UNIT

06

# 저수준 파일 입출력

로봇SW 교육원

최상훈(shchoi82@gmail.com)

# 학습 목표

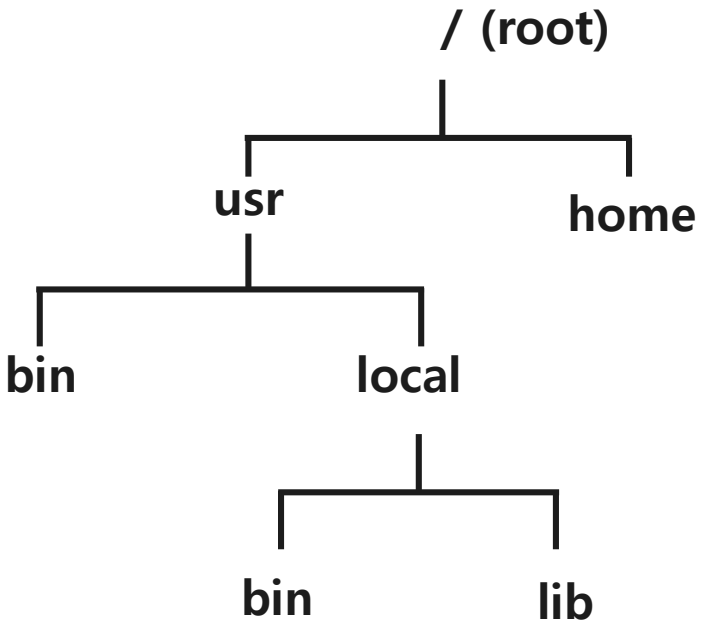


- 저수준 파일 입출력 함수
- 리눅스 파일 시스템의 이해
- 다양한 파일 입출력 실습을 통한 프로그램 능력 향상

# 파일 시스템

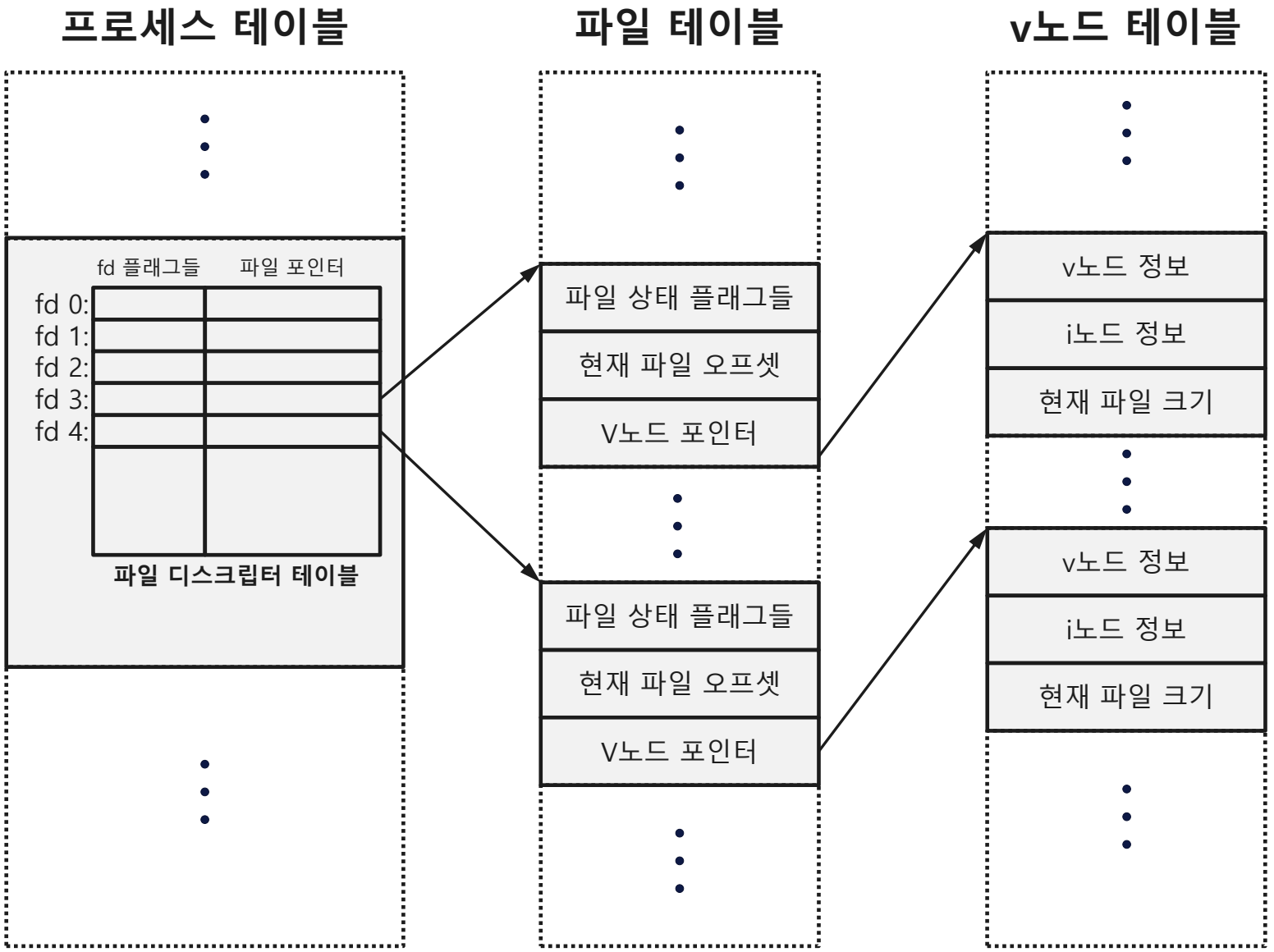


- 리눅스 파일 시스템의 특징
  - “ / ” 로 시작하는 트리 구조



# 열린 파일에 대한 커널 내부 자료구조





# 프로세스 테이블과 파일 테이블



- 프로세스 테이블
  - 시스템에 현재 실행중인 모든 프로세스에 대한 정보를 저장하고 있음
    - 프로세스 ID, 프로세스 권한 정보, 프로세스의 상태 등 프로세스에 대한 모든 정보들이 저장됨
  - 파일 디스크립터 테이블
    - 프로세스는 자신이 작업중인 모든 파일에 대한 파일 디스크립터 테이블 저장
    - 파일 디스크립터 (file descriptor)의 구성요소
      - 상태 플래그 : FD_CLOEXEC 옵션
      - 파일 포인터 : 파일 테이블의 항목을 가리키는 포인터

- 파일 테이블
  - 현재 시스템상에 열린 모든 파일에 대한 정보(시스템 전역)
  - 각 항목의 구성
    - 파일의 상태를 나타내는 플래그 (R, W, RW)
    - 현재 파일 오프셋
    - v-node를 가리키는 포인터

# v-node 테이블



- v-node **테이블**
  - **각 항은 하나의 파일과 대응**
  - **파일에 관련된 모든 정보 저장**
    - **파일의 종류**
    - **파일의 소유 정보**
    - **파일의 크기**
    - **파일의 실제 데이터 위치(예: 디스크상의 위치)**
    - **파일의 접근 권한**(rwxrwxrwx)
    - **파일의 시간정보**(a, m, c)
    - **파일의 하드링크 수**
    - **파일과 연관된 함수 포인터**

※ Linux**에는** v-node**가 없고 일반적** i-node**가 사용됨**

※ v-node**는 하나의 컴퓨터 시스템에서 여러 종류의 파일 시스템을 지원하기 위해 고안됨**

# 파일 디스크립터



- **파일 식별자**
  - **파일 디스크립터를 통해 파일관 과련된 작업**(I/O**처리 등**)**을 수행**
  - **음이 아닌 정수**
  - open,creat **시스템 콜 등을 통해 얻을 수 있음**

- **표준 파일 디스크립터**
  - **표준입력**, **표준출력**, **표준에러의 파일 서술자**
    - unistd.h**에 정의되어 있음**
    - **표준 입력** : STDIN_FILENO
    - **표준 출력** : STDOUT_FILENO
    - **표준 에러** : STDERR_FILENO
  - **표준** I/O **라이브러리의 표준 스트림**
    - **표준 파일 디스크립터와 연결**
    - stdio.h
    - **표준 입력** - **표준 입력 스트림** : stdin
