9. 파이프

상명대학교 소프트웨어학부

파이프

- 시그널은 이상한 사건이나 오류를 처리하는 데는 이용하지만, 한 프로세스로부터 다른 프로세스로 대량의 정보를 전송하는 데는 부적합하다.

- 파이프
  - 한 프로세스를 다른 관련된 프로세스에 연결시켜주는 단방향의 통신 채널

pipe()

Usage
#include <unistd.h> int pipe(int filedes[2]);

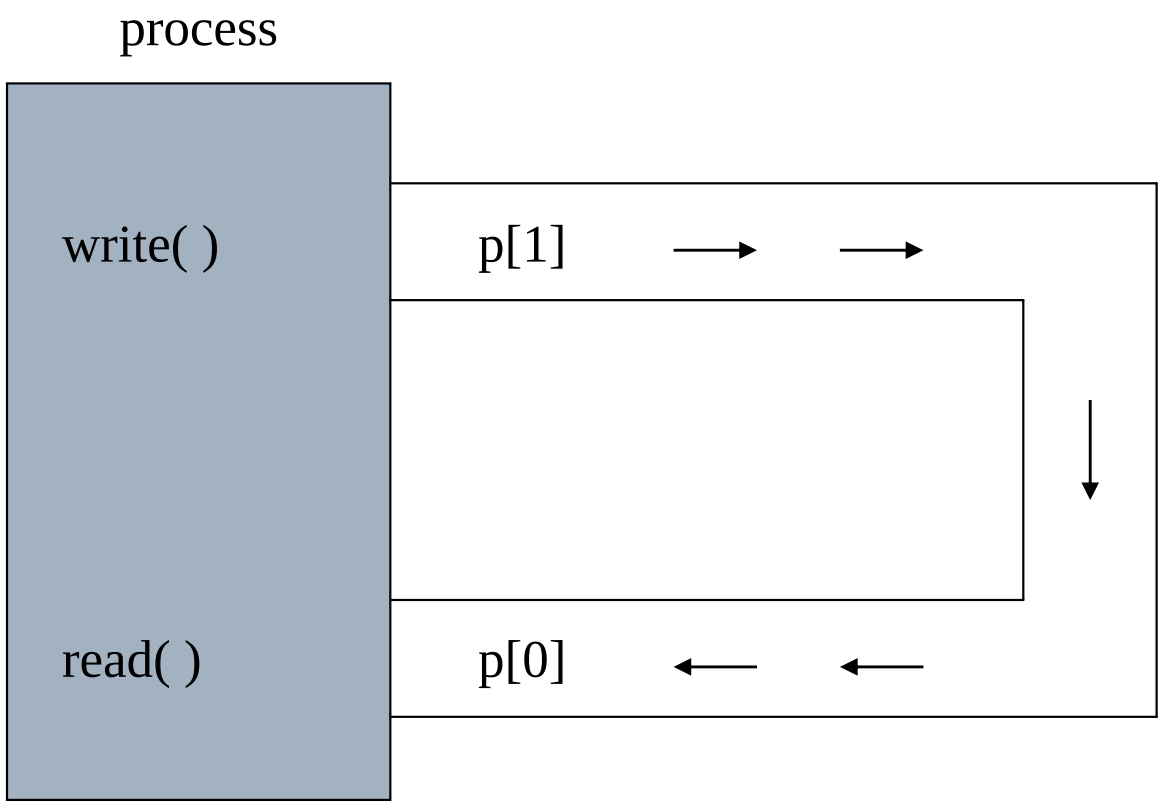
< ex_1.c >

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

#define MSGSIZE  16

char *msg1 = "hello, world #1";
char *msg2 = "hello, world #2";
char *msg3 = "hello, world #3";

main()
{
        char inbuf[MSGSIZE];
        int p[2], j;

         /* 파이프를 개방한다 */
         if(pipe(p) == -1)
         {
                perror("pipe call");
                exit(1);
        }
```

< ex_1.c >

```
        /* 파이프에 쓴다 */
        write(p[1], msg1, MSGSIZE);
        write(p[1], msg2, MSGSIZE);
        write(p[1], msg3, MSGSIZE);

