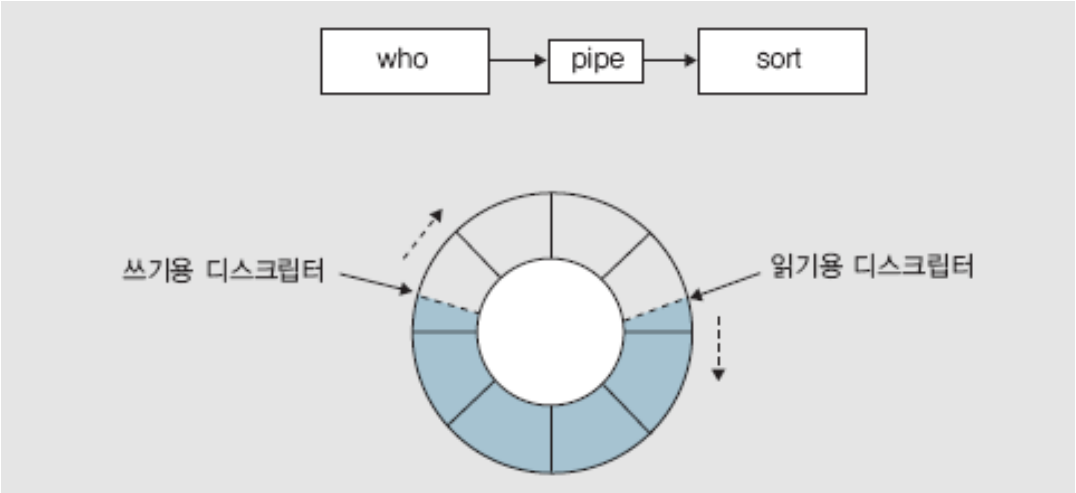
# 12장 파이프

# 12.1 파이프

# 파이프 원리

- $ who | sort



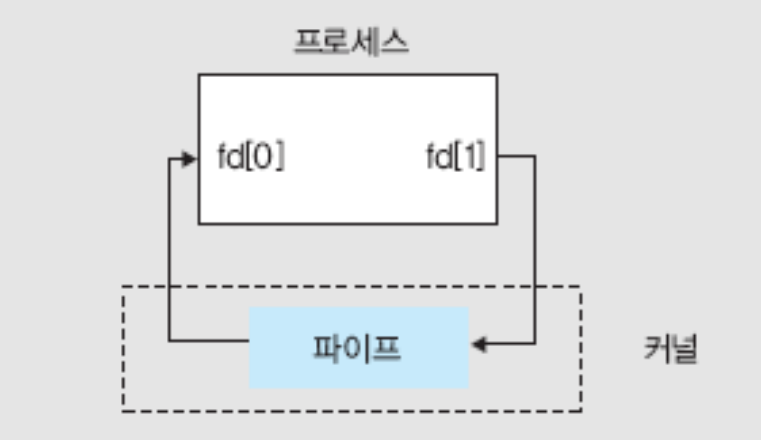

- 파이프
  - 물을 보내는 수도 파이프와 비슷
  - 한 프로세스는 쓰기용 파일 디스크립터를 이용하여 파이프에 데이터를 보내고(쓰고)
  - 다른 프로세스는 읽기용 파일 디스크립터를 이용하여 그 파이프에서 데이터를 받는다(읽는다).
  - 한 방향(one way) 통신

