11장 파이프를 이용한 통신

- 서론
- 예제 프로그램
- 함수
  - pipe
  - pathconf / fpathconf
  - select
  - mkfifo
  - dup2

한빛미디어(주)

- 파이프를 사용하여 통신을 하기 위한 시스템 호출/표준 라이브러리 함수

함수	의미
pipe	프로세스간 통신을 위해 파이프를 생성한다.
fpathconf	파일 시스템의 정보를 알아본다.
pathconf	파일 시스템의 정보를 알아온다.
select	파일 기술자의 집합 내에서 상태가 변한 파일 기술자를 관찰한다.
mkfifo	네임드 파이프인 FIFO를 생성한다.
dup2	파일 기술자의 사본을 생성한다.

【예제11-1】 ex 11-01.c(1/4)


```
01 #include <unistd.h>
02 #include <sys/types.h>
03 #include <fcntl.h>
04
05 #define MSGSIZE 100
06
07 main()
08 {
09     pid_t   pid1, pid2;
10     Int     filedes, p1[2], p2[2];
11     fd_set  initset, newset;
12     int     nread;
13     char    msg[MSGSIZE];
14
15     pipe(p1);
16     pipe(p2);
17
18     pid1 = pid2 = 0;
19     pid1 = fork();
20     if(pid1 > 0)
21         pid2 = fork();
```

05 통신용 메시지의 길이

10 파이프와 관련된 파일 기술자

11 파일 기술자 집합(select 함수에서 사용)

15 두 쌍의 파이프를 생성한다

19 자식 프로세스를 생성한다

20 부모 프로세스는 자식 프로세스를 하나 더 생성한다

표준입력 스트림

【예제11-1】 ex 11-01.c(2/4)


```
22    if(pid1 > 0 && pid2 > 0) {   /* parent */
23       printf("[parent] hello!₩n");
24       close(p1[1]);
25        close(p2[1]);
26
27       FD_ZERO(&initset);
28       FD_SET(p1[0], &initset);
29       FD_SET(p2[0], &initset);
30
31       newset = initset;
32       while(select(p2[0]+1, &newset, NULL, NULL, NULL) > 0)
33        {
34          if(FD_ISSET(p1[0], &newset))
35             if(read(p1[0], msg, MSGSIZE) > 0)
36                printf("[parent] %s₩n", msg);
37          if(FD_ISSET(p2[0], &newset))
38             if(read(p2[0], msg, MSGSIZE) > 0)
39                printf("[parent] %s₩n", msg);
40          newset = initset;
41       }
42    }
```

22 부모 프로세스르 수행할 부분

24 사용하지 않는 파이프를 닫는다

27 파일 기술자 집합을 초기화하고 파이프에 해당하는 두 개의 파일 기술자를 등록한다

32 newset에 등록된 파일 기술자 중에서 읽을거리가 있는 상태로 변한 파일 기술자를 찾는다

36 p1[0]과 p2[0] 중에서 읽을거리가 있는 파일 기술자로부터 메시지를 읽어와 표준출력한다.

표준입력 스트림

【예제11-1】 ex 11-01.c(3/4)


```
43    else if(pid1 == 0 && pid2 == 0)
44    {        /* 1st child */
45       printf("[fork1] hello!₩n");
46       close(p1[0]);
47       close(p2[0]);
48       close(p2[1]);
49
50       dup2(p1[1], 1);
51       execl("ex11-01c", "ex11-01c", (char *)0);
52    }
```

43 첫번째 자식 프로세스

46 사용하지 않는 파이프를 닫는다

50 표준 출력에 해당하는 파일 기술자 1을 쓰기용 파이프의 사본으로 만든다

51 ex11-01c 프로그램을 실행한다

표준입력 스트림

【예제11-1】 ex 11-01.c(4/4)


```
53    else if(pid1 > 0 && pid2 == 0)
54    {          /* 2nd child */
55       printf("[fork2] hello!₩n");
56       close(p1[0]);
57       close(p1[1]);
58       close(p2[0]);
59       write(p2[1], "from fork2 via pipe", MSGSIZE);
60
61       mkfifo("./fifo", 0666);
62       filedes = open("./fifo", O_RDWR);
63       nread = read(filedes, msg, MSGSIZE);
64       printf("%s (%d)₩n", msg, nread);
65       close(filedes);
66       unlink("./fifo");
67    }
68    else
69       exit(1);
70 }
```