        /* 파이프로부터 읽는다. */
        for(j = 0; j < 3; j++)
        {
                read (p[0], inbuf, MSGSIZE);
                printf ("%s\n", inbuf);
         }

        exit (0);
}
```

pipe()



<첫 번째 파이프 사용 예>

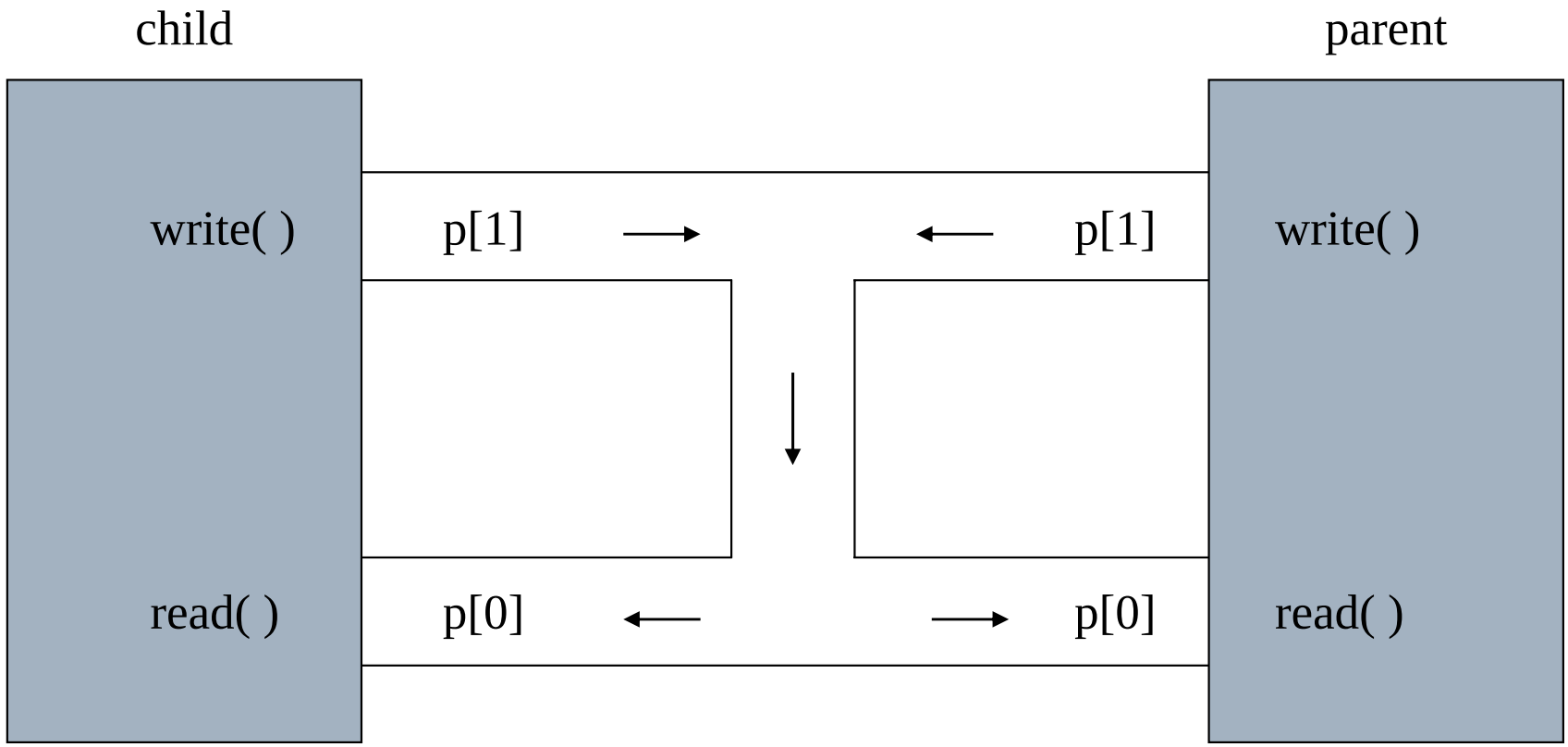
< ex_2.c >

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

#define MSGSIZE  16

char *msg1 = "hello, world #1";
char *msg2 = "hello, world #2";
char *msg3 = "hello, world #3";

main()
{
        char inbuf[MSGSIZE];
         int p[2], j;
         pid_t pid;

        /* 파이프를 개방한다. */
        if (pipe (p) == -1)
        {
                perror ("pipe call");
                exit (1);
         }
```

