IPC (Inter-Process Communication)

컴퓨터과학부

유닉스 프로그래밍

IPC

- **Inter-process Communication**
  - Process간의 신호를 주고 받거나 information의 공유
  - Kernel 의해 지원

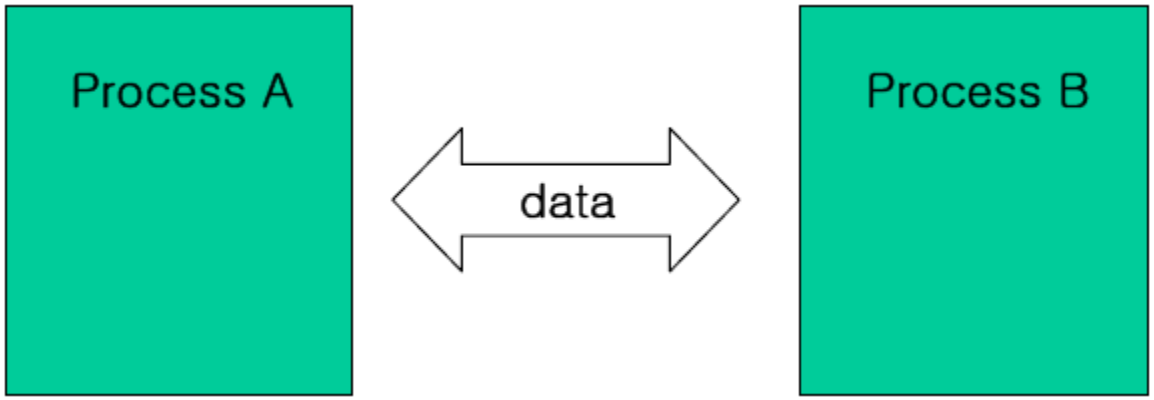
Signal

Pipe & Named Pipe

Message Queue

Shared Memory

Semaphore



Signal

Signal ?

- **Signal 용도**
  - 프로세스간에 신호를 전달
  - 프로세스간 동기화 및 정보 전달 시 이용

- **Signal 생성**
  - 한 프로세스가 다른 프로세스에게 보냄 (kill() 시스템 호출)
  - 커널이 프로세스에게 보냄
    - 사건 발생 알림: Alarm, 프로세스 종료 등
    - 에러 상황: 잘못된 메모리 접근, 오버플로우 등
  - 사용자가 인위적으로 생성
    - 키보드 입력 (Ctrl-C, Ctrl-\ 등)

- **Signal 처리**
  - 무시, 보류, 종료, 등록된 함수 호출
  - 처리시점: 프로세스가 커널 모드에서 유저 모드로 돌아갈 때 시그널 처리

Signal Definition

❑ signal 종류

```
#define SIGHUP  1       /* hangup */
#define SIGINT  2       /* interrupt (rubout) */
#define SIGQUIT 3       /* quit (ASCII FS) */
#define SIGILL  4       /* illegal instruction (not reset when caught) */
#define SIGTRAP 5       /* trace trap (not reset when caught) */
#define SIGIOT  6       /* IOT instruction */
#define SIGABRT 6       /* used by abort, replace SIGIOT in the future */
#define SIGEMT  7       /* EMT instruction */
#define SIGFPE  8       /* floating point exception */
#define SIGKILL 9       /* kill (cannot be caught or ignored) */
#define SIGBUS  10      /* bus error */
#define SIGSEGV 11      /* segmentation violation */
#define SIGSYS  12      /* bad argument to system call */
#define SIGPIPE 13      /* write on a pipe with no one to read it */
#define SIGALRM 14      /* alarm clock */
#define SIGTERM 15      /* software termination signal from kill */
#define SIGUSR1 16      /* user defined signal 1 */
#define SIGUSR2 17      /* user defined signal 2 */
```

Signal Definition

```
#define SIGCLD  18          /* child status change */
#define SIGPWR  19          /* power-fail restart */
#define SIGWINCH 20         /* window size change */
#define SIGURG  21          /* urgent socket condition */
#define SIGPOLL 22          /* pollable event occured */
#define SIGIO   SIGPOLL     /* socket I/O possible (SIGPOLL alias) */
#define SIGSTOP 23          /* stop (cannot be caught or ignored) */
#define SIGTSTP 24          /* user stop requested from tty */
#define SIGCONT 25          /* stopped process has been continued */
#define SIGTTIN 26          /* background tty read attempted */
#define SIGTTOU 27          /* background tty write attempted */
#define SIGVTALRM 28        /* virtual timer expired */
#define SIGPROF 29          /* profiling timer expired */
#define SIGXCPU 30          /* exceeded cpu limit */
#define SIGXFSZ 31          /* exceeded file size limit */
#define SIGWAITING 32       /* process's lwps are blocked */
#define SIGLWP  33          /* special signal used by thread library */
#define SIGFREEZE 34        /* special signal used by CPR */
#define SIGTHAW 35          /* special signal used by CPR */
#define SIGCANCEL 36        /* thread cancellation signal used by libthread */
#define SIGLOST 37          /* resource lost (eg, record-lock lost) */
```