# 파이프 생성

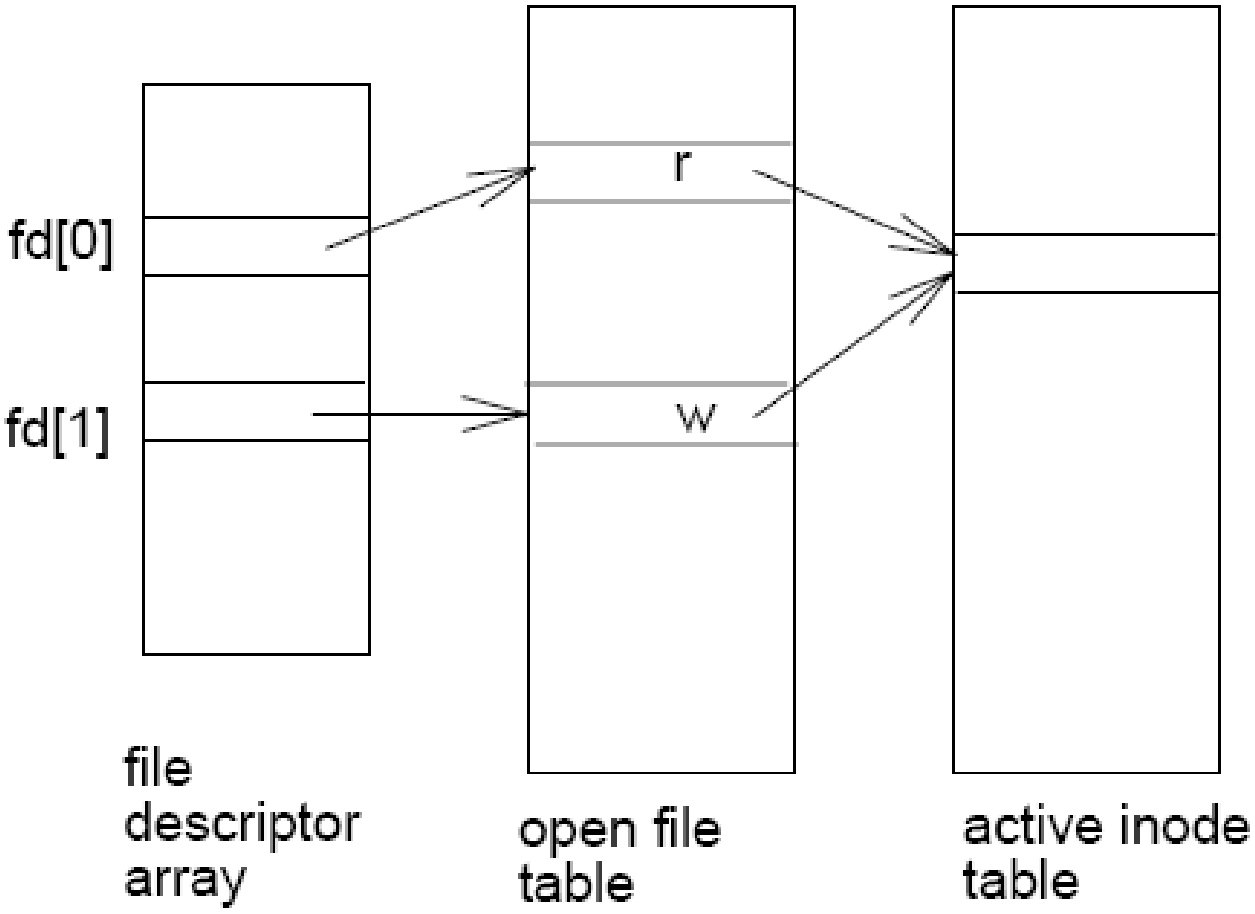
- 파이프는 두 개의 파일 디스크립터를 갖는다.
- 하나는 쓰기용이고 다른 하나는 읽기용이다.

```
#include <unistd.h>

int pipe(int fd[2])
```

파이프를 생성한다. 성공하면 0을 실패하면 -1을 반환한다.



# How to implement pipes



**프로세스가 파이프를 생성한다.**

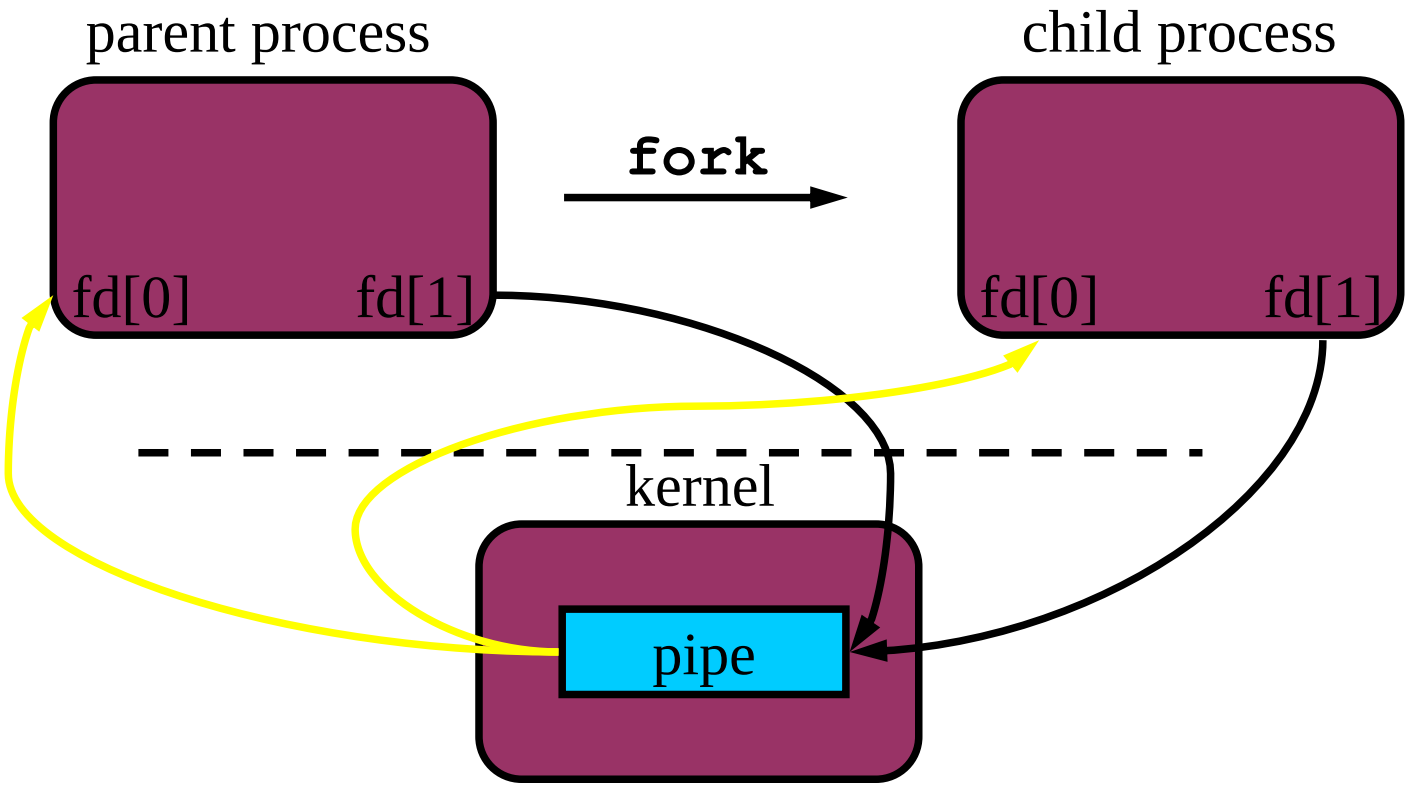
**자신이 파이프에 기록한 메시지 자신이 읽어온다.**

```
01 #include <unistd.h>
02 #include <stdio.h>
03
04 #define SIZE    512
05
06 main()
07 {
08     char msg[SIZE];
09     int filedes[2];
10     int i;
11
12     if(pipe(filedes) == -1) {
13         printf("fail to call pipe()₩n");
14         exit(1);
15     }
16     printf("input a message₩n");
17     for(i = 0; i < 3; i++) {
18
19         fgets(msg, SIZE, stdin);
20         write(filedes[1], msg, SIZE);
21     }
22     printf("₩n");
23     for(i = 0; i < 3; i++) {
24         read(filedes[0], msg, SIZE);
25         printf("%s", msg);
26     }
27 }
```

```
$ ex11-02
input a message
apple is red
banana is yellow
cherry is red

apple is red
banana is yellow
cherry is red
$
```

