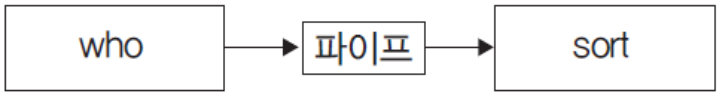
Linux Programming
12장. 파이프

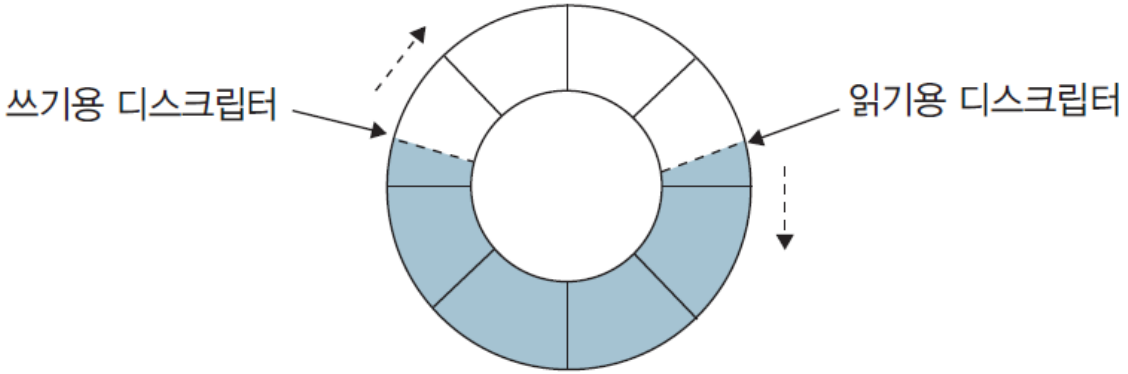
sisong@ut.ac.kr
한국교통대학교 컴퓨터공학전공
송석일

12.1 파이프

파이프 원리

- $ who | sort





- 파이프
  - 물을 보내는 수도 파이프와 비슷
  - 한 프로세스는 쓰기용 파일 디스크립터를 이용하여 파이프에 데이터를 송신(쓰기)
  - 다른 프로세스는 읽기용 파일 디스크립터를 이용하여 그 파이프에서 데이터 수진(읽기)
  - 한 방향(one way) 통신, 큐