53 두번째 자식 프로세스

56 사용하지 않는 파이프를 닫는다

59 파이프로 메시지를 송신한다

61 네임드 파이프 “fifo”를 만든다

62 fifo를 열고 메시지를 읽어와 표준 출력한다.

66 fifo를 삭제한다

표준입력 스트림

【예제11-1】 ex 11-01c.c


```
01 #include <unistd.h>
02 #include <fcntl.h>
03
04 #define MSGSIZE 100
05
06 main()
07 {
08     int filedes;
09
10     printf("[exec] standard output₩n");
11    sleep(1);
12
13    filedes = open("./fifo", O_WRONLY);
14     write(filedes, "from exec via FIFO", MSGSIZE);
15     close(filedes);
16 }
```

04 메시지의 길이

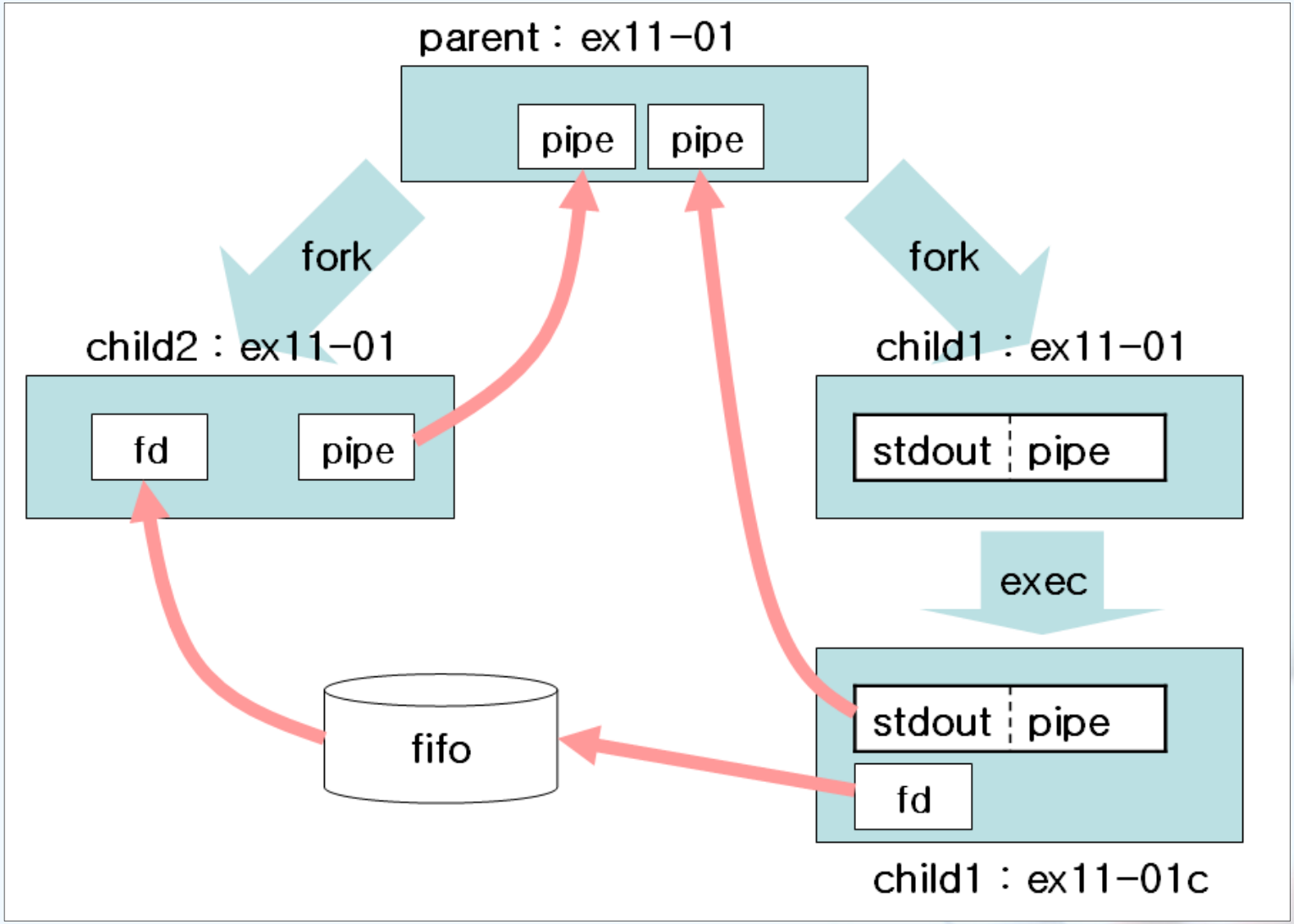
10 표준 출력으로 문자열을 출력한다. 표준 출력이 부모 프로세스와 연결된 파이프의 사본이 되어 있으므로 파이프를 통한 출력이 된다