    - **표준 출력** - **표준 출력 스트림** : stdout
    - **표준 에러** - **표준 에러 스트림** : stderr

# open



```
#include <fcntl.h>
int open ( const char *path, int oflag, … /* mode_t  mode */ );
                                        Returns : file descriptor if OK, -1 on error
```

- **기능** : **존재하는 파일을 열거나**, **새로운 파일을 생성**
- **리턴 값** : **성공하면 파일 디스크립터**, **실패하면** -1
- *path* : **열거나 생성하고자 하는 파일**
- *oflag* : **플래그**
- *mode* : **새로운 파일을 만드는 경우**, **접근 권한** (permission)

# open flag



- O_RDONLY - **읽기만 가능**
- O_WRONLY - **쓰기만 가능**
- O_RDWR - **읽기와 쓰기 모두 가능**
- O_EXEC - ?
- O_SEARCH - ?

※ **위의 다섯가지 모드 중 반드시 하나 선택 해야 함**(mutual exclusive)

- O_APPEND
  - **모든 쓰기 작업**(write)**을 파일의 끝에서 수행**
- O_CLOEXEC
  - file descriptor flag**에** FD_CLOEXEC**를 설정함**
- O_CREAT
  - **파일이 없을 경우 파일을 생성 (세 번째 인자 필요)**
- O_DIRECTORY
  - **디렉토리가 아니면** error**를 발생시킴**

# open flag



- O_EXCL
  - O_CREAT**와 같이 쓰이며, 파일이 있는 경우에** error**를 발생**
- O_TRUNC
  - **파일이 있는 경우에 기존 파일의 내용을 지움**

- **기타**
  - O_NOCTTY
  - O_NOFOLLOW
  - O_NONBLOCK
  - O_SYNC
  - O_TTY_INIT
  - O_DYSNC
  - O_RSYNC

# open



- open**에 사용 가능한 옵션**

  /usr/include/bits/fcntl.h

```
#define O_ACCMODE       0003
#define O_RDONLY        00
#define O_WRONLY        01
#define O_RDWR          02
#define O_CREAT         0100
#define O_EXCL          0200
#define O_NOCTTY        0400
#define O_TRUNC         01000
#define O_APPEND        02000
#define O_NONBLOCK      04000
#define O_NDELAY        O_NONBLOCK
#define O_SYNC          04010000
#define O_FSYNC         O_SYNC
#define O_ASYNC         020000
```

