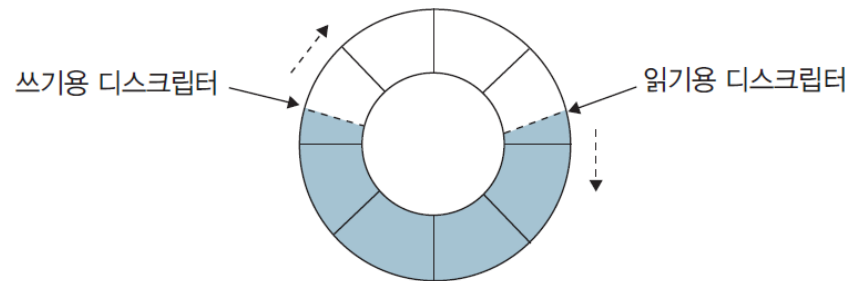
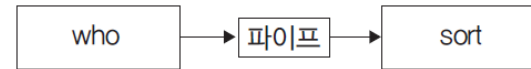


# 12장 파이프

## 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**

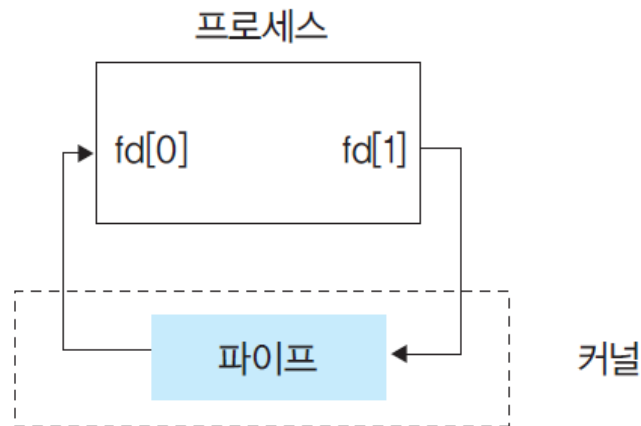
# 파이프 생성

- 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.



