10장. 프로세스간 통신

프로세스간 통신

- 프로세스간 통신 기법
  - 시그널, 파일 잠금, 파이프, 메시지 큐, 세마포어, 공유 메모리, 소켓 등

- 파일을 이용한 레코드 잠금
  - 레코드 잠금**(record locking)**
    - 프로세스가 특정 파일의 일부 레코드에 대하여 잠금 기능 설정
    - 다른 프로세스로 하여금 이 파일에 접근하지 못하도록 함
  - 종류
    - 읽기 잠금 – 다른 프로세스들이 해당 영역에 쓰기 잠금 불가
    - 쓰기 잠금 – 다른 프로세스들이 해당 영역에 읽기와 쓰기 잠금 모두 불가

파일을 이용한 레코드 잠금

fcntl 시스템 호출

기능

- 파일 제어 : **fd** 가 가리키는 파일을 **cmd** 명령에 따라 제어한다.

사용법

```
#include <fcntl.h>

int fcntl(int fd, int cmd, struct flock *lock);
```

- **cmd : fcntl** 이 어떻게 동작할 것인지 결정
  - **F_GETLK :** 레코드 잠금 정보 획득, 정보는 셋째 인자 **lock** 에 저장
  - **F_SETLK :** 파일에 레코드 잠금 적용, 불가하면 즉시 **–1** 반환
  - **F_SETLKW :** 파일에 레코드 잠금 적용, 불가하면 잠금 해제를 기다림
- 성공적인 호출에 대하여, 반환값은 동작에 달려 있다

파일을 이용한 레코드 잠금

◈ Lock

➢ **struct flock :** 레코드 잠금에 대한 내용 기술

```
struct flock {
        short l_type;      /* 잠금 유형 */
        shor l_whence;     /* 잠금 영역을 정하는 기준 */
        off_t l_start;     /*  잠금 영역의 시작 위치 */
        off_t l_len;       /* 잠금 영역의 바이트 단위 길이 */
        pid_t l_pid;       /* 명령어 F_GETLK 일 때 잠금 설정하는 프로세스ID */
}
```

➢ **l_type :** 잠금 유형

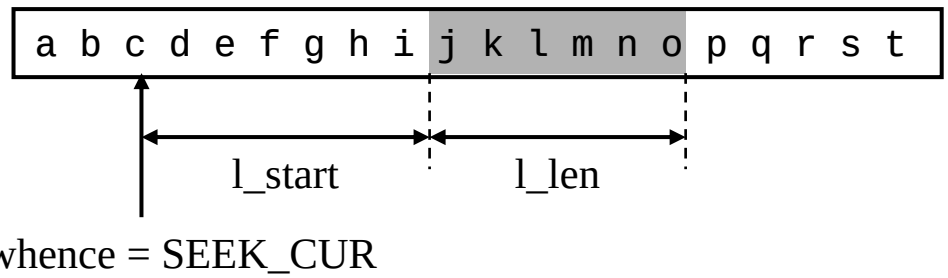
- **F_RDLCK :** 읽기 잠금 지정
- **F_WRLCK :** 쓰기 잠금 지정
- **F_UNLCK :** 잠금 해제

➢ **l_whence, l_start, l_len**

- 잠금 위치 지정

➢ **l_pid**

- 잠근 프로세스 **ID**



l_whence = SEEK_CUR

레코드 잠금

▣ 레코드 잠금 사용 예

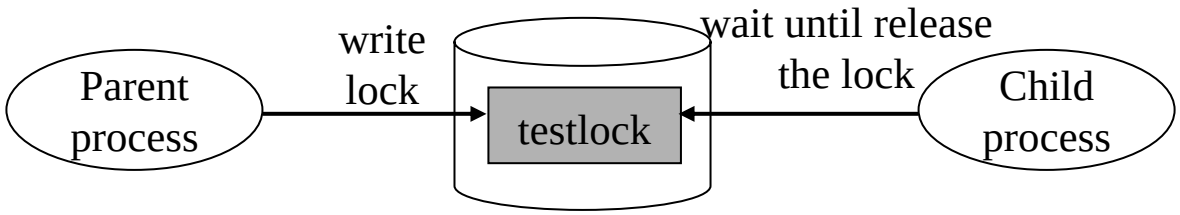
◈ 레코드 잠금 예제 프로그램

```
/* filelock.c */
/* record lock example */
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>

main()
{
    int fd;
    struct flock testlock;
    int pid;

    /* 쓰기 잠금의 인자 지정 */
    testlock.l_type = F_WRLCK;
    testlock.l_whence = SEEK_SET;
    testlock.l_start = 0;
    testlock.l_len = 10;
    /* open file */
    fd = open ("testlock", O_RDWR | O_CREAT,
0666);
    if (fcntl (fd, F_SETLKW, &testlock) == -1) {
        perror ("parent: locking");
        exit (1);
    }
    printf ("parent: locked record₩n");
    pid = fork();
    if (pid == 0) { /* child process */
        testlock.l_len = 5;
        if (fcntl (fd, F_SETLKW, &testlock) == -1) {
            perror ("child: locking");
            exit (1);
        }
        printf ("child: locked₩n");
        sleep(5);
        printf ("child: exiting₩n");
    }
    else if (pid > 0) {
        sleep(5);
        printf ("parent: exiting₩n");
    }
    else
        perror ("fork failed");
}
```

레코드 잠금

▣ 레코드 잠금 사용 예

◈ 레코드 잠금 예제 프로그램 동작 과정



◈ 레코드 잠금 예제 프로그램 실행 결과

```
[cprog2@seps5 ipcs]$ gcc filelock.c –o filelock
[cprog2@seps5 ipcs]$ ./filelock
parent: locked record
parent: exiting
[cprog2@seps5 ipcs]$ child: locked
child: exiting

[cprog2@seps5 ipcs]$
```

레코드 잠금을 이용한 데이터 전달

▣ 레코드 잠금 사용 예

◈ 레코드 잠금 예제 프로그램

```
/* lockdata1.c */
/* record lock example */
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>

#define THIS_PROCESS    1
#define THAT_PROCESS    2

main()
{
    int fd;
    struct flock testlock;
    char buf[20];

    /* 쓰기 잠금의 인자 지정 */
    testlock.l_type = F_WRLCK;
    testlock.l_whence = SEEK_SET;
    testlock.l_start = 0;
    testlock.l_len = 0;
    /* open file */
    fd = open ("testlock", O_RDWR | O_CREAT, 0666);
    if (fcntl (fd, F_SETLKW, &testlock) == -1) {
        perror ("process %d: lock failed", THIS_PROCESS);
        exit (1);
    }

    printf ("process %d: locked successfully₩n", THIS_PROCESS);
    sprintf(buf,"Hello, process %d", THAT_PROCESS);
    write (fd, buf, 17);
    printf ("process %d: wrote ₩"%s₩" to testlock₩n", THIS_PROCESS, buf);

    sleep (5);
    printf ("process %d: unlocking₩n", THIS_PROCESS);
}
```

레코드 잠금을 이용한 데이터 전달

▣ 레코드 잠금 사용 예

◈ 레코드 잠금 예제 프로그램

```
/* lockdata2.c */
/* record lock example */
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>

#define THIS_PROCESS    2
#define THAT_PROCESS    1

main()
{
    int fd;
    struct flock testlock;
    int len;
    char buf[20];

    /* 읽기 잠금의 인자 지정 */
    testlock.l_type = F_RDLCK;
    testlock.l_whence = SEEK_SET;
    testlock.l_start = 0;
    testlock.l_len = 0;

    /* open file */
    fd = open ("testlock", O_RDWR);

    if (fcntl (fd, F_SETLKW, &testlock) == -1) {
        perror ("process %d: lock failed",
THIS_PROCESS);
        exit (1);
    }

    printf ("process %d: locked successfully₩n",
THIS_PROCESS);
    len = read (fd, buf, 20);

    printf ("process %d: read ₩"%s₩" from
testlock₩n", THIS_PROCESS, buf);
    printf ("process %d: unlocking₩n",
THIS_PROCESS);
}
```

