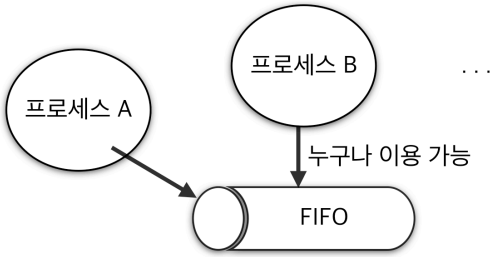
Spring 2019 Operating Systems
Lab 09-1: FIFO

0. 과제 수행 전 유의사항

프로그래밍은 자기 손으로 작성해야 한다. 아이디어를 서로 공유하고 도움을 받는 것은 좋으나, 다른 사람의 코드를 그대로 가져다 쓰는 것은 자신에게도 손해이고 다른 사람에게 피해를 끼치는 결과를 낳게되므로 허용하지 않는다. 과제 체크 시 두 사람의 코드가 유사하면, 두 사람 모두에게 과제 점수를 부여할 수 없으니 유의바란다.

1. 소개

파이프는 강력한 통신 수단임에도 불구하고 부모 프로세스와 자식 프로세스 사이에서만 통신 할 수 있고, 프로세스가 종료되면 사라져 영구적이지 못하다는 문제점이 있다. 이런 문제점을 해결한 통신 수단이 FIFO(이름 있는 파이프, "named pipes")다. FIFO에는 이름이 부여되므로 부모 자식 관계가 아닌 프로세스에서도 파이프 이름만 알면 이용할 수 있고, 삭제하지 않는 한 영구적으로 존재한다.



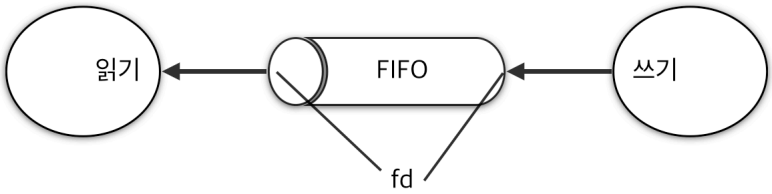
FIFO는 mkfifo( ) 함수로 생성되며, 다음과 같이 pathname 이름을 갖는 파이프가 생성되고 접근 권한은 mode를 갖는다.

```
mkfifo 함수
기능
        FIFO를 생성한다.
기본형
        int mkfifo(const char *pathname, mode_t mode);
        pathname: 생성할 FIFO의 이름
        mode: 생성될 FIFO의 접근 권한
반환값
        성공: 0
        실패: -1
헤더파일
        <sys/types.h>
        <sys/stat.h>
```

그리고 mkfifo( )로 생성된 FIFO인 fifo를 사용하기 위해서는 다음과 같이 open을 이용해 열어야 한다.

```
mkfifo("fifo", 0666);
fd = open("fifo", O_RDONLY);
```

2. 예제



다음은 프로세스 A에서 "fifo"라는 파일의 FIFO를 생성하여 fifo를 통해 전달되는 데이터를 읽고, 프로세스 B에서 이미 생성된 fifo를 열어 fifo에 데이터를 쓰면 두 프로세스간에 통신이 이루어진다.

[lab8_1.c]

```c
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

#define SIZE 128
#define FIFO "fifo"

main(int argc, char *argv[])
{
    int fd;
    char buffer[SIZE];

    if (mkfifo(FIFO, 0666) == -1) {
        perror("mkfifo failed");
        exit(1);
    }

    if ((fd = open(FIFO, O_RDWR)) == -1) {
        perror("open failed");
        exit(1);
    }

    while (!0) {
        if (read(fd, buffer, SIZE) == -1) {
            perror("read failed");
            exit(1);
        }
        if (!strcmp(buffer, "quit")) {
            exit(0);
        }
        printf("receive message %s\n", buffer);
    }
}
```

이 프로그램에서는 읽기 동작만 수행하는데,

if ((fd = open(FIFO, O_RDWR)) == -1) {

이 라인처럼 읽기/쓰기 모두로 설정한 이유는 무엇 때문일까? 동일한 FIFO를 이용하는 읽기 전용 프로세스와 쓰기 전용 프로세스가 실행되면 쓰기 전용 프로세스가 FIFO에 데이터를 쓸 때까지는 읽기 전용 프로세스는 무한정 기다리게 된다.

그러므로 FIFO에 데이터가 쓰여져야 read( ) 부분이 호출된다. 이 때, 쓰기 전용 프로세스가 종료되어 FIFO에 데이터를 쓸 프로세스가 없는 상황에서 읽기 전용 프로세스가 read( ) 를 호출하면 바로 0을 반환한다. 결국, 불필요한 반복 실행을 하게 된다. 그러나, O_RDWR 로 설정하면 쓰기 전용 프로세스가 종료되더라도 FIFO에 데이터를 쓸 프로세스가 있다고 인식하므로, read( ) 호출은 FIFO에 데이터가 전달될 때까지는 기다리게 되어 불필요한 반복을 하지 않기 때문이다.

3. 과제

[lab8_2.c]

```c
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

#define SIZE 128
#define FIFO "fifo"

main(int argc, char *argv[])
{
    int fd, i;
    char buffer[SIZE];

    if ((fd = open(FIFO, O_WRONLY)) == -1) {
        perror("open failed");
        exit(1);
    }

    for (i = 1; i < argc; i++) {
        strcpy(buffer, argv[i]);
        if (write(fd, buffer, SIZE) == -1) {
            perror("write failed");
            exit(1);
        }
    }

    exit(0);
}
```

이 코드는 프로그램 실행에 추가되는 매개변수를 "fifo" 파일에 쓰는 코드이다.

4. 컴파일 및 실행

$ gcc lab8_1.c -o lab8_1
$ gcc lab8_2.c -o lab8_2

위 두 문장을 실행하면 컴파일된 바이너리 파일 2개가 보일 것이다. 그렇다면, 먼저 lab8_1 을 백그라운드로 실행한다. 참고로 프로그램을 백그라운드로 실행하는 명령은 & 명령이다.

$ ./lab8_1 &

> 만약 오류가 발생한다면, "fifo" 라는 파일이 이미 존재할 가능성이 있다. rm 명령으로 삭제한 다음 실행한다.

lab8_1 을 실행하면 프로세스는 "fifo" 에 데이터가 오기를 기다리고 있는 중이다. 이후 lab8_2 를 실행한다. 단, 매개변수로 메시지를 주어야 한다.

$ ./lab8_2 "My name is OOO" "Really?"

실행과 동시에 백그라운드에서 대기중이던 프로세스가 읽기를 수행하므로 똑같은 문장이 출력된다. 실행은 다음과 같이 하면된다.

```
s_guest@A1409-OSLAB-01:~/lab8$ ./lab8_1 &
[1] 28249
s_guest@A1409-OSLAB-01:~/lab8$ ./lab8_2 "My name is kwp" "Really?"
receive message My name is kwp
receive message Really?
s_guest@A1409-OSLAB-01:~/lab8$ ./lab8_2 "Hello?"
receive message Hello?
s_guest@A1409-OSLAB-01:~/lab8$ ./lab8_2 quit
[1]+  Done                    ./lab8_1
s_guest@A1409-OSLAB-01:~/lab8$ _
```

5. 제출

lab8_1.c, lab8_2.c 를 컴파일하고 실행결과를 보여라. 매개변수는 자유롭게 작성한다. 소스에 주석을 붙여 제출한다.