Inter-process Communication using Pipes

Pipes

- A pipe is a *one-way* communication channel that couples one process to another.
- Used only between processes that have a *common ancestor*.
  - More specifically, for communication between parent and child in general.
- There are two types of pipes: *named* and *unnamed*.
  - Unnamed pipes are used for communication between *related* processes.
  - Named pipes can be used for communication between *unrelated* processes.

Unnamed Pipes (1)

- int pipe(int *fd[2]*);
  - To create an *unnamed pipe*.
  - Needs an array of two *int*'s for two *file descriptors* (r/w).
- A pipe is another generalization of the UNIX file concept.
  - A pipe is a FIFO file: lseek( ) does not work on a pipe.
  - Unnamed pipes come and go.
  - Named pipes (also known as FIFOs) are permanent files.
- The size of a pipe is limited.
  - Max is PIPE_BUF

Unnamed Pipes (2)

- If a read( ) is issued while the pipe is empty, it will block.
- If a write( ) is issued when the pipe is full, it will block.
- If all writers of a pipe are closed, a reader will encounter EOF.
- If all readers of a pipe are closed, a writer will face a broken pipe.
- Non-blocking reads and writes:
  - Issue fcntl( ) with O_NONBLOCK flag.

Example #1: Unnamed Pipe (1)

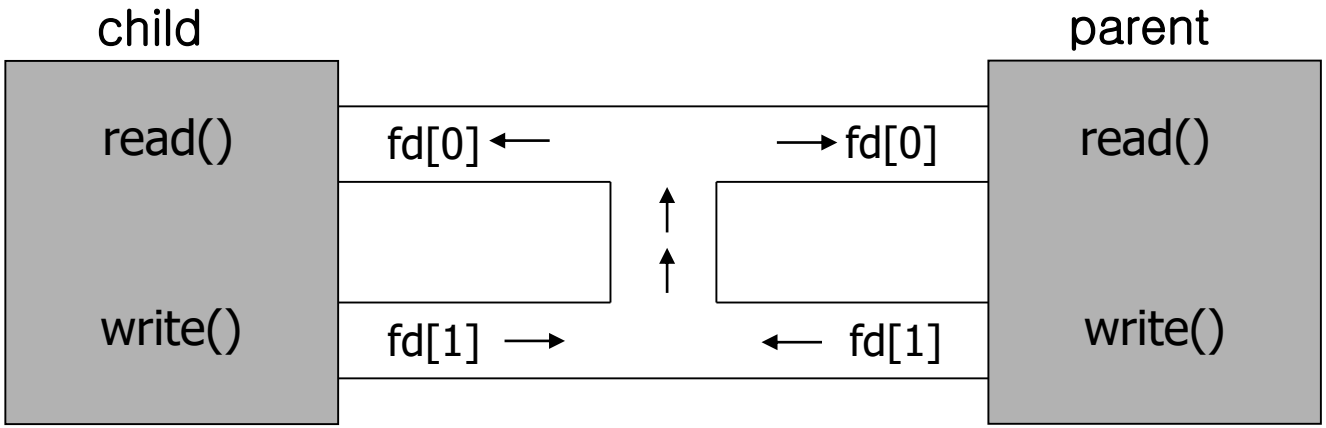
```
#include <stdio.h>

#define MSGSIZE 16

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

int main(void)
{
    char buf[MSGSIZE];
    int fd[2], i, pid;

    /* open unnamed pipe */
    if (pipe(fd) < 0) {
        perror("pipe call");
        exit(1);
    }
```

Example #1: Unnamed Pipe (2)