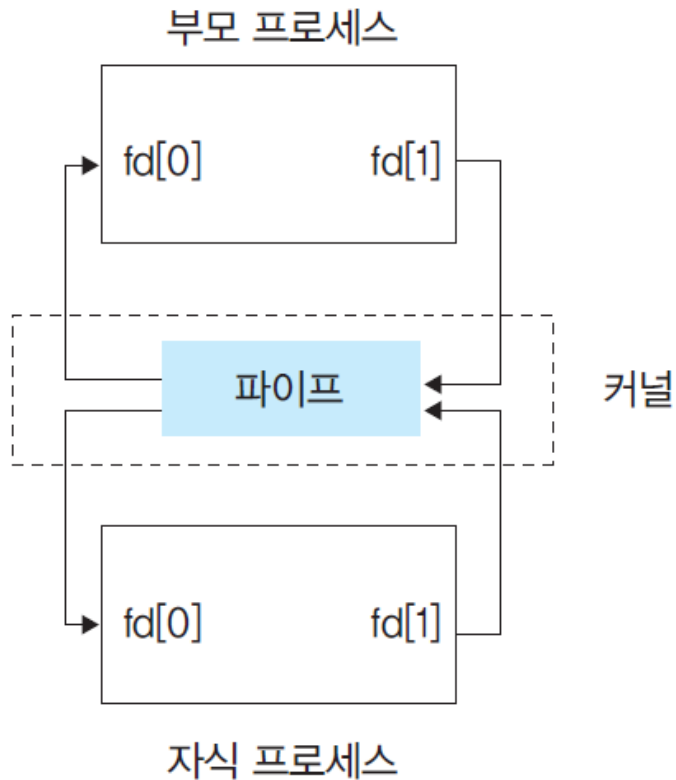
# 부모-자식 프로세스 사이에 파이프 사용법

---

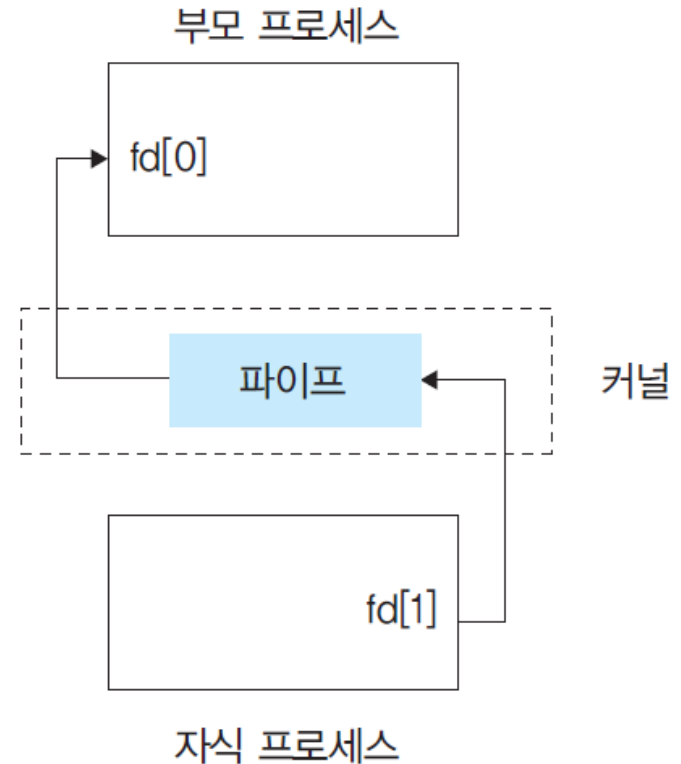
- (1) 한 프로세스가 파이프를 생성한다.
- (2) 그 프로세스가 자식 프로세스를 생성한다.
- (3) 쓰는 프로세스는 읽기용 파이프 디스크립터를 닫는다.  
읽는 프로세스는 쓰기용 파이프 디스크립터를 닫는다.
- (4) write()와 read() 시스템 호출을 사용하여 파이프를 통해 데이터를 송수신한다.
- (5) 각 프로세스가 살아 있는 파이프 디스크립터를 닫는다.

# 파이프 사용법

- 자식 생성 후



- 자식에서 부모로 보내기



# pipe.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <string.h>
5 #define MAXLINE 100
6
7 /* 파이프를 통해 자식에서 부모로
   데이터를 보내는 프로그램 */
8 int main( )
9 {
10  int n, length, fd[2], pid;
11  char message[MAXLINE], line[MAXLINE];
12
13  pipe(fd); /* 파이프 생성 */
14
15  if ((pid = fork()) == 0) { /* 자식 프로세스 */
16      close(fd[0]);
17      sprintf(message, "Hello from PID %d\n",
18              getpid());
19      length = strlen(message)+1;
20      write(fd[1], message, length);
21  } else { /* 부모 프로세스 */
22      close(fd[1]);
23      n = read(fd[0], line, MAXLINE);
24      printf("[%d] %s", getpid(), line);
25  }
26  exit(0);
27 }

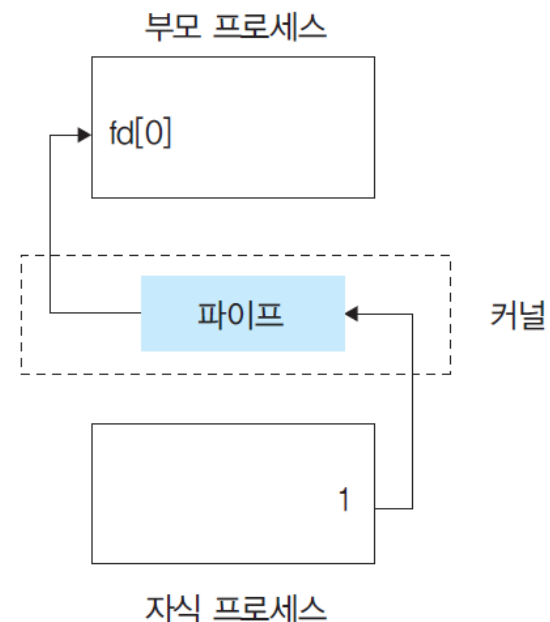
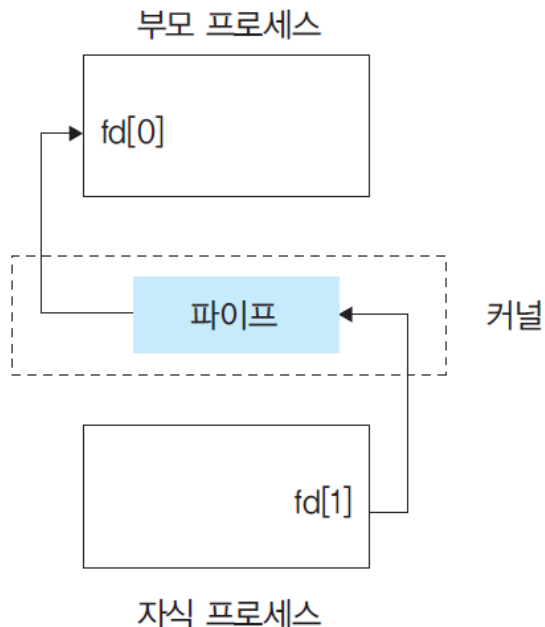
$ pipe
[12555] Hello from PID 12556
```

## 12.2 쉘 파이프 구현



# 표준출력을 파이프로 보내기

- 자식 프로세스의 표준출력을 파이프를 통해 부모 프로세스에 보내기
  - 쓰기용 파이프 디스크립터 `fd[1]`을 표준출력 1번 파일 디스크립터에 복제
  - `dup2(fd[1],1)`



# stdpipe.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #define MAXLINE 100
5
6 /* 파이프를 통해 자식에서 실행되는명
   령어 출력을 받아 프린트 */
7 int main(int argc, char* argv[])
8 {
9     int n, pid, fd[2];
10    char line[MAXLINE];
11
12    pipe(fd); /* 파이프 생성 */
13
14    if ((pid = fork()) == 0) { //자식 프로세스
15        close(fd[0]);
16        dup2(fd[1], 1);
17        close(fd[1]);
18        printf("Hello! pipe\n");
19        printf("Bye! pipe\n");
20    } else { // 부모 프로세스
21        close(fd[1]);
22        printf("자식 프로세스로부터 받은 결과\n");
23        while ((n = read(fd[0], line, MAXLINE)) > 0)
24            write(STDOUT_FILENO, line, n);
25    }
26
27    exit(0);
28 }
```