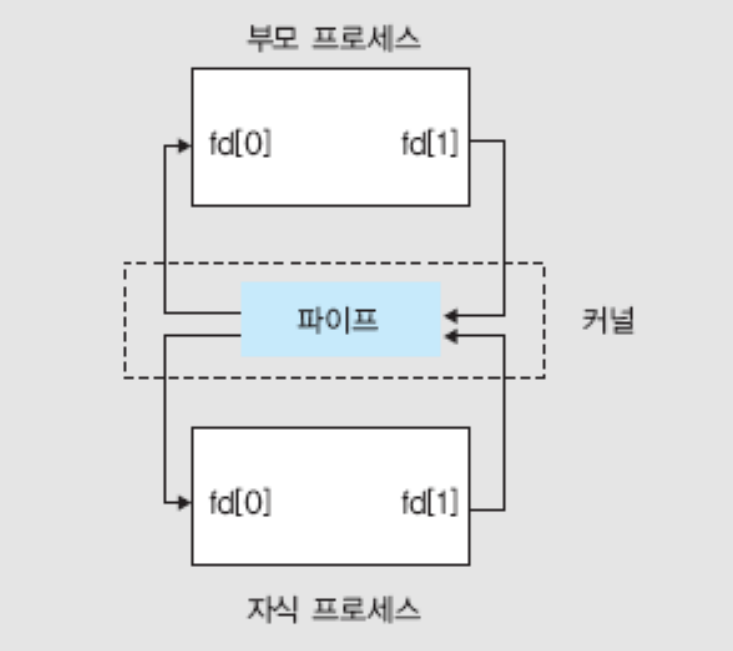
# 파이프 사용법

(1) 한 프로세스가 파이프를 생성한다.
(2) 그 프로세스가 자식 프로세스를 생성한다.
(3) 쓰는 프로세스는 읽기용 파이프 디스크립터를 닫는다.
읽는 프로세스는 쓰기용 파이프 디스크립터를 닫는다.
(4) write()와 read() 시스템 호출을 사용하여 파이프를 통해 데이터를 송수신한다.
(5) 각 프로세스가 살아 있는 파이프 디스크립터를 닫는다.