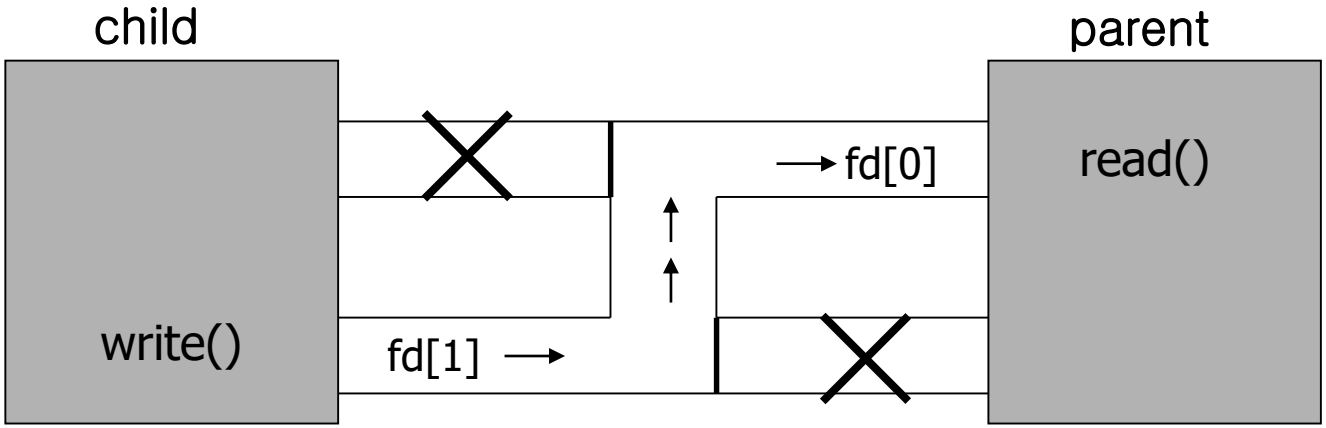
```
if ((pid = fork()) < 0) {
    perror("fork call");
    exit(2);
}

/* if child, then close read file
 * descriptor and write down pipe
 */
if (pid == 0) {
    close(fd[0]);
    write(fd[1], msg1, MSGSIZE);
    write(fd[1], msg2, MSGSIZE);
    write(fd[1], msg3, MSGSIZE);
}
```

```
/* if parent, then close write file
 * descriptor and read from pipe
 */
else {
    close(fd[1]);
    for (i = 0; i < 3; i++) {
        read(fd[0], buf, MSGSIZE);
        printf("%s\n", buf);
    }
   wait(NULL);
}
exit(0);
}
```

Example #1: Unnamed Pipe (3)



Fig(a). Unnamed pipe: before closing unnecessary file descriptor



Fig(b). unnamed pipe: after closing unnecessary file descriptor

Example #2 : Unnamed Pipe

```
#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;

        /* 시그널 핸들러를 구축한다. */
        act.sa_handler = alrm_action;
        sigfillset (&(act.sa_mask));

        if (pipe(p) == 1)
        {
                perror ("pipe call");
                exit (1);
        }

        /* 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);
}
```

Example #3: Unnamed Pipe (1)

```
#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)
                fatal ("pipe call");

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


        switch(fork()){
        case -1:            /* 오류 */
                fatal("fork call");
        case 0:             /* 자식 */
                child(pfd);
        default:            /* 부모 */
                parent (pfd);
         }
}
int parent (int p[2])       /* 부모의 코드 */
{
   int nread;
   char buf[MSGSIZE];
```

Example #3: Unnamed Pipe (2)

```
  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 fatal ("read call");
     case 0:
           /* 파이프가 닫혔음. */
          printf ("End of conversation□n");
          exit (0);
     default:
          printf ("MSG=%s□n", buf);
     }
  }
}
```

```
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);
}
```

select (1)

- #include < sys/types.h>

  #include <sys/time.h>

  #include <unistd.h>

  int select(int *nfds*, fd_set **readfds*, fd_set **writefds*, fd_set *error*fds*,struct timeval_**timeout*);
  - Handles multiple pipes simultaneously
  - Allows device polling.
  - *nfds* gives the number of the descriptors being selected.
  - *readfds*, *writefds* and *errorfds* point to bit masks, each bit representing a file descriptor.
    - If a bit is turned on, it denotes interest in the relevant file descriptor

select (2)

