Lecture 10

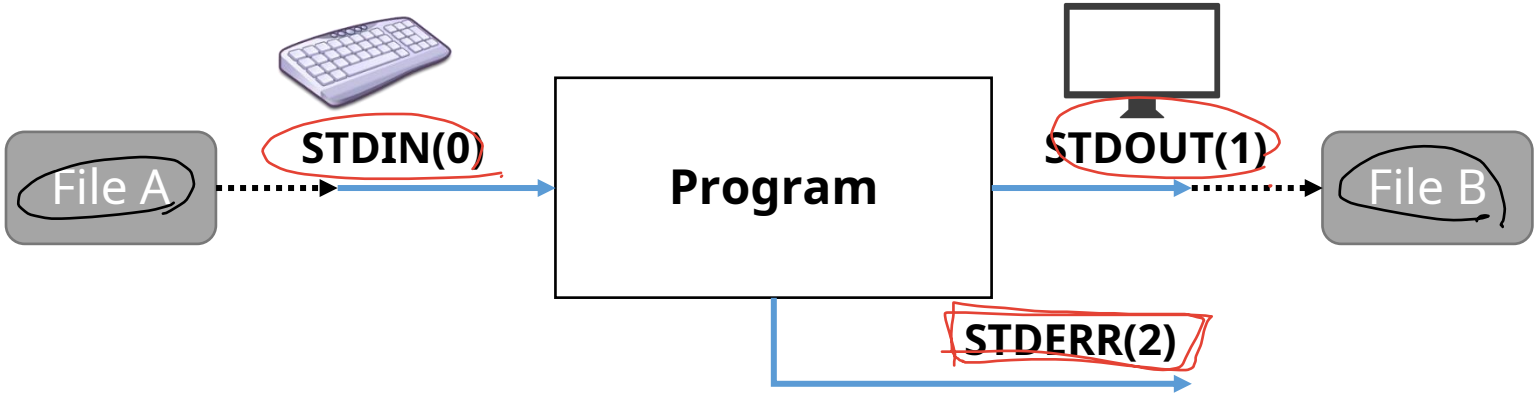
파 이 프

Process A **Process B**

<images from Nitendo>

Redirection

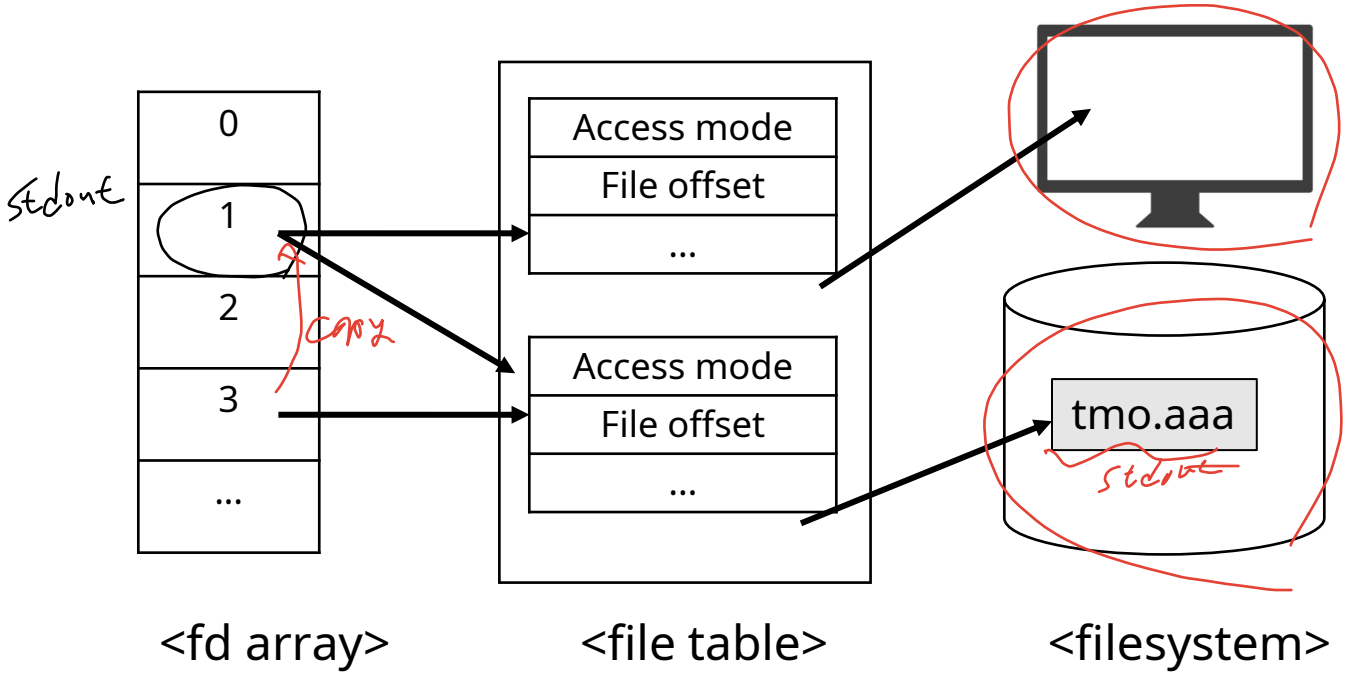
Standard stream



Redirecting to a file

- 프로세스의 standard stream을 파일로 연결

IO Redirection



Stream redirection (shell commands)

- **A > file** : redirect output stream

```
$ ls -al > ls.txt
$ cat ls
total 287044
drwxr-xr-x 1 bluekds bluekds       512 Nov 18 16:16 .
drwxr-xr-x 1 root    root          512 Oct 20 17:21 ..
```

- **A < file** : redirect input stream

```
$ wc -m < ls.txt
2730
```

- **Example**

```
$ wc -m < ls.txt > wc.txt
$ cat wc.txt
2730
```

Outline

- Stream redirection

- **Pipe**
  - Shell commands (|)
  - Anonymous pipe
  - Named pipe

