9 파이프

학습목표

- 파이프를 이용한 IPC 기법을 이해한다. Inter Process Communication
- 이름 없는 파이프를 이용해 통신프로그램을 작성할 수 있다.
- 이름 있는 파이프를 이용해 통신프로그램을 작성할 수 있다.

프로세스간 통신은 메모리 말고도 파이프가 있다

파이프의 용도는 오직 하나 (읽거나 or 쓰거나)

이름 없는 파이프 : 누구나 생성 가능, 부모와 자식만 사용가능

이름 있는 파이프 : 슈퍼유저만 생성 가능, 제 3 자도 사용가능

목차

- 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

- 파이프 생성: popen(3) 더 좋은게 있어서 이제 안씀

```
#include <stdio.h>

FILE *popen(const char *command, const char *mode);
```

command : 쉘 명령
mode : "r"(읽기전용 파이프) 또는 "w"(쓰기전용 파이프)

- 파이프 닫기: pclose(3)

```
#include <stdio.h>

int pclose(FILE *stream);
```

[예제 9-1] popen 함수 사용하기(쓰기 전용)

ex9_1.c

```
<stdlib.h><stdio.h>
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  }
```

wc : word count, 단어가 몇개인가?

-l : 몇 줄인가?

"w"모드로 파이프 생성
자식프로세스는 wc –l
명령 수행

자식 프로세스로 출력

결과는 무엇일까?

[예제 9-2] popen 함수 사용하기(읽기 전용)

ex9_2.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]

파이프 만들기: pipe(2)