14 “fifo”를 열어 메시지를 쓴다. “fifo”에 쓰여진 메시지는 첫 번째 자식 프로세스가 읽어갈 것이다.

표준입력 스트림

예제 프로그램의 동작 과정

(ex11-01.c와 ex11-01c.c)



실행 결과

```
$ ex11-01
[parent] hello!
[fork1] hello!
[fork2] hello!
[parent] from fork2 via pipe
from exec via FIFO (100)
[parent] [exec] standard output

^C
$
```

프로세스 간 통신을 위해 파이프를 생성한다

```
#include <unistd.h>

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

filedes	파일 기술자로 filedes[0]은 읽기용이고 filedes[1]은 쓰기용이다.
반환값	호출이 성공하면 0을 반환하고, 실패하면 -1을 반환한다.

- **pipe 아이노드를 가리키는 한 쌍의 파일 기술자를 생성**
  - **filedes[0]은 읽기용**
  - **filedes[1]은 쓰기용**

  - **※ 일반 파일을 다루기 위한 파일 기술자와 동일하다. 일반 파일이냐 파이프(특수 파일)냐의 차이지만 프로세스 입장에서는 굳이 따질 필요가 없다.**

【예제11-2】 ex 11-02.c

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
17     for(i = 0; i < 3; i++) {
18         printf("input a message₩n");
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
input a message
banana is yellow
input a message
cherry is red

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

【예제11-3】 ex 11-03.c(1/2)


```
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    }
```

표준입력 스트림

【예제11-3】 ex 11-03.c(2/2)


```
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  }
```

표준입력 스트림

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

【예제11-4】 ex 11-04.c(1/2)

※ 모든 프로세스는 하나의 파이프에 대해서 읽기나 쓰기 하나만 가능해야 한다.

```
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[] = {"apple is red", "banana is yellow", "cherry is red"};
10     char buffer[SIZE];
11     int filedes[2], nread, i;
12     pid_t pid;
13
14     if(pipe(filedes) == -1) {
15         printf("fail to call pipe()₩n");
16         exit(1);
17     }
```

【예제11-4】 ex 11-04.c(2/2)


```
18      if((pid = fork()) == -1) {
19          printf("fail to call fork()₩n");
20          exit(1);
21      }
22      else if(pid > 0) {
23          for(i = 0; i < 3; i++) {
24              strcpy(buffer, msg[i]);
25              write(filedes[1], buffer, SIZE);
26          }
27
28          nread = read(filedes[0], buffer, SIZE);
29          printf("[parent] %s₩n", buffer, nread);
30
31          write(filedes[1], buffer, SIZE);
32          printf("[parent] bye!₩n");
33      }
34      else {
35          for(i = 0; i < 3; i++) {
36              nread = read(filedes[0], buffer, SIZE);
37              printf("[child] %s₩n", buffer, nread);
38          }
39          printf("[child] bye!₩n");
40      }
41  }
```

표준입력 스트림

```
$ ex11-04
[parent] apple is red
[parent] bye!
[child] banana is yellow
[child] cherry is red
[child] apple is red
[child] bye!
$
```

【예제11-5】 ex 11-05.c


```
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.