< ex_2.c >

```
        switch (pid = fork()){
        case -1:
                perror ("fork call");
                exit (2);
        case 0:
                write (p[1], msg1, MSGSIZE);
                write (p[1], msg2, MSGSIZE);
                write (p[1], msg3, MSGSIZE);
                break;
        default:
                for (j = 0; j < 3; j++) {
                        read (p[0], inbuf, MSGSIZE);
                        printf ("%s\n", inbuf);
                }
                wait (NULL);
        }

        exit (0);
}
```

pipe()



<두 번째 파이프 사용 예>

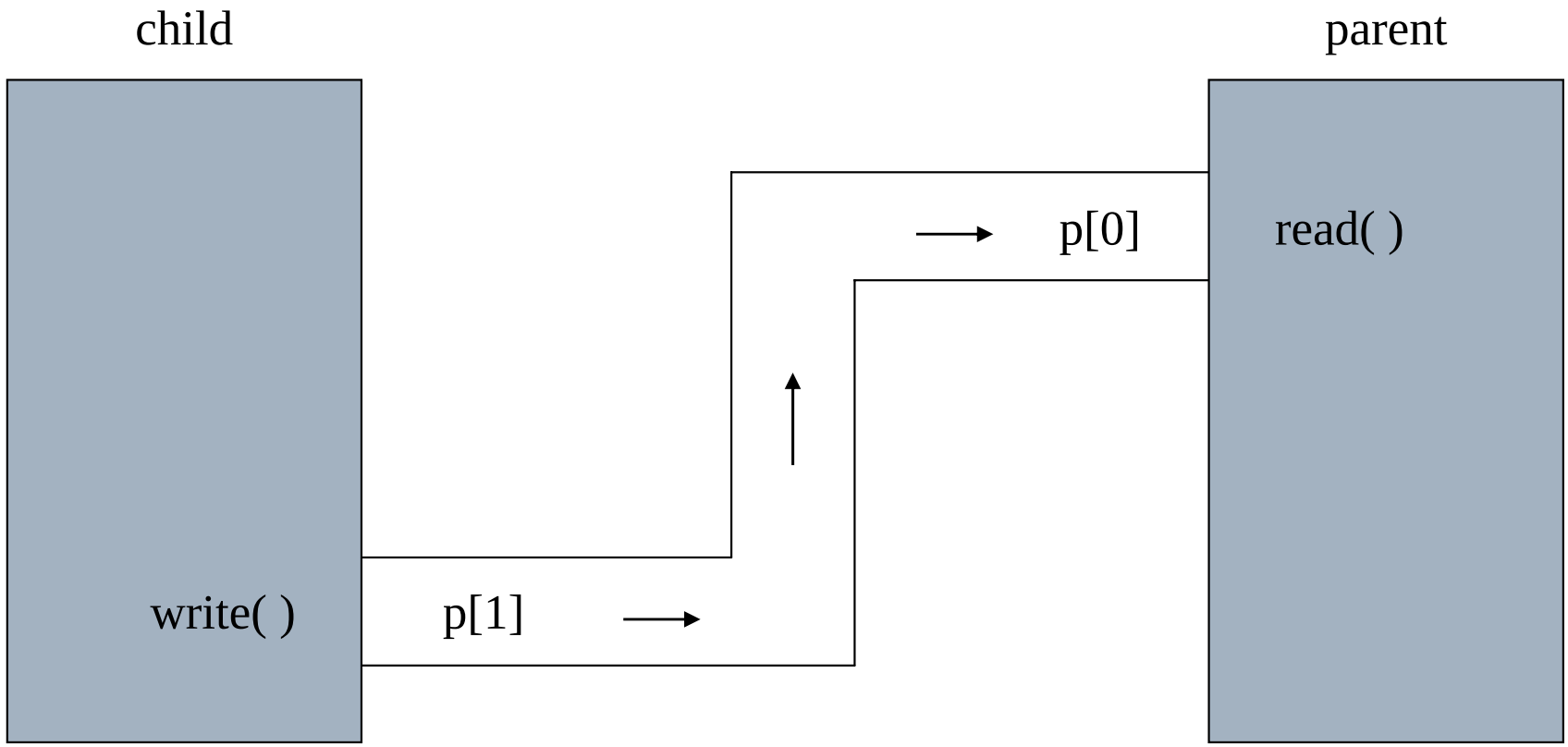
< ex_3.c >

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

#define MSGSIZE  16

char *msg1 = "hello, world #1";
char *msg2 = "hello, world #2";
char *msg3 = "hello, world #3";

main()
{
        char inbuf[MSGSIZE];
        int p[2], j;
        pid_t pid;

        /* 파이프를 개방한다. */
        if (pipe(p) == -1)
        {
                perror ("pipe call");
                exit (1);
        }
```

< ex_3.c >

```
        switch (pid = fork()){
        case -1:
                perror ("fork call");
                exit (2);
        case 0:
                close (p[0]);
                write (p[1], msg1, MSGSIZE);
                write (p[1], msg2, MSGSIZE);
                write (p[1], msg3, MSGSIZE);
                break;
        default:
                close (p[1]);
                for (j = 0; j < 3; j++)
                {
                        read (p[0], inbuf, MSGSIZE);
                        printf ("%s\n", inbuf);
                }
                wait (NULL);
        }

        exit (0);
}
```

pipe()



<세 번째 파이프 사용 예>

파이프의 크기

- 파이프에 들어 있는 자료가 일정량을 초과하면, 그 후의 write는 봉쇄된다.

- 파이프의 용량을 초과할 가능성이 있는 write가 시도되면 프로세스는 다른 프로세스에 의하여 자료가 읽혀져 파이프에 충분한 공간이 마련될 때까지 수행이 일시 중단된다.