레코드 잠금을 이용한 데이터 전달

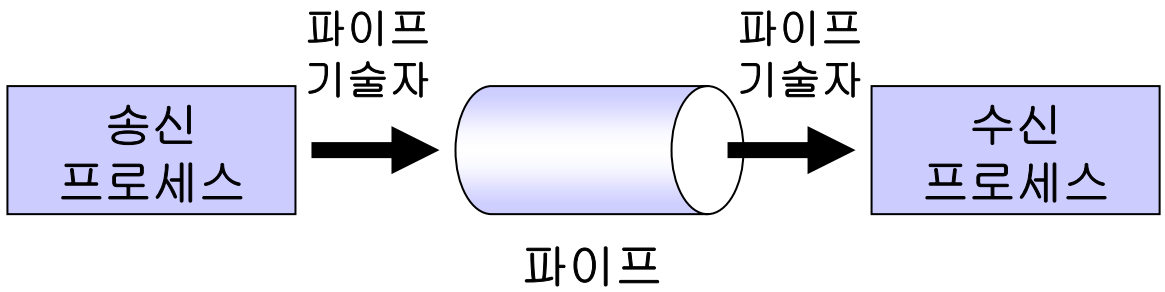
▣ 레코드 잠금 사용 예

◈ 레코드 잠금 예제 프로그램 실행 결과

```
[cprog2@seps5 ipcs]$ gcc lockdata1.c -o lockdata1
[cprog2@seps5 ipcs]$ gcc lockdata2.c -o lockdata2
[cprog2@seps5 ipcs]$ ./lockdata1 &
process 1: locked successfully
[2] 1669
[cprog2@seps5 ipcs]$ process 1: wrote "Hello, process 2" to testlock

[cprog2@seps5 ipcs]$ ./lockdata2
process 1: unlocking
process 2: locked successfully
process 2: read "Hello, process 2" from testlock
process 2: unlocking
[2]+  Exit 21                 ./lockdata1

[cprog2@seps5 ipcs]$
```

파이프

- 레코드 잠금의 단점
  - 파일을 이용 – 추가적인 자원 낭비 및 비효율적
  - 경쟁 문제
  - 외부에서 파일에 접근할 때의 보안 문제
- 파이프
  - UNIX 의 고유한 프로세스간 통신 방식
  - 한 프로세스를 다른 프로세스와 연결시켜 주는 단방향 채널



파이프

▣ 쉘 명령어에서의 파이프 사용

◈ 현재 디렉토리의 파일 수를 출력하는 명령

```
[cprog2@seps5 ipcs]$ ls / wc -l
```

➢ **"ls"** 명령의 표준 출력을 **"wc –l"** 명령의 표준 입력과 연결하는 파이프를 생성하고 자료 전달

파이프를 이용한 프로그래밍

pipe 시스템 호출

기능

- 파이프 생성
- 파이프를 가리키는 파일 기술자 쌍을 생성하고, 이를 **filedes** 에 저장

사용법

```
#include <unistd.h>

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

- **filedes[0]**은 읽기 위한 파이프
- **filedes[1]**은 쓰기 위한 파이프

반환값

- 정상적 종료: **0**, 그렇지 않을 경우: **0** 이 아닌 값
- 오류 번호(**errno**)
  - **EMFILE :** 사용자 프로세스가 너무 많은 파일 기술자를 사용하는 경우
  - **ENFILE :** 시스템 파일 테이블이 꽉 찬 경우
  - **EFAULT : filedes** 변수가 유효하지 못하는 경우

파이프를 이용한 프로그래밍

▣ 파이프 사용 예

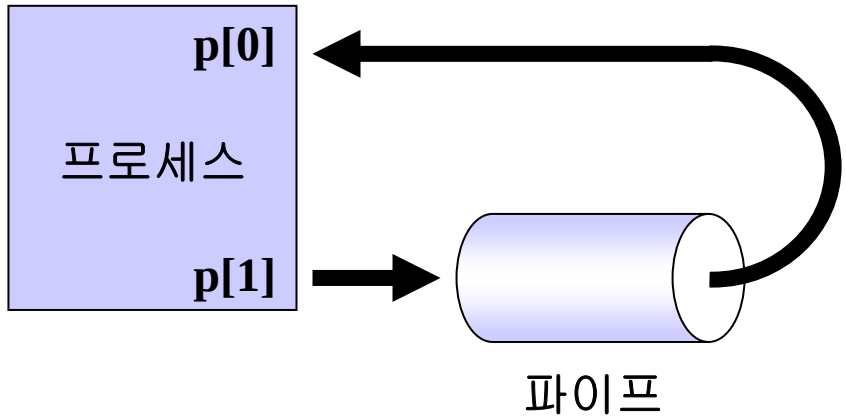
◈ 파이프 예제 프로그램

```
/* selfpipe.c */
#include <unistd.h>
#include <stdio.h>

#define MSGSIZE 16

char *msg[2] = { "Hello", "World"};

main()
{
        char buf[MSGSIZE];
        int p[2], i;

        /* open pipe */
        if (pipe(p) == -1) {
                perror ("pipe call failed");
                exit(1);
        }

        /* write to pipe */
        for (i = 0; i < 2; i++)
                write (p[1], msg[i], MSGSIZE);

        /* read from pipe */
        for (i = 0; i < 2; i++) {
                read (p[0], buf, MSGSIZE);
                printf ("%s₩n", buf);
        }
}
```

파이프를 이용한 프로그래밍

▣ 파이프 사용 예

◈ 파이프 예제 프로그램 동작 과정



◈ 파이프 예제 프로그램 실행 결과

```
[cprog2@seps5 ipcs]$ gcc selfpipe.c –o selfpipe
[cprog2@seps5 ipcs]$ ./selfpipe
Hello
World
[cprog2@seps5 ipcs]$
```

파이프를 이용한 프로그래밍

▣ 파이프 사용 예

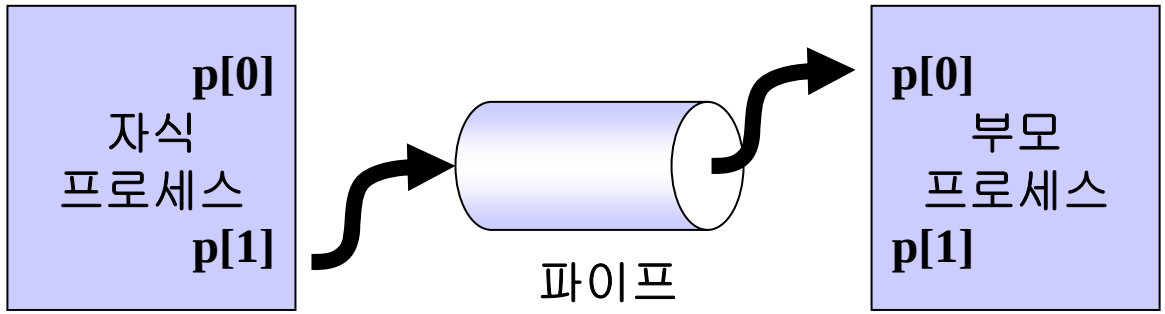
◈ 파이프 예제 프로그램

```
/* pipetest.c */
/* pipe example */
#define MSGSIZE 16

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

        /* open pipe */
        if (pipe(p) == -1) {
                perror ("pipe call failed");
                exit(1);
        }

        pid = fork();
        if (pid == 0) { /* child process */
                close(p[0]);
                /* write to pipe */
                for (i = 0; i < 2; i++) {
                        sprintf(buf, "Hello, world #%d",
i+1);
                        write(p[1], buf, MSGSIZE);
                }
        }
        else if (pid > 0) {
                close(p[1]);
                /* read from pipe */
                for (i = 0; i < 2; i++) {
                        read (p[0], buf, MSGSIZE);
                        printf ("%s₩n", buf);
                }
        }
        else
                perror ("fork failed");
}
```

파이프를 이용한 프로그래밍

▣ 파이프 사용 예

◈ 파이프 예제 프로그램 동작 과정



◈ 파이프 예제 프로그램 실행 결과