<images from Nitendo>

Pipe

- 두 프로세스 사이의 입출력을 연결 해주는 통로

- 동작 원리
  - 단 방향(one way) 통신
  - First-In First-Out
    - 읽어간 내용은 파이프에 남지 않음

<images from Nitendo>

Pipe (Shell command)

- A | B
  - Process A의 STDOUT을 process B의 STDIN으로 연결
- **Examples**

```
$ ls -al |grep bashrc
-rw-r--r--  1  bluekds  bluekds            3888  Oct  14
01:31 .bashrc
```

```
$ ls |sort
Desktop
Documents
Downloads
eclipse
…
```

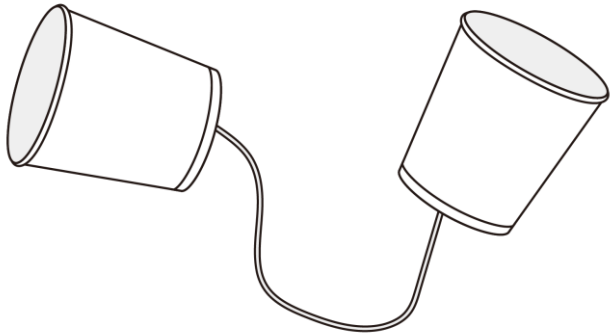
```
$ ls |sort > sort.txt
$ cat sort.txt
```

Anonymous pipe

- **이름이 없는 파이프**
  - 일반적으로 pipe라고 하면, anonymous pipe를 의미

- **부모-자식 프로세스 사이의 통신을 위해 사용 됨**

- **관련 함수**
  - pipe(2)
  - popen(3)
  - pclose(3)



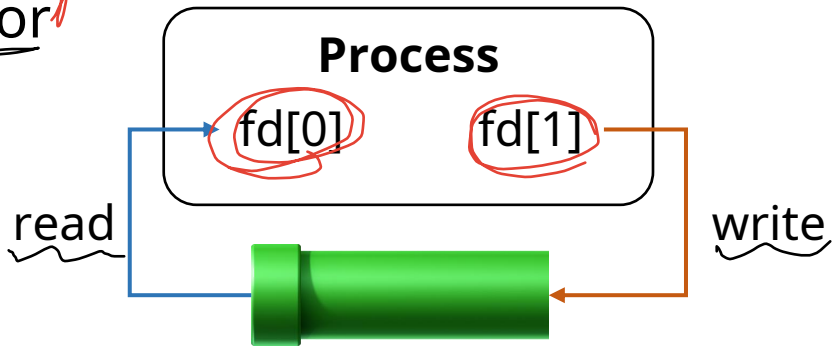
[image form illustAC]