\$ stdpipe  
자식 프로세스로부터 받은 결과  
Hello! pipe  
Bye! pipe

# 명령어 표준출력을 파이프로 보내기

---

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

# pexec1.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #define MAXLINE 100
5
6 /* 파이프를 통해 자식에서 실행되는명
   령어 출력을 받아 프린트 */
7 int main(int argc, char* argv[])
8 {
9     int n, pid, fd[2];
10    char line[MAXLINE];
11
12    pipe(fd); /* 파이프 생성 */
13
14    if ((pid = fork()) == 0) { //자식 프로세스
15        close(fd[0]);
16        dup2(fd[1], 1);
17        close(fd[1]);
18        execvp(argv[1], &argv[1]);
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 }
```

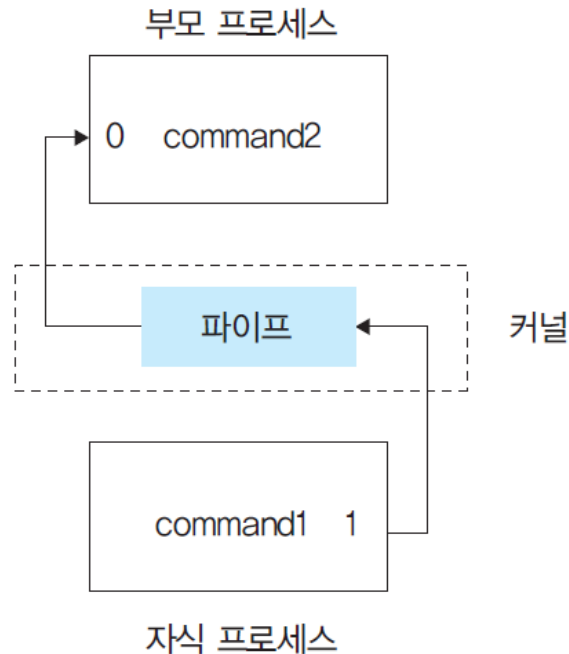
\$ pexec1 date  
자식 프로세스로부터 받은 결과  
2021. 05. 20. (목) 10:19:38 KST

# 셸 파이프

- 셸 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");
}
}
```

\$ shellpipe  
[shell] ls | wc  
17 17 160

## 12.3 파이프 함수



# popen()

- 자식 프로세스에게 명령어를 실행시키고  
그 표준출력을 파이프를 통해 받는 과정을 하나의 함수로 정의

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

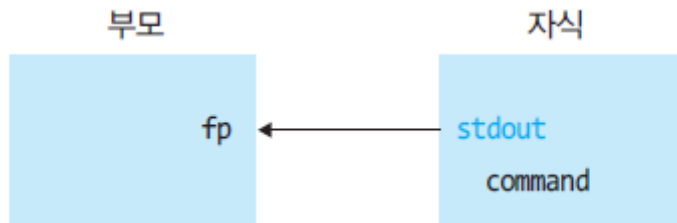
```
FILE *popen(const char *command, const char *type);
```

성공하면 파이프를 위한 파일 포인터를 실패하면 NULL을 리턴한다.

```
int pclose(FILE *fp);
```

성공하면 *command* 명령어의 종료 상태를 실패하면 -1을 리턴한다.

