

# 제4장 파일 입출력

---

리눅스 시스템 프로그래밍

청주대학교 전자공학과  
한철수

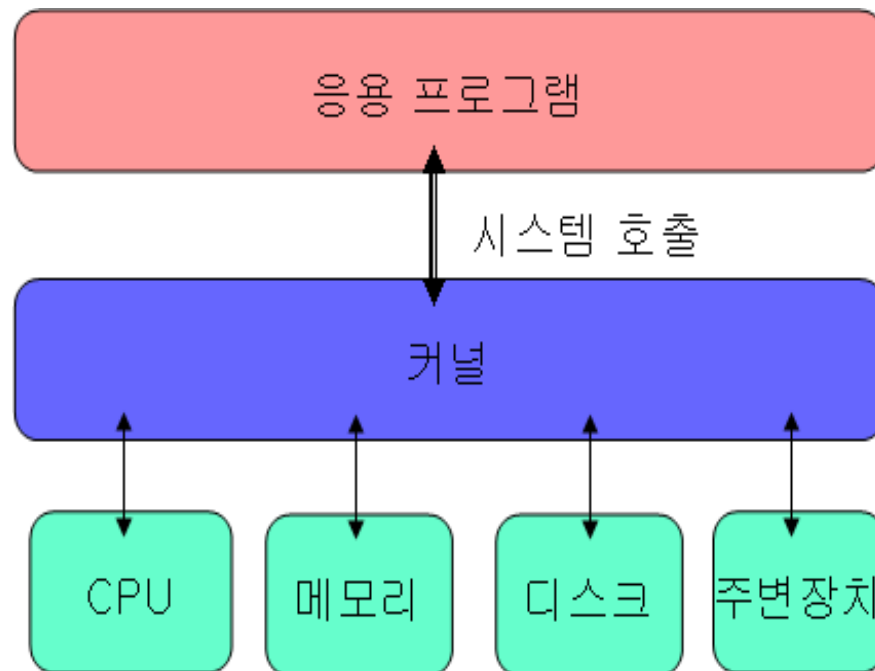
# 체크 포인트

---

- 시스템 호출
- 파일
- 임의 접근

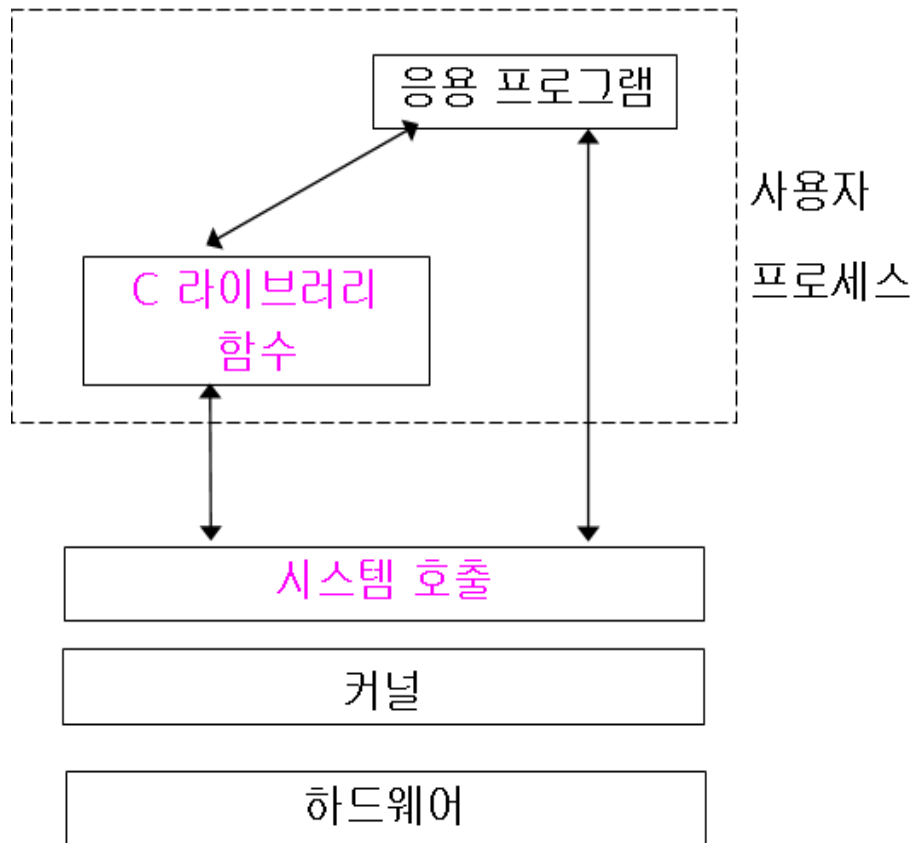
# 유닉스 커널(kernel)