```
[cprog2@seps5 ipcs]$ gcc pipetest.c –o pipetest
[cprog2@seps5 ipcs]$ ./pipetest
Hello, world #1
Hello, world #2
[cprog2@seps5 ipcs]$
```

비봉쇄(nonblocking) 읽기와 쓰기

- 봉쇄(blocking) 읽기/쓰기
  - 파이프로부터 자료를 읽거나 쓸 때까지 프로세스가 대기
- 비봉쇄(nonblocking) 읽기/쓰기
  - 파이프로부터 자료를 읽거나 쓸 때 프로세스가 대기하지 않음
  - 구현 방법
    - **fcntl** 시스템 호출 사용
      - cmd : F_SETFL 명령 – arg 의 값으로 파일 기술자의 플래그 설정
      - arg : O_NONBLOCK 으로 설정하면 비봉쇄 읽기/쓰기 가능

```
if (fcntl(pipe, F_SETFL, O_NONBLOCK) == -1) {
        perror("fcntl failed");...
}
```

    - **select** 시스템 호출 사용

비봉쇄(nonblocking) 읽기와 쓰기

▣ 파이프 사용 예

◈ 비봉쇄 읽기/쓰기 예제 프로그

```
/* nonblockpipe.c */
/* pipe and nonblocking read/write example */
#include <fcntl.h>
#include <errno.h>

#define MSGSIZE 16

char *parent_name = "parent";
char *child_name = "child";
char *parent_msg = "Hello, child!";
char *child_msg = "Hello, parent!";

void nonblock_rw (char *, int, int, char *);

main()
{
        int pp[2][2], i;
        int pid;

        /* open pipe */
        for (i = 0; i < 2; i++) {
                if (pipe(pp[i]) == -1) {
                        perror ("pipe call failed");
                        exit (1);
                }
        }
        pid = fork();
        if (pid == 0) { /* child process */
                close(pp[0][1]);
                close(pp[1][0]);
                nonblock_rw     (child_name,     pp[0][0],
pp[1][1], child_msg);
        }
        else if (pid > 0) { /* parent process */
                close(pp[0][0]);
                close(pp[1][1]);
                nonblock_rw(parent_name,       pp[1][0],
pp[0][1], parent_msg);
        }
        else
                perror ("fork failed");
}
```

비봉쇄(nonblocking) 읽기와 쓰기

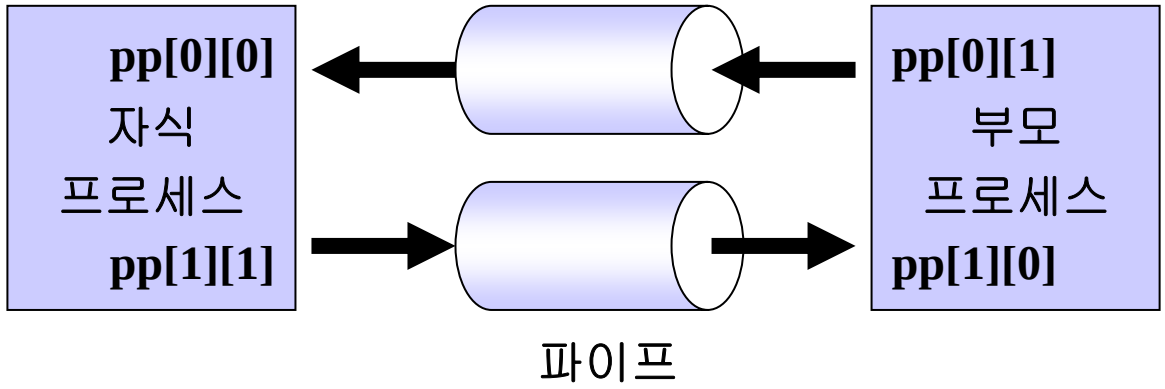
▣ 파이프 사용 예

◈ 비봉쇄 읽기/쓰기 예제 프로그램 계속

```
void nonblock_rw (char *name, int read_pipe,
int write_pipe, char *message)
{
   char buf[MSGSIZE];
   int nread;

    /* set O_NONBLOCK of fcntl system call */
    if (fcntl (read_pipe, F_SETFL, O_NONBLOCK)
== -1) {
            perror("read pipe call");
            exit (1);
    }
    if (fcntl (write_pipe, F_SETFL, O_NONBLOCK)
== -1) {
            perror("write pipe call");
            exit (1);
    }
    for (;;) {
        switch (nread = read(read_pipe, buf,
MSGSIZE)) {
            case -1:
                if (errno == EAGAIN) {
                        printf("%s:      pipe      empty!₩n",
name);
                        sleep (1);
                        break;
                }
                else {
                        perror("read call");
                        exit (1);
                }
            case 0:
                printf ("%s:  read  pipe  closed₩n",
name);
                exit (1);
            default:
              printf("%s: MSG=%s₩n", name, buf);
        }
        write(write_pipe, message, MSGSIZE);
        sleep (1);
    }
}
```

비봉쇄(nonblocking) 읽기와 쓰기

- 파이프 사용 예
  - 비봉쇄 읽기/쓰기 예제 프로그램 동작 과정



비봉쇄(nonblocking) 읽기와 쓰기

▣ 파이프 사용 예

◈ 비봉쇄 읽기/쓰기 예제 프로그램 실행 결과

```
[cprog2@seps5 ipcs]$ gcc nonblockpipe.c –o nonblockpipe
[cprog2@seps5 ipcs]$ ./nonblockpipe
child: pipe empty!
parent: pipe empty!
parent: MSG=Hello, parent!
child: MSG=Hello, child!
parent: MSG=Hello, parent!
child: MSG=Hello, child!
parent: MSG=Hello, parent!
child: MSG=Hello, child!
parent: MSG=Hello, parent!
child: MSG=Hello, child!
parent: MSG=Hello, parent!
child: MSG=Hello, child!
[cprog2@seps5 ipcs]$
```

select 를 이용한 파이프 프로그래밍

select 시스템 호출

기능

- 다중 입출력 관리
- 지정된 파일 기술자 집합 중 읽기와 쓰기가 준비된 것이 있는지, 또는 오류가 있는지 등을 검사

사용법

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

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

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

- **FD_SET :** 파일 기술자를 집합에 추가
- **FD_CLR :** 파일 기술자를 집합에서 뺀다
- **FD_ISSET :** 파일 기술자가 집합의 일부분인지 아닌지를 검사
- **FD_ZERO :** 파일 기술자 집합을 초기화

select 를 이용한 파이프 프로그래밍

▣ 파이프 사용 예

◈ Select 를 이용한 다중 파이프 예제 프로그램

```
/* selectpipe.c */
/* pipe and select example */
#include <sys/time.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>

#define MSGSIZE 16

char *hello_msg = "Hello, parent!";
char *bye_msg = "Bye, parent!";

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

main()
{
        int pp[3][2], i;
        int pid;

        /* open pipe */
        for (i = 0; i < 3; i++) {
                if (pipe(pp[i]) == -1) {
                        perror ("pipe call failed");
                        exit (1);
                }
                pid = fork();
                if (pid == 0) /* child process */
                        child (pp[i]);
                else if (pid == -1) {
                        perror ("fork failed");
                        exit (1);
                }
        }