파이프 생성

```
#include <unistd.h>
int pipe(int fd[2])
```

- 파이프를 생성
- 성공하면 0을 실패하면 -1을 반환

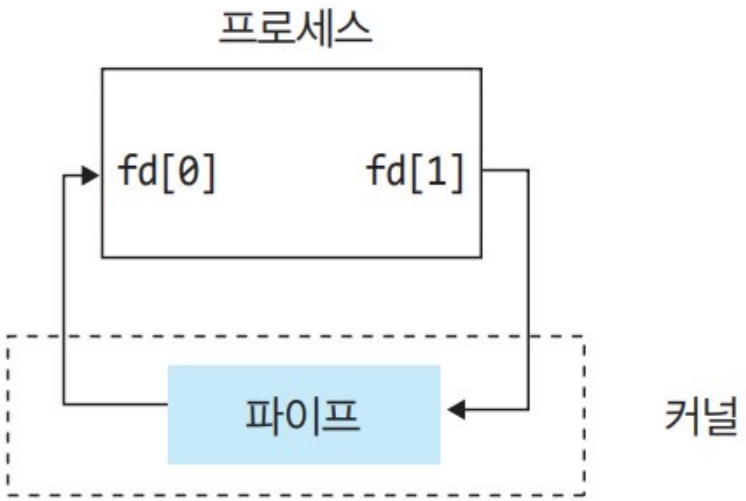


그림 12.2 파이프 생성

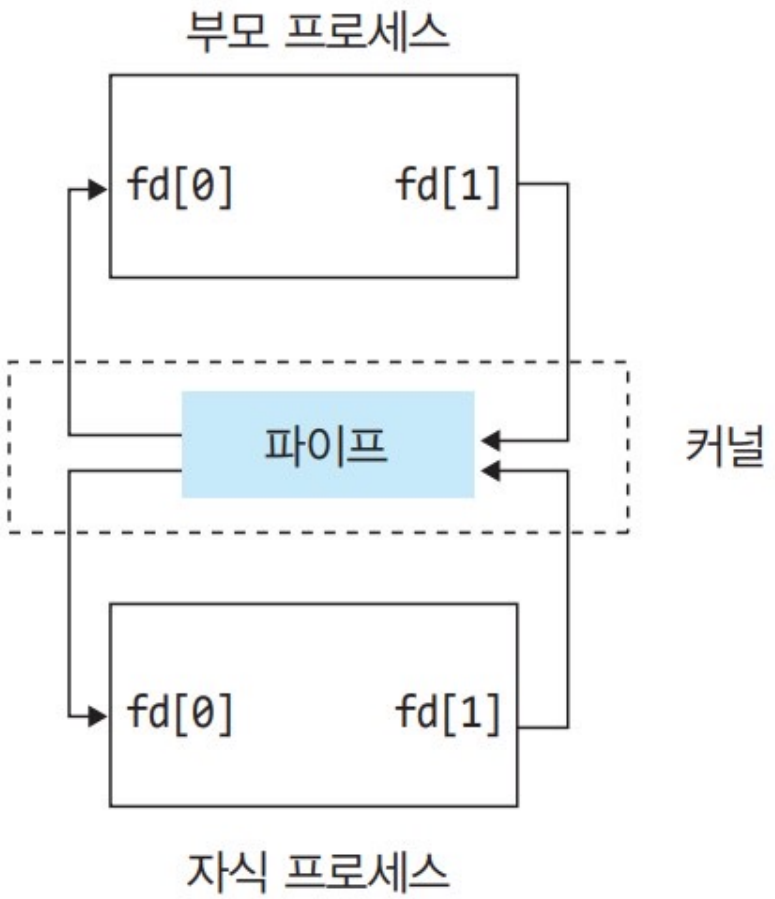
- 파이프는 두 개의 파일 디스크립터 생성
- 하나(fd[1])는 쓰기용이고 다른 하나(fd[0])는 읽기용

부모-자식 프로세스 사이에 파이프 사용법

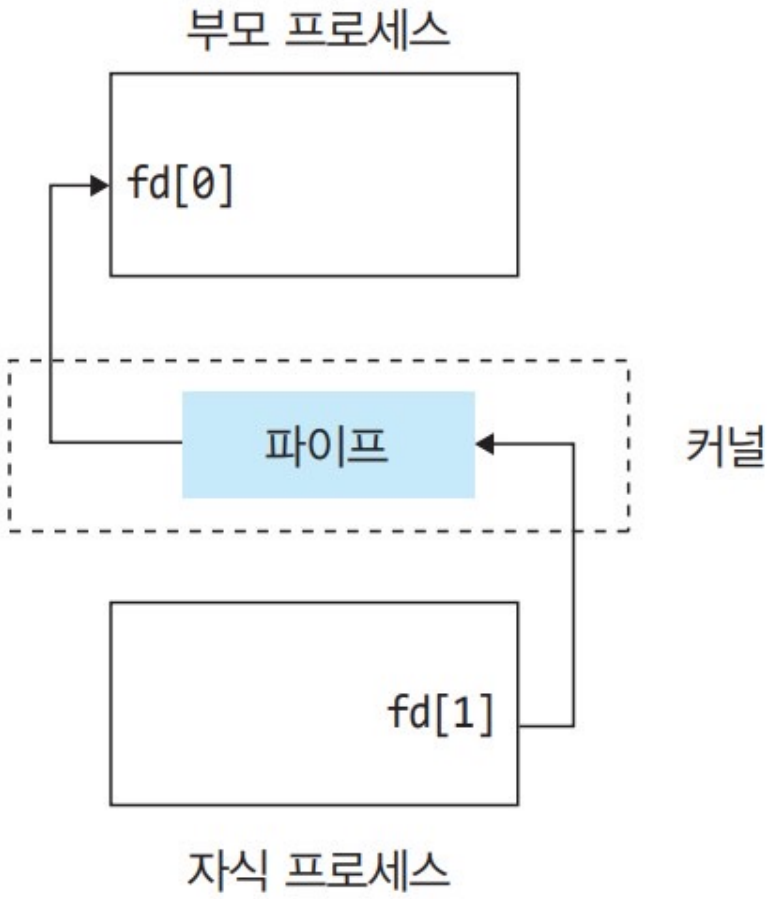
1. 한 프로세스가 파이프를 생성
2. 그 프로세스가 자식 프로세스를 생성
3. 쓰기 프로세스는 읽기 파이프 디스크립터 close
4. 읽기는 프로세스는 쓰기 파이프 디스크립터 close
5. write()와 read() 시스템 호출을 사용하여 파이프를 통해 데이터 송수신
6. 각 프로세스는 파이프 close