< ex_4.c >

```
#include <signal.h>
#include <unistd.h>
#include <limits.h>

int count;
void alrm_action(int);

main() {
        int p[2];
        int pipe_size;
        char c = 'x';
        static struct sigaction act;

        act.sa_handler = alrm_action;
        sigfillset (&(act.sa_mask));

        /* 파이프를 생성한다 */
        if (pipe(p) == -1)
        {
                perror ("pipe call");
                exit (1);
        }
```

< ex_4.c >

```
        /* 파이프의 크기를 결정한다. */
        pipe_size = fpathconf (p[0], _PC_PIPE_BUF);
        printf ("Maximum size of write to pipe: %d bytes\n", pipe_size);

        sigaction (SIGALRM, &act, NULL);

        while (1) {
                alarm (20);
                write(p[1], &c, 1);
                alarm(0);
                if ((++count % 1024) == 0)
                        printf ("%d characters in pipe\n", count);
        }
}

void alrm_action (int signo){
        printf ("write blocked after %d characters\n", count);
        exit (0);
}
```

파이프 닫기

- 쓰기 전용 파일 기술자를 닫았을 때 :
  - 자료를 쓰기 위해 해당 파이프를 개방한 다른 프로세스가 존재하는 경우에는 아무 일도 일어나지 않는다.
  - 파이프에 자료를 쓰는 프로세스가 더 이상 없고, 파이프가 비어 있으면, 그 파이프로부터 자료를 읽으려는 프로세스는 아무 자료도 읽을 수가 없다.
  - 파이프로부터 자료를 읽기를 기다리며 잠들어 있던 프로세스를 모두 깨우고, 이들의 read() 호출은 0을 반환한다. 따라서 자료를 읽는 프로세스에게는 보통 파일의 끝에 도달한 것과 같은 효과가 발생한다.

파이프 닫기

- 읽기 전용 파일 기술자를 닫았을 때 :
  - 자료를 읽기 위해 해당 파이프를 개방한 프로세스가 아직 남아 있는 경우에는 아무 일도 발생하지 않는다.
  - 파이프로부터 자료를 읽어 들이는 프로세스가 더 이상 없으면 그 파이프에 자료를 쓸 수 있기를 기다리던 모든 프로세스는 커널로부터 SIGPIPE 시그널을 받는다. 이때 시그널이 포착되지 않으면 해당 프로세스는 종료한다. 시그널이 포착되면 인터럽트 루틴이 수행된 후에 write()는 -1을 반환한다.
  - 이후에 그 파이프에 자료를 쓰려고 시도하는 프로세스도 역시 SIGPIPE 시그널을 받는다.

봉쇄되지 않는 read와 write

- 파이프에 대한 read, write는 봉쇄될 수 있다.
- 하지만, 어떤 파이프에서 자료를 얻을 때까지 여러 개의 파이프를 차례로 조사(poll)하고자 할 때는 봉쇄되면 안 된다.
- 이에 fcntl()을 사용하여 해결한다.
  - 예) **fcntl(filedec, F_SETFL, O_NONBLOCK)**

< ex_5.c >