Creating a pipe stream

```
#include <unistd.h>

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

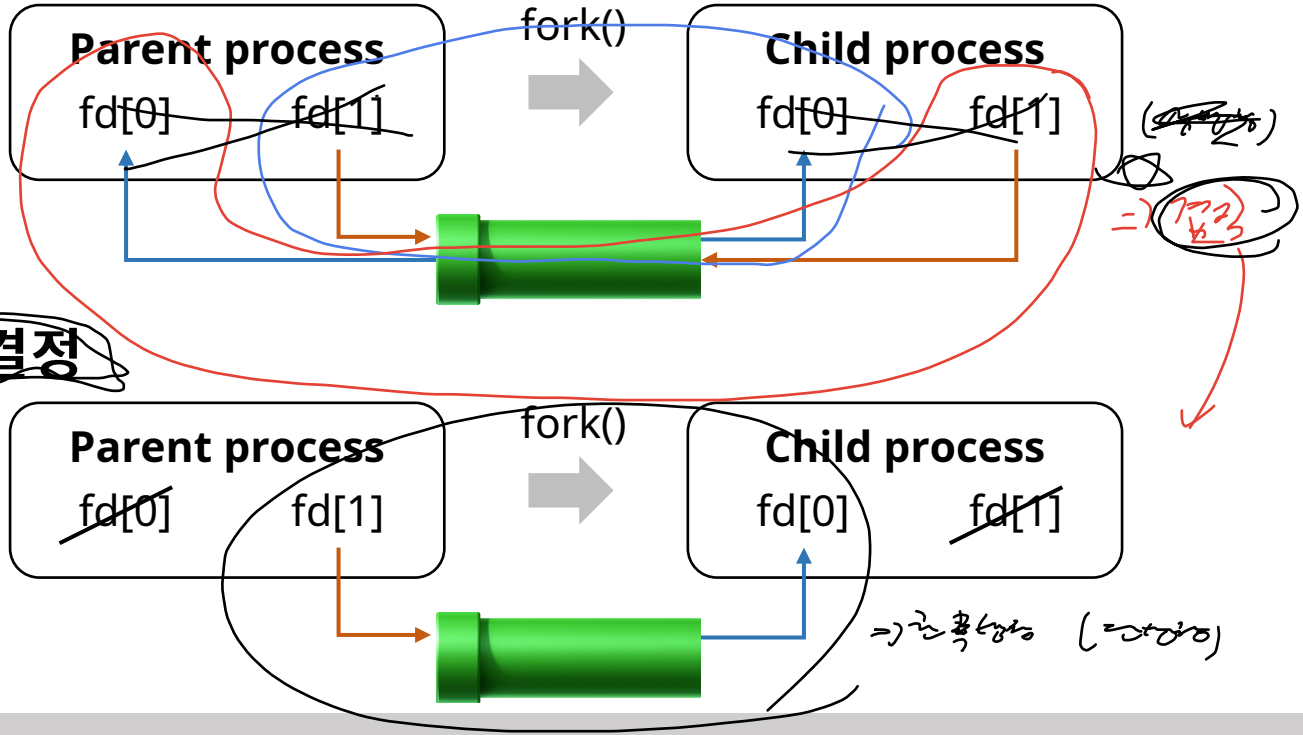
```
$ man -s 2 pipe
```

- **Pipe stream을 열고, 읽기/쓰기 FD를 생성/반환**
- **pipefd[0]** : 읽기 전용 file descriptor
- **pipefd[1]**: 쓰기 전용 file descriptor
- Return
  - 0: Success, -1: error



P-C communication with a pipe

- Pipe stream 생성
- fork()
- 통신 방향 결정



Communication with a pipe

[code link]

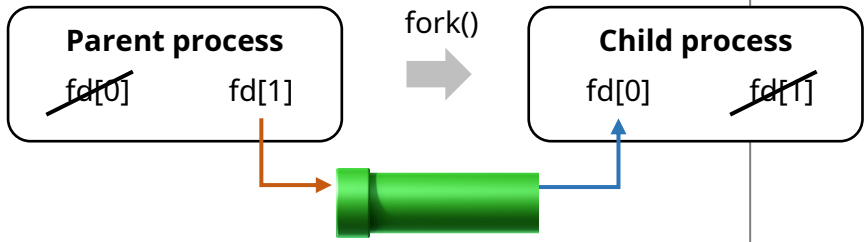
```c
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(void) {
    int fd[2];
    pid_t pid;
    char buf[257];
    int len, status;

    if (pipe(fd) == -1) { perror("pipe"); exit(1); }
...
```

Communication with a pipe (Cont.)

[code link]