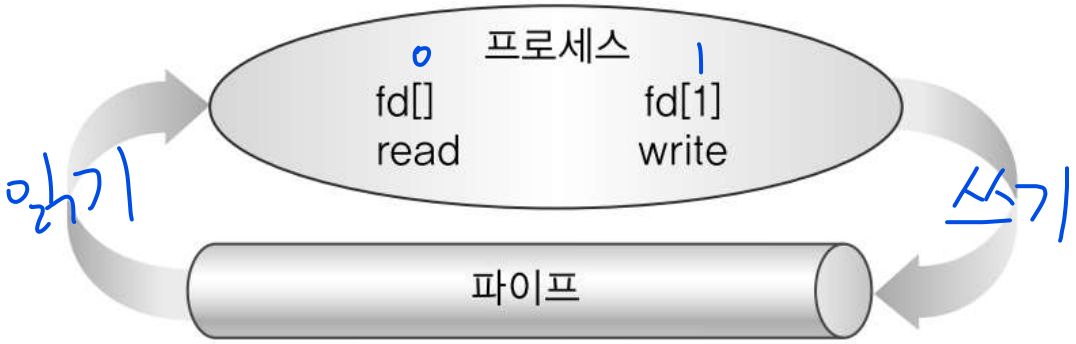
```
#include <unistd.h>

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

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

pipe 함수로 통신과정

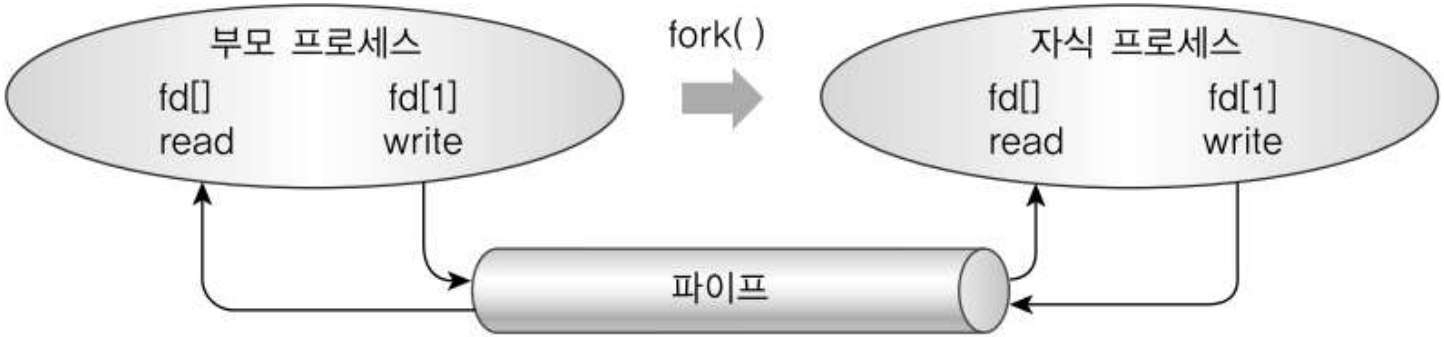
1. piep 함수를 호출하여 파이프로 사용할 파일기술자 생성



[그림 9-1] pipe 함수를 이용한 파이프 생성

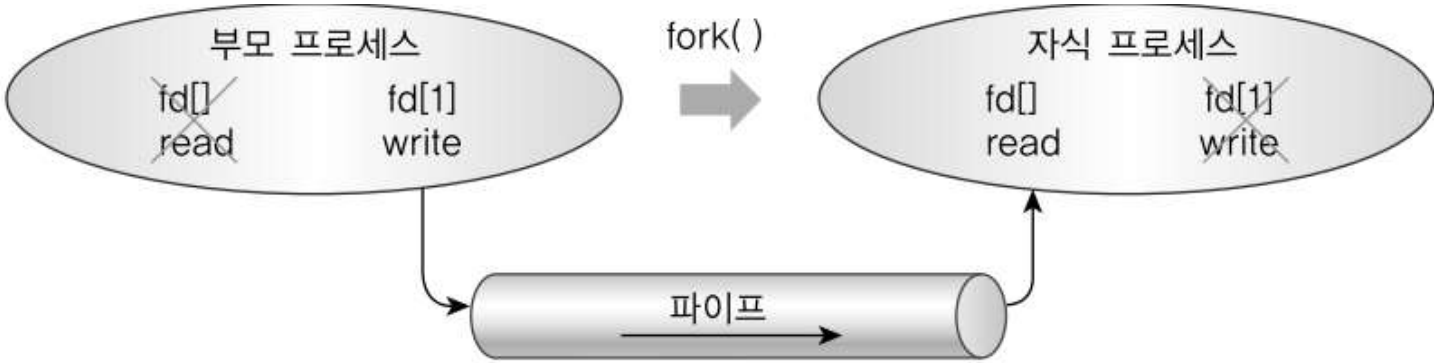
복잡한 파이프 생성[2]

2. fork 함수로 자식 프로세스 생성. pipe도 자식 프로세스로 복사됨



[그림 9-2] 자식 프로세스로 파일 기술자 복사

3. 통신방향 결정(파이프는 기본적으로 단방향)



[그림 9-3] 부모 → 자식 방향으로 통신

[예제 9-3] pipe 함수 사용하기(1)

ex9_3.c

```
<sys/wait.h><unistd.h><stdlib.h><stdio.h>
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(1)

ex9_3.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'; //sleep(2);
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 파일기술자(fd[0])를 닫는다.

파이프에 텍스트를 출력

표준입력 0 // 출력 1

sleep(5); 하면 25번줄에서 read가 기다림

그냥 쓰는게 아닌, '파이프'에

```
# ex9_3.out
Child Process:Test Message
```

[예제 9-4] pipe 함수 사용하기(2)

ex9_4.c

- ps –ef | grep ~~telnet~~ root 동작 구현

현재 활동중인 모든 프로세스들이 나옴, (root 라는 단어가 있는 것만)

```
...<sys/wait.h> <unistd.h> <stdlib.h> <stdio.h>
06  int main(void) {
07      int fd[2];
08      pid_t pid;
09
10      if (pipe(fd) == -1) {
11          perror("pipe");
12          exit(1);
13      }
14
15      switch (pid = fork()) {
16          case -1 :
17              perror("fork");
18              exit(1);
19              break;
20          case 0 : /* child */
21              close(fd[1]);
22              if (fd[0] != 0) {
23                  dup2(fd[0], 0);
24                  close(fd[0]);
25              }
```

