

목차(Chapter 9)

- 파이프의 개념
- 이름없는 파이프 만들기
- 복잡한 파이프 생성
- 양방향 파이프 활용
- 이름있는 파이프 만들기



파이프의 개념

□ 파이프

- 두 프로세스간에 통신할 수 있도록 해주는 특수 파일
- 그냥 파이프라고 하면 일반적으로 이름없는 파이프를 의미
- 이름 없는 파이프는 부모-자식 프로세스 간에 통신할 수 있도록 해줌
- 파이프는 기본적으로 단방향

□ 간단한 파이프 생성

```
#include <stdio.h>
FILE *popen(const char *command, const char *mode);
```

- **command** : 셸 명령
- **mode** : “r” (읽기전용 파이프) 또는 “w” (쓰기전용 파이프)
- 내부적으로 fork함수를 실행해 자식 프로세스를 만들고 command에서 지정한 명령을 exec 함수로 자식이 실행하도록 한다: `execl(“/bin/sh” , “sh” , “-c” , command, (char *)0);`

```
#include <stdio.h>
int pclose(FILE *stream);
```

waitpid함수를 수행하여 자식 프로세스들이 종료하기를 기다린다. 리턴값은 자식 프로세스의 종료상태이며, 실패하면 -1을 리턴한다



[예제 9-1] popen 함수 사용하기(test1.c)

```
...
04 int main(void) {
05     FILE *fp;
06     int a;
07
08     fp = popen("wc -l", "w");
09     if (fp == NULL) {
10         fprintf(stderr, "popen failed\n");
11         exit(1);
12     }
13
14     for (a = 0; a < 100; a++)
15         fprintf(fp, "test line\n");
16
17     pclose(fp);
18
19     return 0;
20 }
```

"w"모드로 파이프 생성
자식프로세스는 wc -l
명령 수행

자식 프로세스로 출력

결과는 무엇일까?

[예제 9-2] popen 함수 사용하기(test2.c)

```
...
04 int main(void) {
05     FILE *fp;
06     char buf[256];
07
08     fp = popen("date", "r");
09     if (fp == NULL) {
10         fprintf(stderr, "popen failed\n");
11         exit(1);
12     }
13
14     if (fgets(buf, sizeof(buf), fp) == NULL) {
15         fprintf(stderr, "No data from pipe!\n");
16         exit(1);
17     }
18
19     printf("line : %s\n", buf);
20     pclose(fp);
21
22     return 0;
23 }
```

자식 프로세스는
date 명령 실행

읽기모드로 파이프생성

파이프에서 데이터 읽기

ex9_2.out

line : 2010년 2월 5일 금요일 오후 11시 20분 40초

복잡한 파이프 생성[1]

- popen은 파이프를 생성하는 것은 간단하지만, 셸을 실행해야 하므로 비효율적이고 주고 받을 수 있는 데이터도 제한적임
- 파이프 만들기: pipe(2)

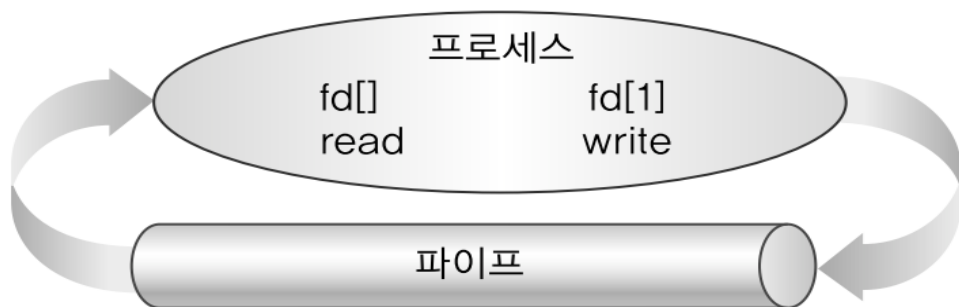
```
#include <unistd.h>

int pipe(int fildes[2]);
```

- 파이프로 사용할 파일기술자 2개를 인자로 지정
- fildes[0]는 읽기, fildes[1]은 쓰기용 파일 기술자

□ pipe 함수로 통신과정 (test3.c)