중요 Signal

- **대표적인 signal**
  - SIGABRT: abort() 함수의 호출에 의해 발생. 프로세스는 비정상적으로 종료
  - SIGALRM: alarm 함수에 의해 설정된 타이머에 의해 발생
  - SIGCHLD: 프로세스가 종료하거나 정지한 경우, 부모프로세스에게 전달
  - SIGFPE: 산술 연산 에러에 의해 발생 (0으로 나눈 경우 등)
  - SIGHUP: 터미널 연결이 단절되는 경우 제어 프로세스에게 전달
  - SIGINT: 인터럽트 키 (Ctrl+C)를 누를 경우 터미널에서 동작하는 프로세스에게 전달
  - SIGKILL: 프로세스를 종료하라는 시그널.
    무시하거나 임의의 처리를 할 수 없는 시그널

중요 Signal

- **대표적인 signal**
  - SIGQUIT: quit 키 (ctrl+\)를 누르면 터미널 드라이버에서 발생
    종료된 프로세스는 core 파일을 생성
  - SIGSEGV: 잘못된 메모리 참조에 의해 발생
  - SIGSTOP: 프로세스를 정지시키는 작업 제어 시그널
    무시하거나 임의의 처리를 할 수 없는 시그널
  - SIGTERM: 작업 종료 시그널
  - SIGUSR1: 응용 프로그램에서 사용자가 정의하여 사용할 수 있는 시그널
  - SIGUSR2: 응용 프로그램에서 사용자가 정의하여 사용할 수 있는 시그널

Signal의 처리

- **시그널 무시 (ignore)**
  - SIGKILL과 SIGSTOP 시그널을 제외한 모든 시그널을 무시할 수 있음
  - 하드웨어 오류에 의해 발생한 시그널에 대해서는 주의해야 함
- **시그널 처리 (catch)**
  - 시그널이 발생하면 미리 등록된 함수(handler)가 수행됨
  - SIGKILL과 SIGSTOP 시그널에는 처리할 함수를 등록할 수 없음
- **기본 처리 (default)**
  - 특별한 처리 방법을 선택하지 않은 경우
  - 대부분 시그널의 기본 처리 방법은 무시나 프로세스 종료

Signal function

NAME

signal – signal handling

SYNOPSIS

```
#include <signal.h>
typedef void (*sighandler_t) (int);
sighandler_t signal(int signo, sighandler_t handler);
```

RETURN VALUE

Return: -1 with errno set to EINTR

Signal function

- **Description**
  - 인자로 받은 시그널에 대한 액션을 정의
  - Signal 무시
    - signal(SIGINT, SIG_IGN);
  - Signal 복구 (시그널 처리 시 기본 동작으로 되돌림)
    - signal(SIGINT, SIG_DFL)
  - Signal 처리 (catch)
    - 두 번째 인자로 미리 정해 놓은 함수(시그널 핸들러)의 포인터를 설정하면 시그널 발생 시, 해당 함수가 실행
    - 시그널 핸들러는 void (*sighandler_t) (int) 다음과 같이 정의됨
      - 리턴: void, 인자: int형 변수 한 개
  - 여러 개의 시그널을 무시할 수도 있음
    - signal(SIGINT, SIG_IGN);
    - signal(SIGQUIT, SIG_IGN);

kill function

NAME

kill - terminate a process

SYNOPSIS

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

int kill(pid_t pid, int sig);
```

RETURN VALUE

Return : 0 if OK, -1 on error

kill fucntion

- **sig 인자**
  - 시그널 번호
  - null signal (0)
    - 실제로 시그널을 보내지 않고 프로세스의 존재여부 파악
    - 프로세스 미 존재시 -1 리턴
- **pid 인자**
  - pid > 0 : 프로세스 ID가 pid인 프로세스에게 시그널 전달
  - pid == 0 : 호출한 프로세스와 같은 그룹 ID를 가지고 있는 모든 프로세스에게 전달
  - pid < 0 : pid의 절대값에 해당하는 프로세스 그룹 ID를 가지고 있는 모든 프로세스에게 시그널을 전달

raise function

NAME

raise - send a signal to the current process

SYNOPSIS

