9장: Pipe, FIFO

이번 장에는 IPC (Interprocess Communication)에 대해서 살펴 본다. 즉, 프로세스들이 서로 통신 하는 방법에 대해 알아본다. 유닉스 시스템은 pipe, FIFO, message queue, semaphore, shared memory, socket 등 여러 가지 프로세스간 통신 방법을 지원하는데, 여기서는 pipe와 FIFO에 대해서 중점적으로 살펴 볼 것이다.

1 pipe를 이용한 client-server 예제

1.1. 목적

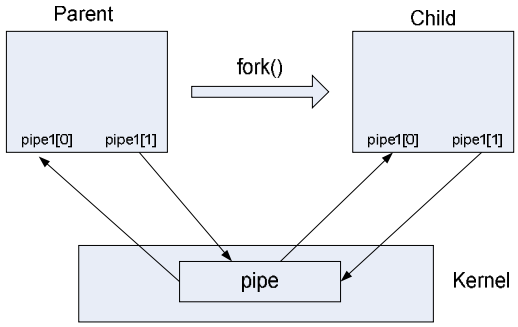
프로세스 간의 통신에서 pipe가 어떻게 이용될 수 있는 지를 알아보고, client-server 예제 프로그램을 작성하여 pipe의 기능을 직접 시험해본다.

1.2. 기초 지식

1.2.1. pipe()

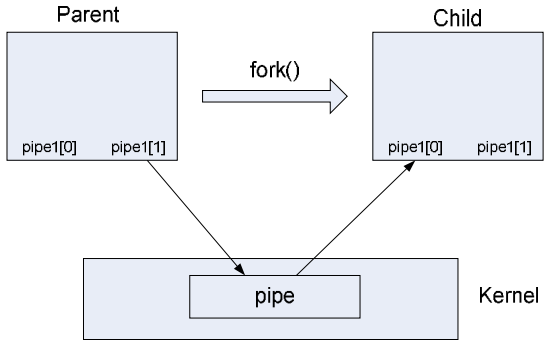
```
#include <unistd.h>

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

pipe란 두개의 프로세스 간에 데이터를 전달할 수 있는 매개체이다. pipe() 함수를 사용하여 pipe를 만들 수 있으며, pipe 생성에 성공하면 만들어진 파이프에 해당하는 파일 식별자를 얻을 수 있다. 이렇게 pipe를 생성한 이후, fork()를 호출하여 child 프로세스를 생성한다. parent 프로세스와 child 프로세스 사이에는 파일 식별자를 공유하므로, 이 둘은 같은 pipe를 통해 서로 통신할 수 있는 것이다. 아래 [그림 9.1]은 parent가 pipe1[0]과 pepe2[0]을 생성하고 fork() 함수를 실행한 상태이다.



[그림 9.1] fork() 호출 후의 pipe 모습

fork() 호출 후에, 데이터가 흘러갈 방향을 정하기 위해 필요가 없는 식별자는 닫아야 한다. 원하는 방향이 parent 프로세스에서 child 프로세스로 데이터가 흘러가는 것이라면 parent 프로세스에서 pipe1[0], child 프로세스에서 pipe1[1]을 닫는다. ([그림9.2])



[그림 9.2] parent 프로세스에서 child 프로세스로의 데이터 흐름

1.2.2. pipe의 단점

pipe는 half-duplex 특성을 가지므로, 데이터는 오직 한 방향으로만 흐른다. 또한 pipe는 일반적으로 하나의 프로세스에 의해 만들어지고, fork()를 통해 생성된 child 프로세스와의 통신에 이용된다. 이러한 특성 때문에 pipe는 공통의 parent 프로세스를 갖는 프로세스들 사이에서만 사용이 가능하다.

1.2.3. popen()

popen() 함수는 하나의 파이프를 만들고 다른 프로세스로 하여금 그 파이프를 통해 읽고 쓸 수 있도록 초기화하는 역할을 한다.

