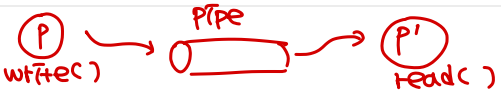
chapter06. UNIX Special Files

objectives

1. Learn about interprocess communication (IPC)
2. Experiment with client-server interactions → 두 process 간의 communication
3. Explore pipes and redirection

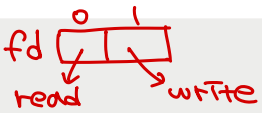
Pipe



the simplest UNIX interprocess communication mechanism.

→ represented by a special file.

- 동일한 시스템에서 실행되는 process들이 정보 공유, 협력할 수 있도록 함.
- filename, offset으로 file의 경로를 알려줌




```
#include <unistd.h>

int pipe(int fd[2]);

//output : two file descriptors -> fd[0], fd[1]
//fd[0] : reading
//fd[1] : writing
//fd[1]에 쓰여진 data를 fd[0]에서 읽을 수 있음 -> FIFO 기반

//return 0 -> successful
//return -1 -> unsuccessful
```

characteristics of pipe

1. pipe : 이름 x ★

   → two file descriptors를 통해서만 program access 가능

   - process가 create (or) fork에 의해 자식 process 상속으로만 사용 가능 (① ②)
2. write → fd[0], read → fd[1] ⇒ POSIX standard에 명시 x

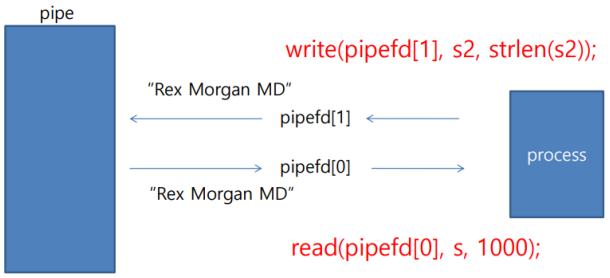
3. process가 pipe에 read하기 위해 call
   a. pipe가 비어 있지 않으면 → block, nonblock 같음
      → read가 immediately하게 return
   b. pipe가 비어 있고 + pipe를 write용으로 fd를 가지고 있는 process가 있다면
      → pipe에 무언가 쓰일 때까지 read block
   c. pipe가 비어 있고 + write 용으로 open한 process가 없다면
      → 비어 있는 pipe read 값은 return 0 (file의 끝을 나타내는 것과 동일)
      ↳기다려봤자 읽을게 X

⇒ pipe는 기본적으로 **blocking I/O** ★

① • **Example** → single process with a pipe is not very useful

```
main(){
  int pipefd[2];
  int i;
  char s[1000];
  char *s2;

  if (pipe(pipefd) < 0) {
    perror("pipe");
    exit(1);
  }

  s2 = "Rex Morgan MD";
  write(pipefd[1], s2, strlen(s2));
  //write(fd, buf, size) -> OS에게 buf의 주소값에 해당하는 size bytes를 보냄
  //fd : open()에 의해 return된 file descriptor

  i = read(pipefd[0], s, 1000);
  s[i] = '\0';
  //end-of-file character -> 다시 read될 때까지 pipe의 끝을 나타냄

  printf("Read %d bytes from the pipe: %s'\n", i, s);
}
```

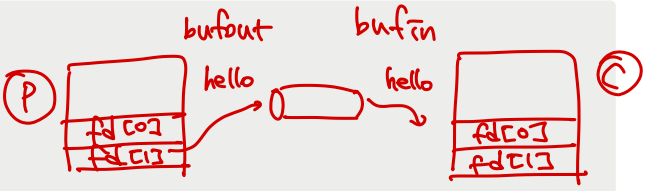
① pipe create → open
② write : string의 길이만큼 파이프에 write
③ Read : 파이프에 buf의 길이만큼 read