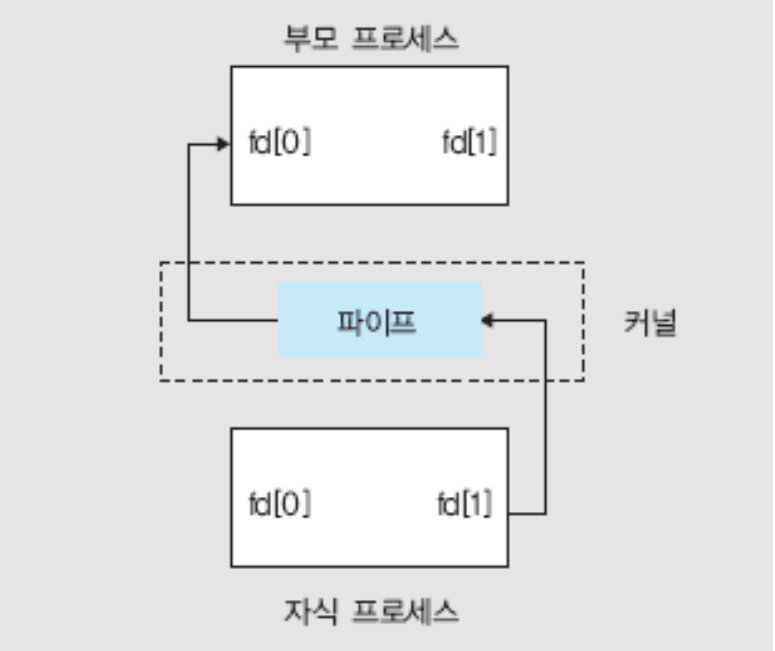

parent process
child process
fork
fd[0]
fd[1]
fd[0]
fd[1]
kernel
pipe

# 파이프 사용법

- 자식 생성 후




- 자식에서 부모로 보내기



```
01 #include <unistd.h>
02 #include <stdio.h>
03 #include <sys/types.h>
04
05 #define SIZE    512
06
07 main()
08 {
09     char msg[SIZE];
10     int filedes[2];
11
12     pid_t pid;
13
14     if(pipe(filedes) == -1)
15     {
16         printf("fail to call pipe()₩n");
17         exit(1);
18    }
```

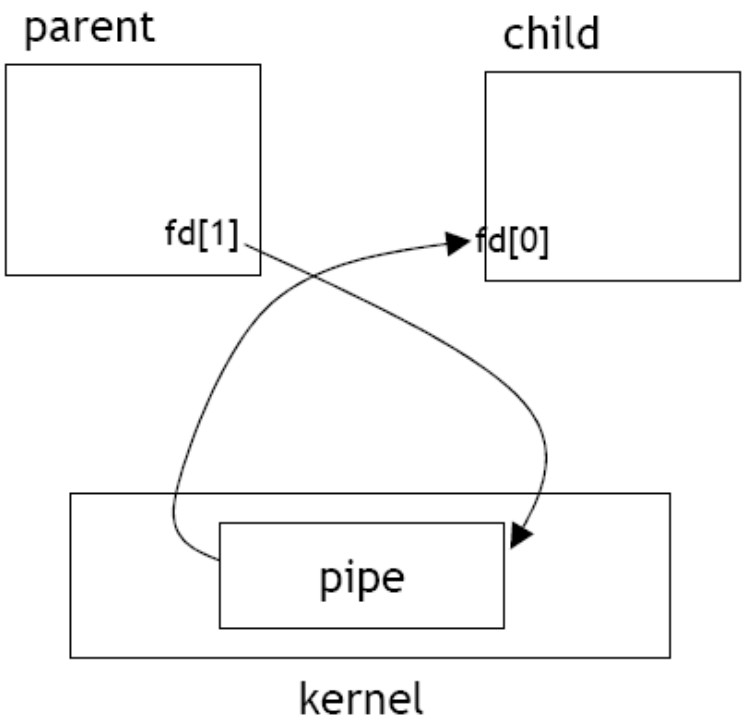
```
19      if((pid = fork()) == -1)
20      {
21          printf("fail to call fork()₩n");
22          exit(1);
23      }
24      else if(pid > 0)
25      {
26          strcpy(msg, "apple is red.₩n");
27          write(filedes[1], msg, SIZE);
28          printf("[parent] %s₩n", msg);
29      }
30      else
31      {
32          sleep(1);
33          read(filedes[0], msg, SIZE);
34          printf("[child] %s₩n", msg);
35      }
36  }
```

```
$ ex11-03
[parent] apple is red.
[child] apple is red.
$
```

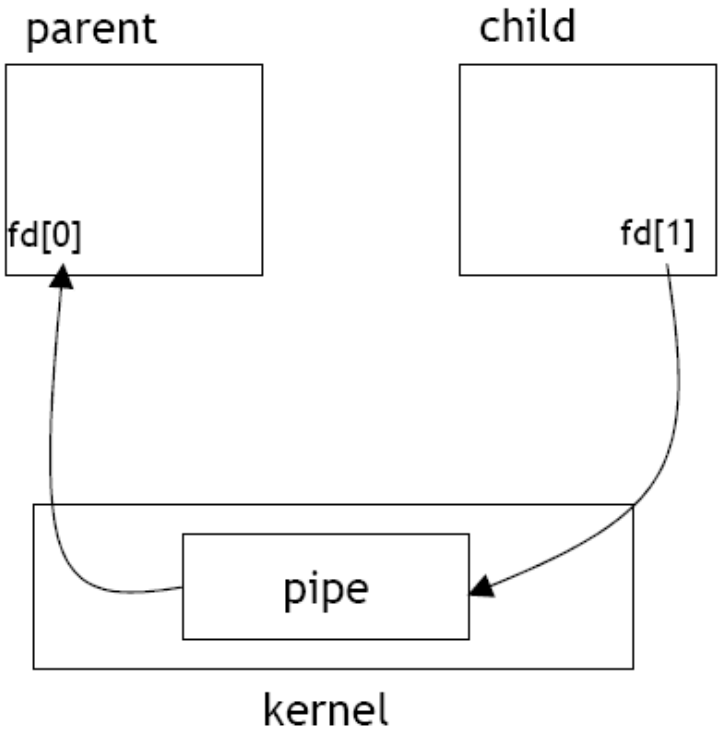
```
01 #include <unistd.h>
02 #include <stdio.h>
03 #include <sys/types.h>
04
05 #define SIZE    512
06
07 main()
08 {
09     int filedes[2];     pid_t pid;
10
11     if(pipe(filedes) == -1) {
12         printf("fail to call pipe()₩n");
13         exit(1);
14     }
15
16     if((pid = fork()) == -1) {
17         /* fork() 호출 실패 */
18     }
19     else if(pid > 0) {
20         close(filedes[0]);
21         /* filedes[1]을 지정하여 파이프에 메시지 쓰기 */
22     }
23     else {
24         close(filedes[1]);
25         /* filedes[0]을 지정하여 파이프로부터 메시지 읽기 */
26     }
27 }
```