# close



```
#include <unistd.h>

int close ( int  filedes );

                                                    Returns : 0 if OK, -1 on error
```

- **기능**: **열려진 파일을 닫음**
- **리턴 값**: **성공하면** 0, **실패하면** -1
- *filedes* : **닫고자 하는 파일의 파일 디스크립터**
- **파일을 닫지 않더라도 프로세스가 종료되면 모든 열린 파일들을 자동적으로 닫음**

# 실습1:open



- open **함수를 이용한 파일 생성**
- **파일명** : openEx1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#define PERM 0644

int main()
{
    int fd;
    char *szNewFilePath = "./newFile";

    if((fd = open(szNewFilePath, O_WRONLY | O_CREAT, PERM)) == -1){
        fprintf(stderr, "Couldn't open... : filePath:%s", szNewFilePath);
        exit(1);
    }

    close(fd);
    return 0;
}
```

# 실습2:open



- open **함수를 이용한 파일 생성**(with O_EXCL)
- **파일명** : openEx2.c

```
#include<stdio.h>
#include<fcntl.h>
#include<stdlib.h>
#include<unistd.h>

int main()
{
    int fd;
    char *szNewFilePath = "./newFile";

    if((fd = open(szNewFilePath, O_RDWR | O_CREAT | O_EXCL | O_TRUNC , 0644)) == -1){
        fprintf(stderr, "Couldn't open.. : filePath=%s\n", szNewFilePath);
        exit(1);
    }

    close(fd);
    return 0;
}
```

# 실습3:open



- open **함수의** O_TRUNC
- **파일명** : openEx3.c

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

int main(void)
{
    int fd;

    if((fd = open("./file1", O_RDONLY | O_TRUNC)) == -1){
        fprintf(stderr, "Couldn't open.. : filePath=%s\n", szFilePath);
        exit(1);
    }

    printf("fd:%d\n", fd);

    close(fd);
    return 0;
}
```

# creat



```
#include <fcntl.h>
int creat ( const char *path,  mode_t  mode );
          Returns : file descriptor opened for write-only if OK, -1 on error
```

- **기능**: **새로운 파일을 생성**
- **리턴 값**: **성공하면 파일 디스크립터**, **실패하면** -1
- *path* : **생성하고자 하는 파일의 이름**
- *mode* : **접근 권한**

- open **함수와 비교**
  - O_WRONLY | O_CREAT | O_TRUNC **플래그 적용한 것과 동일함**

# 실습4:creat



- creat **함수를 이용한 파일 생성**
- **파일명** : creatEx1.c

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<fcntl.h>
#define PERM 0644

int main()
{
    int fd;
    char* szNewFilePath = "./newFile";
    if((fd = creat(szNewFilePath, PERM)) == -1){
        fprintf(stderr, "Couldn't creat : filePath=%s", szNewFilePath);
        exit(1);
    }
    close(fd);
    return 0;
}
```

# read



```
#include <unistd.h>
ssize_t  read (int filedes,  void *buf, size_t nbytes);
```

- **기능**: **파일로부터 데이터를 읽어 들이는 함수** (**파일** offset **증가**)
- **리턴 값**:
  - **성공하면**, **버퍼로 읽어 들인 데이터의 바이트 수**
  - **파일의 끝을 만나면**, 0
  - **실패하면**, -1
- *buff* : **읽어 들인 데이터를 저장하는 버퍼 공간**
- *nbytes* : **읽어 들일 데이터의 최대 바이트의 수**

# write



```
#include <unistd.h>
ssize_t  write (int filedes,  const void *buf, size_t nbytes);
```

- **기능**: **지정된 파일에 데이터를 쓰는 함수** (**파일** offset **증가**)
- **리턴 값**:
  - **성공하면**, **파일에 쓴  데이터의 바이트 수**
  - **실패하면**, -1
- *buff* : **쓸 데이터를 저장하고 있는 버퍼 공간**
- *nbytes* : **쓸 데이터의 바이트의 수**
- write **수행 시 에러가 발생하는 경우**
  - **파일 시스템이 가득 찬 경우**
  - **파일의 크기가 한계 값을 초과 하는 경우**

# lseek



- 모든 열린 파일에는 “현재 파일 오프셋” 이 존재
  - 파일테이블에 저장됨
  - 파일의 처음부터 byte단위로 계산한 정수값
  - 파일이 열릴때, 0으로 초기화됨
  - lseek

```
#include <unistd.h>

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

- 기능: 파일의 현재 파일 오프셋 을 명시적으로 변경
- 리턴 값: 성공하면 이동한 지점의 오프셋, 실패하면 -1
- *whence* : 오프셋을 옮기기 위한 기준점
  - SEEK_SET : 파일의 처음 시작 부분
  - SEEK_CUR : 현재 파일 오프셋
  - SEEK_END : 파일의 재일 마지막 부분
- *offset* : 기준점에서의 상대적인 거리 (byte 단위)
  - SEEK_CUR, SEEK_END 와 같이 쓰일 때는 음수도 허용