```
#include <stdio.h>

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

                                    Returns: file pointer if OK, NULL on error
```

(1) *command*는 쉘(shell) 프로그램에 의해 처리되며, PATH 환경변수를 이용하여 *command*의 위치를 알아낼 수 있다. pipe는 호출하는 프로세스와 특정한 command 사이에서 만들어진다.

(2) popen()에 의해 리턴되는 값은 표준 I/O FILE 포인터이다. 이것은 문자열 포인터 *type*에 의하여 입력 또는 출력으로 사용된다.

(3) *type*이 r이면, 호출하는 프로세스는 command의 표준 출력을 읽는다.

(4) *type*이 w이면, 호출하는 프로세스는 command의 표준 입력으로 쓴다.

1.2.4. pclose()

```
#include <stdio.h>

int pclose(FILE *stream);
                                    Returns: termination status of shell or -1 on error
```

pclose()은 popen()에 의해 만들어진 표준 I/O stream을 닫는다. command가 끝나기를 기다리고 쉘의 종료 상태를 리턴한다.

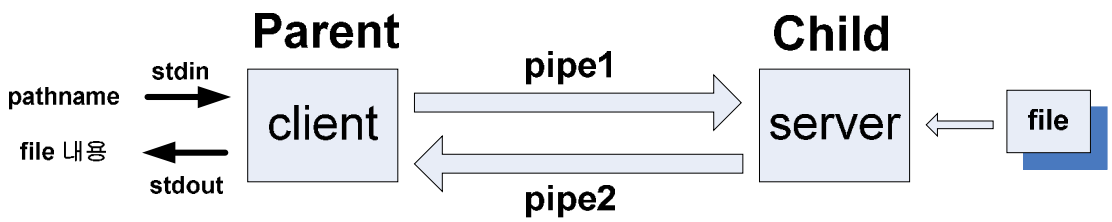
1.3. 수행 절차

1.3.1. 수행 예제

pipe를 사용하여 client에서 pathname을 입력하였을 때, server로부터 그 pathname에 해당하는 file의 내용을 받아서 화면에 출력한다. client와 server 사이에 2개의 pipe로 생성하여 통신이 가능하도록 한다.

1.3.2. 수행 방법

본 예제에서는 먼저 pipe 2개를 생성한 후, fork()를 호출하여 parent와 child 프로세스를 생성한다. 그리고 pipe1을 통해 parent 프로세스에서 child 프로세스로 데이터를 전송하고, pipe2를 통해 child 프로세스에서 parent 프로세스로 데이터를 전송한다. 이를 통해 parent 프로세스와 child 프로세스 사이에 양방향 통신이 가능하게 된다. client에서는 표준입력으로 pathname을 받아 pipe1을 통해 server에게 전달 하고, server로부터 pipe2를 통해 받은 파일의 내용을 표준출력으로 내보낸다. server에서는 client로부터 전달 받은 pathname 파일을 읽고, 그 내용을 pipe2를 통해 client로 전달한다.



[그림 3] 프로그램 구조

1.4. 소스 프로그램

[프로그램 9.1] pipe.c

```
 1 #include <stdio.h>
 2 #include <fcntl.h>
 3 #include <errno.h>
 4 #include <sys/types.h>
 5
 6 #define MAXLINE 4096
 7 #define STDOUT_FILENO 1
 8
 9 void client(int, int), server(int, int);
10
11 int main(int argc, char *argv[])
12 {
13       int pipe1[2], pipe2[2];
14       pid_t childpid;
15
16       pipe(pipe1);
17       pipe(pipe2);
18
19       if((childpid=fork())==0) /* child */
20       {
21                close(       );
22                close(       );
23
24                server(       ,       )
25                exit(0);
26       }
27       /* parent */
28       close(       );
29       close(       );
30
31       client(       ,       );
32
33       waitpid(childpid, NULL, 0); /* wait for child to terminate */
34       exit(0);
35 }
36
37 void client(int readfd, int writefd)
38 {
39       size_t len;
40       size_t n;
41       char buff[MAXLINE];
42
43       /* read pathname */
44       fgets(buff, MAXLINE, stdin);
45       len = strlen(buff);
46       if(buff[len-1] == '₩n')
```

```
47                len--;
48
49        write(writefd, buff, len);
50
51        while((n=read(readfd, buff, MAXLINE))>0)
52                write(STDOUT_FILENO, buff, n);
53 }
54
55 void server(int readfd, int writefd)
56 {
57        int fd;
58        size_t n;
59        char buff[MAXLINE+1];
60
61        if((n=read(readfd, buff, MAXLINE))==0)
62        {
63                printf("end-of-file");
64                exit(0);
65        }
66        buff[n]='₩0';
67
68        if((fd=open(buff,O_RDONLY))<0)
69        {
70                snprintf(buff+n, sizeof(buff)-n, ": can't open, %s₩n", strerror(errno));
71                n=strlen(buff);
72                write(writefd, buff, n);
73        }
74        else
75        {
76                while((n=read(fd, buff, MAXLINE))>0)
77                        write(writefd, buff, n);
78                close(fd);
79        }
80 }
```

1.5. 수행 결과