        /* parent process */
        parent (pp);
}
```

select 를 이용한 파이프 프로그래밍

▣ 파이프 사용 예

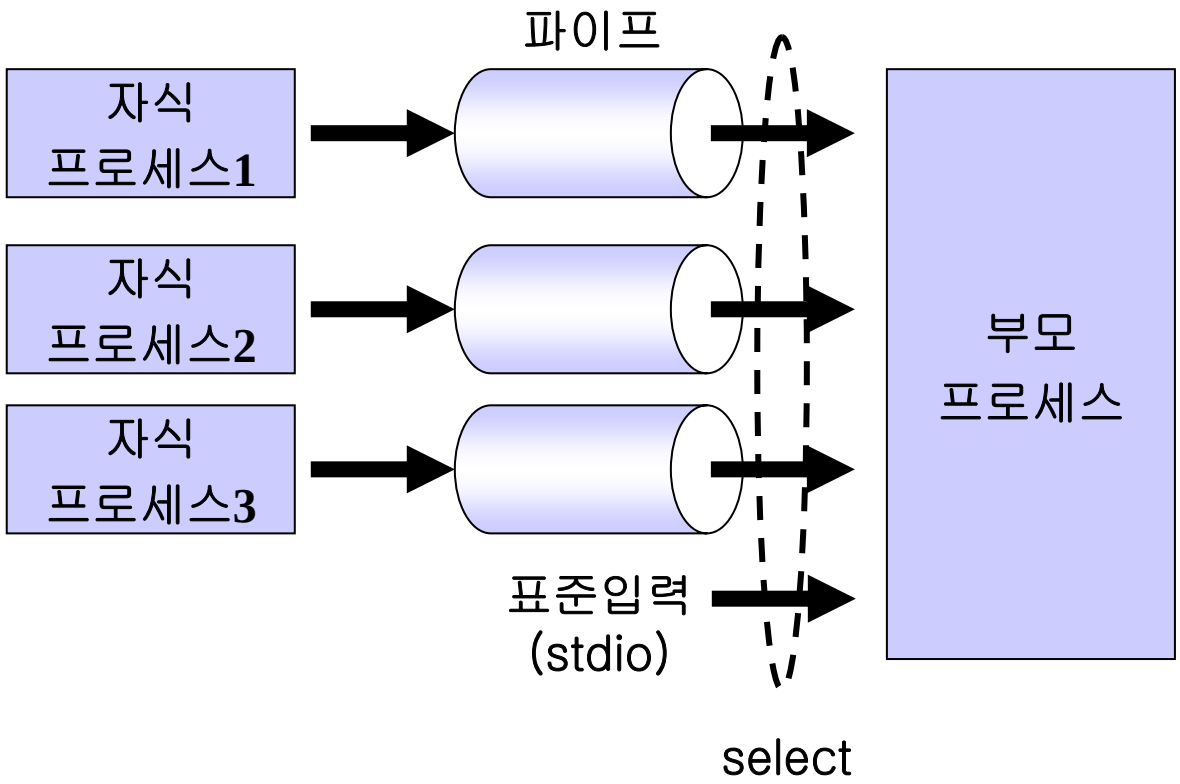
◈ Select 를 이용한 다중 파이프 (

```
void parent(int pp[3][2])
{
        char buf[MSGSIZE], ch;
        fd_set set, master;
        int i;

        for (i = 0; i < 3; i++)
            close(pp[i][1]);
        /* set bit mask of select system call */
        FD_ZERO (&master);
        FD_SET (0, &master);
        for (i = 0; i < 3; i++)
            FD_SET (pp[i][0], &master);
        while (set=master, select (i+1, &set, NULL,
NULL, NULL) > 0) {
            if (FD_ISSET(0, &set)) {
                printf("From standard input: ₩n");
                read (0, &ch, 1);
                printf("%c₩n", ch);
            }
            for (i = 0; i < 3; i++) {
                if (FD_ISSET(pp[i][0], &set))
                    if (read(pp[i][0], buf, MSGSIZE) > 0)
                        printf("message is %s from
child %d₩n", buf, i);
            }
            if (waitpid (-1, NULL, WNOHANG) == -1)
                return;
        }
}
void child(int p[2])
{
        int j;

        /* write to pipe */
        close (p[0]);
        for (j = 0; j < 3; j++) {
                write(p[1], hello_msg, MSGSIZE);
               sleep (getpid() % 4);
        }
        write(p[1], bye_msg, MSGSIZE);
        exit (0);
}
```

select 를 이용한 파이프 프로그래밍

▣ 파이프 사용 예

◈ Select 를 이용한 다중 파이프 예제 프로그램 동작 과정



select 를 이용한 파이프 프로그래밍

▣ 파이프 사용 예

◈ Select 를 이용한 다중 파이프 예제 프로그램 실행 결과

```
[cprog2@seps5 ipcs]$ gcc selectpipe.c –o selectpipe
[cprog2@seps5 ipcs]$ ./selectpipe
message is Hello, parent! from child 0
message is Hello, parent! from child 1
message is Hello, parent! from child 2
message is Hello, parent! from child 0
message is Hello, parent! from child 1
message is Hello, parent! from child 0
message is Hello, parent! from child 2
message is Bye, parent! from child 0
message is Hello, parent! from child 1
a
From standard input:

message is Hello, parent! from child 2
message is Bye, parent! from child 1
message is Bye, parent! from child 2
[cprog2@seps5 ipcs]$
```

표준 파이프 입출력 함수

popen 과 pclose 함수

기능

- **popen : pipe**와 **fork**를 이용하여, 쉘 명령어의 입력 혹은 출력을 파이프와 연결한 다음 쉘 명령어를 실행
- **pclose :** 사용하고 난 파이프와 명령 프로세스를 닫음

사용법

```
#include <stdio.h>

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

- **command :**수행할 명령어를 나타내는 문자열의 포인터
- **type : pipe** 의 읽기**('r')** 또는 쓰기**('w')** 방향

반환값

- **popen** 은 성공적인 호출에 대하여 파이프와 연관된 파일 기술자 반환
- **pclose** 는 성공적인 호출에 대하여 **0**을 반환
- 오류시에는 **-1** 이 반환되고**, errno** 가 적절히 설정

표준 파이프 입출력 함수

▣ 파이프 사용 예

◈ popen/pclose 를 사용하여 로긴 사용자 수를 알아내는 프로그램

```
/* iopipe.c */
/* pipe and popen/pclose example */
#include <stdio.h>
#define BUFSIZE 256

main()
{
        FILE *pin, *pout;
        char buf[BUFSIZE];

        pin = popen("who", "r");
        pout = popen("wc -l", "w");

        while (fgets(buf, BUFSIZE, pin) != NULL)
                fputs(buf, pout);

        pclose(pin);
        pclose(pout);
}
```

◈ 프로그램 실행 결과

```
[cprog2@seps5 ipcs]$ gcc iopipe.c –o iopipe
[cprog2@seps5 ipcs]$ ./iopipe
       2
[cprog2@seps5 ipcs]$
```

FIFO 를 이용한 프로그래밍

- 파이프의 단점
  - 프로그램 내에서는 부모와 자식 프로세스 간 데이터 전달 가능하지만, 프로세스가 종료되면 파이프도 삭제
  - 독립된 두 프로세스 사이에 파이프를 통해 데이터 전달 불가능
- ➔ **FIFO 또는 명명 파이프(named pipe)**
  - 특수한 형태의 파일
  - 입출력 처리 방식은 선입선출**(first-in first-out)** 방식
  - 파일 **inode**를 가지므로 영구적이고 임의의 프로세스가 접근 가능
- **mkfifo** 시스템 호출
  - 기능
    - **FIFO** 파이프 생성
  - 사용법

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

    - **pathname :** 생성할 **FIFO** 파일의 경로 지정
    - **mode :** 이 파일의 허가권을 설정

FIFO 를 이용한 프로그래밍

▣ 파이프 사용 예

◈ FIFO 를 이용한 예제 프로그램

```
/* readfifo.c */
/* named pipe example */
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#define MSGSIZE 64

char *testfifo = "fifo";

main (int argc, char **argv)
{
        int fd;
        char buf[MSGSIZE];

        if (mkfifo(testfifo, 0666) == -1) {
                perror ("mkfifo failed");
                exit (1);
        }

        /* open testfifo, setting O_RDWR */
        if ((fd = open(testfifo, O_RDWR)) < 0) {
                perror("fifo open failed");
                exit (1);
        }

        /* receive message */
        while (1) {
                if (read (fd, buf, MSGSIZE) < 0) {
                        perror ("fifo read failed");
                        exit (1);
                }
                printf ("received message: %s₩n", buf);
        }
}
```