```
...
    switch (pid = fork()) {
        case -1 :
            perror("fork"); exit(1); break;
        case 0 : /* child */
            close(fd[1]);
            write(1, "Child Process:", 15);
            len = read(fd[0], buf, 256);
            write(1, buf, len);
            close(fd[0]); break;
        default :
            close(fd[0]);
            buf[0] = '\0';
            write(fd[1], "Test Message\n", 14);
            close(fd[1]);
            waitpid(pid, &status, 0); break;
    }

    return 0;
}
```

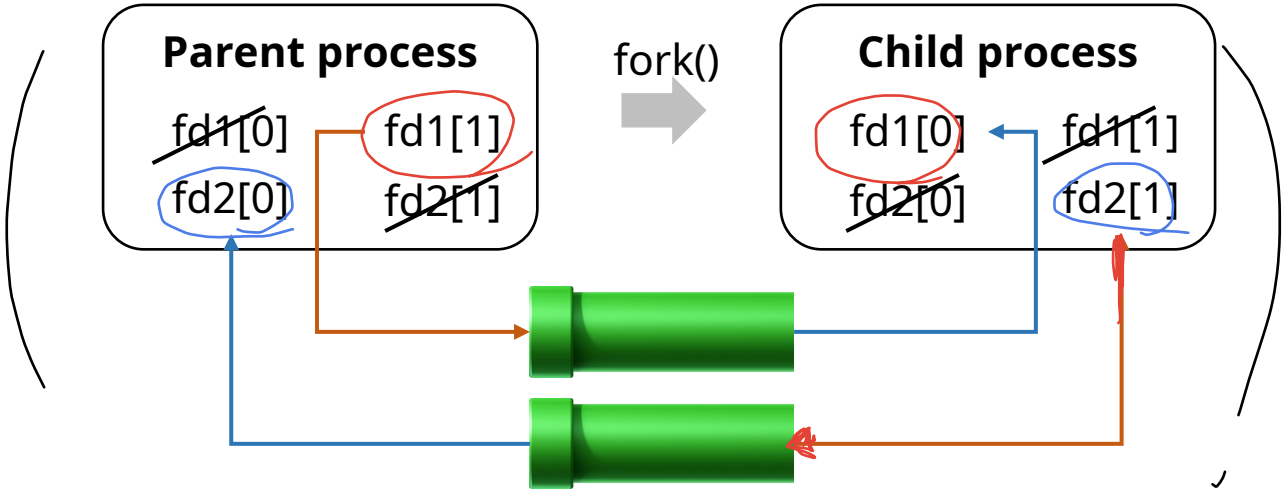
stdout



$ ./pipe.out
Child Process: Test Message

Bidirectional communication

- 양방향 통신을 위해서는 두개의 pipe stream이 필요함



Bidirectional communication

[code link]

```c
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(void) {
    int fd1[2], fd2[2];
    pid_t pid;
    char buf[257];
    int len, status;

    if (pipe(fd1) == -1) { perror("pipe"); exit(1); }
    if (pipe(fd2) == -1) { perror("pipe"); exit(1); }
...
```

Bidirectional communication (Cont.)

[code link]

```
...
    switch (pid = fork()) {
        case -1 :
            perror("fork"); exit(1); break;
        case 0 : /* child */
            close(fd1[1]);
            close(fd2[0]);
            write(1, "Child Process:", 15);
            len = read(fd1[0], buf, 256);
            write(1, buf, len);

            strcpy(buf, "Good\n");
            write(fd2[1], buf, strlen(buf));
            break;
...
```

Bidirectional communication (Cont.)

[code link]