# 실습5:표준 파일 디스크립터



- **표준 파일 디스크립터 입출력**
- **파일명** : stdfd.c

```
#include<stdio.h>
#include<unistd.h>
#define BUFFSIZE    4096

int main(void)
{
    int n;
    char    buf[BUFFSIZE];

    while ((n = read(STDIN_FILENO, buf, BUFFSIZE)) > 0)
        if (write(STDOUT_FILENO, buf, n) != n){
            printf("write error\n");
            return 1;
        }

    if (n < 0){
        printf("read error\n");
        return 1;
    }
    return 0;
}
```

```
$ ./stdfd.out > data2
hello world[Ctrl+d][Ctrl+d]
$ cat data2
hello world
$ ./stdfd.out < data2 > data2.copy
$ cat data2
hello world
$
```

# 실습6:read



- read **함수 기본예제**
- **파일명** : readEx1.c

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
int main()
{
    int fd;
    ssize_t nRead;
    char *szFilePath = "./test2.txt";
    char buf[1024];
    if((fd = open(szFilePath, O_RDONLY)) == -1){
        printf("Couldn't open.. : FilePath=%s\n", szFilePath);
        return 1;
    }
    nRead = read(fd, buf, sizeof(buf));
    if(nRead == -1){
        printf("Couldn't read...");
        return 1;
    }
    printf("nRead = %ud\n", nRead);
    close(fd);
    return 0;
}
```

# 실습7:파일 복사



- **파일 복사**
- **파일명** : mycp.c

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<string.h>
#include<unistd.h>

#define PERM 0644

int main(int argc, char* argv[])
{
    int nSrcFd,nDstFd;
    ssize_t nRead = 0;
    ssize_t nWrite = 0;
    char buf[1024] = {0};
    char* szSrcFilePath = argv[1];
    char* szDstFilePath = argv[2];
    if((nSrcFd = open(szSrcFilePath, O_RDONLY)) == -1)
    {
        printf("Couldn't open : filePath=%s", szSrcFilePath);
        exit(0);
    }
```

# 실습8: 파일 복사



```
    if((nDstFd = creat(szDstFilePath, PERM)) == -1)
    {
        printf("Couldn't creat : filePath=%s", szDstFilePath);

        exit(0);
    }

    while((nRead = read(nSrcFd, buf, 1024)) > 0)
    {
        if(write(nDstFd,buf,nRead) < nRead)
        {
            printf("Couldn't write");
            close(nDstFd);
            close(nSrcFd);
            exit(0);
        }
        nWrite += nRead;
        memset(buf,0,sizeof buf);
    }
    close(nDstFd);
    close(nSrcFd);
    if(nRead == -1)
    {
        printf("Couldn't read");
        exit(0);
    }
    printf("nWrite = %d", nWrite);
    return 1;
}
```

# 실습9:lseek



- lseek **함수를 이용해 파일의 크기 구하기**
- **파일명** : lseekEx1.c

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

int main(void)
{
    char *fname = "test.txt";
    int fd;
    off_t fsize;

    if((fd = open(fname, O_RDONLY)) == -1){
        printf("open error\n");
        return 1;
    }

    fsize = lseek(fd, 0, SEEK_END);
    printf("The size of <%s> is %lu bytes.\n", fname, fsize);
    return 0;
}
```

```
$ ./lseekEx1
The size of <test.txt> is 10 bytes.
```

# 실습10-1:lseek



- **파일의 구멍**
- **파일명** : lseekEx2.c

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

char buf1[] = "abcdefghij";
char buf2[] = "ABCDEFGHIJ";

int main(void)
{
    int fd;
    if((fd = creat("file.hole", 0622)) < 0)
        fprintf(stderr, "creat error");

    if(write(fd, buf1, 10) != 10)
        fprintf(stderr, "buf1 write error");
    if(lseek(fd, 16384, SEEK_SET) == -1)
        fprintf(stderr, "lseek error");

    if(write(fd, buf2, 10) != 10)
        fprintf(stderr, "buf2 write error");

    return 0;
}
```

# 실습10-2:lseek



```
$ ./lseekEx2
$ ls -l file.hole
-rw------- 1 advc advc 16394 2011-07-31 21:01 file.hole
$ od -c file.hole
0000000   a   b   c   d   e   f   g   h   i   j  \0  \0  \0  \0  \0  \0
0000020  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0040000   A   B   C   D   E   F   G   H   I   J
0040012
$ cat < file.hole > file.hole.copy
$ ls -ls file.hole file.hole.copy
 8 -rw------- 1 advc advc 16394 2011-07-31 21:01 file.hole
20 -rw------- 1 advc advc 16394 2011-07-31 21:03 file.hole.copy
```

# 실습11-1:read



- **같은 파일 읽기 예제**
- **파일명** : readEx2.c

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

int main(void)
{
    char *fname = "data";
    int fd1, fd2, cnt;
    char buf[30];

    fd1 = open(fname, O_RDONLY);
    fd2 = open(fname, O_RDONLY);
    if(fd1 == -1 || fd2 == -1) {
        printf("open error\n");
        return 1;
    }

    cnt = read(fd1, buf, 12);
    buf[cnt] = '\0';
    printf("fd1's first printf : %s\n", buf);
```

# 실습11-2:read