```
[piero00@Myth pipe]$ ./pipe_example          <= 프로그램 실행
/etc/hosts                                   <= pathname 입력
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1                Myth localhost.localdomain localhost
```

2 FIFO를 이용한 client-server example

2.1. 목적

앞의 예제에서 pipe의 단점 2가지를 살펴보았다. 이번 예제에서는 공통조상을 갖는 프로세스들 사이에서만 사용될 수 있는 pipe의 단점을 극복하는 FIFO에 대해서 알아본다.

2.2. 기초 지식

2.2.1. pipe의 가장 큰 단점

pipe는 공통적인 하나의 부모를 갖고 있는 프로세스 사이의 통신에만 사용될 수 있다. 부모 프로세스가 다른 두 개의 프로세스들 사이에서는 pipe를 생성할 수 없다.

2.2.2. pipe의 한계를 극복하는 FIFO

FIFO는 *first in, first out*을 나타낸다. 유닉스의 FIFO는 pipe와 비슷하다. pipe와 마찬가지로 데이터의 흐름이 한 방향인 half-duplex의 특징을 갖고 있다. 그러나 pipe와 다르게 FIFO는 그것과 관련된 경로명을 갖고 있어서 하나의 FIFO에 서로 관련이 없는 프로세스들이 접속하는 것을 가능하게 한다. 그래서 FIFO는 *named pipe*라고 불리기도 한다.

2.2.3. mkfifo()

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

int mkfifo(const char *pathname, mode_t mode);

                                        Returns : 0 if OK, -1 on error
```

(1) *pathname*은 정상적인 유닉스의 경로명이다. 이것은 FIFO의 이름을 나타낸다.

(2) *mode*는 파일 퍼미션 비트를 나타낸다. open()의 두번째 파라미터와 비슷하다.

(3) mkfifo()는 O_CREAT | O_EXCL을 의미한다. 그것은 새로운 FIFO를 만들거나 이미 FIFO가 존재한다면 EEXIT의 에러를 리턴한다.

(4) 새로운 FIFO를 생성하고 싶지 않다면, mkfifo() 대신에 open()을 호출한다.

2.2.4. FIFO의 특징

한번 FIFO가 만들어지면 open()이나 fopen() 같은 표준 I/O open 함수를 사용하여 읽기나 쓰기로 열어야 한다. FIFO는 오직 읽기모드나 쓰기모드로 열어야 한다. FIFO가 half-duplex이기 때문에 읽고 쓰기 모드로 열 수 없다.

쓰기 모드로 pipe나 FIFO를 열면, 항상 데이터를 뒤에서 추가하게 되고, 읽기모드로 연다면, 항상 pipe나 FIFO의 시작지점에서 읽게 된다. 그래서 pipe나 FIFO에 대해 lseek()를 호출한다면, ESPIPE 에러가 리턴 된다.

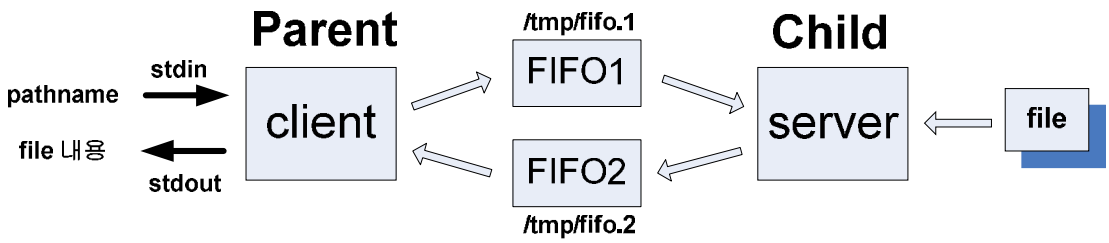
2.3. 수행 절차

2.3.1. 수행 예제

앞의 예제처럼 client에서 pathname 입력받아 server에게 전달하고, server는 그 pathname에 해당하는 file의 내용을 읽어서 다시 client에게 전달하는 프로그램을 만들어 보자. 하지만 앞의 예제와는 다르게 pipe 대신에 FIFO를 사용하여 보자. 이 예제에서도 마찬가지로 client와 server 사이에 2개의 FIFO가 필요하다.