사용하지 않는 파이프는 닫는다.

parent → child:
parent closes fd[0]
child closes fd[1]




parent ← child:
parent closes fd[1]
child closes fd[0]



```
#define MAXLINE 100
int main(void) {
        int n, fd[2];
        int pid;
        char line[MAXLINE];

        if (pipe(fd) < 0)
                perror("pipe error");
        if ( (pid = fork()) < 0)
                perror("fork error");
        else if (pid > 0) { /* parent */
                close(fd[0]);
                write(fd[1], "hello world\n", 12);
        } else { /* child */
                close(fd[1]);
                n = read(fd[0], line, MAXLINE);
                write(1, line, n);
        }
        exit(0);
}
```

# pipe.c

```
1 #include <unistd.h>
2 #define MAXLINE 100
3 /* 파이프를 통해 자식에서 부모로
4    데이터를 보내는 프로그램 */
5 int main( )
6 {
7    int n, length, fd[2];
8    int pid;
9    char message[MAXLINE], line[MAXLINE];
10
11    pipe(fd);    /* 파이프 생성 */
12
```

```
13    if ((pid = fork()) == 0) {  /* 자식 프로세스 */
14      close(fd[0]);
15       sprintf(message, "Hello from PID  %d\n", getpid());
16       length = strlen(message)+1;
17       write(fd[1], message, length);
18    } else {                  /* 부모 프로세스 */
19       close(fd[1]);
20       n = read(fd[0], line, MAXLINE);
21       printf("[%d] %s", getpid(), line);
22    }
23
24    exit(0);
25 }
```

```
01 #include <unistd.h>
02 #include <stdio.h>
03
04 #define SIZE    512
05
06 main()
07 {
08     char *msg1 = "apple is red";
09     char *msg2 = "banana is yellow";
10     char buffer[SIZE];
11
12     int filedes[2];
13     int nread;
14
15     if(pipe(filedes) == -1)
16     {
17         printf("fail to call pipe()₩n");
18         exit(1);
19     }
```

```
20     write(filedes[1], msg1, strlen(msg1) +
       1);
21     write(filedes[1], msg2, strlne(msg2) +
       1);
22
23     nread = read(filedes[0], buffer, SIZE);
24     printf("%d, %s₩n", nread, buffer);
25
26     nread = read(filedes[0], buffer, SIZE);
27     printf("%d, %s₩n", nread, buffer);
}
```

※송신 측이 쓰는 메시지 크기와 수신 측이 읽는 메시지 크기가 서로 다르다.

```
$ ex11-06
30, apple is red
^C
$
```

```
01 #include <unistd.h>
02 #include <stdio.h>
03
04 #define SIZE    512
05
06 main()
07 {
08     char msg1[512] = "apple is red";
09     char msg2[512] = "banana is yellow";
10     char buffer[SIZE];
11
12     int filedes[2], nread;
13     int len1 = strlen(msg1) + 1;
14     int len2 = strlen(msg2) + 1;
15
16     if(pipe(filedes) == -1) {
17         printf("fail to call pipe()₩n");
18         exit(1);
19     }
```

```
20     write(filedes[1], msg1, len1);
21     write(filedes[1], msg2, len2);
22
23     nread = read(filedes[0], buffer,
       len1);
24     printf("%d, %s₩n", nread, buffer);
25     nread = read(filedes[0], buffer,
       len2);
26     printf("%d, %s₩n", nread, buffer);
27 }
```

※송신 측이 쓰는 메시지 크기와
수신 측이 읽는 메시지 크기가 서로 같다.

```
$ ex11-07
13, apple is red
17, banana is yellow
$
```

```
…
01 #define SIZE    512
02
03 main()
04 {
05     char *msg1 = "apple is red";
06     char *msg2 = "banana is yellow";
07     char buffer[SIZE];

    …

20    if(pipe(filedes) == -1)
21        …
22
23     write(filedes[1], msg1, SIZE);
24     write(filedes[1], msg2, SIZE);
25
26     nread = read(filedes[0], buffer, SIZE);
27     printf("%d, %s₩n", nread, buffer);
28
29     nread = read(filedes[0], buffer, SIZE);
30     printf("%d, %s₩n", nread, buffer);
31 }
```

```
$ ex11-08
512, apple is red
512, banana is yellow
$
```