- 하드웨어를 운영 관리하여 다음과 같은 서비스를 제공
  - 파일 관리(File management)
  - 프로세스 관리(Process management)
  - 메모리 관리(Memory management)
  - 통신 관리(Communication management)
  - 주변장치 관리(Device management)

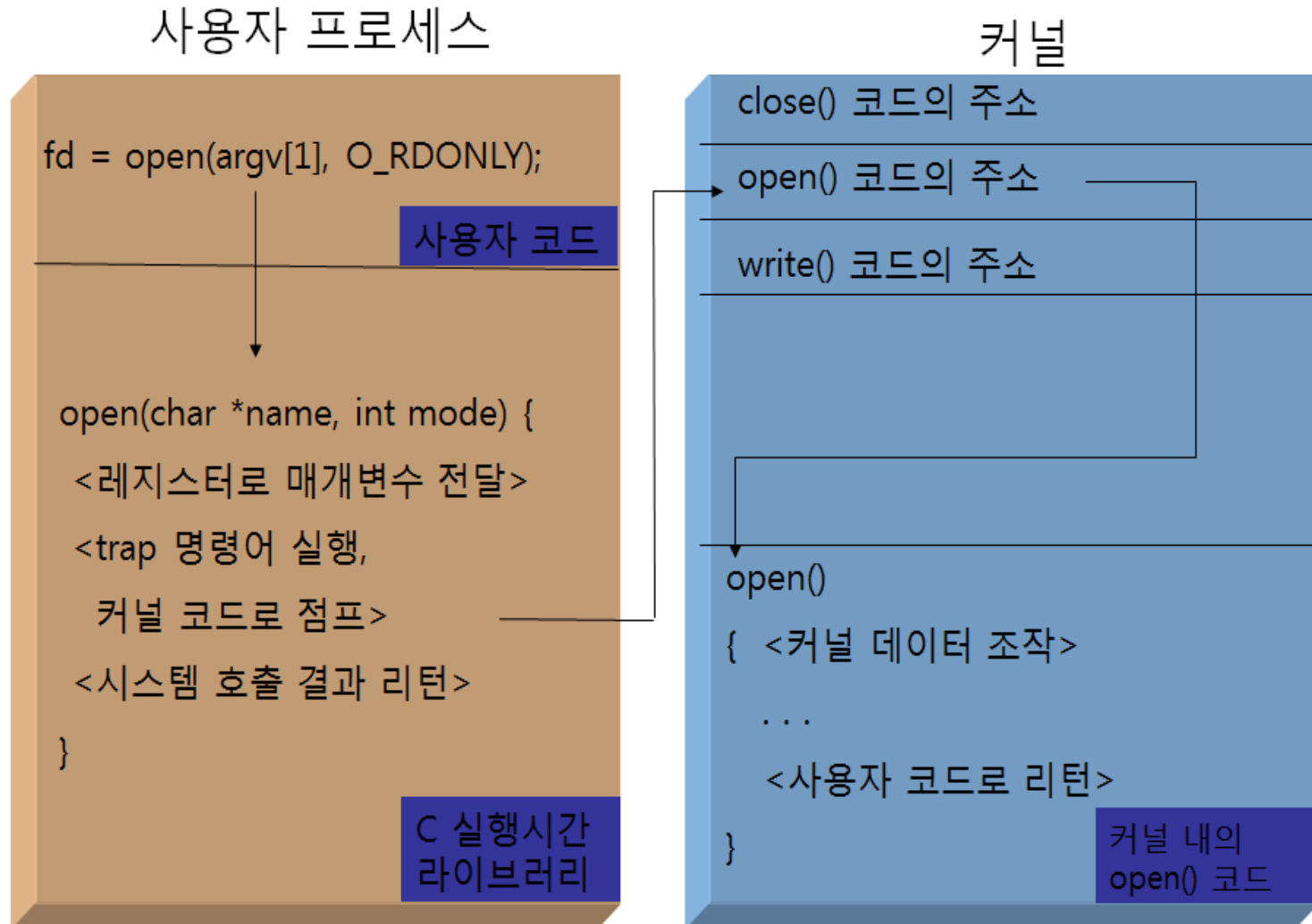


# 시스템 호출(system call)

- 운영체제가 제공하는 서비스에 대한 프로그래밍 인터페이스
- 응용 프로그램은 시스템 호출을 통해서 커널에 서비스를 요청함



# 시스템 호출 과정



# 주요 시스템 호출

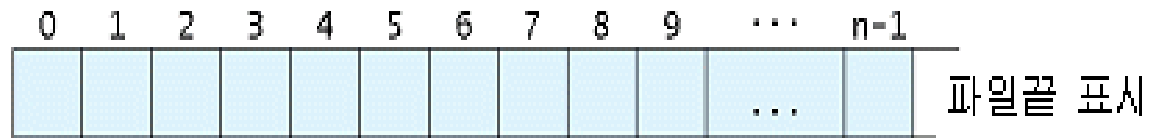
---

주요 자원	시스템 호출
파일	open(), close(), read(), write(), dup(), lseek() 등
프로세스	fork(), exec(), exit(), wait(), getpid(), getppid() 등
메모리	malloc(), calloc(), free() 등
시그널	signal(), alarm(), kill(), sleep() 등
프로세스 간 통신	pipe(), socket() 등

# 리눅스 파일

---

- 연속된 바이트의 나열



# 파일 열기: open()

- 파일을 사용하기 위해서는 먼저 open() 시스템 호출을 이용하여 파일을 열어야 함

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

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

```
#include <fcntl.h>
```

```
int open (const char *path, int oflag, [ mode_t mode ]);
```

파일 열기에 성공하면 파일 디스크립터를, 실패하면 -1을 리턴