- Bit manipulation macros

```
#include <sys/time.h>
/* initialize the mask pointed to by fdset*/
void FD_ZERO(fd_set *fdset);

/* set the bit, fd in the mask pointed to by fdset */
void FD_SET(int fd, fd_set *fdset);

/* is the bit, fd, set in the mask pointed to by fdset */
int FD_ISSET(int fd, fd_set *fdset);

/* turn of the bit, fd, in the mask pointed to by fdset */
void FD_CLR(int fd, fd_set *fdset);
```

select (3)

- *timeout* indicates how long select( ) should sleep, waiting for data to arrive.
  - If data arrives for any file descriptors and the *timeout* value has not expired, select( ) return, indicating in the bit masks which file descriptors were selected.
- For instance, if a user wished to sleep until receiving input on file descriptors 0, 1 or 2, *readfds* would point to the bit mask 7; when select( ) returns, the bit mask would be *overwritten* with a mask indicating which file descriptors had data ready.
- The bit mask *writefds* dose a similar function for write file descriptors, and the bit mask *errorfds* indicates when exceptional conditions exist for particular file descriptors, useful in networking.

select (4)

- *timeout*
  - struct timeval {

    long tv_sec; /*seconds*/

    long tv_usec; /*and microseconds*/

    };
  - Three condition
    - *timeout* == NULL
      - » Wait without a time limit.
    - *timeout* ->tv_sec == 0 && *timeout*->tvusec == 0
      - » No wait
    - *timeout* ->tv_sec != 0 || *timeout*->tvusec != 0
      - » Wait until the time has expired.

select (5)

- Return value
  - -1: Error
  - 0: No descriptor is ready.
  - Positive value: The total count of the number of descriptors that are ready.

Example #4: select (1)

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

#define MSGSIZE   6

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

void parent(int [] []);
int child(int []);

main()
{
        int pip[3] [2];
        int i;
```

```
/* 세 개의 통신 파이프를 생성하고, 세 개의 자식을 낳
    는다. */
        for (i = 0; i < 3; i++)
        {
                if (pipe(pip[i]) == -1)
                        fatal ("pipe call");

                switch (fork()){
                case -1:                        /* 오류 */
                        fatal ("fork call");
                 case 0:                        /* 자식 */
                        child (pip[i]);
                }
        }
        parent (pip);

        exit (0);
}
```

Example #4: select (2)

```
/* 부모는 세 개의 파이프에 전부 귀를 기울이고 있다. */
void parent(int p[3] [2])            /* 부모의 코드 */
{
        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);
```

Example #4: select (3)

```
/* 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);
                  }
            }
      }
```

Example #4: select (4)

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

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);
}
```

Pipes and exec

- Pipe can be set up between two programs at shell level
  - $ ls | wc
  - Open file descriptors are kept open across exec calls
  - Two pipe file descriptors opened prior to a fork/exec will still be open when the child process begins execution of the new program
  - Shell couples the standard output of ls to the write end of the pipe, and the standard input of wc to the read end

Example #5: pipe and exec (1)

```
/* join -- 두 명령을 파이프로 결합한다. */

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

        /* 명령을 수행할 자식을 생성한다. */
        switch (fork()){
        case -1:        /* 오류 */
                fatal ("1st fork call in join");
        case 0:         /* 자식 */
                break;
        default:        /* 부모 */
                wait(&status);
                return (status);
        }

        /* 루틴의 나머지 부분으로 자식에 의해 수행된다. */
        /* 파이프를 만든다. */
        if (pipe(p) == -1)
                fatal ("pipe call in join");
```

Example #5: pipe and exec (2)

```
/* 다른 프로세스를 생성한다. */
        switch (fork()){
        case -1:
                /* 오류 */
                fatal ("2nd fork call in join");
        case 0:
                /* 쓰는 프로세스 */
                dup2 (p[1],1);          /* 표준 출력이 파이프로 가게 한다. */
                close (p[0]);          /* 화일 기술자를 절약한다. */
                close (p[1]);
                execvp (com1[0], com1);
                /* execvp가 복귀하면, 오류가 발생한 것임. */
                fatal("1st execvp call in join");
        default:
                /* 읽는 프로세스 */
                dup2(p[0], 0);          /* 표준 입력이 파이프로부터 오게 한다 */
                close (p[0]);
                close (p[1]);
                execvp (com2[0], com2);
                fatal ("2nd execvp call in join");
        }
}
```