```
#include <fcntl.h>
#include <errno.h>

#define MSGSIZE 6

int parent (int *);
int child (int *);

char *msg1 = "hello";
char *msg2 = "bye!!";

main()
{
        int pfd[2];

        /* 파이프를 개방한다 */
        if(pipe (pfd) == -1)
                printf ("pipe call");

        /* p[0]의 O_NONBLOCK 플래그를 1로 설정한다 */
        if (fcntl (pfd[0], F_SETFL, O_NONBLOCK) == -1)
                printf ("fcntl call");
```

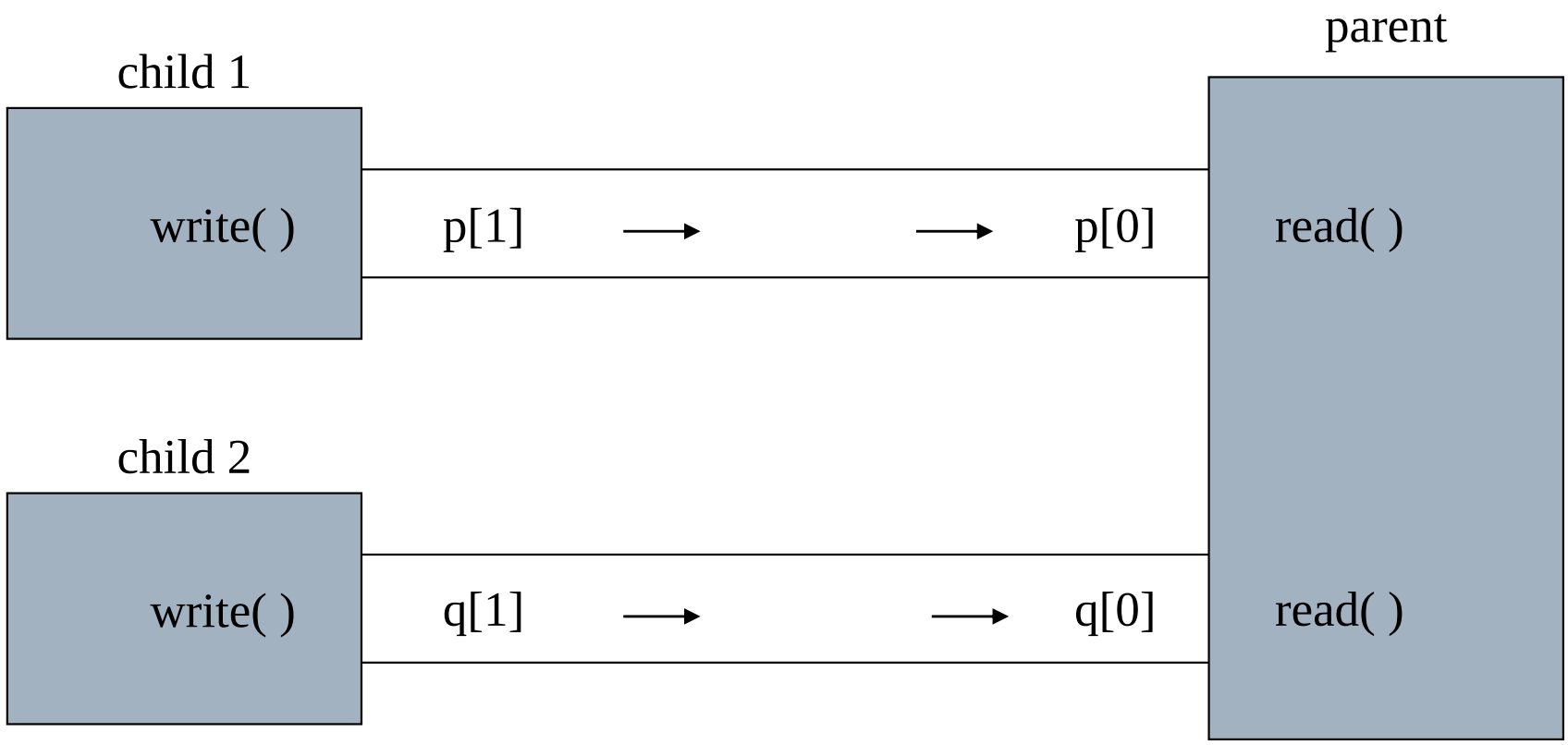
< ex_5.c >