파이프 사용법

- 자식 생성 후



- 자식에서 부모로 보내기



pipe.c

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <unistd.h>
4.  #include <string.h>
5.  #define MAXLINE 100

6.  /* 파이프를 통해 자식에서 부모로 데이터를
    보내는 프로그램 */
7.  int main( )
8.  {
9.      int n, length, fd[2], pid;
10.     char message[MAXLINE], line[MAXLINE];

11.     pipe(fd);  /* 파이프 생성 */
```

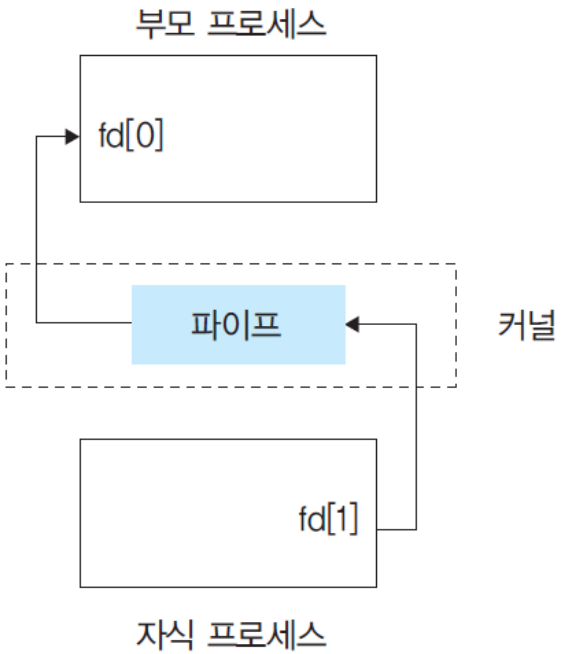
```
12.  if ((pid = fork()) == 0) {   /* 자식 프로세스 */
13.          close(fd[0]);
14.          sprintf(message, "Hello from PID %d\n",
                        getpid());
15.          length = strlen(message)+1;
16.          write(fd[1], message, length);
17.      } else {                    /* 부모 프로세스 */
18.          close(fd[1]);
19.          n = read(fd[0], line, MAXLINE);
20.          printf("[%d] %s", getpid(), line);
21.      }