파이프 생성

fd[0]에 0번(표준입력)을 복사
자식프로세스는 파이프
입력으로 0번 사용

복사 후 원본 삭제

[예제 9-4] pipe 함수 사용하기(2)

ex9_4.c

```
26              execlp("grep", "grep", "telnet", (char *)NULL);
27              exit(1);//telnet을 root로 변경
28              break;
29          default :
30              close(fd[0]);
31              if (fd[1] != 1) {
32                  dup2(fd[1], 1);
33                  close(fd[1]);
34              }
35              execlp("ps", "ps", "-ef", (char *)NULL);
36              wait(NULL);
37              break;
38      }
39
40      return 0;
41  }
```

grep telnet 명령 실행

fd[1]에 1번(표준출력)을 복사
부모로세스는 파이프
출력으로 1번 사용

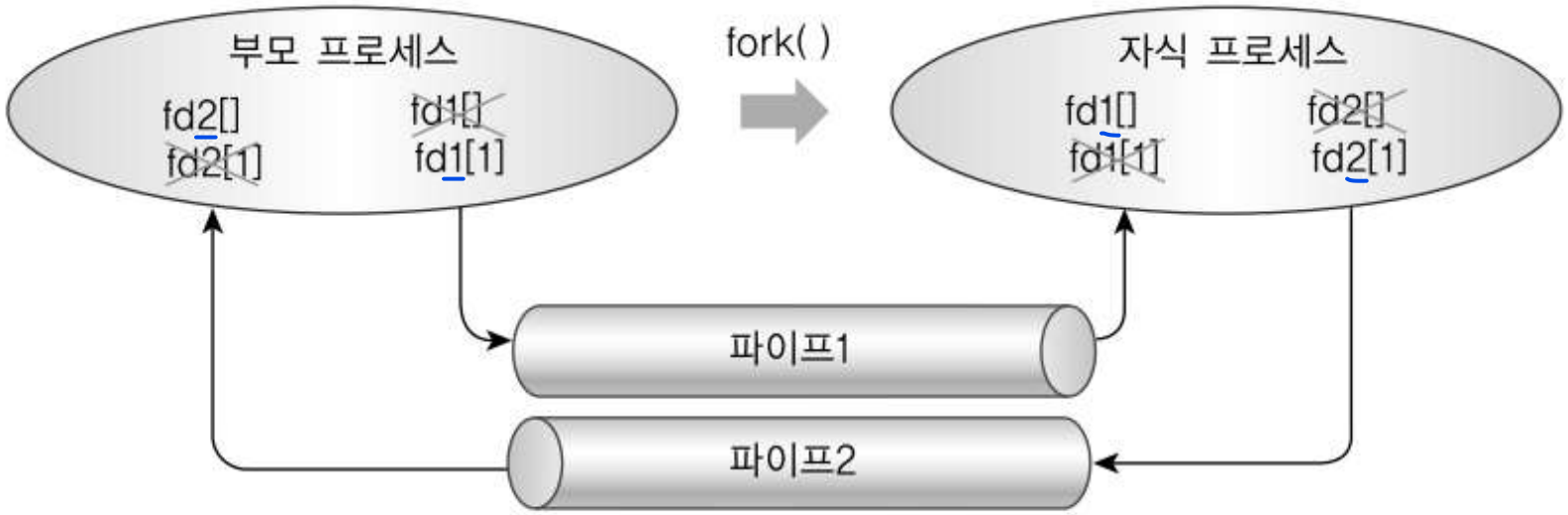
ps -ef 명령 실행

```
# ex9_4.out
root  1568   342   0   2씨 08 ?      0:00 /usr/sbin/in.telnetd
root  1576   342   0   2씨 08 ?      0:00 /usr/sbin/in.telnetd
root  2763   342   0   2씨 10 ?      0:00 /usr/sbin/in.telnetd
root  3782  3781   0 09:40:12 pts/3    0:00 grep telnet
```

양방향 파이프의 활용

양방향 통신

- 파이프는 기본적으로 단방향이므로 양방향 통신을 위해서는 파이프를 2개 생성한다.