- 파일 디스크립터는 열린 파일을 나타내는 번호임



# 입출력방식 지정(oflag)

```
int open (const char *path, int oflag, [ mode_t mode ]);
```

- O\_RDONLY  
읽기 모드
- O\_WRONLY  
쓰기 모드
- O\_RDWR  
읽기/쓰기 모드
- 예  
fd = open("account", O\_RDONLY);  
fd = open(argv[1], O\_RDWR);

# oflag의 다양한 옵션

```
int open (const char *path, int oflag, [ mode_t mode ]);
```

- O\_APPEND
  - 데이터를 쓰면 파일끝에 첨부됨
- O\_CREAT
  - 해당 파일이 없는 경우에 생성함
  - **mode**는 생성할 파일의 사용권한을 나타냄
- O\_TRUNC
  - 파일이 이미 있는 경우 내용을 지움
- O\_EXCL
  - O\_CREAT와 함께 사용되며 해당 파일이 이미 있으면 오류
- O\_NONBLOCK
  - 넌블로킹 모드로 입출력 하도록 함
- O\_SYNC
  - write() 시스템 호출을 하면 디스크에 물리적으로 쓴 후 반환됨

# 파일 열기의 예

---

- `fd = open("account", O_RDONLY);`
- `fd = open(argv[1], O_RDWR);`
- `fd = open(argv[1], O_RDWR | O_CREAT, 0600);`
- `fd = open("tmpfile", O_WRONLY | O_CREAT | O_TRUNC, 0600);`
- `fd = open("/sys/log", O_WRONLY | O_APPEND | O_CREAT, 0600);`
- `if ((fd = open("tmpfile", O_WRONLY | O_CREAT | O_EXCL, 0666)) == -1)`