```
    lseek(fd1, 1, SEEK_CUR);
    cnt = read(fd1, buf, 12);
    buf[cnt] = '\0';
    printf("fd1's second printf : %s\n", buf);

    cnt = read(fd2, buf, 12);
    buf[cnt] = '\0';
    printf("fd2's first printf : %s\n", buf);

    lseek(fd2, 1, SEEK_CUR);
    cnt = read(fd2, buf, 12);
    buf[cnt] = '\0';
    printf("fd2's second printf : %s\n", buf);

    return 0;
}
```

**<실습하기전 data 파일 생성>**

```
$ cat > data
Hello, UNIX!
How are you?
[Ctrl + D]
$ cat data
Hello, UNIX!
How are you?
$
```

```
$ ./readEx2
fd1's first printf : Hello, UNIX!
fd1's second printf : How are you?
fd2's first printf : Hello, UNIX!
fd2's second printf : How are you?
```

# 파일 오프셋과 파일 I/O



- write 호출시 동작
  - 기록된 바이트 수만큼 현재 파일 오프셋 증가
  - 현재 파일 오프셋이 파일 크기를 넘게되면 i-node 테이블 항목의 파일크기에 설정됨

- open 함수 호출시 O_APPEND를 지정한 경우
  - '파일 테이블' 의 '파일 상태 플래그'에 설정됨
  - write 호출시 '현재 파일 오프셋'이 i-node 테이블 항목의 파일크기에 설정됨

- lseek 함수 호출로 파일의 끝으로 이동
  - 현재 파일 오프셋을 단순히 i-node 테이블 항목의 파일크기로 설정
  - I/O연산이 일어나지 않음

# 실습12:파일 오프셋



- O_APPEND **플래그**
- **파일명** : openEx4.c

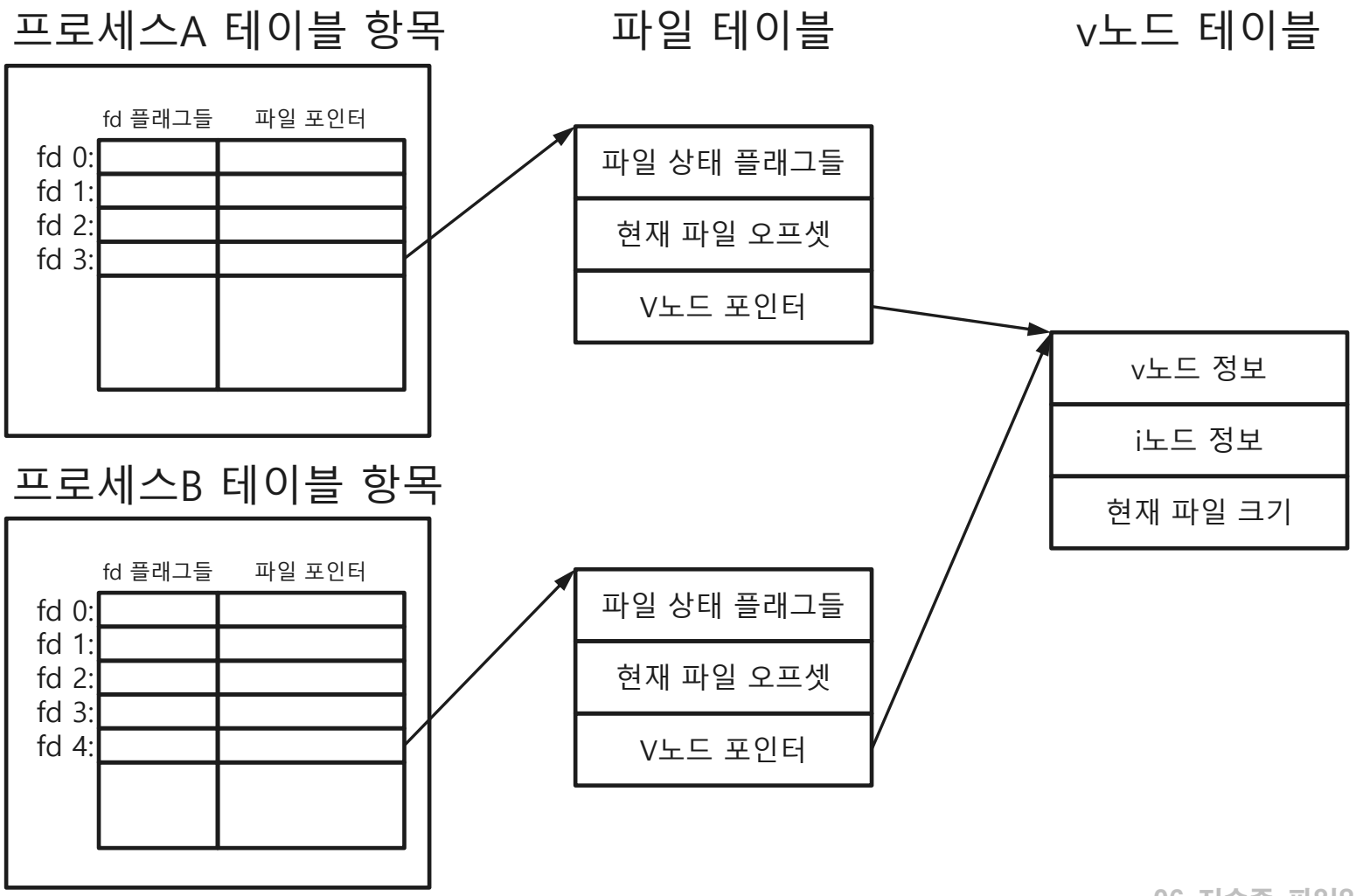
```
$ cat > data
1234567890[Ctrl+d][Ctrl+d]$ ./openEx4
before write offset:0
after write offset:14
file size:14
$
```

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
 int main(void)
{
    int fd;
    long offset;
    if((fd = open("./data", O_WRONLY | O_APPEND)) == -1){
        fprintf(stderr, "open error\n");
        exit(1);
    }
    offset = lseek(fd, 0, SEEK_CUR);
    printf("before write offset:%ld\n", offset);
    if(write(fd, "7777", 4) != 4){
        fprintf(stderr, "write error\n");
        exit(1);
    }
    offset = lseek(fd, 0, SEEK_CUR);
    printf("after write offset:%ld\n", offset);
    offset = lseek(fd, 0, SEEK_END);
    printf("file size:%ld\n", offset);
    close(fd);
    exit(0);
}
```

