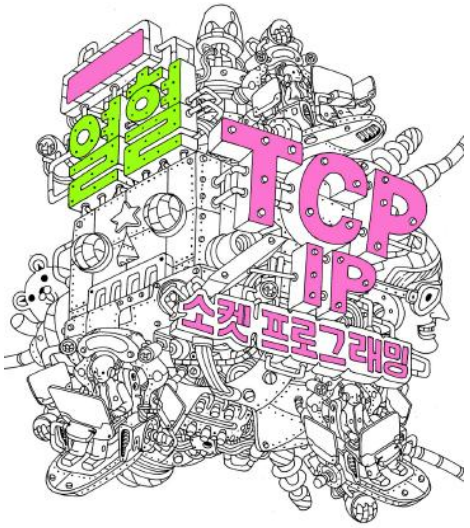
윤성우의 열혈 TCP/IP 소켓 프로그래밍

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

Chapter 11. 프로세스간 통신



Chapter 11-1. 프로세스간 통신의 기본 개념

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

프로세스간 통신의 기본이해

- 프로세스간 통신
  - 두 프로세스 사이에서의 데이터 전달
  - 두 프로세스 사이에서의 데이터 전달이 가능 하려면, 두 프로세스가 함께 공유하는 메모리가 존재해야 한다.

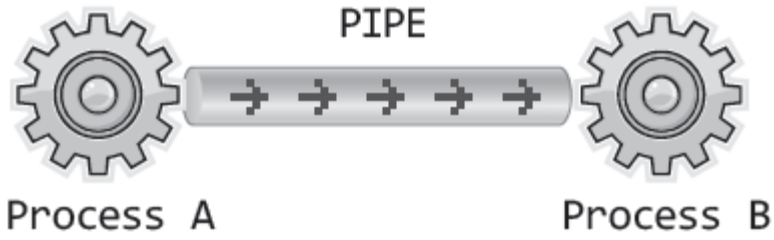
- 프로세스간 통신의 어려움
  - 모든 프로세스는 자신만의 메모리 공간을 독립적으로 구성한다.
  - 즉, A 프로세스는 B 프로세스의 메모리 공간에 접근이 불가능하고, B 프로세스는 A 프로세스의 메모리 공간 접근이 불가능하다.
  - 따라서 운영체제가 별도의 메모리 공간을 마련해 줘야 프로세스간 통신이 가능하다.

파이프 기반의 프로세스간 통신

```
#include <unistd.h>

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

➜ 성공 시 0, 실패 시 -1 반환

- filedes[0] 파이프로부터 데이터를 수신하는데 사용되는 파일 디스크립터가 저장된다. 즉, filedes[0]는 파이프의 출구가 된다.
- filedes[1] 파이프로 데이터를 전송하는데 사용되는 파일 디스크립터가 저장된다. 즉, filedes[1]은 파이프의 입구가 된다.

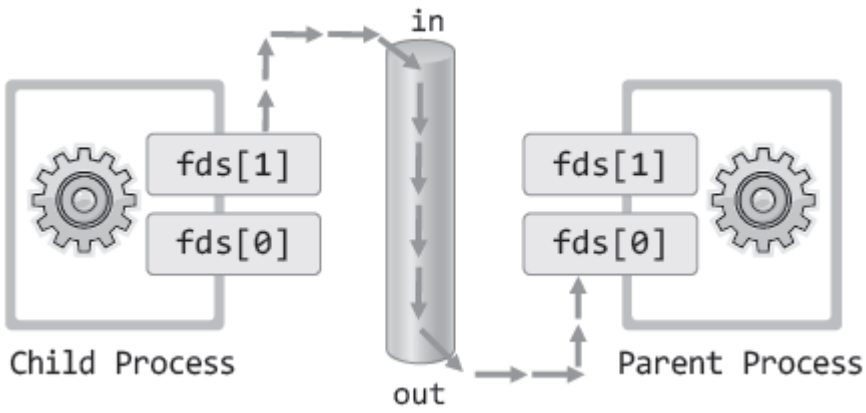


위의 함수가 호출되면, **운영체제는 서로 다른 프로세스가 함께 접근할 수 있는 메모리 공간을 만들고**, 이 공간의 접근에 사용되는 파일 디스크립터를 반환한다.

파이프 생성의 예

예제 pipe1.c