파이프를 통해 전달하는 메시지의 크기(길이)

그 길이가 고정적일 수도 있고 가변적일 수도 있다.

파이프로 메시지를 쓰거나 파이프로부터 읽을 때

- 메시지의 실제 크기로 다룰 경우
  - 가변적인 크기의 실제 메시지 내용만큼만 파이프에 쓰기 때문에 파이프의 최대 크기만큼 효율적으로 사용 가능하다.
  - 송신 측이 보낸 메시지의 실제 크기를 수신 측에서 일일이 확인하는 것이 힘들다.
- 버퍼의 고정된 크기로 다룰 경우 (메시지의 실제 크기는 상관하지 않음)
  - 길이가 짧은 메시지일 경우 낭비되는 공간이 많아 효율적이지 못하다.
  - 송신 측과 수신 측에서 전달되는 메시지의 크기를 미리 약속할 수 있다.

【예제11-6】 ex 11-06.c


```
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
$
```

【예제11-7】 ex 11-07.c


```
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], 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
$
```

【예제11-8】 ex 11-08.c(일부)


```
...
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 }
```

표준입력 스트림

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

지정한 파일과 관련된 여러 가지 정보를 확인한다

```
#include <unistd.h>

long fpathconf(int filedes, int name);
long pathconf(char *path, int name);
```

filedes	파일 기술자이다.
path	파일에 대한 경로이다.
name	파일에 대해 설정된 항목의 이름이다.
반환값	호출이 성공하면 name으로 선택한 항목으로 설정된 한계 값이 반환되고, 한계값이 설정되어 있지 않거나 호출이 실패하면 -1을 반환한다.

- **파일 기술자나 경로로 지정한 파일에 대해서 지정한 항목의 정보를 알아온다.**

int name

이름	의미
_PC_LINK_MAX	링크의 개수를 의미한다.
_PC_NAME_MAX	파일 이름의 최대 길이를 의미한다.
_PC_PATH_MAX	상대 경로의 최대 길이를 의미한다.
_PC_PIPE_BUF	파이프 버퍼의 최대 크기를 의미한다. 우리가 본 장에서 가장 관심을 가져야 할 값이다. filedes는 반드시 파이프나 FIFO를 가리켜야 하고 path는 반드시 FIFO를 가리켜야 한다.

- **파이프의 크기를 미리 확인하여 메시지의 크기를 결정한다.**

【예제11-9】 ex 11-09.c(1/2)


```
01 #include <unistd.h>
02 #include <signal.h>
03 #include <limits.h>
04
05 int nc;
06 void alarm_action(int);
07
08 main()
09 {
10     int filedes[2];
11     char msg = 'A';
12
13     struct sigaction act;
14     act.sa_handler = alarm_action;
15     sigfillset(&(act.sa_mask));
16
17     if(pipe(filedes) == -1) {
18         printf("fail to call pipe()₩n");
19         exit(1);
20     }
```

표준입력 스트림

【예제11-9】 ex 11-09.c(2/2)


```
21      printf("PIPE size : %d bytes₩n", fpathconf(filedes[1], _PC_PIPE_BUF));
22      nc = 0;
23      sigaction(SIGALRM, &act, NULL);
24      alarm(1);
25      while(1) {
26          write(filedes[1], &msg, 1);
27          nc++;
28      }
29  }
30
31  void alarm_action(int signo)
32  {
33      printf("₩n₩nblocked after %d characters₩n", nc);
34      exit(1);
35  }
```

```
$ ex11-09
PIPE size : 4096 bytes
blocked after 4096 characters
$
```