22.      exit(0);
23.  }
```

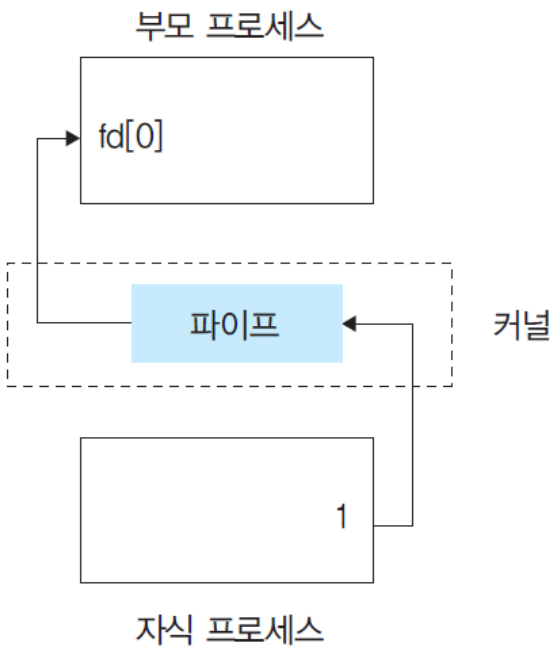
```
$ pipe
[12555] Hello from PID 12556
```

12.2 쉘 파이프 구현

표준출력을 파이프로 보내기

- 자식 프로세스의 표준출력을 파이프를 통해 부모 프로세스에 보내기
  - 쓰기용 파이프 디스크립터 fd[1]을 표준출력 1번 파일 디스크립터에 복제
  - dup2(fd[1],1)





stdpipe.c

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <unistd.h>
4.  #define MAXLINE 100

5.  /* 파이프를 통해 자식에서 실행되는 명령어
    출력을 받아 프린트한다. */
6.  int main(int argc, char* argv[])
7.  {
8.     int n, pid, fd[2];
9.     char line[MAXLINE];

10.    pipe(fd);  /* 파이프 생성 */
```

```
11.    if ((pid = fork()) == 0) { /* 자식 프로세스 */
12.       close(fd[0]);
13.       dup2(fd[1],1);       /* 쓰기용 파이프를
14.                               표준출력에 복제 */
15.       close(fd[1]);
16.       printf("Hello! pipe\n");
17.       printf("Bye! pipe\n");
18.    } else {              /* 부모 프로세스 */
19.       close(fd[1]);
20.       printf("자식 프로세스로부터 받은 결과\n");
21.       while ((n = read(fd[0], line, MAXLINE)) > 0)
22.          // printf("%d %s\n", n, line);
23.          write(STDOUT_FILENO, line, n);
24.    }
25.
26.    exit(0);
27. }
```

```
$ stdpipe
자식 프로세스로부터 받은 결과
Hello! pipe
Bye! pipe
```

pexec1.c

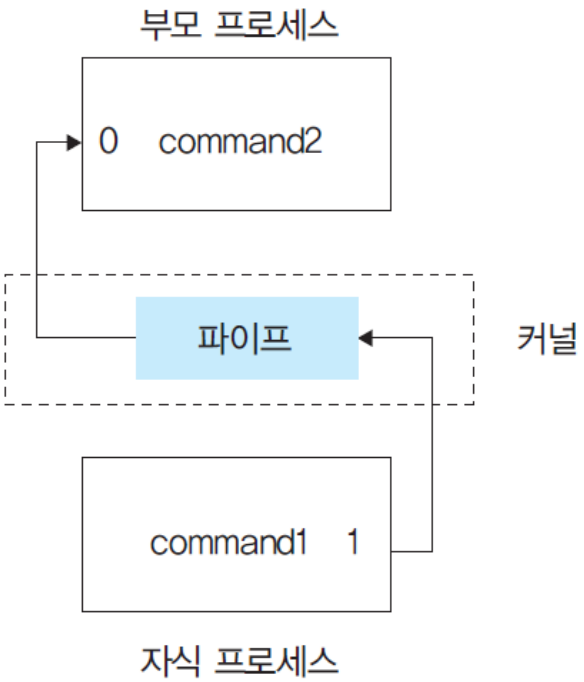
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #define MAXLINE 100
5
6 /* 파이프를 통해 자식에서 실행되는명령어 출력을 받아 프린트 */
7 int main(int argc, char* argv[])
8 {
9    int n, pid, fd[2];
10   char line[MAXLINE];
11
12   pipe(fd);   /* 파이프 생성 */
13
```

```
14   if ((pid = fork()) == 0) { //자식 프로세스
15     close(fd[0]);
16     dup2(fd[1],1);
17     close(fd[1]);
18     execvp(argv[1], &argv[1]);
19   } else {  // 부모 프로세스
20     close(fd[1]);
21     printf("자식 프로세스로부터 받은 결과\n");
22     while ((n = read(fd[0], line, MAXLINE))> 0)
23        write(STDOUT_FILENO, line, n);
24   }
25
26   exit(0);
27 }
```