FIFO 를 이용한 프로그래밍

▣ 파이프 사용 예

◈ FIFO를 이용한 예제 프로그램 계속

```
/* writefifo.c */
/* named pipe example */
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#define MSGSIZE 64

char *testfifo = "fifo";

main (int argc, char **argv)
{
        int fd, i, nwrite;
        char buf[MSGSIZE];

        if (argc < 2) {
                fprintf (stderr, "Usage: ex11_8_s
msg ...₩n");
                exit (1);
        }
        /* open testfifo, setting O_WRONLY */
        if ((fd = open(testfifo, O_WRONLY)) < 0) {
                perror("fifo open failed");
                exit (1);
        }

        /* send message */
        for (i = 1; i < argc; i++) {
                strcpy (buf, argv[i]);
                if ((nwrite = write (fd, buf, MSGSIZE))
< 0) {
                        perror ("fifo write failed");
                        exit (1);
                }
        }
}
```

FIFO 를 이용한 프로그래밍

▣ 파이프 사용 예

◈ FIFO 를 이용한 예제 프로그램 실행 결과

```
[cprog2@seps5 ipcs]$ gcc -o readfifo readfifo.c
[cprog2@seps5 ipcs]$ gcc -o writefifo writefifo.c
[cprog2@seps5 ipcs]$ ./readfifo &
[1] 7179
[cprog2@seps5 ipcs]$ ls -la
 ...
prw-rw-r--    1 cprog2   cprog2          0 11월 25 15:59 fifo
...
[cprog2@seps5 ipcs]$ ./writefifo Hello World!
received message : Hello
received message : World!
[cprog2@seps5 ipcs]$
```

고급 프로세스간 통신

▣ 고급 IPC 개요

◈ UNIX 와 LINUX 에서 지원하는 고급 IPC 자원 (System V IPC)

1. 메시지 큐를 통하여 프로세스들 간에 메시지 전달
2. 세마포어를 통하여 프로세스들 간의 동기화
3. 프로세스 간 메모리 영역 공유

◈ 자원 요청 시 동적으로 데이터 구조 생성

- 명시적으로 삭제하지 않으면 시스템이 종료될 때까지 메모리에 존재
- 독립적인 프로세스 간 통신에 사용 가능

◈ IPC 자원의 구별

- **Key (32 bit)**
  - 파일 경로와 비슷
  - 프로그래머가 자유롭게 선택
- **식별자(32 bit)**
  - 파일 기술자와 비슷
  - 커널에 의해 자원에 부여되며, 시스템 내에서 고유한 값

고급 IPC 개요

▣ 고급 IPC 자원의 사용

◈ IPC 자원의 생성

- 임의의 IPC 키와 생성 함수를 통하여 IPC 자원과 IPC 식별자를 생성
- 생성 함수 - **msgget, semget, shmget** 함수
- IPC 식별자를 통한 자원 접근과 프로세스 간 통신
- 생성 시 오류

오류코드	설명
EACCESS	프로세스가 적절한 접근 권한을 가지지 않는다.
EEXIST	프로세스가 이미 존재하는 키를 가진 IPC 자원의 생성을 시도
EIDRM	자원이 삭제된 것으로 표시되어 있다.
ENOENT	요청한 키의 IPC 자원이 존재하지 않고, 프로세스가 생성을 요구하지 않음
ENOMEM	추가 IPC 자원에 대한 공간이 남아 있지 않다.
ENOSPC	IPC 자원 수에 대한 최대 제한을 초과하였다.

- 생성 함수의 마지막 인자 : **permflags**
  - **IPC_CREATE : IPC_**지정한 IPC 자원이 존재하지 않으면 생성하도록 지시
  - **IPC_EXCL : IPC_CREAT** 와 함께 사용될 때 자원이 이미 존재하지 않으면 실패
  - 하위 9 비트를 IPC 자원에 대한 접근 허가권(단, 실행 허가는 제외)을 지정

고급 IPC 개요

▣ 고급 IPC 자원의 사용

◈ IPC 자원의 상태 정보

- ➢ 해당 자원에 관련된 데이터 구조를 만들어 각종 정보 수록
- ➢ 허가 구조 포함 : **ipc_perm**

```
ushort cuid; /*IPC 객체 생성자의 사용자 id */
ushort cgid; /* 생성자의 그룹 id */
ushort uid; /* 유효 사용자 id */
ushort gid; /* 유효 그룹-id */
mode_t mode; /* 허가권 비트 마스크 */
ushort seq; /* slot usage sequence number */
```

◈ IPC 자원의 제어

- ➢ 자원의 상태 정보를 얻거나 제어하는 데 사용
- ➢ 제어 함수 - **msgctl, semctl, shmctl**

◈ IPC 자원의 동작

- ➢ 실제 IPC 자원을 동작시킴
- ➢ 동작 함수
  - **msgsnd** 와 **msgrcv** 함수 : 메시지 큐를 통해 메시지를 전달하거나 받음
  - **semop** 함수 : 세마포어를 얻거나 줌
  - **shmat** 와 **shmdt** 함수 : 공유 메모리를 프로세스의 주소 공간에 붙이거나 뗌

ipcs 와 ipcrm 명령

▣ 고급 IPC 자원을 위해 쉘에서 제공하는 명령

◈ ipcs - 전체 시스템에서 사용하고 있는 고급 IPC 자원 정보 출력

```
[cprog2@seps5 ipcs]$ ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000049 1146880    cprog2     666        1024       0

------ Semaphore Arrays --------
key        semid      owner      perms      nsems      status
0x00000049 65536      cprog2     666        1

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x00000049 0          cprog2     666        0            0
```

◈ ipcrm - 시스템으로부터 특정한 고급 IPC 자원을 제거

ipcrm	
일반형식	ipcrm [msg|sem|shm] id
주요옵션	msg : 메시지 큐를 삭제하는 경우 sem : 세마포어를 삭제하는 경우 shm : 공유 메모리를 삭제하는경우

메시지 큐

메시지 큐

- 프로세스 간에 문자나 바이트 열로 이루어진 메시지를 전달하기 위한 일종의 버퍼와 같은 큐
- 메시지 큐와 파이프의 차이점
  - 파이프와 달리 메시지의 크기가 제한적
  - **select** 등을 사용할 수 없음
  - 우선순위 등에 따라 메시지 관리 가능

메시지 큐

msgget 시스템 호출

기능

- 메시지 큐 생성

사용법

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

int msgget(key_t key, int permflags);
```

- **key :** 생성할 메시지 큐에 대한 고유 번호
  - 이 값을 통해 프로세스들이 메시지 큐를 공유 가능
  - **IPC_PRIVATE**을 사용 - 다른 프로세스가 생성한 **key** 값과 중복되지 않는 유일한 메시지 큐 생성
- **permflags :** 생성 시 메시지 큐 사용 허가권 지정
  - 메시지를 보내기 위해서는 쓰기 허가권 설정
  - 메시지를 받기 위해서는 읽기 허가권 설정

반환값

- 성공적인 호출에 대하여 메시지 큐 식별자 반환
- 오류시에는 **-1** 이 반환되고, **errno** 가 적절히 설정

메시지 큐

msgsnd 시스템 호출

기능

- 메시지 큐로 메시지 전송

사용법

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

int msgsnd(int msqid, strucg msgbuf *msgp, size_t msgsz, int msgflg);
```

- **msqid** 가 가리키는 메시지 큐에 메시지를 추가
- **msgp :** 전송할 버퍼의 주소

```
struct msgbuf {
    long mtype;    /* 메시지 유형, 0 보다 커야 한다 */
    char mtext[xxx];  /* 메시지 데이터 */
};
```