# Discussion

1. 기본 파이프는 서로 관계 있는(예를 들어 부모 관계) 프로세스간에만 통신이 가능하다. 그렇지 않은 관계에서는 파이프를 이용한 통신이 불가능한가? (Named pipe – mkfifo)

2. 기본 파이프는 양방향 통신이 불가능하다. 양방향 통신을 위해서는 어떻게 해야하나?(이중 파이프)

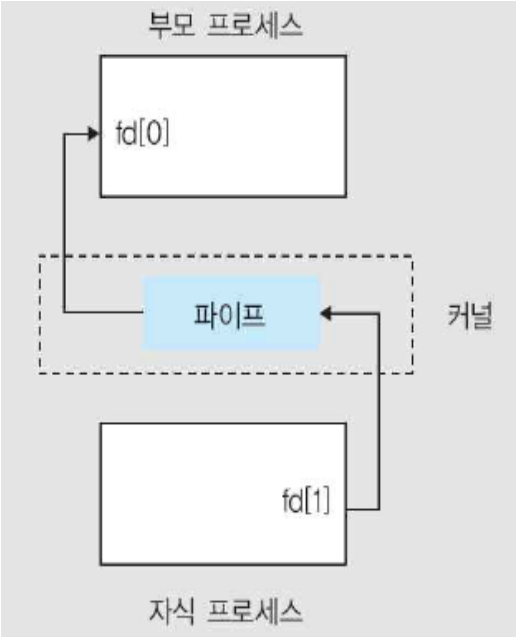
3. 쓰는 프로세스가 종료되고 파이프도 비어있는 경우 읽기를 시도하면 어떤 현상이 벌어지나? (상대 프로세스가 종료되었는가 아니면 아직 쓰지 않았는가에 따라 결정됨)

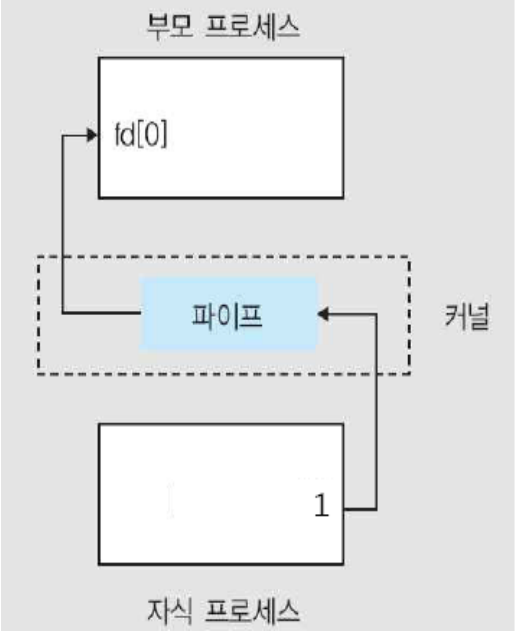
4. 읽는 프로세스가 종료되었을 경우 쓰는 프로세스가 파이프에 쓰기를 시도하면 어떤 현상이 벌어지나?(쓰기시도時 종료)

# 12.2 쉘 파이프 구현

# 표준출력을 파이프로 보내기

- 자식 프로세스의 표준출력을 파이프를 통해 부모 프로세스에게 보내려면 어떻게 하여야 할까?
  - 쓰기용 파이프 디스크립터 fd[1]을 표준출력을 나타내는 1번 파일 디스크립터에 복제
  - dup2(fd[1],1)



# stdpipe.c

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #define MAXLINE 100
4
5 /* 파이프를 통해 자식에서 실행되는명령어 출력을 받아 프린트 */
6 int main(int argc, char* argv[])
7 {
8     int n, pid, fd[2];
9     char line[MAXLINE];
10
11    pipe(fd);   /* 파이프 생성 */
12
13    if ((pid = fork()) == 0) {    //자식 프로세스
14        close(fd[0]);
15        dup2(fd[1],1);
16        close(fd[1]);
17        printf("Hello! pipe\n");
18        printf("Bye! pipe\n");
19    } else {       // 부모 프로세스
20        close(fd[1]);
21        printf("자식 프로세스로부터 받은 결과\n");
22        while ((n = read(fd[0], line, MAXLINE))> 0)
23            write(STDOUT_FILENO, line, n);
24    }
25
26    exit(0);
27 }
```

# 명령어 표준출력을 파이프로 보내기

- 프로그램 pexec1.c는 부모 프로세스가 자식 프로세스에게
- 명령줄 인수로 받은 명령어를 실행하게 하고
- 그 표준출력을 파이프를 통해 받아 출력한다.

# pexec1.c

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #define MAXLINE 100
4
5 /* 파이프를 통해 자식에서 실행되는명령어 출력을 받아 프린트 */
6 int main(int argc, char* argv[])
7 {
8    int n, pid, fd[2];
9    char line[MAXLINE];
10
11  pipe(fd);   /* 파이프 생성 */
12
```