- `fp = popen(command, "r");`



- `fp = popen(command, "w");`



# pexec2.c

---

```
#include <stdio.h>
#define MAXLINE 100

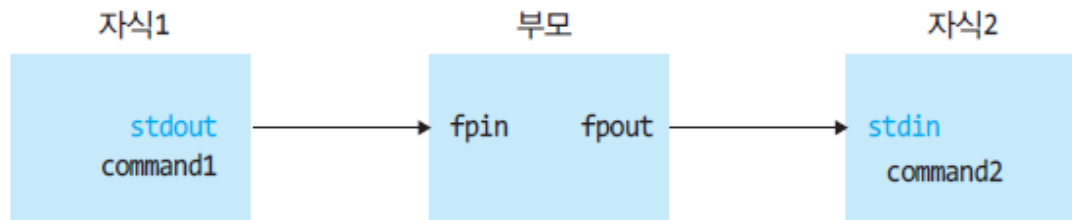
/* popen() 함수를 이용해 자식에서 실행되는 명령어 출력을 받아 프린트 */
int main(int argc, char* argv[])
{
    char line[MAXLINE];
    FILE *fpin;
    if ((fpin = popen(argv[1], "r")) == NULL) {
        perror("popen 오류");
        return 1;
    }
    printf("자식 프로세스로부터 받은 결과\n");
    while (fgets(line, MAXLINE, fpin))
        fputs(line, stdout);
    pclose(fpin);
    return 0;
}
```

\$ pexec2 date  
자식 프로세스로부터 받은 결과  
2021. 05. 20. (목) 10:27:39 KST

# 명령어 파이프 구현 예제

---

- `$ pexec3 command1 command2`



```
1 #include <stdio.h>
2 #define MAXLINE 100
3 /* popen() 함수를 이용해 명령어 파이프 기능을 구현한다. */
4
5 int main(int argc, char* argv[])
6 {
7     char line[MAXLINE];
8     FILE *fpin, *fpout;
9
10    if ((fpin = popen(argv[1], "r")) == NULL) {
11        perror("popen 오류");
12        return 1;
13    }
14
15    if ((fpout = popen(argv[2], "w")) == NULL) {
16        perror("popen 오류");
17        return 1;
18    }
19
20    while (fgets(line, MAXLINE, fpin))
21        fputs(line, fpout);
22
23    pclose(fpin);
24    pclose(fpout);
25    return 0;
26 }
```

---

## 12.4 이름 있는 파이프

# 이름 있는 파이프(named pipe)

---

- (이름 없는) 파이프
  - 이름이 없으므로 부모 자식과 같은 서로 관련된 프로세스 사이의 통신에만 사용될 수 있었다.
- 이름 있는 파이프
  - 다른 파일처럼 이름이 있으며 파일 시스템 내에 존재한다.
  - 서로 관련 없는 프로세스들도 공유하여 사용할 수 있다.

# 이름 있는 파이프를 만드는 방법

---

- p 옵션과 함께 mknod 명령어

```
$mknod myPipe p
```

```
$chmod ug+rw myPipe
```

```
$ls -l myPipe
```

```
prw-rw-r-- 1 chang faculty 0 4월 11일 13:03 myPipe
```

- mkfifo() 시스템 호출

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

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

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

이름 있는 파이프를 생성한다. 성공하면 0을 실패하면 -1을 리턴한다.