```
#include <signal.h>
int raise(int signo);
```

RETURN VALUE

Return : 0 if OK, -1 on error

signal, kill function

signal.c

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
void catchint(int  signo)
{
    printf("SIGINT Received\n");
}

void main()
{
   signal(SIGINT, (void *) catchint);

   printf("sleep call #1\n");     sleep(1);
   printf("sleep call #1\n");     sleep(1);
   printf("sleep call #1\n");     sleep(1);
   printf("sleep call #1\n");     sleep(1);
   printf("Exiting");
}
```

signal, kill function

signal2.c

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

void handler(int  sig)
{
   printf("signal no(%d) Received\n", sig);
}

void main()
{
  if (signal(SIGUSR1, handler) == SIG_ERR) {
                fprintf(stderr, "cannot set USR1\n");
                exit(1);
  }
  if (signal(SIGUSR2, handler) == SIG_ERR) {
                fprintf(stderr, "cannot set USR2\n");
                exit(1);
  }
  for (;;) pause();
}
```


```
% kill   -USR1   PID

% kill   -USR2   PID
```

signal, kill function

sig_parent.c

```
#include <signal.h>   #include <stdio.h>  #include <unistd.h>
#define NUMCHILD 3

void main(int argc, char *argv[])
{
   int   pid, chpid[NUMCHILD];
   int   i, status;

   for (i = 0; i < NUMCHILD; i++) {
      if ((pid = fork()) == 0)
           execlp("./sig_child", "./sig_child", (char *) 0);
      chpid[i] = pid;
    }
    printf("sig_parent: %d child process run\n", NUMCHILD);
    sleep(10);

     for (i = 0; i < NUMCHILD; i++)
         kill(chpid[i], SIGINT);
}
```

sig_child.c

```
#include <signal.h> #include <unistd.h>
#include <stdio.h>
void sig(int sig) {
   printf("child die (%d)\n",  getpid());
}

void main() {
   signal(SIGINT, sig);
   pause();
}
```

인위적 발생

alarm function

NAME

alarm - set an alarm clock for delivery of a signal

SYNOPSIS

```
#include <unistd.h>

unsigned int alarm(unsigned int second);
```

RETURN VALUE

Return : 0 or number of seconds until previously set alarm

alarm function

- **Description**
  - 지정된 시간 후에 SIGALRM 시그널이 발생하도록 타이머 설정
  - SIGALRM의 기본 처리 방법은 프로세스의 종료
  - 일반적으로 시그널 처리 함수를 등록하여 사용
  - 한 프로세스에는 하나의 알람만 존재
  - 이미 알람이 설정된 상태에서 다시 alarm() 함수를 호출하면 이전 알람의 남은 시간이 리턴되고 새로운 알람으로 설정됨
  - second 인자가 0인 경우, 이미 설정된 알람이 존재하면 남은 시간이 리턴되고 알람은 해제됨

pause function

NAME

pause - wait for signal

SYNOPSIS

```
#include <unistd.h>
void pause(void);
```

RETURN VALUE

Return : -1 with errno set to EINTR

alarm, pause function

alarm.c

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

void alm(signo)
int signo;
{
   printf("SIGALRM Received\n");
}

void main()
{
  signal(SIGALRM, alm);
  alarm(10);
  printf("process pause\n");
  pause();
  printf("process wakeup");
}
```

Signal set

NAME

signal set handling

SYNOPSIS

```
#include <signal.h>

int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
int sigismember(const sigset_t *set, int signo);
```

RETURN VALUE

Return : 0 if OK, -1 on error (sigismember : 1 if true else 0)

Signal Set

- **Description**
  - sigemptyset
    - set의 모든 시그널을 0으로 set (모든 시그널 제외)
  - sigfillset
    - set의 모든 시그널을 1로 set (모든 시그널 포함)
  - sigaddset
    - set의 멤버로서signo로 지정된 시그널 추가
  - sigdelset
    - set에서 signo로 지정된 시그널 제거
  - sigismember
    - signo 시그널이 set의 멤버인지 검사

sigprocmask function

NAME

signal handling functions.

SYNOPSIS

```
#include <signal.h>

int sigprocmask(int how, const sigset_t *set, sigset_t *oset);
```