표준입력 스트림

- 지정한 파일 기술자의 집합에서 상태가 바뀐 파일 기술자가 있는지 관찰한다.

```
#include <sys/time.h>
#include <sys/types.h>
#niclude <unistd.h>

int select(int n, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);

FD_CLR(int fd, fd_set *set);
FD_ISSET(int fd, fd_set *set);
FD_SET(int fd, fd_set *set);
FD_ZERO(fd_set *set);
```

int select(int *n*, fd_set **readfds*, fd_set **writefds*,
fd_set **exceptfds*, struct timeval **timeout*);

n	readfds, wirtefds, exceptfds에 포함된 파일 기술자 중 가장 큰 값에 1을 더한 값이다.
readfds	읽을 수 있는 값이 있는지 관찰하게 될 파일 기술자들의 집합이다. (읽기 작업이 블록되지 않을 것인지를 관찰한다.)
writefds	쓰기 작업이 가능한지 관찰하게 될 파일 기술자들의 집합이다. (쓰기 작업이 블록되지 않을 것인지를 관찰한다.)
exceptfds	readfds, writefds와 같이 예외처리를 위한 파일 기술자들의 집합이다.
timeout	select가 리턴할 때까지 주어진 시간으로 지정된 시간만큼 파일 기술자들을 감시한다.
fd	파일 기술자이다.
set	파일 기술자들의 집합이다.
반환값	호출이 성공할 경우 파일 기술자 집합에 포함된 파일 기술자 중 하나의 번호가 반환된다. 주어진 시간이 지나면 0을 반환한다. 호출이 실패할 경우 -1을 반환한다.

하나의 프로세스가 여러 개 프로세스와 동시에 메시지를 주고 받을 경우

- 메시지 흐름마다 하나의 파이프가 필요하다.
  - 한 개의 파이프를 공유하는 것이 아니다.
- 각 파이프를 통해서 전달되는 메시지의 순서가 항상 같지 않다.
  - 파이프를 위해 할당 받은 파일 기술자의 순서대로 메시지를 처리하는 것은 좋지 않다.
  - 메시지가 도착한 순서대로 처리하는 것이 좋다.

- select 사용
  - 파일 기술자의 집합을 만든 뒤에 상태가 변하는 것이 있는지 관찰하다가 상태가 변한 파일 기술자가 발생하면 이를 처리해준다.

    예) 2개의 읽기용 파이프를 관찰하다가 메시지가 도착한 파이프가 발견되면 이를 읽어온 뒤에 다시 관찰 상태가 된다.

struct timeval

```
struct timeval {
    long    tv_sec;         /* seconds */
    long    tv_usec;        /* microseconds */
};
```

- tv_sec나 tv_usec 중에 하나가 0이면 Non-blocking으로 파일 기술자 집합을 감시한다.

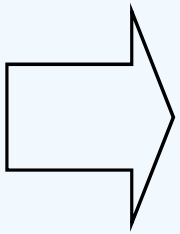
유용한 매크로

파일 기술자의 집합은 비트 마스크 형태이다.

- 특정 비트의 값이 1이면 해당 비트가 가리키는 파일 기술자가 읽기 또는 쓰기가 가능함을 의미한다.

매크로	기능
FD_SET	파일 기술자 집합에 지정한 파일 기술자를 추가한다.
FD_CLR	파일 기술자 집합에서 지정한 파일 기술자를 제거한다.
FD_ISSET	파일 기술자 집합에서 특정 파일 기술자가 설정되어 있는지를 확인한다. select가 리턴했을 때 유용하다.
FD_ZERO	파일 기술자 집합을 초기화한다.

```
fd_set initset;
…
FD_ZERO(&initset);
FD_SET(pipe[0], &initset);
…
```



```
select(…, &initset, …);
…
FD_ISSET(pipe[0], &initset);
```

