

이름 있는 파이프[1]

□ 이름 있는 파이프

- 부모-자식간이 아닌 독립적인 프로세스 간에 통신하기 위해서는 이름 있는 파이프 사용
- 이름 있는 파이프는 FIFO라고도 함
- FIFO로 사용할 특수파일을 명령이나 함수로 먼저 생성해야 함

□ 명령으로 FIFO 파일 생성

- **mknod 명령**

mknod 파일명 p

```
# mknod HAN_FIFO p
# ls -l HAN_FIFO
prw-r--r--  1 root      other          0  2월 13일  12:21 HAN_FIFO
# ls -l
HAN_FIFO |
```

FIFO 표시

- **mkfifo명령**

/usr/bin/mkfifo [-m mode] path...

```
# mkfifo -m 0644 BIT_FIFO
# ls -l BIT_FIFO
prw-r--r--  1 root      other          0  2월 13일  12:28 BIT_FIFO
```

이름 있는 파이프[2]

□ 함수로 특수파일 생성

▪ 특수파일생성: mknod(2)

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

```
int mknod(const char *path, mode_t mode, dev_t dev);
```

▪ mode : 생성할 특수파일의 종류 지정

- S_IFIFO : FIFO 특수 파일
- S_IFCHAR : 문자장치 특수 파일
- S_IFDIR : 디렉토리
- S_IFBLK : 블록장치 특수파일
- S_IFREG : 일반파일

```
if (mknod( "TEST_FIFO" , S_IFIFO|0644, 0) == -1) {
```

▪ FIFO 파일 생성: mkfifo(3)

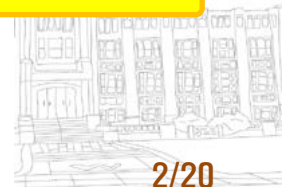
```
#include <sys/types.h>
```

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

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

- mode : 접근권한 지정

```
if (mkfifo( "TEST_FIFO" , 0644) == -1) {
```



[예제 9-7] FIFO로 데이터 주고 받기 - 서버(server.c)

```
...
08 int main(void) {
09     int pd, n;
10     char msg[] = "Hello, FIFO";
11
12     printf("Server =====\n");
13
14     if (mkfifo("./HAN-FIFO", 0666) == -1) {
15         perror("mkfifo");
16         exit(1);
17     }
18
19     if ((pd = open("./HAN-FIFO", O_WRONLY)) == -1) {
20         perror("open");
21         exit(1);
22     }
23
24     printf("To Client : %s\n", msg);
25
26     n = write(pd, msg, strlen(msg)+1);
27     if (n == -1) {
28         perror("write");
29         exit(1);
30     }
31     close(pd);
32
33     return 0;
34 }
```

FIFO 파일 생성

FIFO 파일 쓰기모드로 열기

FIFO 파일에 문자열 출력

[예제 9-7] FIFO로 데이터 주고 받기 -클라이언트(client.c)

```
...
08 int main(void) {
09     int pd, n;
10     char inmsg[80];
11
12     if ((pd = open("./HAN-FIFO", O_RDONLY)) == -1) {
13         perror("open");
14         exit(1);
15     }
16
17     printf("Client =====\n");
18     write(1, "From Server :", 13);
19
20     while ((n=read(pd, inmsg, 80)) > 0)
21         write(1, inmsg, n);
22
23     if (n == -1) {
24         perror("read");
25         exit(1);
26     }
27
28     write(1, "\n", 1);
29     close(pd);
30
31     return 0;
32 }
```

서버측에서 생성한 FIFO 파일열기

서버가 보낸
데이터 읽기

```
# server
Server =====
To Client : Hello, FIFO
#
```

```
# client
Client =====
From Server :Hello, FIFO
#
```

레코드 록킹(advisory locking)

- multiple reader, single writer
- 읽기 록: 다른 프로세스들이 쓰기 록을 적용하지 못하게 함. 여러 프로세스들이 같은 구역에 동시에 읽기 록을 할 수 있음
- 쓰기 록: 다른 프로세스들이 그 구역에 읽거나 쓰기 록을 할 수 없도록 함
파일의 한 구역에는 한 순간에 하나의 쓰기 록 만이 존재할 수 있음