```
        switch(fork()){
                case 0:                 /* 자식 */
                        child(pfd);
                default:                /* 부모 */
                parent (pfd);
        }
 }

int child(int p[2]) {
        int count;
        close (p[0]);

        for (count= 0; count < 3; count++)  {
                write (p[1], msg1, MSGSIZE);
                sleep(3);
        }

        /* 마지막 메시지를 보낸다 */
        write (p[1], msg2, MSGSIZE);
        exit (0);
}
```

< ex_5.c >

```
int parent (int p[2]) {
        int nread;
        char buf[MSGSIZE];

        close (p[1]);
        for(;;)  {
                switch (nread = read(p[0], buf, MSGSIZE)){
                        case -1:    /* 파이프에 아무것도 없는지 검사한다. */
                                    if (errno == EAGAIN){
                                                printf ("(pipe empty)\n");
                                                sleep (1);
                                                break;
                                    }
                                    else
                                                printf ("read call");
                        case 0:     /* 파이프가 닫혔음. */
                                    printf ("End of conversation\n");
                                    exit (0);
                        default:
                                    printf ("MSG=%s\n", buf);
                }
        }
 }
```

다수의 파이프 취급



select()

Usage

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

int select(int nfds, fd_set *readfds, fd_set *writefds,
        fd_set *errorfds, struct timeval *timeout);
```

< ex_6.c >

```
#include <sys/time.h>
#include <sys/wait.h>
#include <stdio.h>

#define MSGSIZE   6

char *msg1 = "hello";
char *msg2 = "bye!!";

void parent(int [][]);
int child(int []);
main() {
        int pip[3] [2];
        int i;

        for (i = 0; i < 3; i++) {
                if (pipe(pip[i]) == -1)
                        printf ("pipe call");

                switch (fork()){
                        case 0:
                                child (pip[i]);
                }
        }

        parent (pip);
        exit (0);
}
```

< ex_6.c >

```
/* 부모는 세 개의 파이프에 전부 귀를 기울이고 있다. */
void parent(int p[3][2]) {
        char buf[MSGSIZE], ch;
         fd_set set, master;
        int i;

        /* 모든 원하지 않는 화일 기술자를 닫는다 */
        for (i = 0; i < 3; i++)
                close (p[i] [1]);

        /* select 시스템 호출의 비트 마스크를 설정한다. */
        FD_ZERO (&master);
        FD_SET (0, &master);

        for (i = 0; i <3; i++)
                FD_SET (p[i] [0], &master);
```