```
...
        default :
            close(fd1[0]);
            close(fd2[1]);
            buf[0] = '\0';
            write(fd1[1], "Hello\n", 6);

            write(1, "Parent Process:", 15);
            len = read(fd2[0], buf, 256);
            write(1, buf, len);
            waitpid(pid, &status, 0);
            break;
    }

    return 0;
}
```

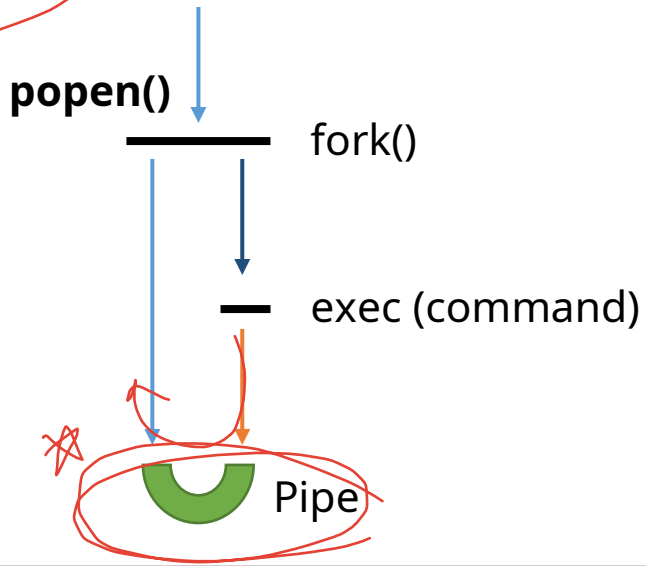
$ ./bidirectional.out
Child Process: Hello
Parent Process : Good

Opening/Closing a pipe stream

```
#include <stdio.h>

FILE *popen (const char *command, const char *type);
int pclose (FILE *stream);
```

```
$ man -s 3 popen
```

- command : 실행할 명령어
- type : pipe access mode
  - "r" : 읽기
  - "w" : 쓰기
- Return : file pointer for the stream



popen(3)/pclose(3)

[code link]

```c
#include <stdlib.h>
#include <stdio.h>

int main(void) {
    FILE *fp;
    char buf[256];

    fp = popen("date", "r");
    if (fp == NULL) {
        fprintf(stderr, "popen failed\n");
        exit(1);
    }

    if (fgets(buf, sizeof(buf), fp) == NULL) {
        fprintf(stderr, "No data from pipe!\n");
        exit(1);
    }

    printf("line : %s\n", buf);
    pclose(fp);

    return 0;
}
```

$ ./popen.out
line : Mon Nov 8 17:29:30 KST 2021

Outline

- **Stream redirection**

- **Pipe**
  - Shell commands (|)
  - Anonymous pipe
  - Named pipe

<images from Nitendo>

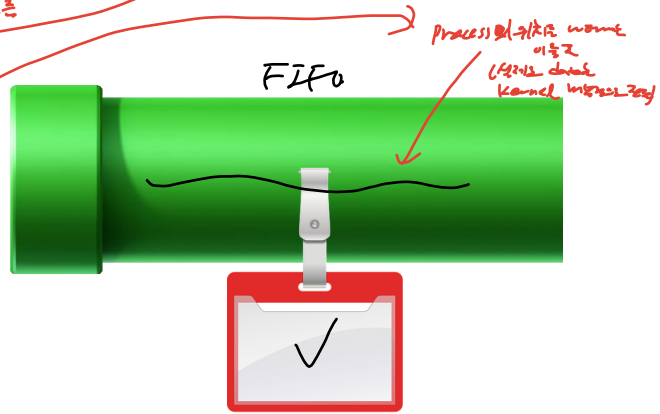
Lecture 10-2.

Named pipe

Pixabay로부터 입수된 Settergren님의 이미지 입니다.

Named Pipe

- 이름이 있는 파이프 (특수 파일)
  - FIFO 파일 (man -s 7 FIFO)
- 독립적인 프로세스 사이의 통신을 위해 사용 가능
  - FIFO 파일을 미리 만들어 두어야 함
  - Kernel 내부적으로 데이터 전달
    - 파일에 내용이 쓰여지진 않음



Pixabay로부터 입수된 Settergren님의 이미지 입니다.

Creating a FIFO file

- **mkfifo** [-m mode] path (shell command)