2.3.2. 수행 방법

본 예제에서는 먼저 mkfifo()를 이용하여 FIFO 2개를 생성한다. FIFO1의 pathname은 "/tmp/fifo.1", FIFO2의 pathname은 "/tmp/fifo.2"로 정한다. 그리고 fork()를 호출하여 parent와 child process를 생성한다. pipe와는 다르게 FIFO는 pathname을 이용하여 접근하므로, open()을 이용해 식별자를 얻는다. 이와 같은 접근 방법으로 인해 관련이 없는 프로세스들도 FIFO를 통해 통신을 할 수 있다. 이를 통해, parent와 child 사이에 양방향 통신이 가능하게 된다. client에서의 표준 입출력과 server에서의 file을 읽는 과정은 앞의 예제와 동일한 방법으로 이루어진다.



[그림 4] 프로그램 구조

2.4. 소스 프로그램

[프로그램 9.2] fifo.c

```
 1 #include <stdio.h>
 2 #include <fcntl.h>
 3 #include <errno.h>
 4 #include <sys/types.h>
 5 #include <sys/stat.h>
 6
 7 #define MAXLINE 4096
 8 #define STDOUT_FILENO 1
 9 #define FIFO1 "/tmp/fifo.1"
10 #define FIFO2 "/tmp/fifo.2"
11 #define FILE_MODE (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
12
13 void client(int, int), server(int, int);
14
15 int main(int argc, char *argv[])
16 {
17      int readfd, writefd;
18      pid_t childpid;
19
20      if ((mkfifo(FIFO1, FILE_MODE) < 0) && (errno != EEXIST))
21      {
22               printf("can't create %s", FIFO1);
23               exit(1);
24      }
25      if ((mkfifo(FIFO2, FILE_MODE) < 0) && (errno != EEXIST))
26      {
27               unlink(FIFO1);
28               printf("can't create %s", FIFO2);
29               exit(1);
30      }
31      if ((childpid=fork())==0)
32      {   /* child */
33               readfd = open(        ,        , 0);
34               writefd = open(        ,        , 0);
35
36               server(readfd, writefd);
37               exit(0);
38      }
39
40      /* parent */
41      writefd = open(        ,        , 0);
42      readfd = open(        ,        , 0);
43
44      client(readfd, writefd);
45
46      waitpid(childpid, NULL, 0); /* wait for child to terminate */
```

```
47
48      close(readfd);
49      close(writefd);
50
51      unlink(FIFO1);
52      unlink(FIFO2);
53      exit(0);
54 }
55
56 void client(int readfd, int writefd)
57 {
58      size_t len;
59      size_t n;
60      char buff[MAXLINE];
61
62      /* read pathname */
63      fgets(buff, MAXLINE, stdin);
64      len = strlen(buff);
65      if(buff[len-1] == '₩n')
66               len--;
67
68      write(writefd, buff, len);
69
70      while((n=read(readfd, buff, MAXLINE))>0)
71               write(STDOUT_FILENO, buff, n);
72 }
73
74 void server(int readfd, int writefd)
75 {
76      int fd;
77      size_t n;
78      char buff[MAXLINE+1];
79
80      if((n=read(readfd, buff, MAXLINE))==0)
81      {
82               printf("end-of-file");
83               exit(0);
84      }
85      buff[n]='₩0';
86
87      if((fd=open(buff,O_RDONLY))<0)
88      {
89               snprintf(buff+n, sizeof(buff)-n, ": can't open, %s₩n", strerror(errno));
90                        n=strlen(buff);
91               write(writefd, buff, n);
92      }
93      else
94      {
95               while((n=read(fd, buff, MAXLINE))>0)
96                        write(writefd, buff, n);
97               close(fd);
```

```
98      }
99 }
```

2.5. 수행 결과

```
[piero00@Myth FIFO]$ ./fifo_example                    <= 프로그램 실행
/etc/hosts                                              <= pathname 입력
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1                Myth localhost.localdomain localhost
```

3 연습 문제

1. 예제 1에서 pipe()를 이용하여 작성한 client-server 프로그램을 popen()을 이용하여 수정한다. 그리고, Server 프로그램에서 file의 내용을 보일 때 “cat” 유틸리티를 사용한다.

2. FIFO를 사용하여 서로 다른 조상에 속해 있는 프로세스들끼리 통신이 가능하도록 한다. Client에서 프로세스 아이디와 내용을 알고자 하는 파일명을 server에게 보내고 server는 해당 내용을 client에게 전달한다. server에서 client의 명령을 받는 부분과 client에서 server로부터 파일 내용을 받는 부분을 FIFO로 구현한다.