# fopen.c

---

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int main(int argc, char *argv[])
{
    int fd;
    if ((fd = open(argv[1], O_RDWR)) == -1)
        perror(argv[1]);
    printf("파일 %s 열기 성공\n", argv[1]);
    close(fd);
    exit(0);
}
```

# 커맨드라인 인수

- 커맨드라인에서 프로그램을 실행시키면서 인수를 전달할 때 사용
- 커맨드라인 인수를 사용하지 않을 때
  - 메인 함수 형태: `int main(void)`
  - 프로그램 실행 :     파일명.exe                   sample.exe  
                          파일명                   sample
- 커맨드라인 인수를 사용할 때
  - 메인 함수 형태: `int main(int argc, char *argv[])`
  - 프로그램 실행 :     파일명.exe 인수                   sample.exe abc  
                          파일명 인수1 인수2           sample 1 2  
   sample -w

# 커맨드라인 인수 예제

```
#include<stdio.h>

int main(int argc, char *argv[])
{
    int i;

    printf("argc=%d\n", argc);

    for (i = 0; i < argc; i++)
        printf("*argv[%d]=%s\n", i, argv[i]);

    return 0;
}
```

```
C:\#>cjuc lan.exe a b c
argc=4
*argv[0]=cjuc lan.exe
*argv[1]=a
*argv[2]=b
*argv[3]=c
```

# 파일 생성: creat()

- creat() 시스템 호출
  - path가 나타내는 파일을 생성하고 쓰기 전용으로 엮
  - 생성된 파일의 사용권한은 mode로 정함
  - 기존 파일이 있는 경우에는 그 내용을 삭제하고 엮
  - 다음 시스템 호출과 동일함  
`open(path, WRONLY | O_CREAT | O_TRUNC, mode);`

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

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

```
#include <fcntl.h>
```

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

파일 생성에 성공하면 파일 디스크립터를, 실패하면 -1을 리턴

# 파일 닫기: close()

---

- close() 시스템 호출은 fd(파일 디스크립터)가 나타내는 파일을 닫음

```
#include <unistd.h>
```

```
int close( int fd );
```

fd가 나타내는 파일을 닫음

성공하면 0, 실패하면 -1을 리턴함



# 데이터 읽기: read()

---

- read() 시스템 호출은 fd가 나타내는 파일에서 nbytes 만큼의 데이터를 읽고, 읽은 데이터는 buf에 저장함

```
#include <unistd.h>
```

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

파일 읽기에 성공하면 읽은 바이트 수, 파일 끝을 만나면 0,  
실패하면 -1을 리턴

## fsize.c

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#define BUFSIZE 512
int main(int argc, char *argv[])
{
    char buffer[BUFSIZE];
    int fd;
    ssize_t nread;
    long total = 0;
    if ((fd = open(argv[1], O_RDONLY))
        == -1)
        perror(argv[1]);

    while( (nread = read(fd, buffer,
                        BUFSIZE)) > 0)
        total += nread;
    close(fd);
    printf ("%s 파일 크기 : %ld 바이트\n", argv[1], total);
    exit(0);
}
```

# 데이터 쓰기: write()

---

- write() 시스템 호출은 buf에 있는 nbytes 만큼의 데이터를 fd가 나타내는 파일에 씀

```
#include <unistd.h>
```

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

파일에 쓰기를 성공하면 실제 쓰여진 바이트 수를 리턴하고,  
실패하면 -1을 리턴

## copy.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
main(int argc, char *argv[])
{
    int fd1, fd2, n;
    char buf[BUFSIZ];
    if (argc != 3) {
        fprintf(stderr, "사용법: %s file1\n", argv[0]);
        exit(1);
    }
    if ((fd1 = open(argv[1], O_RDONLY))
        == -1) {
        perror(argv[1]);
        exit(2);
    }
    if ((fd2 = open(argv[2], O_WRONLY |
        O_CREAT | O_TRUNC 0644)) == -1)
    {
        perror(argv[2]);
        exit(3);
    }
    while ((n=read(fd1, buf, BUFSIZ))>0)
        write(fd2, buf, n);
    exit(0);
}
```

# 파일 디스크립터 복제

- dup()과 dup2() 호출은 기존의 파일 디스크립터를 복제함