# npreader.c

---

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define MAXLINE 100
/* 이름 있는 파이프를 통해 읽은 내용을
   프린트한다. */
int main( )
{
    int fd;
    char str[MAXLINE];
    unlink("myPipe");
    mkfifo("myPipe", 0660);
    fd = open("myPipe", O_RDONLY);
```

```
    while (readLine(fd, str))
        printf("%s\n", str);
    close(fd);
    return 0;
}

int readLine(int fd, char *str)
{
    int n;
    do {
        n = read(fd, str, 1);
    } while (n > 0 && *str++ != NULL);
    return (n > 0);
}
```



# npwriter.c

---

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define MAXLINE 100
/* 이름 있는 파이프를 통해 메시지를 출력
   한다. */
Int main( )
{
    int fd, length;
    char message[MAXLINE];
    sprintf(message, "Hello from PID %d",
        getpid());
    length = strlen(message)+1;

    do {
        fd = open("myPipe", O_WRONLY);
        if (fd == -1) sleep(1);
    } while (fd == -1);

    for (int i = 0; i <= 3; i++) {
        write(fd, message, length);
        sleep(3);
    }
    close(fd);
    return 0;
}
```

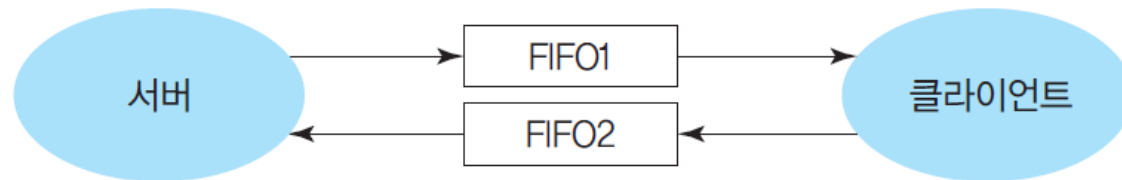
# npreader, npwriter 실행

---

```
$ npwriter &  
[1] 13079  
$ npreader  
Hello from PID 13079  
Hello from PID 13079  
Hello from PID 13079  
Hello from PID 13079  
[1]+  Done                npwriter
```

# 파이프를 이용한 일대일 채팅

- 이 프로그램은 채팅 서버와 채팅 클라이언트로 구성된다.
- 2 개의 파이프가 필요함
  - FIFO1 : 채팅 서버 → 채팅 클라이언트로 데이터를 보내는데 사용
  - FIFO2 : 채팅 클라이언트 → 채팅 서버로 데이터를 보내는데 사용



## 실행결과

\$chatserver

[서버] : 안녕하세요. 클라이언트  
[클라이언트] -> 반갑습니다. 서버  
...

\$chatclient

\* 클라이언트 시작  
[서버] -> 안녕하세요. 클라이언트  
[클라이언트] : 반갑습니다. 서버  
...

# chatserver.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#define MAXLINE 256
main() {
    int fd1, fd2, n;
    char msg[MAXLINE];

    if (mkfifo("./chatfifo1", 0666) == -1) {
        perror("mkfifo");
        exit(1);
    }
    if (mkfifo("./chatfifo2", 0666) == -1) {
        perror("mkfifo");
        exit(2);
    }

    fd1 = open("./chatfifo1", O_WRONLY);
    fd2 = open("./chatfifo2", O_RDONLY);
    if (fd1 == -1 || fd2 == -1) {
        perror("open");
        exit(3);
    }

    printf("* 서버 시작 \n");
    while(1) {
        printf("[서버] : ");
        fgets(msg, MAXLINE, stdin);
        n = write(fd1, msg, strlen(msg)+1);
        if (n == -1) {
            perror("write");
            exit(1);
        }
        n = read(fd2, msg, MAXLINE);
        printf("[클라이언트] -> %s\n", msg);
    }
}
```

# chatclient.c

---

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#define MAXLINE 256
main() {
    int fd1, fd2, n;
    char inmsg[MAXLINE];
    fd1 = open("./chatfifo1", O_RDONLY);
    fd2 = open("./chatfifo2", O_WRONLY);

    if(fd1 == -1 || fd2 == -1) {
        perror("open");
        exit(1);
    }

    printf("* 클라이언트 시작 \n");
    while(1) {
        n = read(fd1, inmsg, MAXLINE);
        printf("[서버] -> %s\n", inmsg);
        printf("[클라이언트] :");
        fgets(inmsg, MAXLINE, stdin);
        write(fd2, inmsg, strlen(inmsg)+1);
    }
}
```

# 핵심 개념

---

- 파이프는 데이터를 한 방향으로 보내는데 사용된다.
- 파이프는 두 개의 파일 디스크립터를 갖는다.  
하나는 쓰기용이고 다른 하나는 읽기용이다.
- 이름 있는 파이프는 파일처럼 파일 시스템 내에 존재하고 이름이 있으며 서로 관련 없는 프로세스들도 공유하여 사용할 수 있다.