# 파일의 공유



- UNIX 는 multi-user/multi-tasking **시스템이기 때문에**
  **다른 프로세스에 의해 같은 파일을 동시에 접근할 있음**



# 원자적 연산



- **원자적**(atomic) **연산**
  - **연산 수행중에 다른연산에 의해 가로채지지 않는것**
  - **모두 수행되거나 또는 모두 수행되지 않거나**

- open**시** O_APPEND **플래그 지원되지 않는다면**?

```
if (lseek(fd, 0L, SEEK_END) == -1) {          /* position to EOF */
    fprintf(stderr, "lseek error\n");
    exit(1);
}
<<<<<<<<<<<<<<<<  >>>>>>>>>>>>>>>>
if (write(fd, buf, 100) != 100) {             /* ...and write */
    fprintf(stderr, "write error\n");
    exit(1);
}
```

  - lseek**와** write**사이에 다른 프로세스가 실행될 가능성 이 있음**
  - semaphore, mutex**등을 사용해야 함**

# 실습13:원자적 연산



- **원자적 연산**
- **파일명** : atomicEx1.c

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

int main(void)
{
    char *fname = "shared";
    int fd;
    off_t curOffset;
    long i;

    if((fd = open(fname, O_WRONLY)) == -1){
        printf("open error\n");
        return 1;
    }

    for(i = 0 ; i < 100000 ; i++){
        curOffset = lseek(fd, 0, SEEK_END);
        if(write(fd, "0", 1) != 1)
            exit(1);
        printf("%ld\r", i);
    }
    return 0;
}
```

# 실습14:원자적 연산



- **원자적 연산**
- **파일명** : atomicEx2.c

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

int main(void)
{
    char *fname = "shared";
    int fd;
    off_t curOffset;
    long i;

    if((fd = open(fname, O_WRONLY | O_APPEND)) == -1){
        printf("open error\n");
        return 1;
    }

    for(i = 0 ; i < 100000 ; i++){
        if(write(fd, "0", 1) != 1)
            exit(1);
        printf("%ld\r", i);
    }
    return 0;
}
```

# 실습15-1:구조체 파일 출력



- **구조체 파일** write
- **파일명** : structureWrite.c

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

typedef struct TAG_GPIOCTL_DATA{
    int pinNo;
    int onTime;
    int offTime;
    int repeat;
}GPIOCTL_DATA;

int main(void)
{
    char *fname = "gpio_data";
    int fd;
    const int DATA_SIZE = sizeof(GPIOCTL_DATA);

    GPIOCTL_DATA data;
```

# 실습15-2:구조체 파일 출력



```
    printf("pinNo:");   scanf("%d", &data.pinNo);
    printf("onTime:");  scanf("%d", &data.onTime);
    printf("offTime:"); scanf("%d", &data.offTime);
    printf("repeat:");  scanf("%d", &data.repeat);

    if((fd = open(fname, O_WRONLY | O_CREAT | O_TRUNC, 0644)) == -1){
        fprintf(stderr, "open error\n");
        return 1;
    }
    if(write(fd, &data, DATA_SIZE) != DATA_SIZE){
        fprintf(stderr, "write error\n");
        exit(1);
    }

    return 0;
}
```

# 실습16-1:구조체 파일 입력



- **구조체 파일** read
- **파일명** : structureRead.c

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

typedef struct TAG_GPIOCTL_DATA{
    int pinNo;
    int onTime;
    int offTime;
    int repeat;
}GPIOCTL_DATA;

int main(void)
{
    char *fname = "gpio_data";
    int fd;
    const int DATA_SIZE = sizeof(GPIOCTL_DATA);

    GPIOCTL_DATA data;
```

# 실습16-2:구조체 파일 입력



```
    if((fd = open(fname, O_RDONLY)) == -1){
        fprintf(stderr, "open error\n");
        return 1;
    }
    if(read(fd, &data, DATA_SIZE) != DATA_SIZE){
        fprintf(stderr, "read error\n");
        exit(1);
    }
    printf("pinNo:%d\n", data.pinNo);
    printf("onTime:%d\n", data.onTime);
    printf("offTime:%d\n", data.offTime);
    printf("repeat:%d\n", data.repeat);

    return 0;
}
```