```
#include <unistd.h>
```

```
int dup(int oldfd);
```

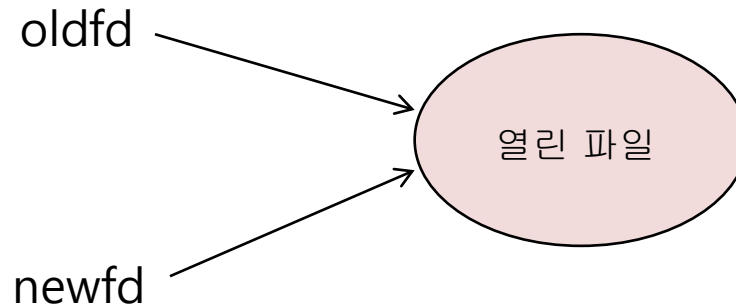
oldfd에 대한 복제본인 새로운 파일 디스크립터를 생성하여 반환함

```
int dup2(int oldfd, int newfd);
```

oldfd을 newfd에 복제하고 복제된 새로운 파일 디스크립터를 반환함

실패하면 -1을 반환함

- oldfd와 복제된 새로운 디스크립터는 하나의 파일을 공유함



## dup.c

```
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
int main()
{
    int fd, fd2;
    if((fd = creat("myfile", 0600)) == -1)
        perror("myfile");
    write(fd, "Hello! Linux", 12);
    fd2 = dup(fd);
    write(fd2, "Bye! Linux", 10);
    exit(0);
}
```

```
$ ./dup
$ cat myfile
Hello! LinuxBye! Linux
```

# 파일의 임의 접근

- 파일 위치 포인터(file position pointer)
  - 파일 위치 포인터는 파일 내에서 읽거나 쓸 위치인 현재 파일 위치(current file position)를 가리킴



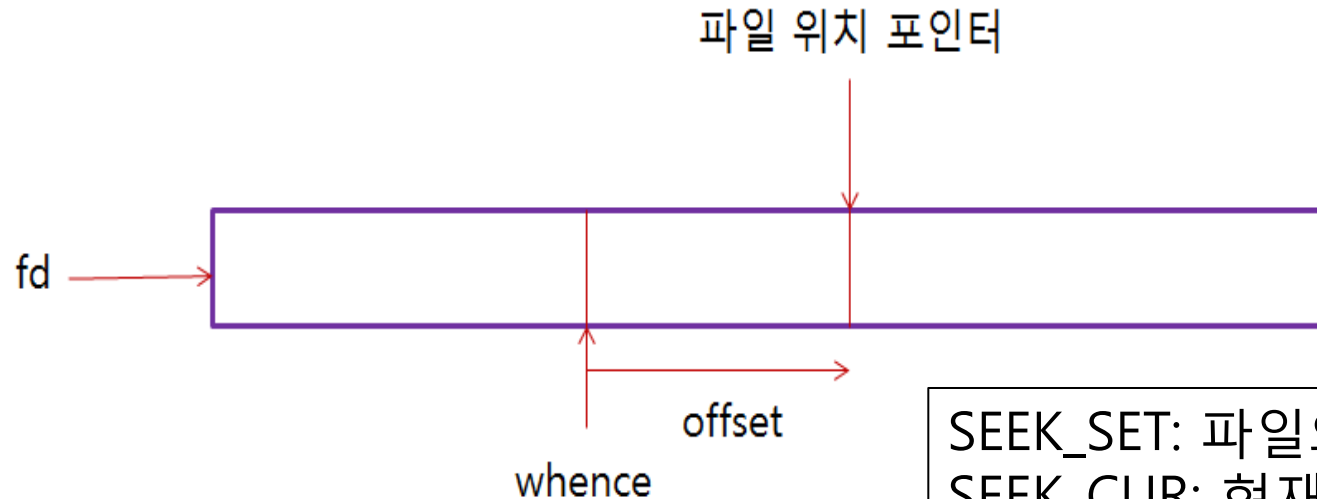
# 파일 위치 포인터의 이동: lseek()

- lseek() 시스템 호출
  - 임의의 위치로 파일 위치 포인터를 이동시킬 수 있음