```
int main(int argc, char *argv[])
{
    int fds[2];
    char str[]="Who are you?";
    char buf[BUF_SIZE];
    pid_t pid;

    pipe(fds);
    pid=fork();
    if(pid==0)
    {
        write(fds[1], str, sizeof(str));
    }
    else
    {
        read(fds[0], buf, BUF_SIZE);
        puts(buf);
    }
    return 0;
}
```



핵심은 파이프의 생성과 디스크립터의 복사에 있다!

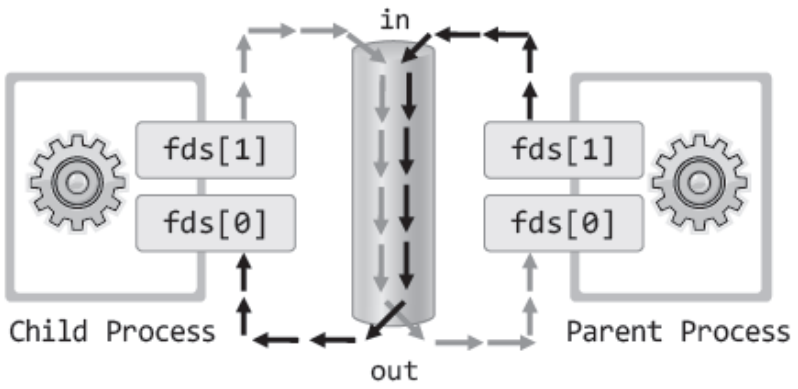
실행결과

```
root@my_linux:/tcpip# gcc pipe1.c -o pipe1
root@my_linux:/tcpip# ./pipe1
Who are you?
```

프로세스간 양방향 통신: 잘못된 방식

예제 pipe2.c

```
int main(int argc, char *argv[])
{
    int fds[2];
    char str1[]="Who are you?";
    char str2[]="Thank you for your message";
    char buf[BUF_SIZE];
    pid_t pid;

    pipe(fds);
    pid=fork();
    if(pid==0)
    {
        write(fds[1], str1, sizeof(str1));
        sleep(2);
        read(fds[0], buf, BUF_SIZE);
        printf("Child proc output: %s \n", buf);
    }
    else
    {
        read(fds[0], buf, BUF_SIZE);
        printf("Parent proc output: %s \n", buf);
        write(fds[1], str2, sizeof(str2));
        sleep(3);
    }
    return 0;
}
```

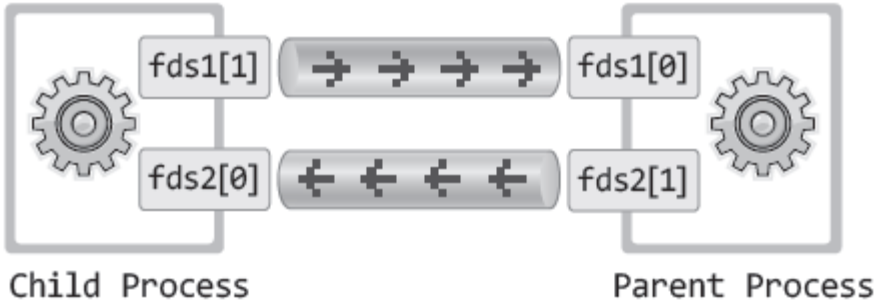


하나의 파이를 이용해서 양방향 통신을 하는 경우, 데이터를 쓰고 읽는 타이밍이 매우 중요해진다. 그런데 이를 컨트롤 하는 것은 사실상 불가능하기 때문에 이는 적절한 방법이 될 수 없다. 왼쪽의 예제에서 sleep 함수의 호출문을 주석처리 해 버리면 문제가 있음을 쉽게 확인할 수 있다.

프로세스간 양방향 통신: 적절한 방식

예제 pipe3.c

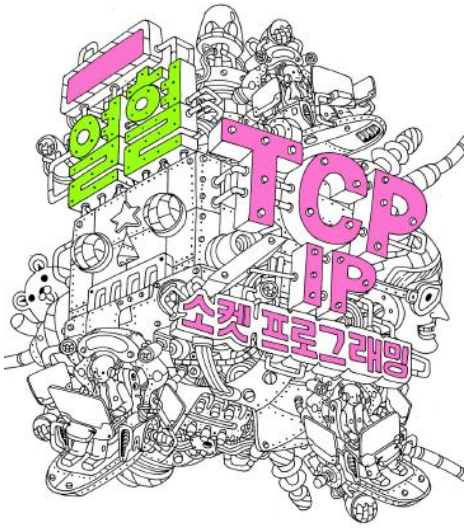
```
int main(int argc, char *argv[])
{
    int fds1[2], fds2[2];
    char str1[]="Who are you?";
    char str2[]="Thank you for your message";
    char buf[BUF_SIZE];
    pid_t pid;