# dup, dup2



```
#include <unistd.h>
int  dup (int filedes);
int  dup2 (int filedes , int filedes2);
```

- **기능**: **사용 중인 파일 디스크립터를 복사**
- **리턴 값**: **성공하면 할당 받은 파일 디스크립터 번호**, **실패하면** -1
  - dup( ) **함수는 할당 가능한 가장 작은 번호를 리턴**
  - dup2( ) **함수는** filedes2**를 리턴**, fd1**를** fd2**로 복사함**
- **파일 디스크립터 항목의 플래그** (FD_CLOEXEC)**는 초기값** (0) **설정됨**

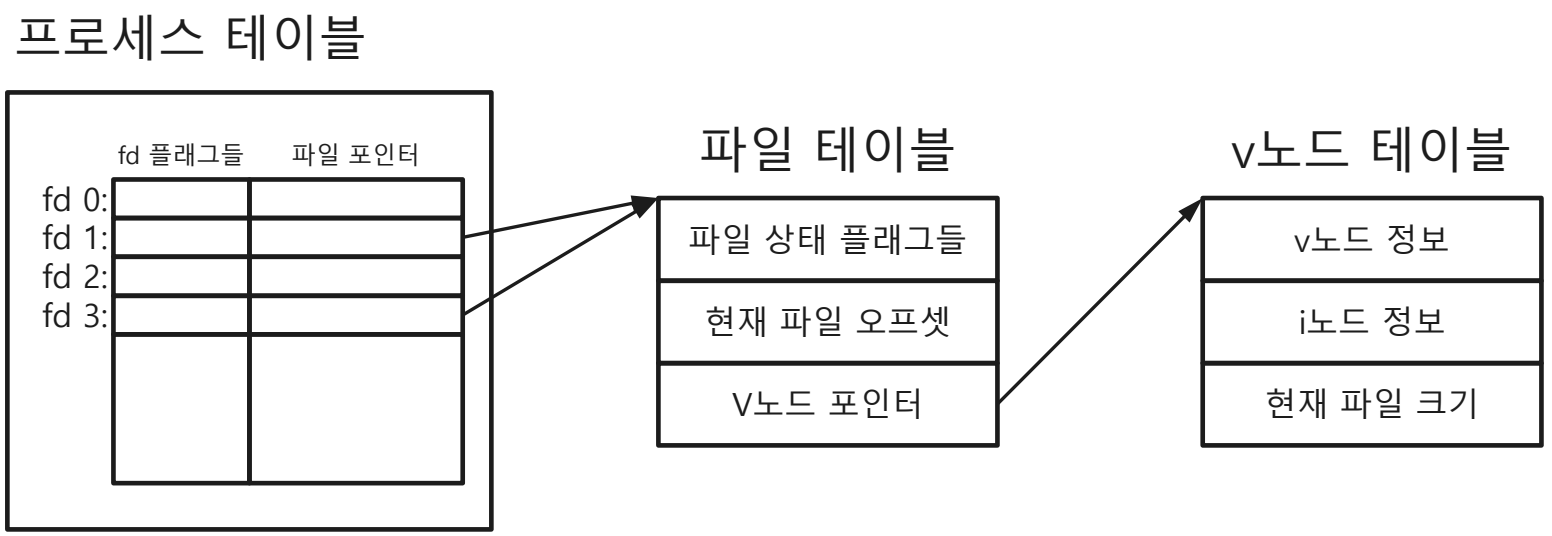
# dup, dup2



- **동일한 '파일 테이블' 항목을 공유**

  ex) newfd = dup(1);

  newfd = dup2(1,3);



# 실습17:dup



- **파일명** : dupEx1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

int main(void)
{
    char *fname = "data2";
    int fd1, fd2, cnt;
    char buf[30];

    if((fd1 = open(fname, O_RDONLY)) == -1){
        printf("open error/n");
        return 1;
    }

    fd2 = dup(fd1);
    cnt = read(fd1, buf, 4);
    buf[cnt] = '\0';
    printf("fd1's printf : %s\n", buf);
    printf("current file offeset:%ld\n", lseek(fd1, 0, SEEK_CUR));

    cnt = read(fd2, buf, 4);
    buf[cnt] = '\0';
    printf("fd2's printf : %s\n", buf);

    return 0;
}
```

```
$ cat > data2
abcd0123[Ctrl + D][Ctrl + D]
$ ./dupEx1
fd1's printf : abcd
current file offeset:4
fd2's printf : 0123
$
```

# 실습18:dup



- **파일명** : dupEx2.c

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
int main(void)
{
    char *fname = "data3";
    int fd1, fd2;

    if((fd1 = creat(fname, 0666)) < 0) {
        printf("creat error\n");
        return 1;
    }
    printf("First printf is on the screen.\n");
    fd2 = dup2(fd1,1);
    printf("Second printf is in this file.\n");
    printf("fd2:%d\n", fd2);
    return 0;
}
```

```
$ ./dupEx2
First printf is on the screen.
$ cat data3
Second printf is in this file.
fd2:1
```