- **msgflg : 0** 이거나 **IPC_NOWAIT**
  - **IPC_NOWAIT** 가 설정되어 있으면, 즉시 반환
  - 그렇지 않으면, 메시지 큐가 메시지를 저장할 수 있을 때까지 대기
- **msgsz :** 전송할 메시지의 최대 바이트 크기를 지정

메시지 큐

msgrcv 시스템 호출

기능

- 메시지 큐로부터 메시지 수신

사용법

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

ssize_t msgrcv(int msqid, struct msgbuf *msgp, size_t msgsz, long msgtyp, int msgflg);
```

- **msqid** 가 가리키는 메시지 큐로부터 메시지를 읽음
- **msgp :** 수신할 버퍼의 주소
- **msgsz : msgp** 가 가리키는 구조체에 저장할 수 있는 최대 바이트 크기를 지정
- **msgtyp :** 수신할 메시지의 유형 결정
  1. **0 :** 메시지 큐의 제일 앞에 있는 메세지를 읽음
  2. 양수 **:** 큐에서 같은 유형을 가진 첫 메시지를 읽음**.** 다만**, msgflg** 인자에 **MSG_EXCEPT** 가 지정되어 있으면**, msgtyp**와 같지 않은 유형의 첫 메시지를 읽음
  3. 음수 **:** 큐에서 **msgtyp** 의 절대값보다 작거나 같은 유형의 첫 메시지를 읽음
- **msgflg : 0** 이거나 **IPC_NOWAIT, IPC_NOERROR** 또는 **MSG_EXCEPT**
  - **IPC_NOWAIT** 가 설정되어 있으면**,** 즉시 반환
  - 그렇지 않으면**,** 메시지 큐가 메시지를 저장할 수 있을 때까지 대기

메시지 큐

msgctl 시스템 호출

기능

- 메시지 큐 제어 (메시지 큐의 정보를 얻거나, 변경하거나, 삭제)

사용법

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

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

- **msqid**가 가리키는 메시지 큐에 **cmd** 가 지정한 동작을 수행
- **cmd :** 수행할 명령
  - **IPC_STAT : msqid** 와 관련된 메시지 큐의 정보를 **buf** 의 위치에 저장
  - **IPC_SET :** 메시지 큐의 정보를 **buf** 가 가리키는 값으로 변경, 변경 가능한 값은 **msg_perm.uid, msg_perm.gid, msg_perm.mode, msg_qbyte**
  - **IPC_RMID :** 메시지 큐와 관련된 데이터 구조를 즉시 삭제
- **buf :** 메시지 큐 데이터 구조의 주소

```
struct ipc_perm msg_perm;
time_t msg_stime;      /* 최근 msgsnd 시간 */
time_t msg_rtime;      /* 최근 msgrcv 시간 */
time_t msg_ctime;      /* 최근 변경 시간 */
unsigned short msg_qnum;       /* 큐의 메시지 수 */
unsigned short msg_qbytes;     /* 큐의 최대 바이트 수 */
pid_t msg_lspid;   /* 최근 msgsnd 의 pid */
pid_t msg_lrpid;   /* 최근 수신 pid */
```

메시지 큐

▣ 메시지 큐 사용 예

◈ 메시지 큐를 이용한 예제 프로그램

```
/* sendmq.c */
/* message queue example */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <string.h>

#define BUFSIZE 16
#define QKEY    (key_t)0111

struct msgq_data {
        long type;
        char text[BUFSIZE];
} ;
struct msgq_data send_data = {1, "Hello, world"};
```

```
main()
{
        int qid, len;
        char buf[BUFSIZE];

        if ((qid = msgget(QKEY, IPC_CREAT | 0666))
== -1) {
                perror ("msgget failed");
                exit (1);
        }

        if
(msgsnd(qid,&send_data,strlen(send_data.text),0)
== -1) {
                perror ("msgsnd failed");
                exit (1);
        }
}
```

메시지 큐

▣ 메시지 큐 사용 예

◈ 메시지 큐를 이용한 예제 프로그램 계속

```
/* receivemq.c */
/* message queue example */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <string.h>

#define BUFSIZE 16
#define QKEY    (key_t)0111

struct msgq_data {
        long type;
        char text[BUFSIZE];
} ;
struct msgq_data recv_data;
```

```
main()
{
        int qid, len;

        if ((qid = msgget(QKEY, IPC_CREAT | 0666))
== -1) {
                perror ("msgget failed");
                exit (1);
        }
        if ((len=msgrcv(qid, &recv_data, BUFSIZE, 0,
0)) == -1) {
                perror ("msgrcv failed");
                exit (1);
        }
        printf("received from message queue: %s₩n",
recv_data.text);
        if (msgctl(qid, IPC_RMID, 0) == -1) {
                perror ("msgctl failed");
                exit (1);
        }
}
```

메시지 큐

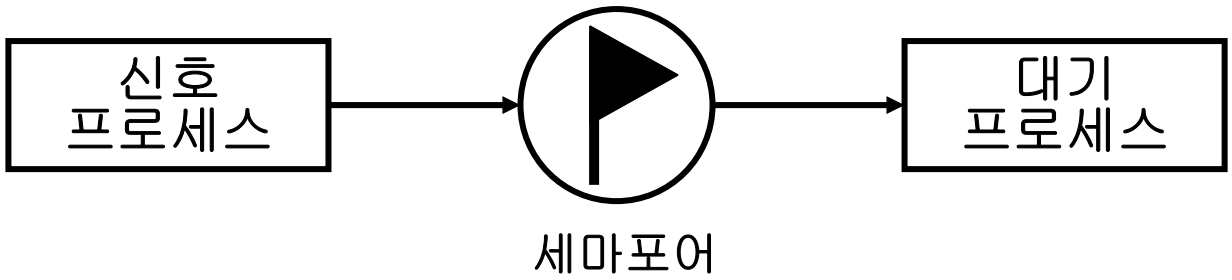
▣ 메시지 큐 사용 예

◈ 메시지 큐를 이용한 예제 프로그램 실행 결과

```
[cprog2@seps5 ipcs]$ gcc sendmq.c -o sendmq
[cprog2@seps5 ipcs]$ gcc receivemq.c -o receivemq
[cprog2@seps5 ipcs]$ ./sendmq
[cprog2@seps5 ipcs]$ ./receivemq
received from message queue: Hello, world
[cprog2@seps5 ipcs]$
```

세마포어

- 세마포어
  - 프로세스 간 효율적인 동기화 방식
  - 열쇠와 비슷하여, 어떤 프로세스가 세마포어를 획득하면 공유 자원에 접근하거나 동작을 수행하도록 허용하고, 그렇지 않으면 세마포어가 해제되기를 기다리며 대기
  - 기능
    - 프로세스들 간의 공유 자원 접근 제공 (상호 배제 요구 조건 만족)
    - 파이프나 메시지 큐는 대규모 자료 전송에는 적합하지만 자원 낭비가 크므로 동기화 수단으로는 적합하지 않음



세마포어

▣ 세마포어의 구현

- ◈ 두 가지 연산을 가지는 어떤 정수값
- ◈ 데이터 구조

```
unsigned short  semval;   /* 세마포어 값 */
unsigned short  semzcnt;  /* 0 이 되기를 기다리는 프로세스의 수 */
unsigned short  semncnt;  /* 증가하기를 기다리는 프로세스의 수 */
pid_t           sempid;   /* 최근 연산을 수행한 프로세스 ID */
```

세마포어

◈ 세마포어의 연산

wait(S) 또는 **P(S)** :
세마포어의 값이 영이 아니면,
세마포어의 값을 일만큼 감소시킨다;
그렇지 않으면,
세마포어의 값이 영이 아닐 때까지 기다린다.
그런 다음 세마포어의 값을 일만큼 감소시킨다;

signal(S) 또는 **V(S)** :
세마포어의 값을 일만큼 증가시킨다;
대기하고 있는 프로세스가 있으면,
대기 리스트로부터 한 프로세스를 깨운다;

◈ 세마포어 사용 예

- 프로세스 간 동기화
- 프로세스 간 공유 자원 접근