RETURN VALUE

Return : 0 if OK, -1 on error

sigprocmask function

- **Description**
  - signalmask를 변경하거나 검사한다.
    - signalmask: 프로세스에게 전달되지 않도록 블록된 시그널의 집합
  - how: 시그널 set을 변경시키는 방법
    - SIG_BLOCK: set인자로 지정된 시그널들을 시그널 마스크에 추가
    - SIG_UNBLOCK: set 인자로 지정된 시그널들을 시그널 마스크에서 제외(빼기)
    - SIG_SETMASK: set 인자로 시그널 마스크를 대체
  - set: 변경될 시그널 마스크
  - oset: sigprocmask 함수 호출 이전의 시그널 마스크 내용
  - oset이 NULL이 아니면 이전의 블록된 시그널 set 값이 저장
  - set이 NULL이면 how는 의미가 없으면 기존 마스크 값을 얻기 위해 사용

sigprocmask function

sigprocmask.c

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

void show_mask() {
  sigset_t  sigset;

  if (sigprocmask(0, NULL, &sigset) < 0)
    printf("sigprocmask error\n");

  if (sigismember(&sigset, SIGINT)) printf("SIGINT");
  if (sigismember(&sigset, SIGQUIT)) printf("SIGQUIT");
  if (sigismember(&sigset, SIGUSR1)) printf("SIGUSR1");
  if (sigismember(&sigset, SIGALRM)) printf("SIGALRM");

  printf("\n");
}
```

sigprocmask function

sigprocmask.c

```
int main(void) {
  sigset_t   newmask, oldmask;

  sigemptyset(&newmask);
  sigaddset(&newmask, SIGQUIT);

  /* add SIGQUIT signal to blocked signal list */
  if (sigprocmask(SIG_BLOCK, &newmask, &oldmask) < 0)
    printf("sigprocmask error\n");

  show_mask();

  if (sigprocmask(SIG_SETMASK, &oldmask, NULL) < 0)
    printf("sigprocmask error\n");
}
```

sigpending function

NAME

signal handling functions.

SYNOPSIS

```
#include <signal.h>

int sigpending(sigset_t *set);
```

RETURN VALUE

Return : 0 if OK, -1 on error

sigpending function

- **Description**
  - 호출된 프로세스에 대해 발생한 후 블록되어 있는 시그널 집합을 리턴
  - 블록된 시그널 집합은 시그널 세트로 표현
  - 중복 발생된 시그널은 누적되지 않음

sigpending function

sigpending.c

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

void sig_quit(int);

int main(void)
{
  sigset_t newmask, oldmask, pendmask;

  if (signal(SIGQUIT, sig_quit) == SIG_ERR)
    printf("can't catch SIGQUIT\n");

  sigemptyset(&newmask);
  sigaddset(&newmask, SIGQUIT);

  /*block SIGQUIT and save current signal mask */
  if (sigprocmask( SIG_BLOCK, &newmask, &oldmask) < 0)
    printf("SIG_BLOCK error\n");

  sleep(5);    /* SIGQUIT here will remain pending */

  if (sigpending(&pendmask) < 0)
    printf("sigpending error\n");
  if (sigismember(&pendmask, SIGQUIT))
    printf("sigQUIT pending\n");

  /* reset signal mask which unblocks SIGQUIT */
  if (sigprocmask(SIG_SETMASK, &oldmask, NULL) < 0)
    printf("SIG_SETMASK error\n");
  printf("SIGQUIT unblocked\n");
  sleep(5);
}

void sig_quit(int signo)
{
  printf("caught SIGQUIT\n");
  if (signal(SIGQUIT, SIG_DFL) == SIG_ERR)
    printf("can't reset SIGQUIT\n");
}
```

SIGSUSPEND function

NAME

sigsuspend – wait signal after sigmask set

SYNOPSIS

```
#include <signal.h>

int sigsuspend(const sigset_t *set);
```

RETURN VALUE

Return: -1 with errno set to EINTR

-1 with errno set to EFAULT if *set* is invalid pointer

Race Condition 없는 SIGALRM

norace_alm.c

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

void alm(int signo)
{
   printf("SIGALRM Received\n");
}
```

```
void main()
{
  sigset_t  newmask, oldmask;

  signal(SIGALRM, alm);
  sigemptyset(&newmask);
  sigaddset(&newmask, SIGALRM);
  sigprocmask(SIG_BLOCK, &newmask, &oldmask);