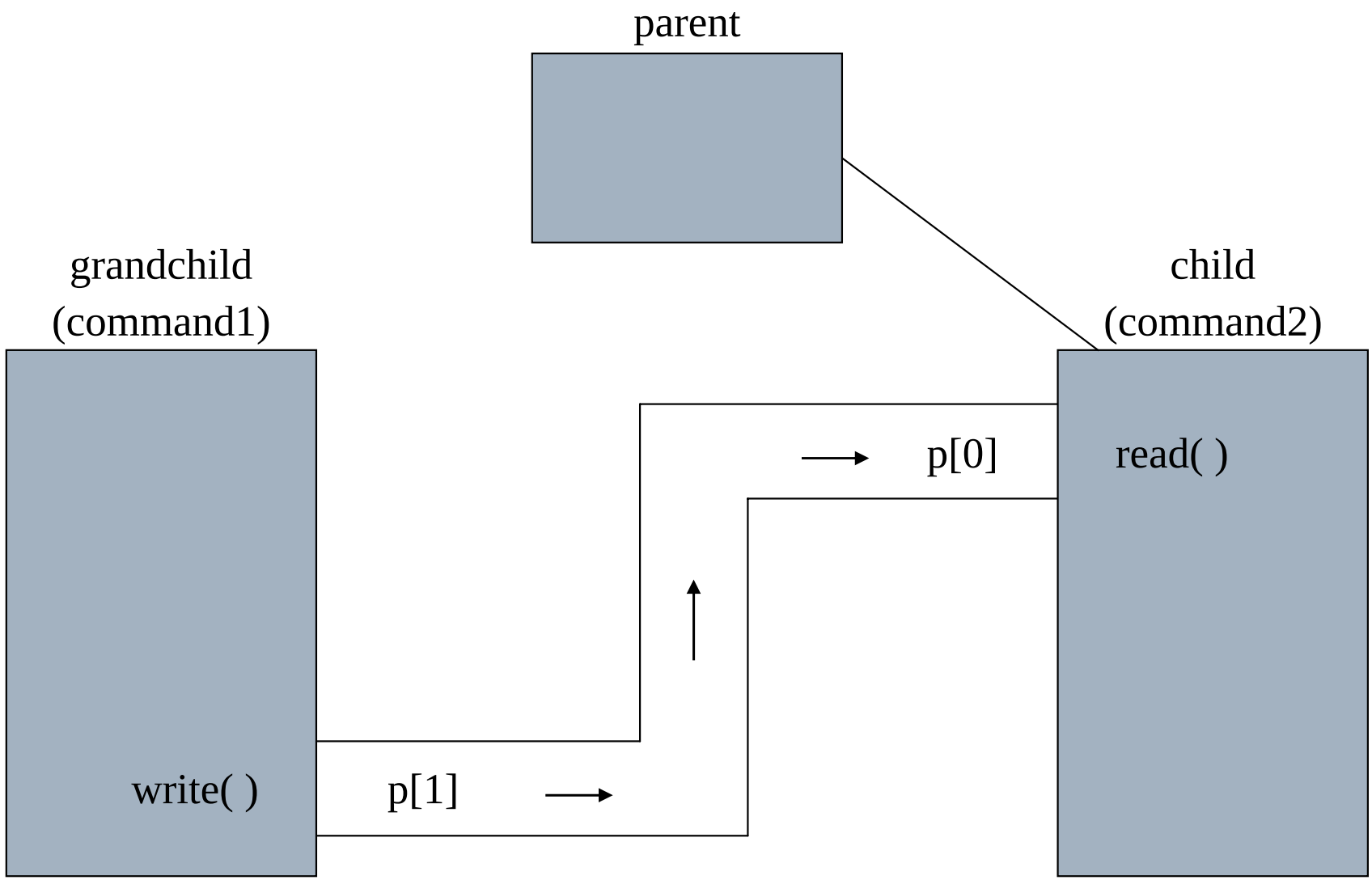
< ex_6.c >

```
        /* 타임아웃 없이 select를 호출한다. 사건이 발생할 때까지 select는 봉쇄될 것이다 */
        while (set = master, select (p[2] [0]+1, &set, NULL, NULL, NULL) > 0) {
                /* 표준 입력, 즉 화일 기술자 0에 있는 정보를 잊어버리면 안됨. */
                if (FD_ISSET(0, &set)){
                        printf ("From standard input...");
                        read (0, &ch, 1);
                        printf ("%c\n", ch);
                }

                for (i = 0; i < 3; i++){
                        if (FD_ISSET(p[i] [0], & set)){
                                if (read(p[i][0], buf[MSGSIZE])>0) {
                                        printf ("Message from child%d\n", i);
                                        printf ("MSG=%s\n",buf);
                                }
                        }
                }

                /* 서버는 모든 자식이 죽으면 주 프로그램으로 복귀한다. */
                if (waitpid (-1, NULL,WNOHANG) == -1)
                        return;
        }
}
```