```
p (S);
 공유 자원을 사용하여 필요한 일을 수행한다;
v (S);
```

세마포어

semget 시스템 호출

기능

- 세마포어 집합 생성

사용법

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

int semget(key_t key, int nsems, int permflags);
```

- **key :** 생성할 세마포어 집합에 대한 고유 번호
  - 이 값을 통해 프로세스들이 세마포어 공유 가능
  - **IPC_PRIVATE**을 사용 - 다른 프로세스에서 생성한 **key** 값과 중복되지 않는 유일한 세마포어 집합 생성
- **nsems :** 세마포어 집합에서 생성할 세마포어 개수
- **permflags :** 생성 시 세마포어 사용 허가권 지정

반환값

- 성공적인 호출에 대하여 세마포어 집합 식별자 반환
- 오류시에는 **-1** 이 반환되고**, errno** 가 적절히 설정

세마포어

semop 시스템 호출

기능

- 세마포어 연산

사용법

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

int semop(int semid, struct sembuf *sops, size_t nsops);
```

- **semid** 가 가리키는 세마포어 집합 중에서 **sops** 가 가리키는 구조체 배열에서 **nsops** 만큼 지정된 것들에 대하여 세마포어 연산을 수행
- **sops : sembuf** 구조체의 주소

```
struct sembuf {
    unsigned short sem_num;  /* 세마포어 번호 */
    short sem_op;            /* 세마포어 연산 */
    short sem_flg;           /* 연산 플래그 */
    .... };
```

- **semflg : IPC_NOWAIT** 또는 **SEM_UNDO**
  - **IPC_NOWAIT** 가 설정되어 있으면, 즉시 반환
  - **SEM_UNDO :** 프로세스가 종료**(exit)** 할 때 이 연산이 수행되지 않음

세마포어

semop 시스템 호출

sem_op : 수행하는 연산

- 양수 : "세마포어 값을 증가시키는 연산" 수행.
  - 이 값을 **semval** 에 더함. 이 연산은 항상 진행되며, 세마포어 값이 증가하기를 기다리는 프로세스들을 깨움.
- **0 : "0을 기다리는 연산" 수행.**
  - **semval** 이 **0**이면 그대로 진행
  - 그렇지 않고 **IPC_NOWAIT** 플래그가 설정되어 있으면 즉시 오류값 반환
  - 그렇지 않으면, **semzcnt** 가 하나 증가하고, **semval** 이 **0**이 될 때까지 대기
- 음수 : "세마포어 값을 감소시키는 연산" 수행.
  - **semval** 값이 **sem_op** 의 절대값보다 크거나 같으면, 연산은 즉시 진행되어 **semval** 은 **sem_op** 의 절대값만큼 감소
  - **semval** 값이 **sem_op** 의 절대값보다 작고 **IPC_NOWAIT** 가 설정되어 있으면, 즉시 반환되고 오류값은 **EAGAIN**
  - 그렇지 않으면 **semncnt** 값이 하나 증가하고, 세마포어 값이 커질 때까지 프로세스가 대기

세마포어

semctl 시스템 호출

기능

- 세마포어 집합 제어 (세마포어의 정보를 얻거나, 변경하거나, 삭제)

사용법

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

int semctl(int semid, int sem_num, int cmd, union semun arg);
```

- **semid**가 가리키는 세마포어 집합 중 **sem_num** 이 지정하는 세마포어에 대하여 **cmd** 가 지정한 동작을 수행
- **arg :** 다음 유니온으로 지정

```
union semun {
    int val;                    /* SETVAL 을 위한 값 */
    struct semid_ds *buf;       /* IPC_STAT, IPC_SET 을 위한 버퍼 */
    unsigned short int *array;  /* GETALL, SETALL 을 위한 배열 */
    struct seminfo *__buf;      /* IPC_INFO 를 위한 버퍼 */
};
```

- **semid_ds** 구조체 : 세마포어 집합의 정보를 가짐. 다음 요소 포함

```
struct ipc_perm sem_perm;
time_t sem_otime;               /* 최근 연산 시간 */
time_t sem_ctime;               /* 최근 변경 시간 */
ushort sem_nsems;               /* 세마포어의 수 */
```

세마포어

semctl 시스템 호출

- **cmd : 수행할 명령**
  - **IPC_STAT : semid** 와 관련된 세마포어 집합 정보를 **arg.buf** 의 위치에 저장
  - **IPC_SET :** 세마포어 집합의 정보를 **arg.buf** 가 가리키는 값으로 변경, 변경가능한 값은 **sem_perm.uid, sem_perm.gid, sem_perm.mode**
  - **IPC_RMID :** 세마포어 집합과 관련된 데이터 구조를 즉시 삭제
  - **GETVAL :** 세마포어 집합에서 **sem_num** 번째 세마포어 값 반환
  - **SETVAL :** 세마포어 집합에서 **sem_num** 번째 세마포어 값을 **arg.val** 로 설정
  - **GETPID :** 세마포어 집합에서 **sem_num** 번째 세마포어의 **sempid** 값 반환
  - **GETNCNT :** 세마포어 집합에서 **sem_num** 번째 세마포어의 **semncnt** 값 반환
  - **GETZCNT :** 세마포어 집합에서 **sem_num** 번째 세마포어의 **semzcnt** 값 반환
  - **GETALL :** 세마포어 집합의 모든 세마포어의 값을 **arg.array** 에 저장
  - **SETALL :** 세마포어 집합의 모든 세마포어의 값을 **arg.array** 으로 설정

세마포어

▣ 세마포어 사용 예

◈ 세마포어를 이용한 예제 프로그램

```
/* testsem.c */
/* semaphore example */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
#include <string.h>

#define SEMKEY  (key_t)0111
void testsem(int semid);
void p(int semid);
void v(int semid);

main()
{
        int semid, i;
        union semun {
                int value;
                struct semid_ds *buf;
                unsigned short int *array;
        } arg;
        if ((semid = semget(SEMKEY, 1, IPC_CREAT |
0666)) == -1) {
                perror ("semget failed");
                exit (1);
        }
        arg.value = 1;
        if (semctl(semid, 0, SETVAL, arg) == -1) {
                perror ("semctl failed");
                exit (1);
        }
        for (i = 0; i < 3; i++) {
                if (!fork())
                        testsem(semid);
        }
        sleep(10);
        if (semctl(semid, 0, IPC_RMID, arg) == -1) {
                perror ("semctl failed");
                exit (1);
        }
}
```

세마포어

▣ 세마포어 사용 예

◈ 세마포어를 이용한 예제 프로그램 계속

```
void testsem (int semid)
{
        srand((unsigned int) getpid());
        p(semid);
        printf("process %d : semaphore in use₩n",
getpid());
        sleep(rand()%5);
        printf("process %d : putting
semaphore₩n", getpid());
        v(semid);
        exit(0);
}

void p (int semid)
{
        struct sembuf pbuf;
        pbuf.sem_num = 0;
        pbuf.sem_op = -1;
        pbuf.sem_flg = SEM_UNDO;
        if (semop (semid, &pbuf, 1) == -1) {
                perror ("semop failed");
                exit (1);
        }
}

void v (int semid)
{
        struct sembuf vbuf;
        vbuf.sem_num = 0;
        vbuf.sem_op = 1;
        vbuf.sem_flg = SEM_UNDO;
        if (semop (semid, &vbuf, 1) == -1) {
                perror ("semop failed");
                exit (1);
        }
}
```