  alarm(10);
  printf("process pause\n");
  sigsuspend(&oldmask);
  printf("process wakeup");
}
```

Pipe

Pipe ?

- **Pipe ?**
  - 한 프로세스의 표준 출력을 다른 프로세스의 표준 입력에 연결
    - 주로 부모/자식 혹은 동일한 부모의 자식 프로세스 사이의 통신을 위함
  - 프로세스간 단 방향 통신의 한 방법
  - 동기화를 기본적으로 제공
  - 가득 차거니 비어 있을 때 자동으로 block

Pipe 생성

NAME

pipe - create pipe

SYNOPSIS

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

RETURN VALUE

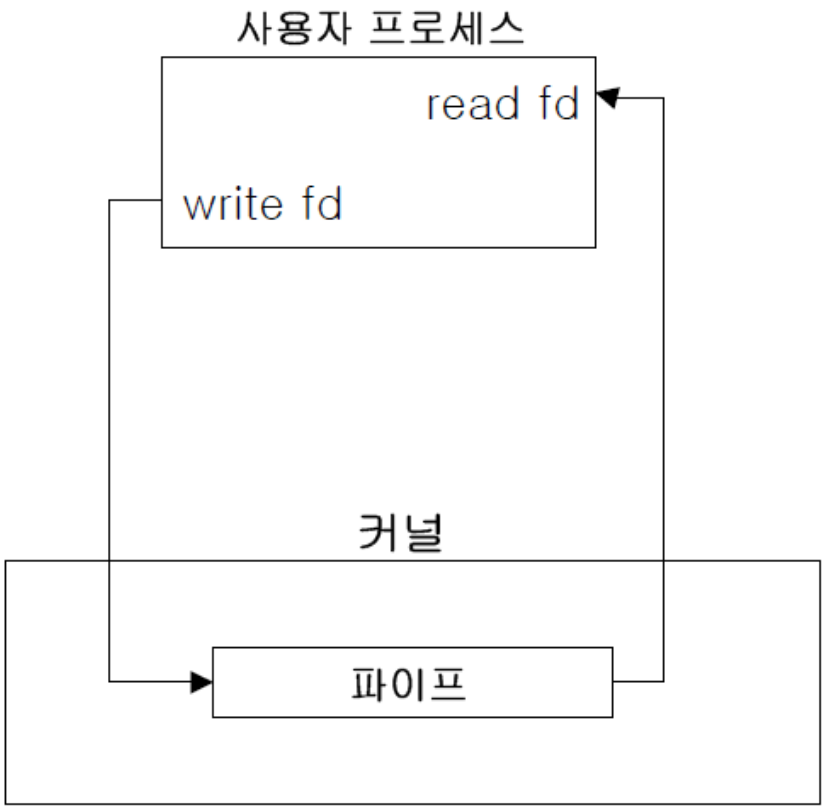
Return : 0 if OK, -1 on error

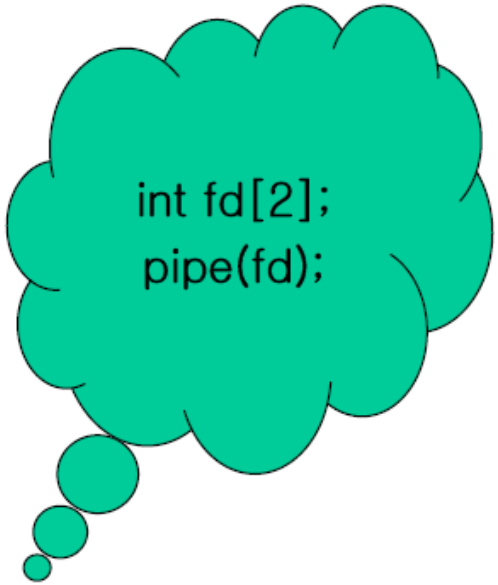
Pipe 생성

- **Pipe 생성**
  - pipe() 시스템 호출을 이용하여 만들어짐
  - 파일처럼 동작
  - data를 FIFO 방식으로 처리
  - filedes[0]: 읽기 위해 사용
  - filedes[1]: 쓰기 위해 사용
  - 파이프의 사용을 마쳤을 때: close()
  - 파이프에서 데이터를 읽을 때: read()
  - 파이프에 데이터를 쓸 때: write()
  - 파이프와 표준 입출력을 연결할 때: dup()
  - block을 해제할 때: fcntl()

단일 프로세스의 pipe

❑ 프로세스와 파이프와의 관계





단일 프로세스의 Pipe

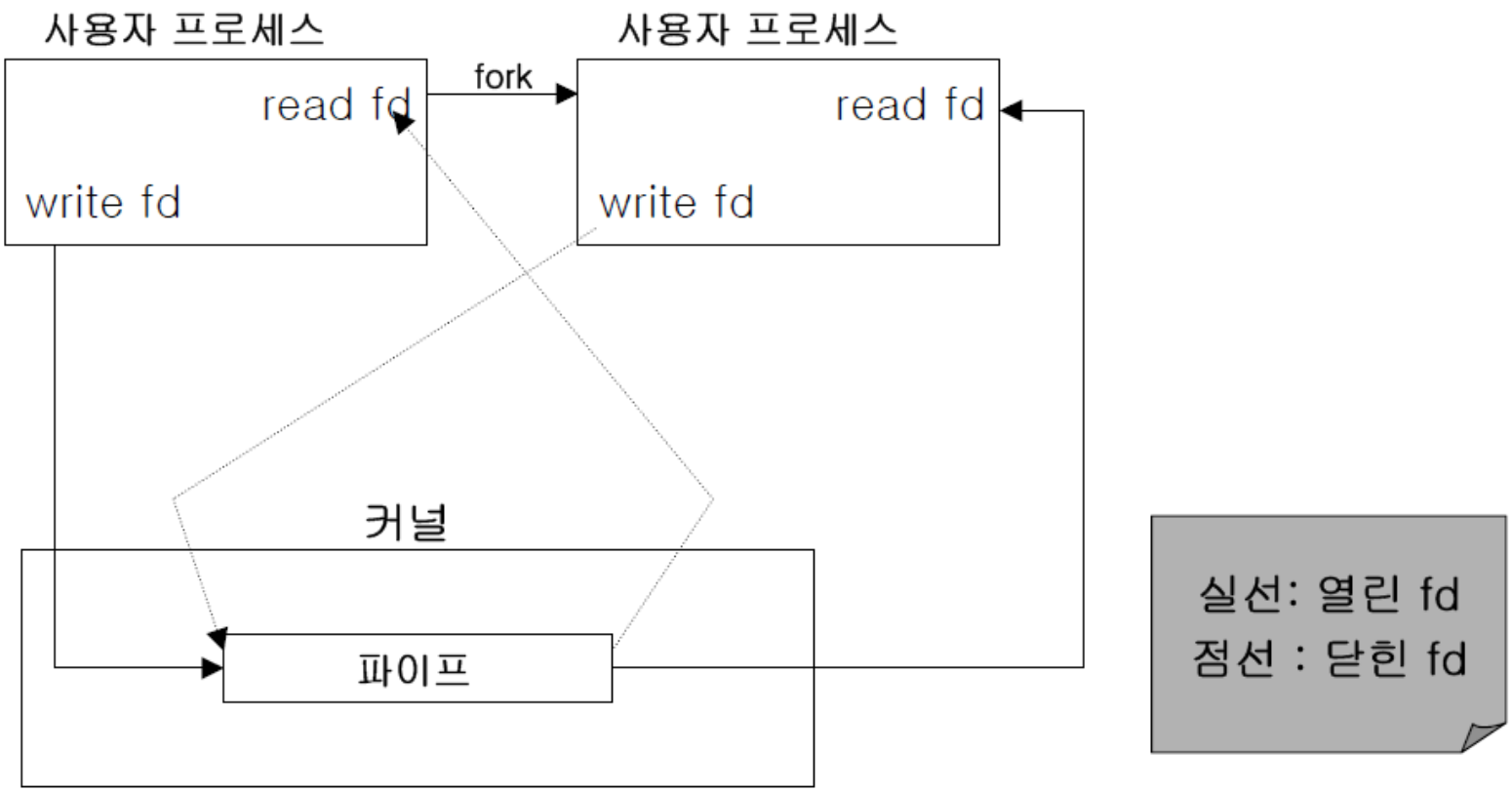
pipe1.c

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

#define MSGSIZE 17

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

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

  if (pipe(p) < 0) {
    perror("pipe call");
    exit(1);
  }
```