```
$ pexec1 date
자식 프로세스로부터 받은 결과
2021. 05. 20. (목) 10:19:38 KST
```

쉘 파이프

- 쉘 커맨드 라인에서 파이프 생성 가능
- $ command1 | command2
  - 자식 프로세스가 실행하는 command1의 표준출력을 파이프를 통해서 부모 프로세스가 실행하는 command2의 표준입력으로 전달



shellpipe.c

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  #include <unistd.h>
4.  #define READ 0
5.  #define WRITE 1

6.  int main(int argc, char* argv[])
7.  {
8.      char str[1024];
9.      char *command1, *command2;
10.     int fd[2];

11.     printf("[shell]");
12.     fgets(str,sizeof(str),stdin);
13.     str[strlen(str)-1] ='\0';

14.     if(strchr(str,'|') != NULL) {
15.         //파이프 사용하는 경우
16.         command1 = strtok (str,"| ");
17.         command2 = strtok (NULL, "| ");
18.     }
```

```
1.      pipe(fd);

2.      if (fork() ==0) {
3.          close(fd[READ]);
4.      dup2(fd[WRITE],1);
5.      close(fd[WRITE]);
6.      execlp(command1, command1, NULL);
7.      perror("pipe");
8.       } else {
9.      close(fd[WRITE]);
10.     dup2(fd[READ],0);
11.     close(fd[READ]);
12.     execlp(command2, command2, NULL);
13.     perror("pipe");
14.     }
15. }
```

```
$ shellpipe
[shell] ls | wc
      17       17      160
```

12.3 파이프 함수

popen()

#include <stdio.h>

FILE *popen(const char **command*, const char **type*);

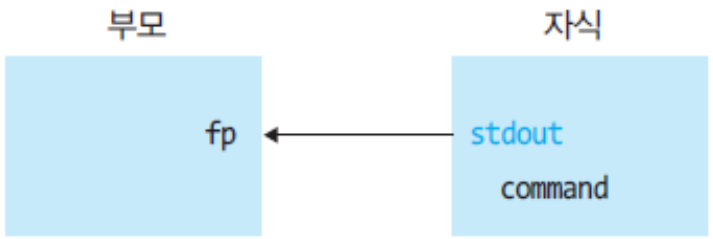
- 성공하면 파이프를 위한 파일 포인터를 실패하면 NULL을 반환

Int pclose(FILE **fp*);

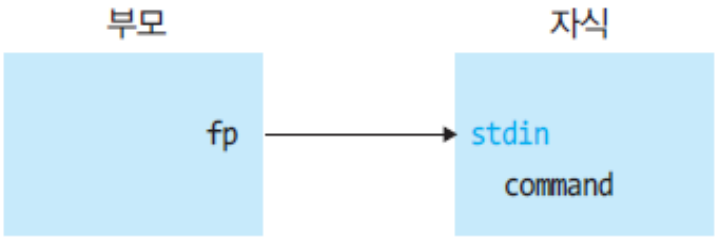
- 성공하면 *command* 명령어의 종료 상태를 실패하면 -1을 반환

- 자식 프로세스에게 명령어를 실행시키고 그 표준 입출력을 파이프를 통해 송수신 하는 과정을 하나의 함수로 정의

fp = popen(command, "r");



fp = popen(command, “w");



pexec2.c

```
1.  #include <stdio.h>
2.  #define MAXLINE 100

3.  /* popen() 함수를 이용해 자식에서 실행되는 명령어 출력을 받아 프린트 */
4.  int main(int argc, char* argv[])
5.  {
6.    char line[MAXLINE];
7.    FILE *fpin;
8.    if ((fpin = popen(argv[1],"r")) == NULL) {
9.      perror("popen 오류");
10.     return 1;
11.   }
12.   printf("자식 프로세스로부터 받은 결과\n");
13.   while (fgets(line, MAXLINE, fpin))
14.     fputs(line, stdout);
15.   pclose(fpin);
16.   return 0;
17. }
```

```
$ pexec2 date
자식 프로세스로부터 받은 결과
2021. 05. 20. (목) 10:27:39 KST
```