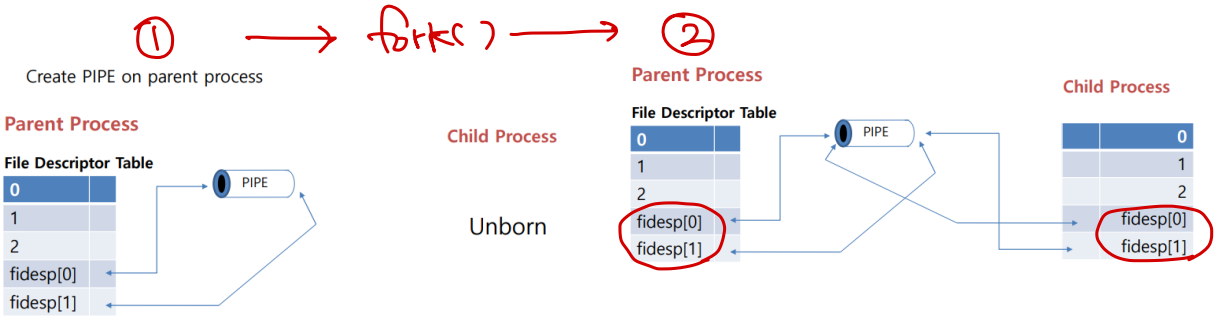
② • **Example** → fork() : 'hello' message를 parent에서 child로 보냄

- bufin → parent: 'empty', bufin이 전달 → child: 'hello'
- read()가 signle write call에 의해 쓰여진 모든 것을 가지고 올 것이라는 보장X
  - 일부만 읽고 return도 가능(cuz, read() : 하나라도 읽으면 success)

    → 다 읽었는지 확인
  - ex) empty → (hel만 읽음) → helty 가능




```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#define BUFSIZE 10

int main(void) {
  char bufin[BUFSIZE] = "empty";
  char bufout[] = "hello";
  int bytesin;
  pid_t childpid;
  int fd[2];
       ① pipe create → open
  if (pipe(fd) == -1) {
    perror("Failed to create the pipe");
    return 1;
  }

  bytesin = strlen(bufin);
  childpid = fork();  ② fork()
  if (childpid == -1) {
    perror("Failed to fork");
    return 1;
  }

                    ③ parent : write
  if (childpid) /* parent code */
    write(fd[1], bufout, strlen(bufout));
  else /* child code */  ④ child : read
    bytesin = read(fd[0], bufin, BUFSIZE);

  fprintf(stderr, "[%ld]:my bufin is {%.*s}, my bufout is {%s}\n",
    (long)getpid(), bytesin, bufin, bufout);
  return 0;
}
```



parent
child

```
ccslab@ccslab-linux:~/programs/usp_all/chapter06$ parentwritepipe
bufin size: 5
[3090]:my bufin is {empty}, my bufout is {hello}
[3091]:my bufin is {hello}, my bufout is {hello}
ccslab@ccslab-linux:~/programs/usp_all/chapter06$
```

↔ pipe : process가 종료되면 pipe도 함께 사라짐 (open한 process가 없다면)

FIFO

named pipe → 모든 process가 FIFO를 다 닫더라도 유효한 pipe

- 이름을 알고 open 권한이 있다면 모든 process가 사용 가능함.
- 다른 file처럼 ls 명령어에 따라 FIFO도 나타남

```
#include <stdio.h>

int mkfifo(const char *path, mode_t mode);
//creates a new FIFO → mkfifo
//1. command from a shell
//2. calling function from a program

//path : 만들고자 하는 FIFO의 file 경로
//mode : permission

//remove a FIFO → file 삭제랑 똑같음
//1. Execute the rm command from a shell
//2. call unlink from a program
```

- **Example**

```
//1. create FIFO(myfifo) in the current working dir,
//and everyone can be read but owner만 wirte 가능

#define FIFO_PERMS (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) → read 다 가능 write는 owner만

if (mkfifo("myfifo", FIFO_PERMS) == -1)
  perror("Failed to create myfifo");

//2. remove FIFO(myfifo) from the current working dir.
if (unlink("myfifo") == -1)
  perror("Failed to remove myfifo");
```