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

  for (j = 0; j < 3; j++) {
    read(p[0], inbuf, MSGSIZE);
    printf("%s", inbuf);
  }
}
```

프로세스간 통신

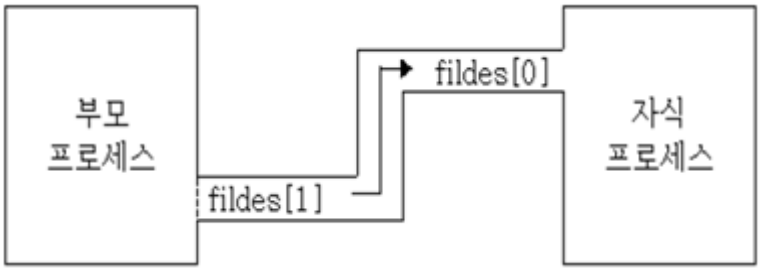
❑ fork후에 프로세스와 파이프와의 관계



프로세스간 통신

pipe2.c

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

#define MSGSIZE 20

int main()
{
  int fd[2], pid;
  char msgout[MSGSIZE] = "Hello world\n";
  char msgin[MSGSIZE];

  if (pipe(fd) == -1) {
    perror("pipe()");    exit(1);
  }
  if ((pid=fork()) > 0) {
    close(fd[0]);
    write(fd[1], msgout, MSGSIZE);
  }
  else if (pid == 0) {
    close(fd[1]);
    read(fd[0], msgin, MSGSIZE);
    puts(msgin);
  }
  else {
    perror("fork()");    exit(2);
  }
}
```