【예제11-10】 ex 11-10.c(1/3)


```
01 #include <sys/time.h>
02 #include <sys/wait.h>
03 #include <sys/types.h>
04 #include <unistd.h>
05
06 #define MSGSIZE 16
07
08 void parent(int [][]);
09 int child(int []);
```

```
10 void onerror(char *msg)
11 {
12     printf("%s");
13     exit(1);
14 }
```

```
15 main()
16 {
17     int p1[2], p2[2];
18     char msg[MSGSIZE];
19     int i;
20     pid_t pid1, pid2;
21     fd_set initset, newset;
22
23     pid1 = pid2 = 0;
24
25     if(pipe(p1) == -1)
26         onerror("fail to call pipe() #1₩n");
27     if(pipe(p2) == -1)
28         onerror("fail to call pipe() #2₩n");
```

표준입력 스트림

[예제11-10] ex 11-10.c(2/3)


```
29      if((pid1 = fork()) == -1)
30          onerror("fail to call fork() #1₩n");
31      if(pid1 > 0)
32      if((pid2 = fork()) == -1)
33          onerror("fail to call fork() #2₩n");
34
35      if(pid1 > 0 && pid2 > 0) {  /* parent process */
36          printf("parent: %d₩n", getpid());
37          close(p1[1]); close(p1[1]);
38
39          FD_ZERO(&initset);
40          FD_SET(p1[0], &initset);
41          FD_SET(p2[0], &initset);
42
43          newset = initset;
44          while(select(p2[0] + 1, &newset, NULL, NULL, NULL) > 0) {
45              if(FD_ISSET(p1[0], &newset))
46                  if(read(p1[0], msg, MSGSIZE) > 0)
47                      printf("[parent] %s from child1₩n", msg);
48              if(FD_ISSET(p2[0], &newset))
49                  if(read(p2[0], msg, MSGSIZE) > 0)
50                      printf("[parent] %s from child2₩n", msg);
51              newset = initset;
52          }
53      }
```

[예제11-10] ex 11-10.c(3/3)


```
54      else if(pid1 == 0 && pid2 == 0) {     /* first child */
55          printf("child1: %d₩n", getpid());
56          close(p1[0]); close(p2[0]); close(p2[1]);
57
58          for(i = 0; i < 3; i++) {
59              sleep((i + 1) % 4);
60              printf("child1: send message %d₩n", i);
61              write(p1[1], "i'm child1", MSGSIZE);
62          }
63          printf("child1: bye!₩n");
64          exit(0);
65      }
66      else if(pid1 > 0 && pid2 == 0) {     /* second child */
67          printf("child2: %d₩n", getpid());
68          close(p1[0]); close(p1[1]); close(p2[0]);
69
70          for(i = 0; i < 3; i++) {
71              sleep((i + 3) % 4);
72              printf("child2: send message %d₩n", i);
73              write(p2[1], "i'm child2", MSGSIZE);
74          }
75          printf("child2: bye!₩n");
76          exit(0);
77      }
78  }
```

표준입력 스트림

실행 결과

```
$ ex11-10
parent: 5347
child1: 5348
child2: 5349
child1: send message 0
[parent] i'm child1 from child1
child2: send message 0
child2: send message 1
[parent] i'm child2 from child2
[parent] i'm child2 from child2
child1: send message 1
[parent] i'm child1 from child1
child2: send message 2
child2: bye!
[parent] i'm child2 from child2
child1: send message 2
child1: bye!
[parent] i'm child1 from child1
^C
$
```

두 개의 자식 프로세스가 무작위 순서로 보내는 메시지를 부모 프로세스가 select를 사용하여 도착하자 마자 처리해주고 있다.

첫 번째 자식 프로세스
1초, 2초, 3초 간격

두 번째 자식 프로세스
3초, 0초, 1초 간격

두 개의 서로 다른 프로세스가 파이프를 통해서 메시지를 주고 받는 방법

예) $ ls –al | more

파일 기술자의 계승

- 부모 프로세스가 개방한 파일의 상태는 fork나 exec를 호출하여 생성된 자식 프로세스에게 그대로 계승된다.
- 부모 프로세스에 의해 개방된 파일은 여전히 자식 프로세스에게도 개방되어져 있다.
  - fork일 경우 파일 기술자가 담긴 변수도 계승되므로 부모 프로세스가 개방한 파일을 자식 프로세스가 쉽게 사용할 수 있다.
  - exec일 경우 변수를 계승할 수 없다.