Example #5: pipe and exec (3)

```
#include <stdio.h>

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);
}
```

FIFOs or Named Pipes

- Drawbacks of unnamed pipes
  - Only be used to connect processes that share a common ancestry, such as a parent and its child process
  - Not permanently
- FIFO or named pipes
  - $mkfifo channel (or $mknod channel p)
  - $ls –la channel
  - prw-r--r-- 1 dhlee adm 0 11월 3 14:51 channel|
  - $cat < channel /* this command would be blocked */
  - $cat < channel &
  - $ls –la >! channel; wait (or $ls –la >> channel; wait)

mkfifo (1)

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

- Create a FIFO file (*named pipe)* named by the first parameter pathname with mode permissions
- Can be used between unrelated processes for data exchange.
- Once created, a FIFO must be opened using open
  - mkfifo("/tmp/fifo", 0666)
  - Fd = open("/tmp/fifo", O_WRONLY);
- The open will block until another process opens the FIFO for reading
- Non-blocking open calls are possible with O_NONBLOCK flag
  - fd = open("/tmp/fifo", O_WRONLY|O_NONBLOCK);

mkfifo (2)

- Named pipe can be also created with mknod( ).
  - A value of octal 010000 must be added to the mode value to signify a FIFO.
  - if (mknod("fifo", 010600, 0) < 0)

    perror("mknod failed");

Example #6: FIFO (1)

```
#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 (argc < 2)
        {
                fprintf (stderr, "Usage: sendmessage msg ... □n");
                exit(1);
        }
```

Example #6: FIFO (2)

```
    /* O_NONBLOCK을 설정하여 fifo를 개방한다. */
    if ((fd = open(fifo, O_WRONLY | O_NONBLOCK)) < 0)
        fatal ("fifo open failed");

    /* 메시지를 보낸다. */
    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)
            fatal ("message write failed");
    }
    exit (0);
}
```

Example #6: FIFO (3)

```
/* rcvmessage -- fifo를 통해 메시지를 받는다. */

#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];

      /* fifo가 이미 존재하지 않으면, 생성한다 */
      if (mkfifo(fifo, 0666) == -1)
      {
            if (errno != EEXIST)
                  fatal ("receiver: mkfifo");
      }
```

Example #6: FIFO (3)

```
        /* fifo를 읽기와 쓰기용으로 개방한다. */
        if ((fd = open(fifo, O_RDWR)) < 0)
                fatal ("fifo open failed");


        /* 메시지를 받는다 */
        for(;;)
        {
                if (read(fd, msgbuf, MSGSIZ+1) <0)
                        fatal ("message read failed");

        /*
          * 메시지를 프린트한다 ; 실제로는 보다 흥미 있는 일이 수행된다.
          */

                printf ("message received:%s□n", msgbuf);
        }
}
```

Sample Run : FIFO

```
dhlee@kde:~/Course/SP/example>rcvmsg &
[1] 32615
dhlee@kde:~/Course/SP/example>sndmsg "msg1" "msg2"
dhlee@kde:~/Course/SP/example>message received:msg1
message received:msg2

dhlee@kde:~/Course/SP/example>sndmsg "msg3"
message received:msg3
dhlee@kde:~/Course/SP/example>
```

More Examples #1: Pipe (1)

```
/* pipe.c
   This program illustrates how to use the pipe() system call.

   This example shows a way of implementing who | sort using pipe().
   The output from who is redirected to a system pipe and sort gets its
   input from the system pipe.
   Algorithm outline:
     a. to create a pipe
     b. to fork a child
     c. to duplicate a file descriptor
     d. to close unused ends of the pipe
     e. to exec a process to execute the command
*/

#include <stdio.h>

int main(void)
{
    int fd[2];

    pipe(fd);     /* a pipe is created: fd[0] for read; fd[1] for write */
```