< ex_6.c >

```
int child(int p  [2]) {
        int count;

        close (p[0]);

        for (count = 0; count < 2; count++)
        {
                write (p[1], msg1, MSGSIZE);
                /* 임의의 시간 동안 중지한다. */
                sleep (getpid() % 4);
        }

        /* 최종 메시지를 보낸다. */
        write (p[1], msg2, MSGSIZE);
        exit (0);
}
```

파이프와 exec( )호출


parent
grandchild
(command1)
child
(command2)
p[0]
read( )
write( )
p[1]

< ex_7.c >

```
#include <stdio.h>

int join (char *com1[], char *com2[]) {
        int p[2], status;

        switch (fork()){
                case 0:     /* 자식 */
                            break;
                default:    /* 부모 */
                            wait(&status);
                            return (status);
        }

        if (pipe(p) == -1)
                printf ("pipe call in join");
```

< ex_7.c >

```
        switch (fork()){
                case 0:
                        dup2 (p[1],1);      /* 표준 출력이 파이프로 가게 한다. */

                        close (p[0]);       /* 화일 기술자를 절약한다. */
                        close (p[1]);

                        execvp (com1[0], com1);
                        printf("1st execvp call in join");
                default:
                        dup2(p[0], 0);      /* 표준 입력이 파이프로부터 오게 한다 */
                        close (p[0]);
                        close (p[1]);
                        execvp (com2[0], com2);
                        printf ("2nd execvp call in join");
        }
}
```

< ex_7.c >

```
main()
{
        char *one[4] = {"ls", "-l", "/usr/lib", NULL};
        char *two[3] = {"grep", "^d", NULL};
        int ret;

        ret = join (one, two);
        printf ("join returned %d\n", ret);
        exit (0);
}
```

FIFO와 이름형 파이프

- 파이프의 결점
  - 부모와 자식 프로세스간에만 사용할 수 있다.
  - 파이프는 영구히 존재할 수 없다.

- FIFO
  - 파이프의 결점을 보완하기 위한 파이프의 변종
  - 영구적이며, UNIX 파일 이름을 부여받는다.

mkfifo()

Usage
#include <sys/types.h> **#include <sys/stat.h>** **int mkfifo(const chat *pathname, mode_t mode);**

< sendmsg.c >

```
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#define MSGSIZ                  63

char *fifo = "fifo";

main (int argc, char **argv) {
        int fd, j, nwrite;
        char msgbuf[MSGSIZ+1];

        if ((fd = open(fifo, O_WRONLY | O_NONBLOCK)) < 0)
                printf ("fifo open failed");

        for ( j = 1; j < argc; j++)  {
                if (strlen(argv[j]) > MSGSIZ) {
                        fprintf (stderr, "message too long %s\n", argv[j]);
                        continue;
                }

                strcpy (msgbuf, argv[j]);

                if ((nwrite = write (fd, msgbuf, MSGSIZ+1)) == -1)
                        printf ("message write failed");
        }
        exit (0);
}
```

< receivemsg.c >

```
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#define MSGSIZ                  63

char *fifo = "fifo";

main (int argc, char **argv) {
        int fd;
        char msgbuf[MSGSIZ+1];

        if (mkfifo(fifo, 0666) == -1)  {
                if (errno != EEXIST)
                        printf ("receiver: mkfifo");
        }

        if ((fd = open(fifo, O_RDWR)) < 0)
                printf ("fifo open failed");

        for(;;)  {
                if (read(fd, msgbuf, MSGSIZ+1) <0)
                        printf ("message read failed");
                printf ("message received:%s\n", msgbuf);
        }
}
```