644 permission

삭제

- **Example** → 자식이 부모에게 전달

① named pipe (FIFO) 생성
② fork
③ child → write
④ parent → read

```
#define BUFSIZE 256
#define FIFO_PERM (S_IRUSR | S_IWUSR)
#define FIFO_MODES O_RDONLY
```

```
//child
int dofifochild(const char *fifoname, const char *idstring) {
  char buf[BUFSIZE];
  int fd;
  int rval;
  ssize_t strsize;

  fprintf(stderr, "[%ld]:(child) about to open FIFO %s...\n",
    (long)getpid(), fifoname);
  while (((fd = open(fifoname, O_WRONLY)) == -1) && (errno == EINTR)) ;

  if (fd == -1) {
    fprintf(stderr, "[%ld]:failed to open named pipe %s for write: %s\n",
      (long)getpid(), fifoname, strerror(errno));
    return 1;
  }

  rval = snprintf(buf, BUFSIZE, "[%ld]:%s\n", (long)getpid(), idstring);

  if (rval < 0) {
    fprintf(stderr, "[%ld]:failed to make the string:\n", (long)getpid());
    return 1;
  }

  strsize = strlen(buf) + 1;
  fprintf(stderr, "[%ld]:about to write...\n", (long)getpid());
  rval = r_write(fd, buf, strsize);
  if (rval != strsize) {
    fprintf(stderr, "[%ld]:failed to write to pipe: %s\n",
      (long)getpid(), strerror(errno));
    return 1;
  }

  fprintf(stderr, "[%ld]:finishing...\n", (long)getpid());
  return 0;
}

//parent
int dofifoparent(const char *fifoname) {
  char buf[BUFSIZE];
  int fd;
  int rval;
  fprintf(stderr, "[%ld]:(parent) about to open FIFO %s...\n",
  (long)getpid(), fifoname);
  while (((fd = open(fifoname, FIFO_MODES)) == -1) && (errno == EINTR));

  if (fd == -1) {
    fprintf(stderr, "[%ld]:failed to open named pipe %s for read: %s\n",
    (long)getpid(), fifoname, strerror(errno));
    return 1;
  }

  fprintf(stderr, "[%ld]:about to read...\n", (long)getpid());
  rval = r_read(fd, buf, BUFSIZE);
  if (rval == -1) {
    fprintf(stderr, "[%ld]:failed to read from pipe: %s\n",
    (long)getpid(), strerror(errno));
```

```
    return 1;
  }

  fprintf(stderr, "[%ld]:read %.*s\n", (long)getpid(), rval, buf);
  return 0;
}

//main - parentchildfifo.c
int main (int argc, char *argv[]) {
  pid_t childpid;

  if (argc != 2) { /* command line has pipe name */
    fprintf(stderr, "Usage: %s pipename\n", argv[0]);
    return 1;
  }
  if (mkfifo(argv[1], FIFO_PERM) == -1) { /* create a named pipe */
    if (errno != EEXIST) {
      fprintf(stderr, "[%ld]:failed to create named pipe %s: %s\n",
        (long)getpid(), argv[1], strerror(errno));
      return 1;
    }
  }
  if ((childpid = fork()) == -1){
    perror("Failed to fork");
    return 1;
  }

  if (childpid == 0) /* The child writes */
    return dofifochild(argv[1], "this was written by the child");
  else
    return dofifoparent(argv[1]);
}
```

① fifo 생성
file 이름
같은 이름을 가진 pipe가 있다면 mkfifo → return -1
But, 타입이 EEXIST면 그냥 넘어가고 정상작동
②
③ child code
↳ pipe에 쓰기
④ parent code

```
ccslab@ccslab-linux:~/programs/usp_all/chapter06$ parentchildfifo
Usage: parentchildfifo pipename
ccslab@ccslab-linux:~/programs/usp_all/chapter06$ parentchildfifo myfifo
[4171]:(parent) about to open FIFO myfifo...
[4172]:(child) about to open FIFO myfifo...
[4172]:about to write...
[4172]:finishing...
[4171]:about to read...
[4171]:read [4172]:this was written by the child
```

⇒이 예제에서 보았듯 자식 모두 FIFO에 대해 unlink X
↳ FIFO는 계속 존재하여 재사용 가능