    pipe(fds1), pipe(fds2);
    pid=fork();
    if(pid==0)
    {
        write(fds1[1], str1, sizeof(str1));
        read(fds2[0], buf, BUF_SIZE);
        printf("Child proc output: %s \n", buf);
    }
    else
    {
        read(fds1[0], buf, BUF_SIZE);
        printf("Parent proc output: %s \n", buf);
        write(fds2[1], str2, sizeof(str2));
        sleep(3);
    }
    return 0;
}
```



양방향 통신을 위해서는 이렇듯 두 개의 파이프를 생성해야 한다. 그래야 입출력의 타이밍에 따라서 데이터의 흐름이 영향을 받지 않는다.

실행결과

```
root@my_linux:/tcpip# gcc pipe3.c -o pipe3
root@my_linux:/tcpip# ./pipe3
Parent proc output: Who are you?
Child proc output: Thank you for your message
```



Chapter 11-2. 프로세스간 통신의 적용

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

메시지를 저장하는 형태의 에코 서버

예제 echo_storeserv.c의 핵심코드

```
pipe(fds);
pid=fork();
if(pid==0)
{
    FILE * fp=fopen("echomsg.txt", "wt");
    char msgbuf[BUF_SIZE];
    int i, len;

    for(i=0; i<10; i++)
    {
        len=read(fds[0], msgbuf, BUF_SIZE);
        fwrite((void*)msgbuf, 1, len, fp);
    }
    fclose(fp);
    return 0;
}
```

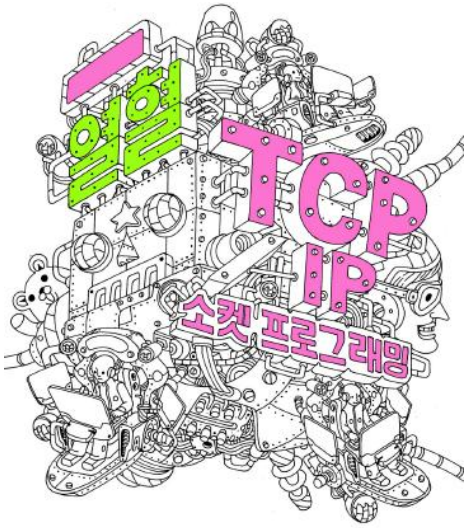
서버에서 처음으로 생성하는 자식 프로세스

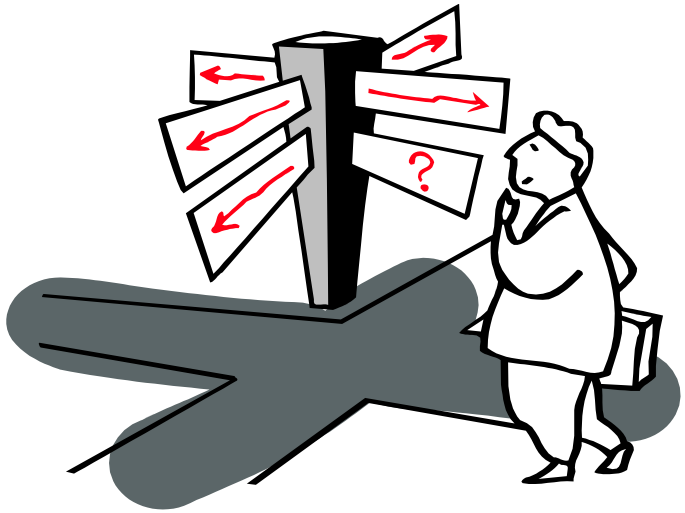
파이프를 생성하고 자식 프로세스를 생성해서, 자식 프로세스가 파이프로부터 데이터를 읽어서 저장하도록 구현되어 있다.

```
clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr, &adr_sz);
. . . .
pid=fork();
if(pid==0)
{
    close(serv_sock);
    while((str_len=read(clnt_sock, buf, BUF_SIZE))!=0)
    {
        write(clnt_sock, buf, str_len);
        write(fds[1], buf, str_len);
    }

    close(clnt_sock);
    puts("client disconnected...");
    return 0;
 }
 else
     close(clnt_sock);
```

서버에서 연결 허용 시마다 생성하는 자식 프로세스

accept 함수 호출 후 fork 함수호출을 통해서 파이프의 디스크립터를 복사하고, 이를 이용해서 이전에 만들어진 자식 프로세스에게 데이터를 전송한다.





Chapter 11이 끝났습니다. 질문 있으신지요?