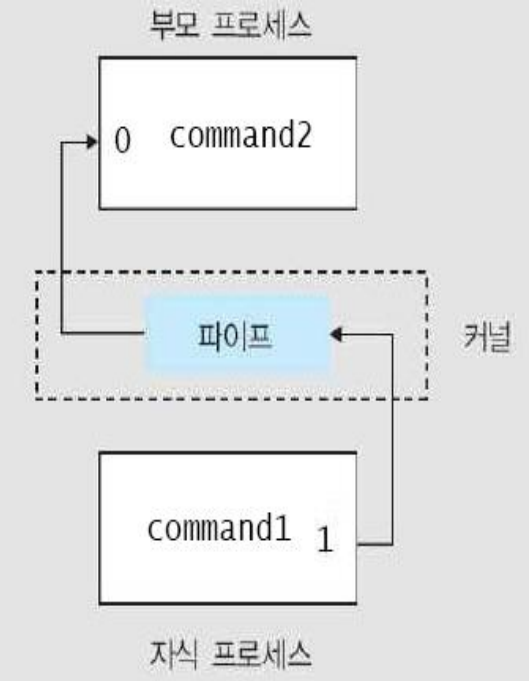
```
13    if ((pid = fork()) == 0) { //자식 프로세스
14       close(fd[0]);
15       dup2(fd[1],1);
16       close(fd[1]);
17       execvp(argv[1], &argv[1]);
18    } else {  // 부모 프로세스
19       close(fd[1]);
20       printf("자식 프로세스로부터 받은 결과\n");
21       while ((n = read(fd[0], line, MAXLINE))>
      0)
22          write(STDOUT_FILENO, line, n);
23    }
24
25    exit(0);
26 }
```

# 쉘 파이프

- 쉘 파이프 기능

  [shell] command1 | command2

  - 자식 프로세스가 실행하는 command1의 표준출력을 파이프를 통해서 부모 프로세스가 실행하는 command2의 표준입력으로 전달



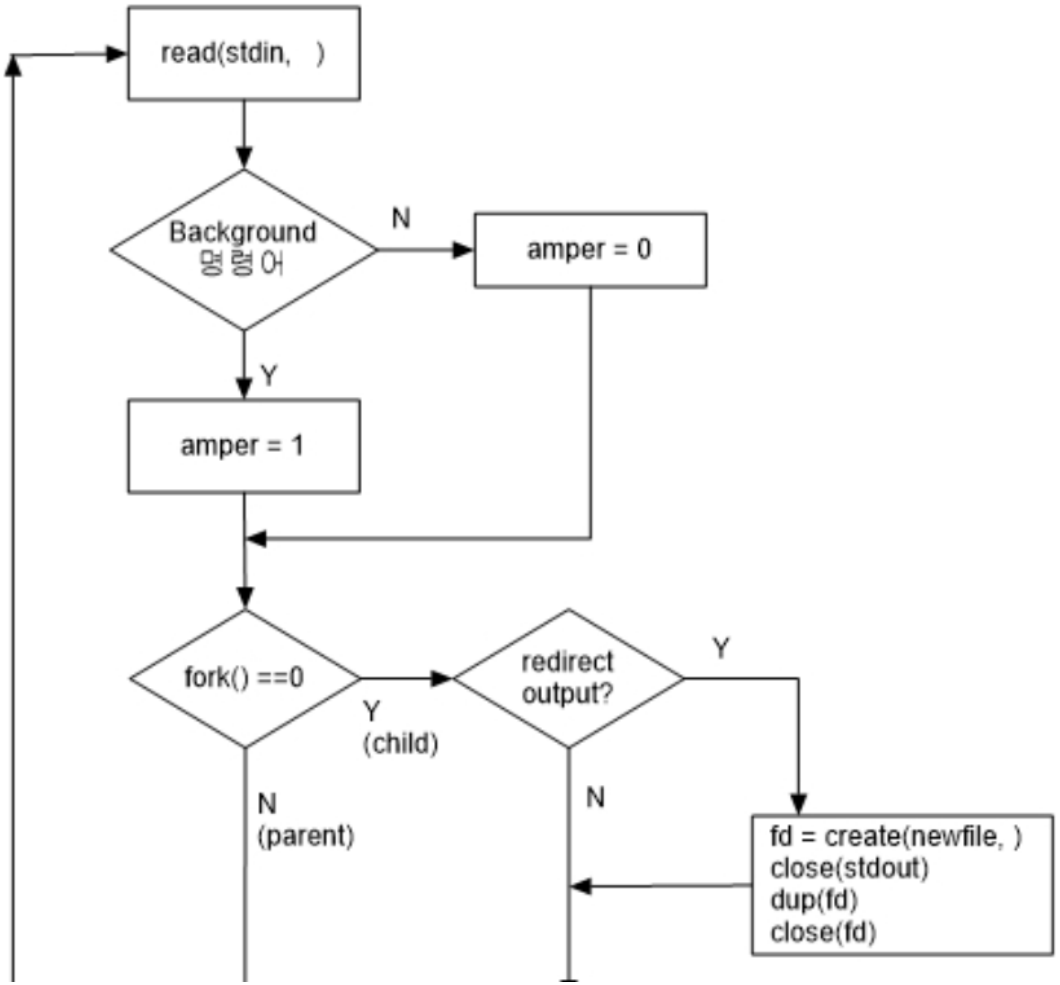
# shellpipe.c

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#define READ 0
#define WRITE 1