[그림 9-4] 양방향 통신 개념도

[예제 9-5] 양방향 통신하기

ex9_5.c

```
<sys/wait.h><unistd.h><stdlib.h><stdio.h><string.h>
07  int main(void) {
08      int fd1[2], fd2[2];
09      pid_t pid;
10      char buf[257];
11      int len, status;
12
13      if (pipe(fd1) == -1) {
14          perror("pipe");
15          exit(1);
16      }
17
18      if (pipe(fd2) == -1) {
19          perror("pipe");
20          exit(1);
21      }
22
23      switch (pid = fork()) {
24          case -1 :
25              perror("fork");
26              exit(1);
27              break;
```

파이프 2개를 생성하기 위해 배열2개 선언

파이프 2개 생성

[예제 9-5] 양방향 통신하기

ex9_5.c

```
28          case 0 : /* child */
29             close(fd1[1]);
30             close(fd2[0]);
31             write(1, "Child Process:", 15);
32             len = read(fd1[0], buf, 256);
33             write(1, buf, len);
34
35             strcpy(buf, "Good\n");
36             write(fd2[1], buf, strlen(buf));
37             break;
38          default :
39             close(fd1[0]);
40             close(fd2[1]);
41             buf[0] = '\0';
42             write(fd1[1], "Hello\n", 6);
43             //sleep(1);
44             write(1, "Parent Process:", 15);
45             len = read(fd2[0], buf, 256);
46             write(1, buf, len);
47             waitpid(pid, &status, 0);
48             break;
49       }
50
51       return 0;
52  }
```

sleep

sleep

자식 프로세스
-fd1[0]으로 읽기
-fd2[1]로 쓰기

부모 프로세스
-fd1[1]로 쓰기
-fd2[0]으로 읽기

```
# ex9_5.out
Child Process:Hello
Parent Process:Good
```

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 생성해야함

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 -F
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

- FIFO 파일 생성: mkfifo(3)

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

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

  - mode : 접근권한 지정

[예제 9-6] 함수로 FIFO 파일 생성하기

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
int main(void) {
    if(mknod("HAN-FIFO",S_IFIFO|0644, 0)==-1) {
        perror("mknod");
        exit(1);
     }

     if (mkfifo("BIT-FIFO", 0644) == -1) {
        perror("mkfifo");
        exit(1);
      }
     return 0;
}
```

[예제 9-7] FIFO로 데이터 주고 받기 -서버프로그램

ex9_7s.c

```
<sys/types.h><fcntl.h><unistd.h><stdlib.h><stdio.h><string.h>
08  int main(void) {
09      int pd, n;
10      char msg[] = "Hello, FIFO";
11
12      printf("Server =====\n");
13
14      if (mkfifo("./HAN-FIFO1", 0666) == -1) {
15          perror("mkfifo");
16          exit(1);
17      }
18
19      if ((pd = open("./HAN-FIFO1", 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 프로그램

ex9_7c.c

```
<fcntl.h><unistd.h><stdlib.h><stdio.h>...
08  int main(void) {
07      int pd, n;
08      char inmsg[80];
09
10      if ((pd = open("./HAN-FIFO1", O_RDONLY)) == -1) {
11          perror("open");
12          exit(1);
13      }
14
15      printf("Client =====\n");
16      write(1, "From Server :", 13);
17
18      while ((n=read(pd, inmsg, 80)) > 0)
19          write(1, inmsg, n);
20
21      if (n == -1) {
22          perror("read");
23          exit(1);
24      }
25
26      write(1, "\n", 1);
27      close(pd);
28
29      return 0;
30  }
```

서버측에서 생성한 FIFO 파일열기

서버가 보낸 데이터 읽기

```
# ex9_7s.out
Server =====
To Client : Hello, FIFO
#
```

```
# ex9_7c.out
Client =====
From Server :Hello, FIFO
#
```