1. pipe 함수를 호출하여 파이프로 사용할 파일기술자 생성

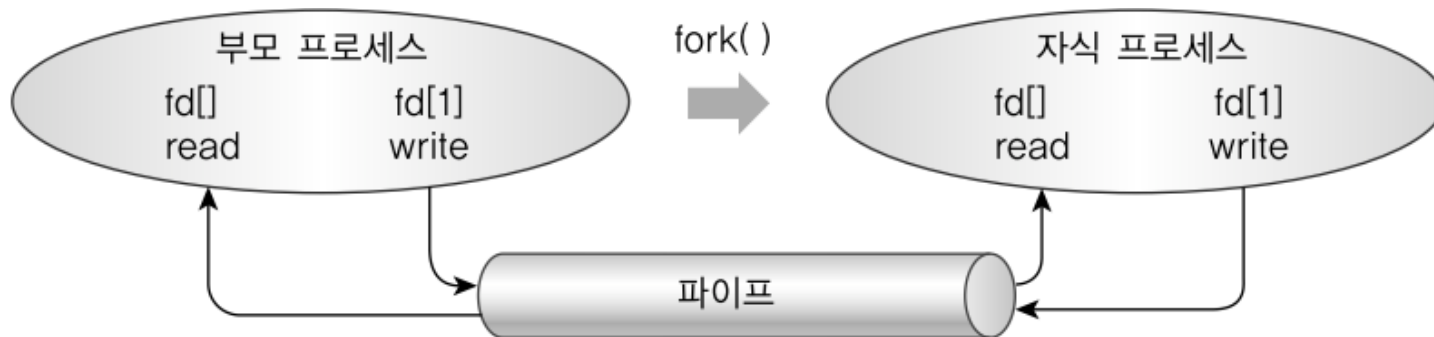


[그림 9-1] pipe 함수를 이용한 파이프 생성



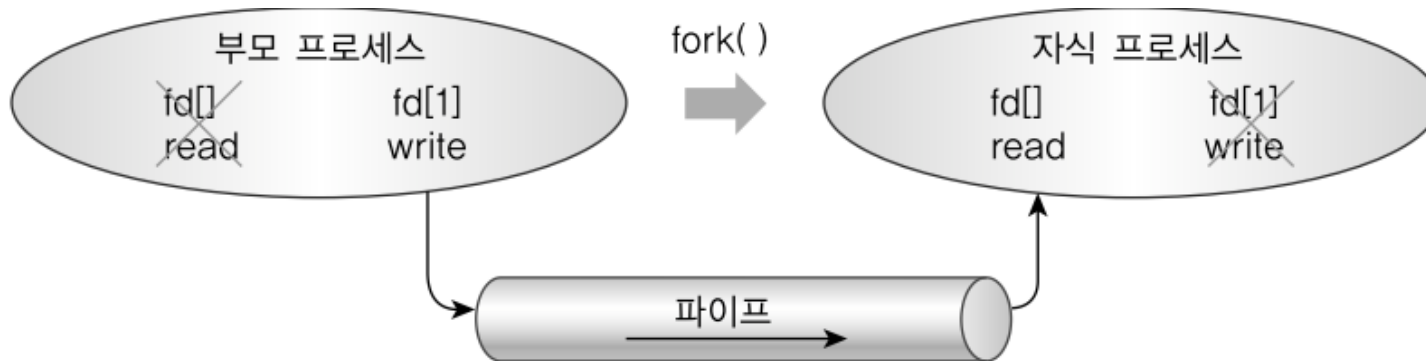
복잡한 파이프 생성[2]

2. fork 함수로 자식 프로세스 생성. pipe도 자식 프로세스로 복사됨(test4.c)



[그림 9-2] 자식 프로세스로 파일 기술자 복사

3. 통신방향 결정(파이프는 기본적으로 단방향)(test5.c)



[그림 9-3] 부모 → 자식 방향으로 통신



```
...
06 int main(void) {
07     int fd[2];
08     pid_t pid;
09     char buf[257];
10 int len, status;
11
12     if (pipe(fd) == -1) {
13         perror("pipe");
14         exit(1);
15     }
16
17     switch (pid = fork()) {
18         case -1 :
19             perror("fork");
20             exit(1);
21             break;
```

파이프 생성

fork로 자식 프로세스
생성



[예제 9-3] pipe 함수 사용하기(test6.c)

```
22     case 0 : /* child */
23         close(fd[1]);
24         write(1, "Child Process:", 15);
25         len = read(fd[0], buf, 256);
26         write(1, buf, len);
27         close(fd[0]);
28         break;
29     default :
30         close(fd[0]);
31         buf[0] = '\0';
32         write(fd[1], "Test Message\n", 14);
33         close(fd[1]);
34         waitpid(pid, &status, 0);
35         break;
36     }
37
38     return 0;
39 }
```

파이프에서
읽기

자식 프로세스는 파이프에서
읽을 것이므로 쓰기용 파일
기술자(fd[1])를 닫는다.

부모 프로세스는 파이프에
쓸 것이므로 읽기용 파일기
술자(fd[0])를 닫는다.

파이프에 텍스트를 출력

ex9_3.out

Child Process:Test Message

파이프

- Write시 파이프에 충분한 공간이 있으면 파이프에 저장
- 공간이 없으면 다른 프로세스에 의해 자료가 읽혀져 파이프에 충분한 공간이 마련될 때까지 수행이 일시 중단 (test7.c)
- 쓰기전용 파일기술자를 닫았을 때:
 - 자료를 쓰기 위해 해당 파이프를 개방한 다른 프로세스가 아직 존재하면 OK
 - 쓰기 프로세스가 더 이상 없으면, 그 파이프로부터 자료를 읽으려는 프로세스를 깨우고 0를 복귀. 파일의 끝에 도달한 것 같은 효과를 발생
- 읽기전용 파일기술자를 닫았을 때:
 - 자료를 읽기 위해 해당 파이프를 개방한 다른 프로세스가 존재하면 OK
 - 없으면 자료 쓰기를 기다리는 모든 프로세스는 커널로부터 SIGPIPE시그널을 받고 -1로 복귀. 오류발생