    표준 입력과 표준 출력을 파이프와 연결한 후에 exec를 호출하여 자식 프로세스를 생성한다.

    자식 프로세스의 표준 입,출력에 대한 작업은 파이프에 대한 작업이 된다.

특정 파일 기술자를 지정한 다른 파일 기술자의 사본으로 만든다

```
#include <unistd.h>

int dup2(int oldfd, int newfd);
```

oldfd	원본에 해당하는 파일 기술자이다.
newfd	oldfd의 사본에 해당하는 파일 기술자이다.
반환값	호출이 성공하면 파일 기술자를 반환하고, 실패하면 -1을 반환한다.

- **newfd로 지정한 파일 기술자를 oldfd로 지정한 파일 기술자의 사본으로 만든다.**

  즉, newfd에 대한 작업이 oldfd에 대한 작업으로 바뀐다.

표준 입출력을 파이프와 연결

```
pipe(p);              /* 파이프 생성 : p[0] 읽기용, p[1] 쓰기용 */
…
dup2(p[0], 0);        /* 파일 기술자 0인 표준 입력을 p[0]의 사본으로 만든다. */
…
dup2([[1], 1);        /* 파일 기술자 1인 표준 출력을 p[1]의 사본으로 만든다. */
```

- 위의 코드를 실행하고 난 후
  - 표준 입력에서 읽는 것은 p[0]에서 읽는 것과 같다.
  - 표준 출력으로 쓰는 것은 p[1]로 쓰는 것과 같다.

[예제11-11] ex 11-11.c(1/2)


```
01 #include <sys/types.h>
02 #include <unistd.h>
03
04 main()
05 {
06     char *msg[3] = {"apple is red₩n",
07                     "banana is yellow₩n", "cherry is red₩n"};
08     int p[2];
09     pid_t pid;
10     int cnt;
11
12     if(pipe(p) == -1) {
13         printf("fail to call pipe()₩n");
14         exit(1);
15     }
16
17     if((pid = fork()) == -1) {
18         printf("fail to call fork()₩n");
19         exit(1);
20     }
```

[예제11-11] ex 11-11.c(2/2)


```
21    else if(pid > 0) {
22        printf("[parent]₩n");
23        close(p[0]);
24        for(cnt = 0; cnt < 3; cnt++)
25            write(p[1], msg[cnt], strlen(msg[cnt]) + 1);
26    }
27    else {
28        printf("[child]₩n");
29        close(p[1]);
30        dup2(p[0], 0);
31        execlp("wc", "wc", (char *)0);
32        printf("[child] fail to call execlp()₩n");
33    }
34 }
```

```
$ ex11-11
[parent]
[child]
       3       9      47

$
```

※ 자식 프로세스는 표준 출력을 읽기용 파이프의 사본으로 만든 후에 "wc"를 실행한다. "wc"는 실행되면서 표준 출력에서 무언가를 읽으려고 한다.

【예제11-11】 ex 11-11.c(수정판)(1/2)


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

표준입력 스트림

【예제11-11】 ex 11-11.c(수정판)(2/2)


```
18    else if(pid > 0)
19    {
20        printf("[parent]₩n");
21        close(p[0]);
22        dup2(p[1], 1);
23        execlp("ls", "ls", "-al", (char *)0);
24        printf("[parent] fail to call execlp()₩n");
25    }
26    else
27    {
28        printf("[child]₩n");
29        close(p[1]);
30        dup2(p[0], 0);
31        execlp("wc", "wc", (char *)0);
32        printf("[child] fail to call execlp()₩n");
33    }
34 }
```

```
$ ex11-12
[parent]
[child]
      46      407     2856

$ ls -al | wc
      46      407     2856
$
```

네임드 파이프를 생성한다.

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

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

pathname	특수 파일인 FIFO의 경로 이름으로 일반 파일의 경로를 지정하는 것과 같다.
mode	pathname으로 지정한 특수 파일의 초기 접근 권한이다.
반환값	호출이 성공하면 0을 반환하고, 실패하면 -1을 반환한다.