More Examples #1: Pipe (1)

```
    if (fork() == 0) {          /* 1st child - output redirection */
        dup2(fd[1], 1);          /* connect stdout to pipe */
        close(fd[0]);           /* close unneeded file descriptors */
        close(fd[1]);
        printf("\nThis is the write end of the pipe:\n");
        fflush(stdout);          /* write out the msg to terminal */
        execl("/bin/who", "who", (char *)0);
        printf("\nThis line shouldn't be here");
    }
    if (fork() == 0) {          /* 2nd child - input redirection */
        dup2(fd[0], 0);          /* connect stdin to pipe */
        close(fd[0]);           /* close unneeded file descriptors */
        close(fd[1]);
        printf("\nThis is the read end of the pipe:\n");
        execl("/bin/sort", "sort", (char *)0);
        printf("\nThis line shouldn't be here");
    }
    close(fd[0]);               /* parent is taking it easy */
    close(fd[1]);
    printf("\nParent is terminating\n");
    return(0);
}
```

More Examples #2: Pipe (1)

```
/* np.c
   This program shows an example of creating named pipes.

   A named pipe is known as FIFO. It is implemented as a disk file.
   Therefore, it can outlive the program that creates it.
   Also, most of the file manipulation commands apply to FIFOs.
   However, the length of a FIFO file is alway shown as 0, and all reads
   from FIFO are destructive.
   Any of the following two sytem calls can be used to create FIFO:
      makefifo() or mknod()
*/

#include <stdio.h>
#include <fcntl.h>

#define  NPNAM  "fifo"
#define  BSIZ    BUFSIZ
#define  ERR    -1
#define  EMPTY   0

char *mesg[] = {"yesterday", "today", "tomorrow"};
```

More Examples #2: Pipe (2)

```
int main(void)
{
    int npid;
    int npcreate(char *, int), nperror(int, char *),
        npread(int, char *), npwrite(int, int, char *[]);

    if ((npcreate(NPNAM, 0664)) == ERR)  /* to create a fifo */
        nperror(1, "Can't create named pipe\n");

    if (fork() == 0) {  /* writer */
        if ((npid = open(NPNAM, O_WRONLY)) == ERR)
            nperror(2, "Can't open named pipe\n");

        if (npwrite(npid, sizeof(mesg)/sizeof(char *), mesg) == ERR)
            nperror(3, "Can't write named pipe\n");
        exit(0);
    }

    if (fork() == 0) {  /* reader */
        char rbuf[BSIZ];

        if ((npid = open(NPNAM, O_RDONLY)) == ERR)
            nperror(4, "Can't open named pipe\n");
```

More Examples #2: Pipe (3)

```
        while(1) {
            switch(npread(npid, rbuf)) {
                case ERR:
                    nperror(5, "Can't read named pipe\n");
                case EMPTY:
                    break;
                default:
                    printf("Got: %s\n", rbuf);
                    continue;
            }
            break;
        }
        exit(0);
    }
    wait((int *)0);
    return(0);
}  /* main */
```

More Examples #2: Pipe (4)

```
int npcreate(char *npnam, int perms)
{
    umask(0);         /* permissions to be determined by perms  */
    return(mkfifo(npnam, perms));      /* either one of these two works */
    /* return(mknod(npnam, 010000 | perms));  010000: fifo special */
}  /* npcreat */

int nperror(int errno, char *errmsg)
{
    write(2, errmsg, strlen(errmsg));
    exit(errno);
}  /* nperror  */

int npread(int npid, char *rbuf)
{
    return(read(npid, rbuf, BSIZ));
}  /* npread  */
```

More Examples #2: Pipe (5)

```
Int npwrite(int npid, int siz, char *mesg[])
{
    int i;
    for (i = 0; i < siz; i++) {
        if (write(npid, mesg[i], strlen(mesg[i])+1) == ERR)
            return(ERR);
        sleep(1);
    }                          /* write out one mesg at a time */
    return(0);
}  /* npwrite  */
```