세마포어

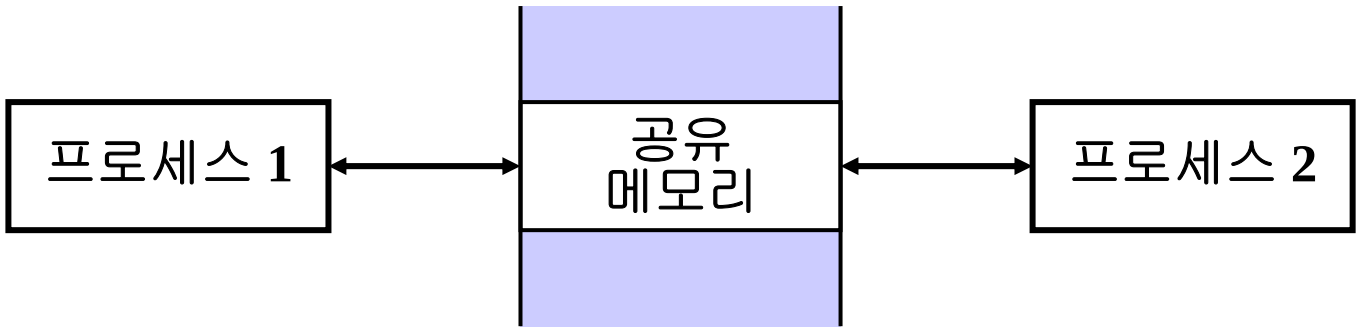
▣ 세마포어 사용 예

◈ 세마포어를 이용한 예제 프로그램 실행 결과

```
[cprog2@seps5 ipcs]$ gcc -o testsem testsem.c
[cprog2@seps5 ipcs]$ ./testsem
process 8107 : semaphore in use
process 8107 : putting semaphore
process 8108 : semaphore in use
process 8108 : putting semaphore
process 8109 : semaphore in use
process 8109 : putting semaphore
[cprog2@seps5 ipcs]$
```

공유 메모리

- 공유 메모리
  - 둘 이상의 프로세스가 특정 메모리 영역을 공유하여 자료에 접근
  - 세 가지 고급 **IPC** 기법 중에서 가장 유용
  - 기능
    - 프로세스들 간의 자료 공유
- 공유 메모리 사용
  - **IPC** 공유 메모리 영역 **(shared memory region)** 에 대한 자료 구조
  - 프로세스 접근을 위해 프로세스 주소 공간에 공유 메모리 영역 추가
  - 사용이 끝나면 주소 공간에서 공유 메모리를 제거



공유 메모리

shmget 시스템 호출

기능

- 공유 메모리 생성

사용법

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

int shmget(key_t key, size_t size, int permflags);
```

- **key :** 생성할 공유 메모리 영역에 대한 고유 번호
  - 이 값을 통해 프로세스들이 공유 메모리 영역을 공유 가능
  - **IPC_PRIVATE**을 사용 - 다른 프로세스가 생성한 **key** 값과 중복되지 않는 유일한 공유 메모리 영역 생성
- **size :** 생성하는 공유 메모리 영역의 최소 바이트 크기를 지정
- **permflags :** 생성 시 공유 메모리 영역 사용 허가권 지정
  - 공유 메모리 영역에 쓰기 위해서는 쓰기 허가권 설정
  - 공유 메모리 영역을 읽기 위해서는 읽기 허가권 설정

반환값

- 성공적인 호출에 대하여 공유 메모리 영역 식별자 반환
- 오류시에는 **-1** 이 반환되고, **errno** 가 적절히 설정

공유 메모리

shmat, shmdt 시스템 호출

기능

- 공유 메모리 연산

사용법

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

int *shmat(int shmid, const void *shmaddr, int shmflg);
int shmdt(const void *shmaddr);
```

- **shmat**는 **shmid**가 가리키는 메모리 영역을 호출 프로세스 주소공간에 붙임
- **shmdt**는 **shmaddr**의 메모리 영역을 호출 프로세스 주소공간으로부터 떼어냄
- **shmaddr :** 프로세스 주소공간에 붙이거나 떼어낼 공유 메모리 영역의 주소
  - **NULL** 일 때 시스템이 공유 메모리 영역이 부착될 적절한 주소를 선택
- **shmflg**
  - **SHM_RND : shmaddr** 의 값에 가장 가까운 주소에 공유 메모리 영역이 부착
  - **SHM_RDONLY :** 공유 메모리 영역은 해당 프로세스에 대해 읽기만을 허용

반환값

- **shmat** 는 성공적인 호출에 대하여 부착된 공유 메모리 주소를 반환
- **shmdt** 는 성공적인 호출에 대하여 **0**을 반환
- 오류시에는 **-1** 이 반환되고, **errno** 가 적절히 설정된다.

공유 메모리

shmctl 시스템 호출

기능

- 공유 메모리 제어 (공유 메모리 영역의 정보를 얻거나, 변경하거나, 삭제)

사용법

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

int shmctl(int mqid, int cmd, struct shmid_ds *buf);
```

- **shmid**가 가리키는 메모리 영역에 **cmd** 가 지정한 동작을 수행
- **cmd :** 수행할 명령
  - **IPC_STAT : shmid** 와 관련된 공유 메모리 영역의 정보를 **buf** 의 위치에 저장
  - **IPC_SET :** 공유 메모리 영역의 정보를 **buf** 가 가리키는 값으로 변경, 변경 가능한 값은 **shm_perm** 의 멤버들과 **shm_ctime**
  - **IPC_RMID :** 공유 메모리 영역과 관련된 데이터 구조를 즉시 삭제
- **buf :** 공유 메모리 객체 데이터 구조의 주소

```
struct ipc_perm shm_perm;  /* 연산 허가권 */
int shm_segsz;             /* 영역의 크기 ( 바이트 수 ) */
time_t shm_atime;          /* 최근 부착 시간 */
time_t shm_dtime;          /* 최근 떼어낸 시간 */
time_t shm_ctime;          /* 최근 변경 시간 */
unsigned short shm_cpid;   /* 생성자의 pid */
unsigned short shm_lpid;   /* 최근 연산자의 pid */
short shm_nattch;          /* 현재 부착 횟수 */
```

공유 메모리

▣ 공유 메모리 사용 예

◈ 공유 메모리를 이용한 예제 프로그램

```
/* writeshm.c */
/* shared memory example */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <string.h>

#define SHMSIZE 1024
#define SHMKEY  (key_t)0111

main()
{
        int shmid, len;
        void *shmaddr;
        if ((shmid = shmget(SHMKEY, SHMSIZE,IPC_CREAT|0666)) == -1) {
                perror ("shmget failed");
                exit (1);
        }
        if ((shmaddr = shmat(shmid, NULL, 0)) == (void *)-1) {
                perror ("shmat failed");
                exit (1);
        }
        strcpy((char *)shmaddr, "Hello, world");
        if (shmdt(shmaddr) == -1) {
                perror ("shmdt failed");
                exit (1);
        }
}
```

공유 메모리

▣ 공유 메모리 사용 예

◈ 공유 메모리를 이용한 예제 프로그램 계속

```
/* readshm.c */
/* shared memory example */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <string.h>

#define SHMSIZE 1024
#define SHMKEY  (key_t)0111

main()
{
        int shmid, len;
        void *shmaddr;

        if ((shmid = shmget(SHMKEY,
SHMSIZE,IPC_CREAT|0666)) == -1) {
                perror ("shmget failed");
                exit (1);
        }
        if ((shmaddr=shmat(shmid, NULL, 0)) ==
(void *)-1) {
                perror ("shmat failed");
                exit (1);
        }

        printf("received from shared
memory: %s₩n",(char *)shmaddr);

        if (shmdt(shmaddr) == -1) {
                perror ("shmdt failed");
                exit (1);
        }

        if (shmctl(shmid, IPC_RMID, 0) == -1) {
                perror ("shmctl failed");
                exit (1);
        }
}
```

공유 메모리

▣ 공유 메모리 사용 예

◈ 공유 메모리를 이용한 예제 프로그램 실행 결과

```
[cprog2@seps5 ipcs]$ gcc writeshm.c -o writeshm
[cprog2@seps5 ipcs]$ gcc readshm.c -o readshm
[cprog2@seps5 ipcs]$ ./writeshm
[cprog2@seps5 ipcs]$ ./readshm
received from shared memory: Hello, world
[cprog2@seps5 ipcs]$
```