- **pipe와 다른 점**
  - **pipe를 사용하여 만든 파이프는 프로세스가 종료하면 사라지는 임시 파이프이나 mkfifo로 만든 파이프는 프로세스의 생사와 상관없이 항상 존재한다.**

네임드 파이프(FIFO)를 사용한 통신

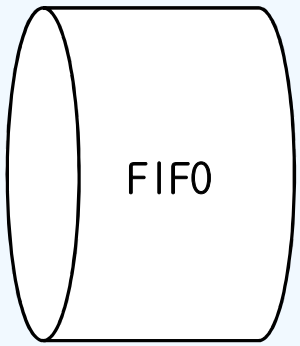
- fork나 exec로 호출되는 프로세스가 아닌, 서로 관련이 없는 프로세스도 네임드 파이프를 통해 메시지를 주고 받을 수 있다.

네임드 파이프로 통신을 수행하려는 프로세스는

- 네임드 파이프의 경로를 알고 있어야 한다.
- 네임드 파이프에 대한 권한이 있어야 한다.
- 일반 파일을 개방하여 읽고 쓰듯이 네임드 파이프를 사용하면 된다.

1. 쓰기용으로 개방

3. FIFO에 쓰기



2. 읽기용으로 개방 (해당 FIFO가 쓰기용으로 개방되지 않았으면 읽기용 개방은 블록됨)

4. FIFO에서 읽기 (해당 FIFO에서 읽을거리가 없을 경우 읽기 작업은 블록됨)

【예제11-13】 ex 11-13.c(1/2)

메시지를 수신하는 측

```
01 #include <fcntl.h>
02 #include <sys/stat.h>
03 #include <unistd.h>
04
05 #define MSGSIZE 64
06
07 main()
08 {
09     char msg[MSGSIZE];
10     int filedes;
11     int nread, cnt;
12
13     if(mkfifo("./fifo", 0666) == -1) {
14         printf("fail to call fifo()₩n");
15         exit(1);
16     }
```

표준입력 스트림

[예제11-13] ex 11-13.c(2/2)


```
17      if((filedes = open("./fifo", O_RDWR)) < 0) {
18          printf("fail to call fifo()₩n");
19          exit(1);
20      }
21
22      for(cnt = 0; cnt < 3; cnt++) {
23          if((nread = read(filedes, msg, MSGSIZE)) < 0) {
24              printf("fail to call read()₩n");
25              exit(1);
26          }
27
28          printf("recv: %s₩n", msg);
29      }
30
31      unlink("./fifo");
32  }
```

표준입력 스트림

【예제11-14】 ex 11-14.c(1/2)

메시지를 송신하는 측

```
01 #include <fcntl.h>
02 #include <sys/stat.h>
03 #include <unistd.h>
04
05 #define MSGSIZE 64
06
07 main()
08 {
09          char msg[MSGSIZE];
10          int filedes;
11          int cnt;
12
13          if((filedes = open("./fifo", O_WRONLY)) < 0)
14          {
15                  printf("fail to call open()₩n");
16                  exit(1);
17          }
```

【예제11-14】 ex 11-14.c(2/2)


```
18      for(cnt = 0; cnt < 3; cnt++)
19      {
20          printf("input a message: ");
21          scanf("%s", msg);
22
23          if(write(filedes, msg, MSGSIZE) == -1)
24          {
25              printf("fail to call write()₩n");
26              exit(1);
27          }
28
29          sleep(1);
30      }
31  }
```

표준입력 스트림

실행 결과

```
$ ls -al fifo
ls: fifo: No such file or directory
$ ex11-13 &
[1] 17296
$ ls -al fifo
prw-r--r--    1 usp      student          0 Nov 14 17:20 fifo
$ ex11-14
input a message: apple_is_red
recv: apple_is_red
input a message: banana_is_yellow
recv: banana_is_yellow
input a message: cherry_is_red
recv: cherry_is_red
[1]+  Done                    ex11-13
$
```