```
#include <unistd.h>
```

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

이동에 성공하면 현재 위치를 리턴하고, 실패하면 -1을 리턴함



SEEK\_SET: 파일의 시작점  
SEEK\_CUR: 현재 파일 위치  
SEEK\_END: 파일의 끝



# 파일 위치 포인터의 이동 예

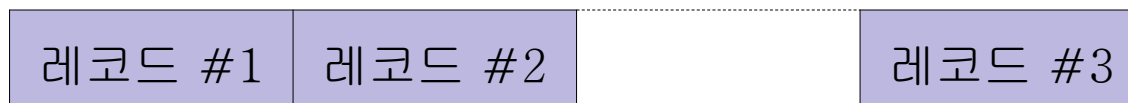
---

- 파일 위치 이동
  - `lseek(fd, 0L, SEEK_SET);`    파일 시작으로 이동(rewind)
  - `lseek(fd, 100L, SEEK_SET);`    파일 시작에서 100바이트 위치로 이동
  - `lseek(fd, 0L, SEEK_END);`    파일 끝으로 이동(append)
- 레코드 단위로 이동
  - `lseek(fd, n*sizeof(record), SEEK_SET);`    n+1번째 레코드 시작위치로 이동
  - `lseek(fd, sizeof(record), SEEK_CUR);`    다음 레코드 시작위치로 이동
  - `lseek(fd, -sizeof(record), SEEK_CUR);`    전 레코드 시작위치로 이동
- 파일끝 이후로 이동
  - `lseek(fd, sizeof(record), SEEK_END);`    파일 끝에서 한 레코드 다음 위치로 이동

# 레코드 저장 예

---

- `write(fd, &record1, sizeof(record));`
- `write(fd, &record2, sizeof(record));`
- `lseek(fd, sizeof(record), SEEK_END);`
- `write(fd, &record3, sizeof(record));`



## dbcreate.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include "student.h"
int main(int argc, char *argv[])
{
    int fd;
    struct student record;
    if (argc < 2) {
        fprintf(stderr, "사용법 : %s file\n",
            argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1],
        O_WRONLY|O_CREAT|O_EXCL, 0640))
        == -1) {
        perror(argv[1]);
        exit(2);
    }
    printf("%-9s %-8s %-4s\n", "학번", "
    이름", "점수");
    while (scanf("%d %s %d", &record.id,
        record.name, &record.score) == 3) {
        lseek(fd, (record.id - START_ID) *
            sizeof(record), SEEK_SET);
        write(fd, (char *) &record,
            sizeof(record));
    }
    close(fd);
    exit(0);
}
```

## student.h

```
#define MAX 24
#define START_ID 1401001
struct student {
    char name[MAX];
    int id;
    int score;
};
```