int main(int argc, char* argv[])
{
    char str[1024];
    char *command1, *command2;
    int fd[2];

    printf("[shell]");
    fgets(str,sizeof(str),stdin);
    str[strlen(str)-1] ='\0';
    if(strchr(str,'|') != NULL) {  // 파이프 사용하는 경우
        command1 = strtok (str,"| ");
        command2 = strtok (NULL, "| ");
    }
```

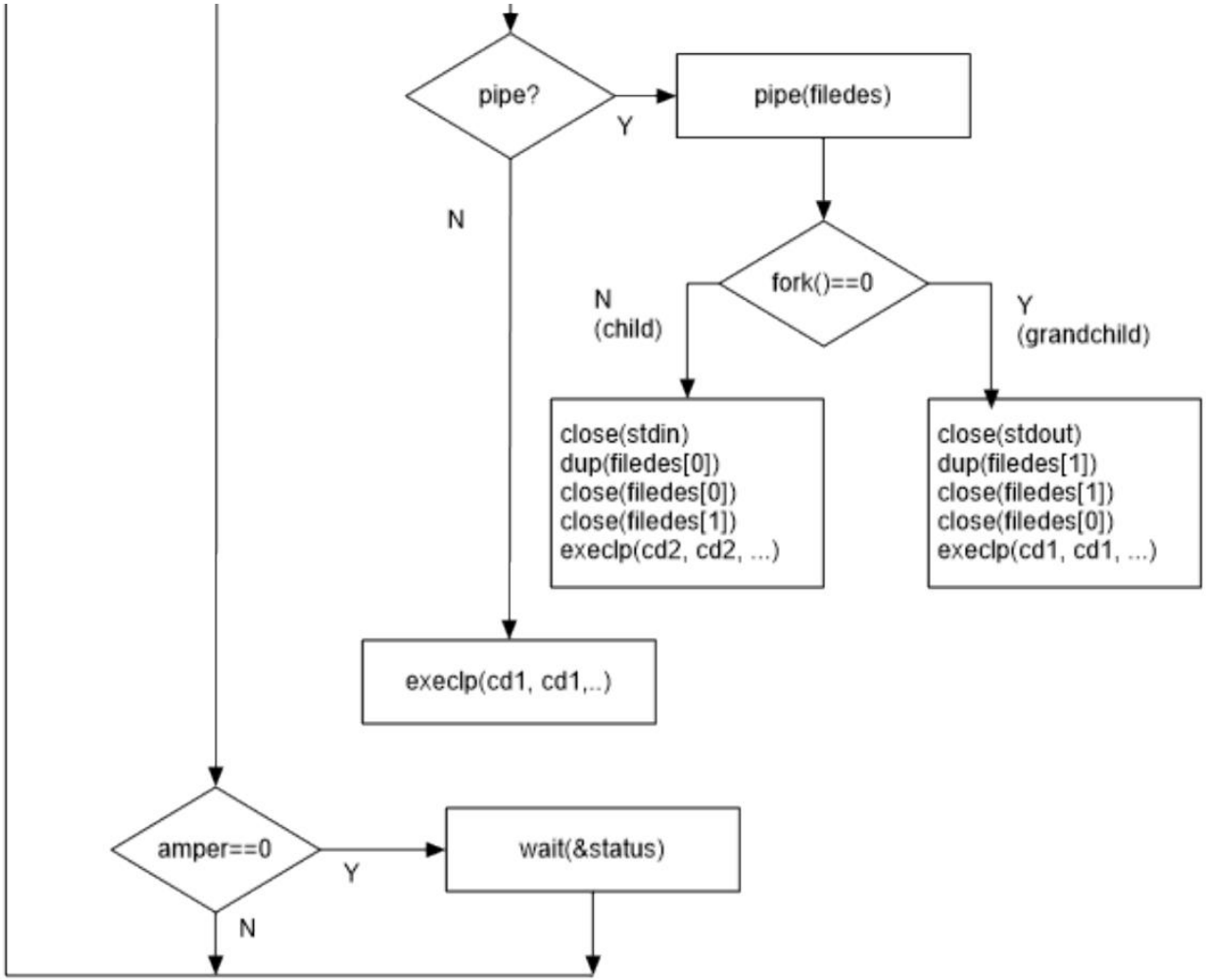
# shellpipe.c

```
pipe(fd);

if (fork() ==0) {
    close(fd[READ]);
    dup2(fd[WRITE],1);    // 쓰기용 파이프를 표준출력에 복제
    close(fd[WRITE]);
    execlp(command1, command1, NULL);
    perror("pipe");
} else {
    close(fd[WRITE]);
    dup2(fd[READ],0);    // 읽기용 파이프를 표준입력에 복제
    close(fd[READ]);
    execlp(command2, command2, NULL);
    perror("pipe");
}
}
```

# Shell의 main loop (1/2)



# Shell의 main loop (2/2)



# 기말과제

- Shell의 동작과정을 나타낸 flowchart를 C언어를 이용하여 작성하고 테스트 하라.
  - Background/Foreground 프로세스 생성
  - Rediction (input/output)
  - Pipe (하나의 파이프만 생성 가정)