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

```
int fcntl (int filedes, int cmd, struct flock *ldata);
```

- filedes: 유효한 개방된 파일기술자. 읽기 록인 경우 O_RDONLY나 O_RDWR로 개방. 쓰기 록을 위해서는 O_WRONLY나 O_RDWR로 개방
- cmd :
 - . F_GETLK : ldata를 통해 전달된 록 정보를 획득
 - . F_SETLK : 파일에 록을 적용하고, 불가능하면 즉시 -1로 돌아옴
 - . F_SETLKW : 파일에 록을 적용하고, 이것이 만약 다른 프로세스가 소유하고 있으면 수면



레코드 록킹

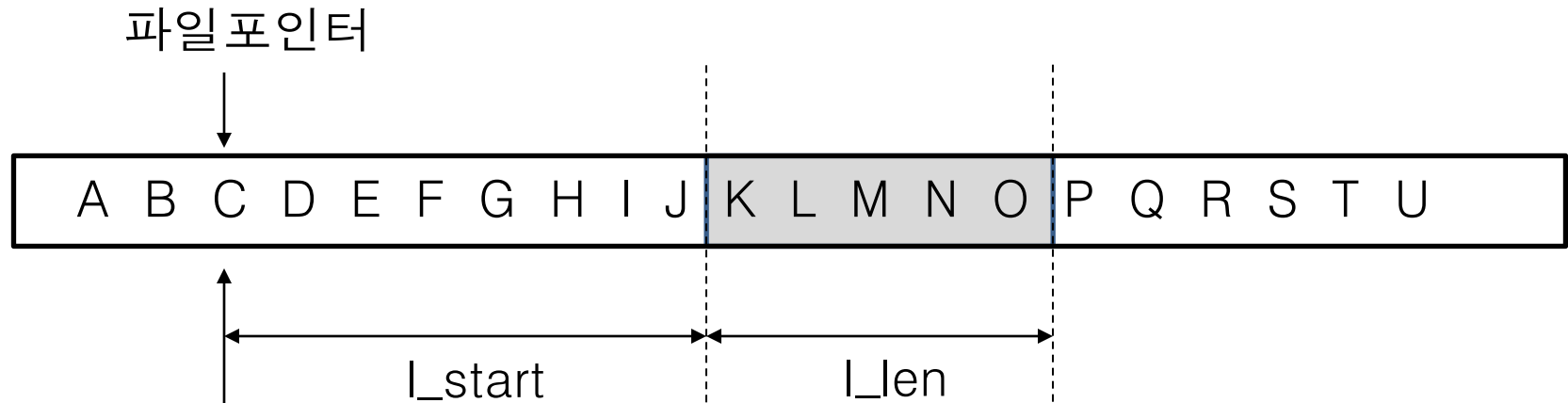
- ldata

```
short l_type;          /* 록의 유형 */
short l_whence;        /* lseek와 동일 */
off_t l_start;         /* 바이트로 표시된 offset */
off_t l_len;           /* 바이트 단위의 세크먼트 크기 */
pid_t l_pid;           /* 명령에 의해 설정 */
```

l_type : F_RDLCK - 읽기 록 적용
F_WRLCK - 쓰기 록 적용
F_UNLCK - 록 제거



레코드 록킹



`l_whence == SEEK_CUR`

```
struct flock my_lock;
```

```
my_lock.l_type = F_WRLCK;
```

```
my_lock.l_whence = SEEK_CUR;
```

```
my_lock.l_start = 0;
```

```
my_lock.l_len = 512;
```

```
fcntl(fd, F_SETLKW, &my_lock);
```



레코드 록킹

□ test1.c

- 록 정보는 fork호출에 의해 계승되지 않음
- fcntl호출의 파일포인터를 변경시키지 않음
- 한 프로세스에 속한 모든 록은 그 프로세스가 죽을 때 자동적으로 제거됨

□ test2.c

- 프로세스는 F_GETLK를 사용해 어느 프로세스가 록을 가지고 있는지 결정할 수 있음

□ test3.c

- Deadlock avoidance test