## dbquery.c (1)

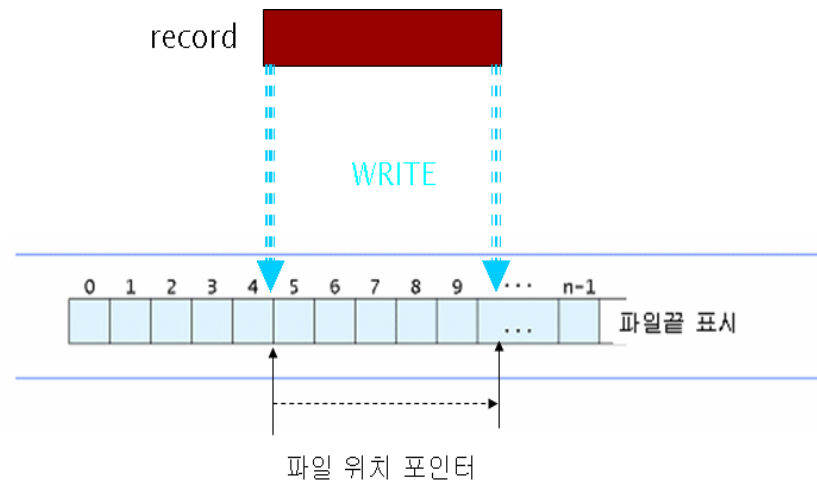
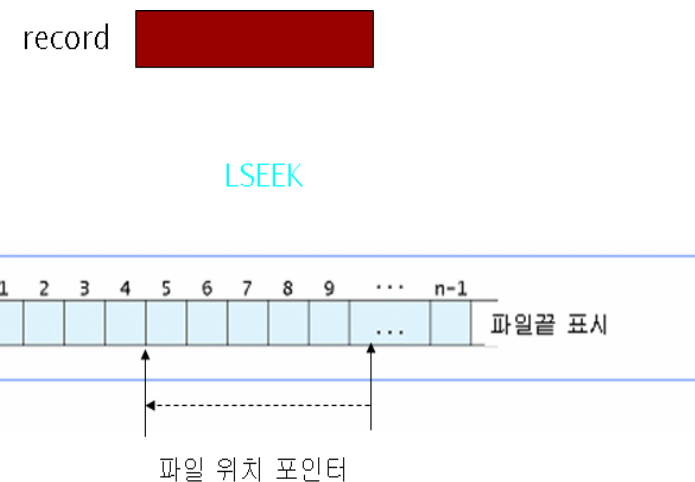
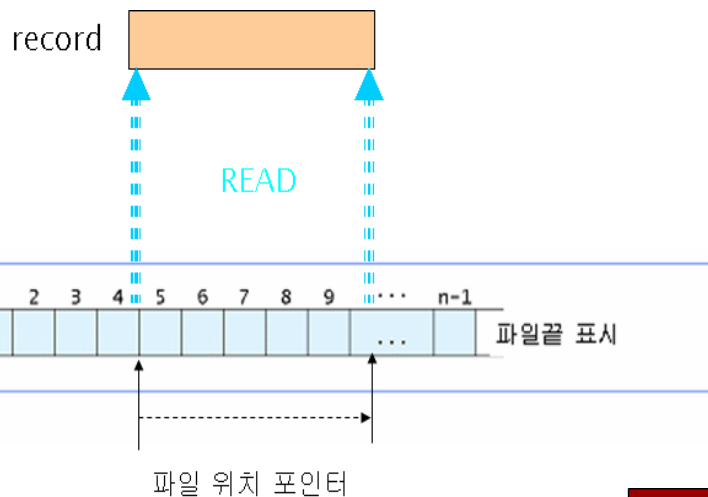
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include "student.h"
int main(int argc, char *argv[])
{
    int fd, id;
    struct student record;
    if (argc < 2) {
        fprintf(stderr, "사용법 : %s file\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_RDONLY)) == -1) {
        perror(argv[1]);
        exit(2);
    }
}
```

## dbquery.c (2)

```
do {
    printf("\n검색할 학생의 학번 입력:");
    if (scanf("%d", &id) == 1) {
        lseek(fd, (id-START_ID)*sizeof(record), SEEK_SET);
        if ((read(fd, (char *) &record, sizeof(record)) > 0) && (record.id != 0))
            printf("이름:%s\t 학번:%d\t 점수:%d\n", record.name, record.id,
                    record.score);
        else printf("레코드 %d 없음\n", id);
    } else printf("입력 오류");
    printf("계속하겠습니까?(Y/N)");
    scanf(" %c", &c);
} while (c=='Y');
close(fd);
exit(0);
}
```

# 레코드 수정 과정

1. 파일로부터 해당 레코드를 읽음
2. 이 레코드를 수정함
3. 수정된 레코드를 다시 파일 내의 원래 위치에 써야 함



## dbupdate.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include "student.h"
int main(int argc, char *argv[])
{
    int fd, id;
    char c;
    struct student record;
    if (argc < 2) {
        fprintf(stderr, "사용법 : %s file\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_RDWR)) == -1) {
        perror(argv[1]);
        exit(2);
    }
}
```



## dbupdate.c (2)

```
do {
    printf("수정할 학생의 학번 입력: ");
    if (scanf("%d", &id) == 1) {
        lseek(fd, (long) (id-START_ID)*sizeof(record), SEEK_SET);
        if ((read(fd, (char *) &record, sizeof(record)) > 0) && (record.id != 0)) {
            printf("학번:%8d\\t 이름:%4s\\t 점수:%4d\\n", record.id, record.name,
record.score);
            printf("새로운 점수: ");
            scanf("%d", &record.score);
            lseek(fd, (long) -sizeof(record), SEEK_CUR);
            write(fd, (char *) &record, sizeof(record));
        } else printf("레코드 %d 없음\\n", id);
    } else printf("입력오류\\n");
    printf("계속하겠습니까?(Y/N)");
    scanf(" %c",&c);
} while (c == 'Y');
close(fd);
exit(0);
}
```

Q&A