```
$ mkfifo -m 0644 pipeFile
$ ls -l pipeFile
prw-r--r-- 1 bluekds bluekds 0 Nov 21 21:55 pipeFile
```

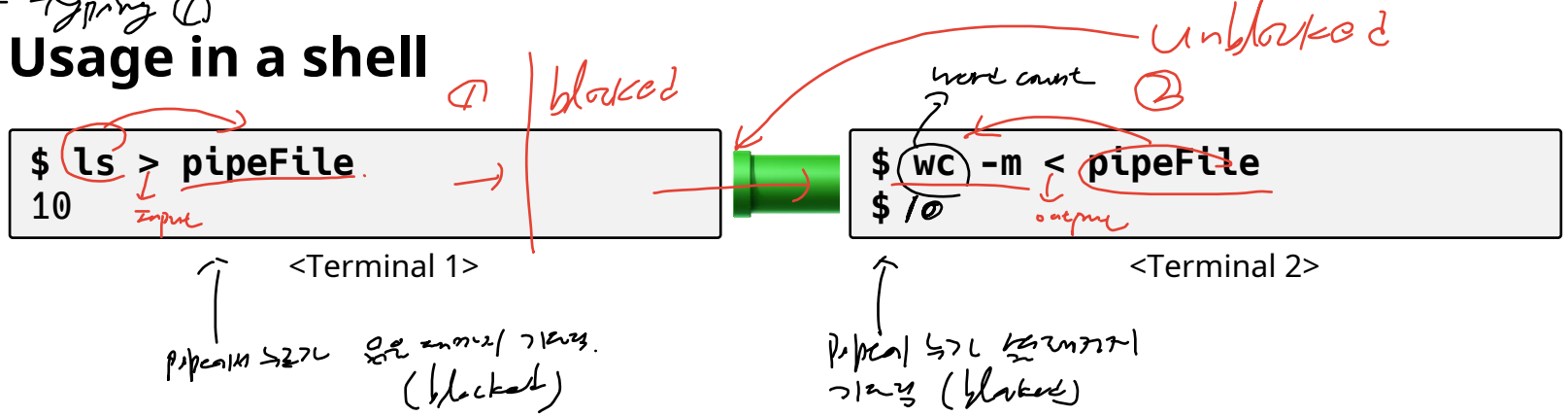
- **mkfifo(3) function**

```
$ man -s 3 mkfifo
```

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

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

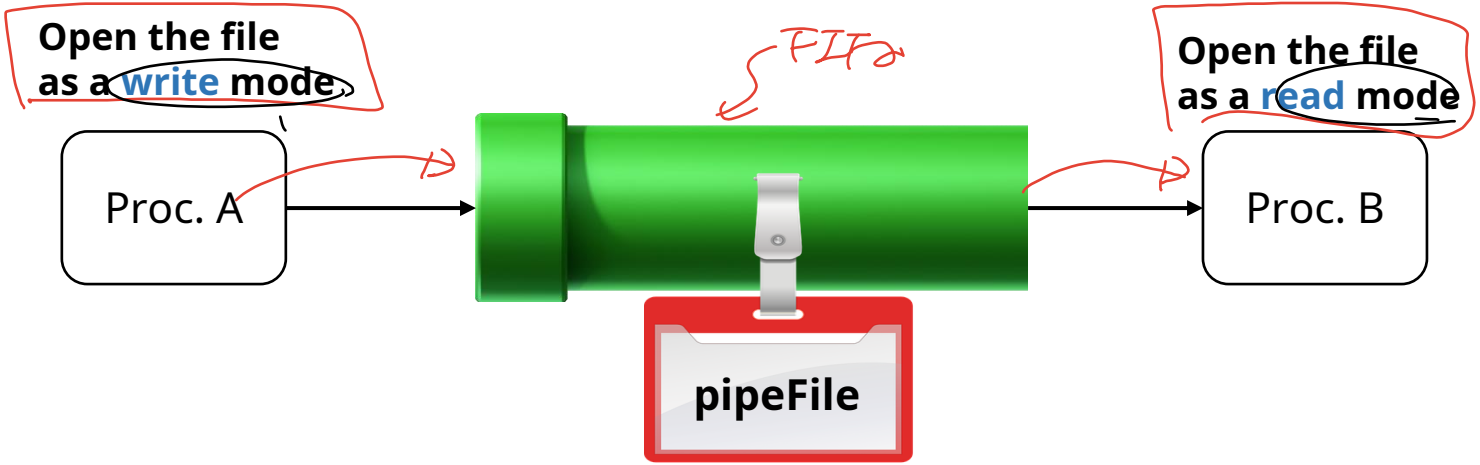
Creating a FIFO file

- **Usage in a shell**



IPC with a named pipe

- 두 개의 프로세스가 하나의 FIFO 파일을 통해 통신 가능
  - **Proc. A** : 쓰기 모드로 file open
  - **Proc. B** : 읽기 모드로 file open



IPC with a named pipe – Server

[code link]

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

int main(void) {
    int pd, n;
    char msg[] = "Hello, FIFO";

    printf("Server =====\n");
    if ((pd = open("./pipeFile", O_WRONLY)) == -1) {
        perror("open"); exit(1);
    }

    printf("To Client : %s\n", msg);
    n = write(pd, msg, strlen(msg)+1);

    if (n == -1) {
        perror("write"); exit(1);
    }
    close(pd);

    return 0;
}
```