명령어 파이프 구현 예제

- $ pexec3 command1 command2

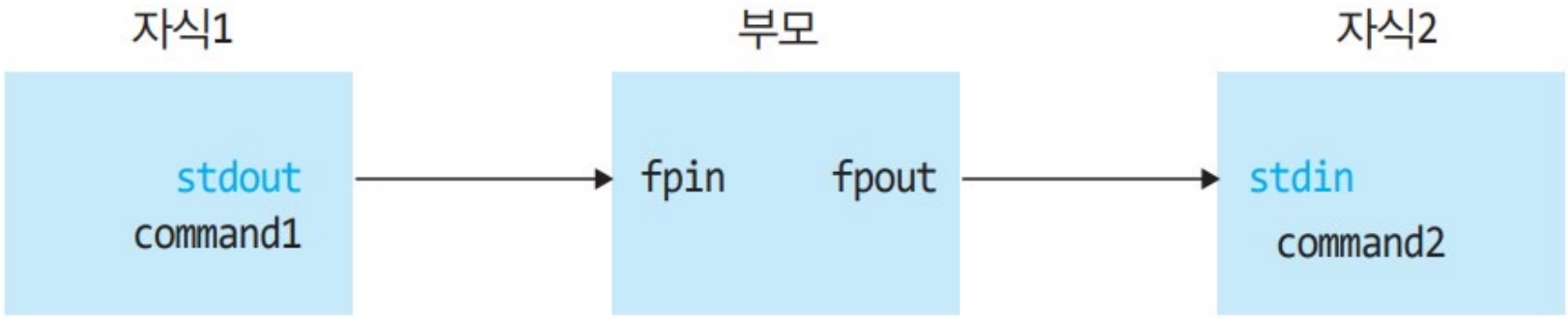


그림 12.10 명령어 파이프 구현

```c
#include <stdio.h>
#define MAXLINE 100
/* popen() 함수를 이용해 명령어 파이프 기능을 구현한다. */

int main(int argc, char* argv[])
{
   char line[MAXLINE];
   FILE *fpin, *fpout;

   if ((fpin = popen(argv[1],"r")) == NULL) {
      perror("popen 오류");
      return 1;
   }

   if ((fpout = popen(argv[2],"w")) == NULL) {
      perror("popen 오류");
      return 1;
   }

   while (fgets(line, MAXLINE, fpin))
      fputs(line, fpout);

   pclose(fpin);
   pclose(fpout);
   return 0;
}
```

12.4 이름 있는 파이프

이름 있는 파이프(named pipe)

- (이름 없는) 파이프
  - 이름이 없으므로 부모 자식과 같은 서로 관련된 프로세스 사이의 통신에만 사용
- 이름 있는 파이프
  - 다른 파일처럼 이름이 있으며 파일 시스템 내에 존재
  - 서로 관련 없는 프로세스들도 공유하여 사용 가능

이름 있는 파이프를 만드는 방법

- p 옵션과 함께 mknod 명령어

```
$mknod myPipe p
$chmod ug+rw myPipe
$ls -l myPipe
prw-rw-r-- 1 lect faculty 0 4월 11일 13:03 myPipe
```

- mkfifo() 시스템 호출

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *pathname, mode_t mode);
```

- 이름 있는 파이프를 생성한다. 성공하면 0을 실패하면 -1을 반환

npreader.c

```
1.  #include <stdio.h>
2.  #include <sys/types.h>
3.  #include <sys/stat.h>
4.  #include <fcntl.h>
5.  #define MAXLINE 100
6.  /* 이름 있는 파이프를 통해 읽은 내용을 프린
    트한다. */
7.  int main( )
8.  {
9.    int fd;
10.   char str[MAXLINE];
11.   unlink("myPipe");
12.   mkfifo("myPipe", 0660);
13.   fd = open("myPipe", O_RDONLY);
14.   while (readLine(fd, str))
15.     printf("%s \n", str);
16.   close(fd);
17.   return 0;
18. }

19. int readLine(int fd, char *str)
20. {
21.   int n;
22.   do {
23.     n = read(fd, str, 1);
24.   } while (n > 0 && *str++ != NULL);
25.   return (n > 0);
26. }
```

npwriter.c

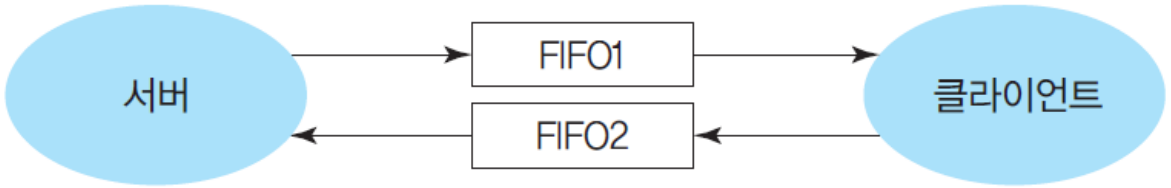
```
1.  #include <sys/types.h>
2.  #include <sys/stat.h>
3.  #include <fcntl.h>
4.  #define MAXLINE 100
5.  /* 이름 있는 파이프를 통해 메시지를 출력한
    다. */
6.  Int main( )
7.  {
8.    int fd, length;
9.    char message[MAXLINE];
10.   sprintf(message, "Hello from PID %d", getpid());
11.   length = strlen(message)+1;
```