프로세스간 통신

pipe3.c

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

int main(int argc, char *argv[])  {
  int   pid, fd[2];

  if (pipe(fd) == -1) {
    perror("pipe()");    exit(1);
  }
  if ((pid = fork()) == 0) { /* child process */
    close(fd[1]);
    dup2(fd[0], 0);
    execlp(argv[2], argv[2], (char *) 0);
    exit(1);
  }
        /* parent process */
  close(fd[0]);
  dup2(fd[1], 1);
  execlp(argv[1], argv[1], (char *) 0);
}
```

```
# w
# w | sort
# ./pipe3  w  sort
```

프로세스간 통신 (양방향)

bipipe.c

```
int main()
{
  int childpid, pipe1[2], pipe2[2];

  if (pipe(pipe1) < 0 || pipe(pipe2) < 0)
    printf("pipe error\n");
  if ((childpid = fork()) < 0)
    printf("fork error\n");
  else if (childpid > 0) {    /* parent process */
    close(pipe1[0]);
    close(pipe2[1]);
    client(pipe2[0], pipe1[1]);
    while (wait((int *) 0) != childpid);
    close(pipe1[1]);
    close(pipe2[0]);
    exit(0);
  }
```

```
else {        /* child process */
    close(pipe1[1]);
    close(pipe2[0]);
    server(pipe1[0], pipe2[1]);
    close(pipe1[0]);
    close(pipe2[1]);
    exit(0);
  }
}
```

프로세스간 통신 (양방향)

bipipe.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>

#define MAXBUFF 1024

void client(int readfd, int writefd)
{
  char buff[MAXBUFF];
  int  n;

  printf("Client: Enter file name: ");
  if (fgets(buff, MAXBUFF, stdin) == NULL)
    printf("Client: filename read error\n");
  n = strlen(buff);
  if (buff[n-1] == '\n')
    n--;
  if (write(writefd, buff, n) != n)
     printf("Client: filename write error\n");
  while ((n=read(readfd, buff, MAXBUFF)) > 0)
      if (write(1, buff, n) != n)
        printf("Client: data write error\n");
  if (n < 0)
    printf("client: data read error\n");
}
```

프로세스간 통신 (양방향)

bipipe.c

```
void server(int readfd, int writefd)
{
  char buff[MAXBUFF];
  int  n, fd;

  if ((n=read(readfd, buff, MAXBUFF)) <= 0)
    printf("Server: filename read error\n");
  buff[n] = '\0';

  if ((fd=open(buff, 0)) < 0) {
    strcat(buff, " can't open\n");
    n = strlen(buff);
    if (write(writefd, buff, n) != n)
      printf("Server: errmsg write error\n");
  }
```

```
else {
    while ((n=read(fd, buff, MAXBUFF)) > 0)
      if (write(writefd, buff, n) != n)
        printf("Server: data write error\n");
    if(n < 0)
      printf("Server: read error\n");
  }
}
```

popen function

NAME

popen, pclose - process I/O

SYNOPSIS

```
#include <stdio.h>
FILE *popen(const char *cmdstring,const char *mode);
int pclose(FILE *fd);
```

RETURN VALUE

Return : popen - file pointer if OK, NULL on error
pclose - termination status of cmdstring

popen function

Result of fp = popen(command,"r");

• command의 표준 출력을 반환된 화일 포인터로 읽음

Result of fp = popen(command,"w");

•반환된 화일 포인터로의 출력을 command의 표준 입력으로

명령어("command")를 사용하여 파이프를 만들고 fork/exec를 수행한다.

popen function

popen.c

```
#include <stdio.h>

int main(void)
{
   FILE  *pipein_fp, *pipeout_fp;
   char  readbuf[80];

   /* create one way pipe with popen() */
   pipein_fp = popen("ls", "r");

   /* create one way pipe with popen() */
   pipeout_fp = popen("sort", "w");

/* read from pipein_fp and write to pipeout_fp */
   while (fgets(readbuf, 80, pipein_fp))
      fputs(readbuf, pipeout_fp);

   /* close pipes */
   pclose(pipein_fp);
   pclose(pipeout_fp);

   return 0;
}
```

pipe와 Block

- **Pipe와 Block**
  - 파이프에 자료가 없을 때 read: block 됨
  - 파이프에 빈 공간이 없을 때 write: block 됨

- **Block되지 않는 pipe 입출력**
  - alarm()을 이용한 대기시간 조절
  - fcnt(filedes, F_SETFL, O_NDELAY)

