
  - 파일의 종류에 상관없이 무조건 바이트 단위로 읽어온다.

파일 읽기와 쓰기

- 파일 쓰기 : write(2)

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t nbytes);
```

  - fd가 지정하는 파일에,
  - buf가 가리키는 메모리로부터,
  - nbytes로 지정한 크기만큼 쓰기

  - Return value
    - 실제로 쓰기를 수행한 바이트 수를 리턴

[예제 3] 파일 읽기

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char* argv[]) {
    int fd, count, line=0;
    char buf[80];       //80 = 1 line for standard console

    if(argc != 2) {
            printf("< Usage: ./file3 filename >\n");
            return 1;
    }

    fd = open(argv[1], O_RDWR);
    if (fd == -1) {
        perror("Open");
        exit(1);
    }

    printf("%s is opened! fd = %d\n", argv[1], fd);

    while ((count = read(fd, buf, 10)) > 0 ) {
        printf("%d: %s\n", count, buf);
    }

    close(fd);

    return 0;
}
```

```
ubuntu@41983:~/hw2$ gcc -o file3 file3.c
ubuntu@41983:~/hw2$ ./file3
< Usage: ./file3 filename >
ubuntu@41983:~/hw2$ ./file3 nofile
Open: No such file or directory
ubuntu@41983:~/hw2$ ./file3 file3.c
file3.c is opened! fd = 3
10: #include <
10: sys/types.
10: h>
#includ
10: e <sys/sta
10: t.h>
#incl
```

```
ubuntu@41983:~/hw2$ gcc -o file3 file3.c && ./file3 novel.txt
novel.txt is opened! fd = 3
10:    Once up
10: on a time
10: . . . ther
10: e were thr
10: ee little
10: pigs, who
10: left their
10:  mummy
and
10:  daddy to
10: see the wo
10: rld.
   Al
```

[예제 4] 파일 읽고 쓰기

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char* argv[]) {
    int rfd, wfd, count, line=0;
    char buf[80];       //80 = 1 line for standard console

    if(argc != 3) {
            printf("< Usage: ./file3 file_for_read file_for_write >\n");
            return 1;
    }

    rfd = open(argv[1], O_RDONLY);
    if (rfd == -1) {
        perror("Open file for read");
        exit(1);
    }

    wfd = open(argv[2], O_RDWR|O_CREAT|O_EXCL, 0644);
    if (wfd == -1) {
        perror("Open file for write");
        exit(1);
    }

    printf("%s and %s are opened! rfd = %d wfd = %d\n", argv[1], argv[2], rfd, wfd);

    while ((count = read(rfd, buf, 10)) > 0 ) {
            write(wfd, buf, count);
    }

    close(rfd);
    close(wfd);

    return 0;
}
```

[예제 4] 파일 읽고 쓰기

```
ubuntu@41983:~/hw2$ gcc -o file4 file4.c
ubuntu@41983:~/hw2$ ./file4 unix.txt new.txt
Open file for write: File exists
ubuntu@41983:~/hw2$ rm new.txt
ubuntu@41983:~/hw2$ ./file4 unix.txt new.txt
unix.txt and new.txt are opened! rfd = 3 wfd = 4
ubuntu@41983:~/hw2$ cat unix.txt
hello
ubuntu@41983:~/hw2$ cat new.txt
hello
ubuntu@41983:~/hw2$ ls -al unix.txt new.txt
-rw-r--r-- 1 ubuntu ubuntu 6 Oct 15 02:19 new.txt
-rw-r--r-- 1 ubuntu ubuntu 6 Oct 14 14:01 unix.txt
```

```
ubuntu@41983:~/hw2$ ./file4 file4.c file5.c
file4.c and file5.c are opened! rfd = 3 wfd = 4
ubuntu@41983:~/hw2$ ls -al file4.c file5.c
-rw-rw-r-- 1 ubuntu ubuntu 830 Oct 15 02:17 file4.c
-rw-r--r-- 1 ubuntu ubuntu 830 Oct 15 02:23 file5.c
ubuntu@41983:~/hw2$ ./file4 novel.txt newfile.txt
novel.txt and newfile.txt are opened! rfd = 3 wfd = 4
ubuntu@41983:~/hw2$ ls -al novel.txt newfile.txt
-rw-r--r-- 1 ubuntu ubuntu 5362 Oct 15 02:24 newfile.txt
-rw-rw-r-- 1 ubuntu ubuntu 5362 Oct 15 01:46 novel.txt
ubuntu@41983:~/hw2$
```

파일 오프셋 지정

- 파일 오프셋 위치 지정 : lseek(2)

```
#include <sys/types.h>
#include <unistd.h>

off_t lseek(int fd, off_t offset, int whence);
```

  - offset으로 지정한 크기만큼 오프셋을 이동시킨다.
  - offset의 값은 whence값을 기준으로 해석한다.

```
lseek(fd, 5, SEEK_SET);
lseek(fd, 0, SEEK_END);
```

파일의 시작에서 5번째 위치로 이동

파일의 끝에서 0번째, 즉 끝으로 이동

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

  - 파일 오프셋의 현재 위치를 알려면?

```
cur_offset = lseek(fd, 0, SEEK_CUR);
```

[예제 5] 파일 오프셋 사용하기

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

char buf[80];
int fd, count;

void read_five_bytes(void) {
    if ((count = read(fd, buf, 5)) <= 0 ) {
        perror("Read Error");
        exit(1);
    }
}

int main(int argc, char* argv[]) {
    if(argc != 2) {
            printf("< Usage: ./file3 filename >\n");
            return 1;
    }

    fd = open(argv[1], O_RDWR);
    if (fd == -1) {
        perror("Open");
        exit(1);
    }

    printf("%s is opened! fd = %d\n", argv[1], fd);
    read_five_bytes();
    printf("\n%d: %s\n", count, buf);
    printf("Current position: %d\n", lseek(fd, 0, SEEK_CUR));

    lseek(fd,1,SEEK_SET);
    read_five_bytes();
    printf("\n%d: %s\n", count, buf);
    printf("Current position: %ld\n", lseek(fd, 0, SEEK_CUR));

    lseek(fd,2,SEEK_SET);
    read_five_bytes();
    printf("\n%d: %s\n", count, buf);
    printf("Current position: %ld\n", lseek(fd, 0, SEEK_CUR));

    close(fd);

    return 0;
}
```

[예제 5] 파일 오프셋 사용하기

```
ubuntu@41983:~/hw2$ ./file5 unix.txt
unix.txt is opened! fd = 3

5: hello
Current position: 5

5: ello

Current position: 6

4: llo


Current position: 6
```

```
ubuntu@41983:~/hw2$ ./file5 file5.c
file5.c is opened! fd = 3

5: #incl
Current position: 5

5: inclu
Current position: 6

5: nclud
Current position: 7
ubuntu@41983:~/hw2$
```