```
1.    do {
2.       fd = open("myPipe", O_WRONLY);
3.       if (fd == -1) sleep(1);
4.     } while (fd == -1);

5.    for (int i = 0; i <= 3; i++) {
6.      write(fd, message, length);
7.      sleep(3);
8.    }
9.    close(fd);
10.   return 0;
11. }
```

```
$ npwriter &
[1] 13079
$ npreader
Hello from PID 13079
Hello from PID 13079
Hello from PID 13079
Hello from PID 13079
[1]+   Done                    npwriter
```

파이프를 이용한 일대일 채팅

- 이 프로그램은 채팅 서버와 채팅 클라이언트로 구성된다.
- 2 개의 파이프가 필요함
  - FIFO1 : 채팅 서버 → 채팅 클라이언트로 데이터를 보내는데 사용
  - FIFO2 : 채팅 클라이언트 → 채팅 서버로 데이터를 보내는데 사용



실행결과

```
$chatserver
[서버] : 안녕하세요. 클라이언트
[클라이언트] -> 반갑습니다. 서버
...
```

```
$chatclient
* 클라이언트 시작
[서버] -> 안녕하세요. 클라이언트
[클라이언트] : 반갑습니다. 서버
...
```

chatserver.c

```
1.  #include <sys/types.h>
2.  #include <sys/stat.h>
3.  #include <fcntl.h>
4.  #include <stdio.h>
5.  #include <string.h>
6.  #include <stdlib.h>
7.  #include <unistd.h>
8.  #define MAXLINE 256
9.  main() {
10. int fd1, fd2, n;
11. char msg[MAXLINE];

12. if (mkfifo("./chatfifo1", 0666) == -1) {
13.     perror("mkfifo");
14.     exit(1);
15. }
16. if (mkfifo("./chatfifo2", 0666) == -1) {
17.     perror("mkfifo");
18.     exit(2);
19. }
```

```
1.      fd1 = open("./chatfifo1", O_WRONLY);
2.      fd2 = open("./chatfifo2", O_RDONLY);
3.      if (fd1 == -1 || fd2 == -1) {
4.          perror("open");
5.          exit(3);
6.      }

7.      printf("* 서버 시작 \n");
8.      while(1) {
9.        printf("[서버] : ");
10.       fgets(msg, MAXLINE, stdin);
11.       n = write(fd1, msg, strlen(msg)+1);
12.       if (n == -1) {
13.           perror("write");
14.           exit(1);
15.       }
16.       n = read(fd2, msg, MAXLINE);
17.       printf("[클라이언트] -> %s\n", msg);
18.   }
19. }
```

chatclient.c

```
1.  #include <sys/types.h>
2.  #include <sys/stat.h>
3.  #include <fcntl.h>
4.  #include <stdio.h>
5.  #include <string.h>
6.  #include <stdlib.h>
7.  #include <unistd.h>
8.  #define MAXLINE 256
9.  main() {
10.     int fd1, fd2, n;
11.     char inmsg[MAXLINE];
12.     fd1 = open("./chatfifo1", O_RDONLY);
13.     fd2 = open("./chatfifo2", O_WRONLY);
```

```
1.      if(fd1 == -1 || fd2 == -1) {
2.         perror("open");
3.         exit(1);
4.      }

5.      printf("* 클라이언트 시작 \n");
6.      while(1) {
7.         n = read(fd1, inmsg, MAXLINE);
8.         printf("[서버] -> %s\n", inmsg);
9.         printf("[클라이언트] : ");
10.        fgets(inmsg, MAXLINE, stdin);
11.        write(fd2, inmsg, strlen(inmsg)+1);
12.     }
13. }
```