Non-blocking I/O

nb_pipe.c

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

int fd[2];

void alarm_handler(int sig) {
        fcntl(fd[0], F_SETFL, O_NDELAY);
}

int main(void)
{
        char buf[100];

        signal(SIGALRM, alarm_handler);
        pipe(fd);
        alarm(3);
        read(fd[0], buf, 100);

        close(fd[0]);
        close(fd[1]);

        return 0;
}
```

Named Pipe

Named Pipe (FIFO)

- **Pipe의 단점**
  - 부모 자식 프로세스 사이 혹은 동일한 부모의 자식 프로세스 사이의 통신만을 지원
  - 즉, 서로 연관이 있는 프로세스 사이의 통신 지원

- **FIFO (Named Pipe)**
  - 파이프와 거의 유사
  - 파일 시스템에 특수 파일로 등록
  - 다른 프로세스에서 정규 파일을 사용하듯이 읽기/쓰기 수행 가능
  - mkfifo()를 통해 생성

Named Pipe (FIFO)

NAME

mkfifo – create fifo

SYNOPSIS

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

RETURN VALUE

Return: 0 if OK, -1 on error

Named Pipe (FIFO)

- **Description**
  - FIFO 특수 파일을 생성
  - pathname
    - FIFO 파일을 생성할 경로
  - mode
    - 접근 모드 (e.g. 0666, 0777)

fifo_reader.c

```
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

#define MSGSIZE 64

int main(void) {
  char   msg[MSGSIZE];
  int    fd, nread, cnt;

  if (mkfifo("./fifo", 0666) == -1) {
    printf("fail to create fifo file\n");      exit(1);
  }
  if ((fd = open("./fifo", O_RDWR)) < 0) {
    printf("fail to open fifo file\n");       exit(1);
  }
  for (cnt= 0 ; cnt < 3; cnt++) {
    if ((nread = read(fd, msg, MSGSIZE)) < 0) {
      printf("fail to read from fifo file\n");          exit(1);
    }
    printf("recv: %s\n", msg);
  }
  unlink("./fifo");
}
```

fifo_writer.c

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

#define MSGSIZE 64

int main(void) {
  char  msg[MSGSIZE];
  int    fd, cnt;

  if((fd = open("./fifo", O_WRONLY)) < 0) {
    printf("fail to open fifo file\n");        exit(1);
  }

  for (cnt = 0; cnt < 3; cnt++) {
    printf("input a message: ");
    scanf("%s", msg);
    if (write(fd, msg, MSGSIZE) == -1) {
      printf("fail to write to fifo\n");          exit(1);
    }
    sleep(1);
  }
}
```

IPC Programming 실습

컴퓨터과학부
유닉스 프로그래밍

실습 1

- **SIGINT (Ctrl+C)가 발생하면 시그널 핸들러 함수가 호출되도록 하라**
  - Signal이 발생하는 것을 기다리기 위하여 sleep(100)을 이용할 것

- **SIGINT가 한번 더 발생하면 프로그램이 종료되도록 하라 (SIGINT의 default action을 수행하도록 하라)**

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

int catchint(int signo)
{
        printf("SIGINT Received\n");
        signal(SIGINT, SIG_DFL);
}
int main(void) {

        signal(SIGINT, catchint);

        for (;;) pause();

        printf("Bye!~~\n");

        return 0;
}
```

실습 2

- **부모 프로세스가 자식 프로세스를 생성**
- **자식 프로세스는 화면에 1~100까지의 합을 출력하고 부모 프로세스에게 시그널을 전송한 후 종료**
  - SIGUSR1 이용할 것
- **부모프로세스는 자식 프로세스에게 시그널이 도착할 때까지 대기**
- **시그널이 도착하면 시그널을 받았다는 메시지를 출력한 후 종료**

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

void usr1_handler(int signo) {
        printf("signal recieved from childl!\n");
}

int main(void) {

        int pid;

        if( (pid = fork()) > 0 ) {          /* Parent */
                signal(SIGUSR1, usr1_handler);
                pause();
        }
        else if( pid == 0 ) {               /* Child */
                int i, sum = 0;

                for( i = 1; i <= 100; i++ )
                        sum += i;

                printf("sum result from 1 to 100: %d\n", sum);

                kill(getppid(), SIGUSR1);
        }

}
```

실습 3

- **Mini shell을 작성할 것**
  - fork, exec, waitpid 사용
  - pipe 또는 popen 사용

- **Pipe를 통해서 하나의 프로그램의 결과를 다른 프로그램의 input으로 넣을 수 있어야 함**
  - Ex) cat aaa.txt | more

실습 4

- **Named pipe np를 만들고 'ls –al > np' 명령어를 수행하고 다음을 확인하라**
  - 1. 결과가 어떻게 되는가?
  - 2. 'cat < np'를 수행하고 그 결과는 어떻게 되는가?
  - 3. 왜 이런 결과가 나타나는가 생각해 보라.

- **위의 예제를 직접 해 보기 전에 결과를 예상해 볼 것!!**

- **mkfifo np**

- **2가지 방법**
  - 1.
    - ls –al > np &
    - cat < np
  - 2.
    - ls –al > np
    - 터미널을 새로 띄우고 띄운 터미널에서
    - cat < np