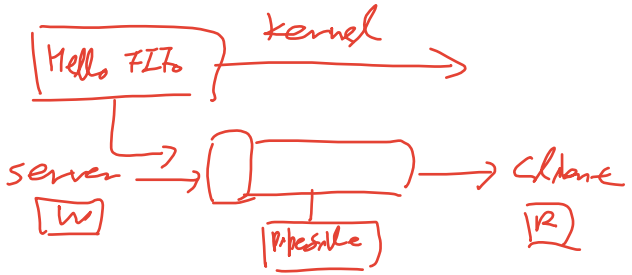
```
$ ./server.out
Server =====
To Client : Hello, FIFO
```

IPC with a named pipe – Client

[code link]

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
```

```c
int main(void) {
    int pd, n;
    char inmsg[80];

    printf("Client =====\n");
    if ((pd = open("./pipeFile", O_RDONLY)) == -1) {
        perror("open"); exit(1);
    }

    write(1, "From Server :", 13);
    while ((n=read(pd, inmsg, 80)) > 0)
        write(1, inmsg, n);

    if (n == -1) {
        perror("read");
        exit(1);
    }

    write(1, "\n", 1);
    close(pd);

    return 0;
}
```



$ ./client.out
Client =====
From server : Hello, FIFO

FIFO file and open(2)

- **FIFO 파일에 대한 open(2)은 기본적으로 blocking mode**
  - Pipe가 양끝(write and read ends)에서 열릴 때 까지 다른 한끝의 open()이 blocking 됨
- **Non-blocking mode로 열기**
  - open(2)의 flag 에 O_NONBLOCK 설정

```
pd = open("./pipeFile", O_RDONLY | O_NONBLOCK))
```

Under Linux, opening a FIFO for read and write will succeed both in blocking and nonblocking mode. **POSIX leaves this behavior undefined**. This can be used to open a FIFO for writing while there are no readers available. A process that uses both ends of the connection in order to communicate with itself should be very careful to avoid deadlocks.

[man -s 7 FIFO]

Summary

- **Stream redirection**

- **Pipe**
  - Shell commands (|)
  - Anonymous pipe
  - Named pipe

이미지 출처

- **본 슬라이드에 사용된 이미지들은,**
  - 다음 출처로 부터 가져 왔으며, 상업적 사용 및 출처 표시 제한이 없는 이미지만 사용 했습니다
    - Pixarbay
    - illustAC

Pixabay로부터 입수된 Peggy und Marco Lachmann-Anke님의 이미지 입니다.

폰트 정보

- **기본 폰트** [link]
  - Noto Sans, Noto Sans KR
    - Google 제공, 상업적 사용 제한 없음

- **제목** [link]
  - 검은고딕 폰트
    - ZESSTYPE 제공, 사업적 사용 제한 없음

- **Source code 폰트** [link]
  - Hack
    - 오픈소스 폰트, 사업적 사용 제한 없음