# fcntl



```
#include <fcntl.h>
int  fcntl (int filedes, int cmd, … /* int arg */ );
```

- **기능** : **이미 열려져 있는 파일에 대한 속성을 알아내거나 변경함**
  cmd**에 따라 기능이 달라짐**

- **리턴 값**: **성공인 경우에** cmd **값에 따라 다름** (**다음 슬라이드 참고**),
  **실패하면** -1

# fcntl 명령의 종류



- **명령의 종류** (cmds)
  - F_DUPFD : **파일 디스크립터를 복사. 세 번째 인수 보다 크거나 같은 값 중 가장 작은 미사용의 값을 리턴한다.**
  - F_GETFD : **파일 디스크립터의 플래그를 반환** (FD_CLOEXEC)
  - F_SETFD : **파일 디스크립터의 플래그를 설정**
  - F_GETFL : **파일 테이블에 저장되어 있는 파일 상태 플래그를 반환**
  - F_SETFL : **파일 상태 플래그의 설정** (O_APPEND, O_NONBLOCK, O_SYNC **등을 지정**)
  - F_GETLK/F_SEKLK : **레코드 자물쇠를 조회/설정함**(14**장 고급** I/O)

# 실습19-1:fcntl (파일명 : fcntlEx.c)



```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
int main(void)
{
    char *fname = "data";
    int fd;
    int flag;

    if((fd = open(fname, O_RDWR | O_APPEND)) < 0) {
        printf("open error\n");
        return 1;
    }
    flag = fcntl(fd, F_GETFL, 0);

    switch(flag & O_ACCMODE){
        case O_RDONLY:
            printf("O_RDONLY flag is set.\n");
            break;
        case O_WRONLY:
            printf("O_WRONLY flag is set.\n");
            break;
        case O_RDWR:
            printf("O_RDWR flag is set.\n");
            break;
        default:
            printf("unknown accemode");
            break;
    }
```

# 실습19-2:fcntl



```
    if (flag & O_APPEND)
        printf("fd: O_APPEND flag is set. \n");
    else
        printf("fd: O_APPEND flag is NOT set. \n");

    flag = fcntl(fd, F_GETFD, 0);
    if (flag & FD_CLOEXEC)
        printf("fd: FD_CLOEXEC flag is set. \n");
    else
        printf("fd: FD_CLOEXEC flag is NOT set. \n");

    fcntl(fd, F_SETFD, FD_CLOEXEC);
    flag = fcntl(fd, F_GETFD, 0);
    if (flag & FD_CLOEXEC)
        printf("fd: FD_CLOEXEC flag is set. \n");
    else
        printf("fd: FD_CLOEXEC flag is NOT set. \n");

    return 0;
}
```

```
$ ./fcntlEx.out
O_RDWR flag is set.
fd: O_APPEND flag is set.
fd: FD_CLOEXEC flag is NOT set.
fd: FD_CLOEXEC flag is set.
$
```

# 참고자료 : 열수 있는 파일의 개수



- **프로세스가 열 수 있는 파일의 최대 개수**
- **파일명** : maxOpen.c

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

int main(void)
{
    printf("OPEN_MAX:%ld\n", sysconf(_SC_OPEN_MAX));
    return 1;
}
```

# 참고자료 : 파일명과 경로명의 최대길이



- **파일의 경로 최대 길이** : PATH_MAX
- **파일의 이름 최대 길이** : NAME_MAX

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

int main(void)
{
    printf("PATH_MAX:%ld\n", pathconf("/",_PC_PATH_MAX));
    printf("NAME_MAX:%ld\n", pathconf("/",_PC_NAME_MAX));

    printf("PATH_MAX:%d\n", PATH_MAX);
    printf("NAME_MAX:%d\n", NAME_MAX);
    return 0;
}
```

# 참고자료 : 파일명과 경로명의 최대길이



- **파일의 경로 최대 길이** : PATH_MAX
- **파일의 이름 최대 길이** : NAME_MAX

```
#include<stdio.h>
#include<errno.h>
#include<unistd.h>
#include<fcntl.h>
extern int errno;
int main(void)
{
    if(_POSIX_NO_TRUNC){
        printf("PATH_MAX:%ld\n", pathconf("/",_PC_PATH_MAX));
        printf("NAME_MAX:%ld\n", pathconf("/",_PC_NAME_MAX));
    }

if(open("0123456789012345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789\
0123456789012345678901234567890123456789012345678901234567890123456789012345
6789012345678901234567890123456789\
0123456789012345678901234567890123456789012345678901234567890123456789012345
6789012345678901234567890123456789.data",O_RDWR | O_CREAT, 0777) == -1){
        if(errno == ENAMETOOLONG)
            perror("");
    }
    return 1;
}
```

# 미션



- **실습** 15 **프로그램은** gpio_data **파일을 읽고 내용을 출력한다.**
  gpio_data **파일을 읽어 실제** LED**를 제어하도록 프로그램을 수정하시오.**
  (wiringPi **라이브러리 사용**)
  - pinNo : GPIO **핀번호**
  - onTime : GPIO **핀이** On **상태인 시간(초)**
  - offTime : GPIO **핀이** Off **상태인 시간(초)**
  - repeat : **반복 회수**