6장. 고급 프로세스간 통신

목 차

- Record locking
- Advanced Inter Process Communication
  - Message Passing
  - Semaphore
  - Shared Memory

Record Locking

- 현재 사용중인 파일의 일부를 잠금
- Atomic operation
- 종류
  - 읽기 록
    - 여러 프로세스들이 같은 구역에 동시에 설정 가능(쓰기 록 적용 불가)
  - 쓰기 록
    - 다른 프로세스들의 읽기나 쓰기 록을 불허
- fcntl 을 사용

#include <fcntl.h>

*int fcntl(int filedes, int cmd, struct flock *ldata);*

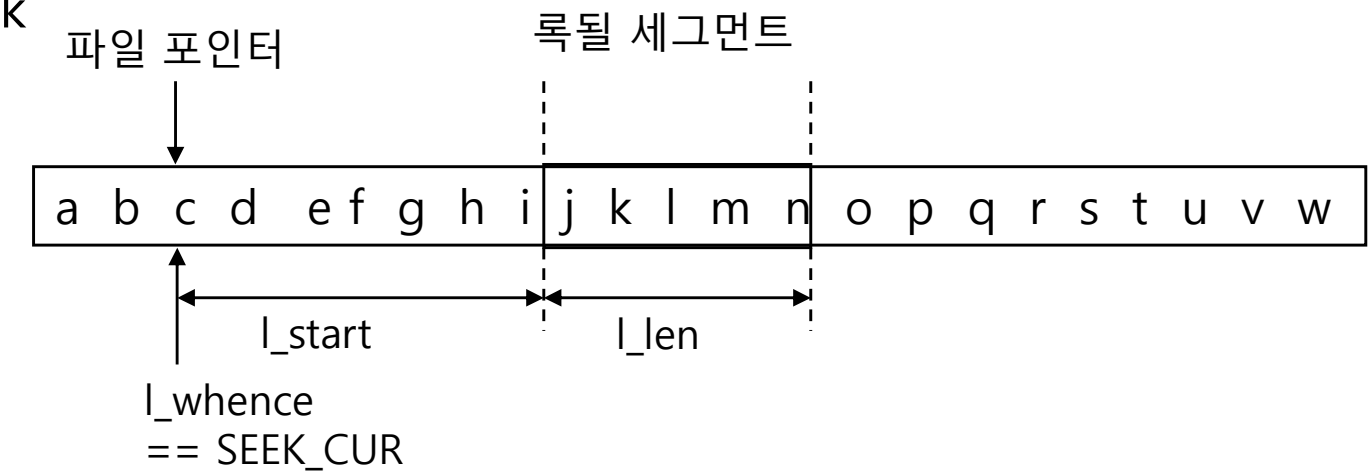
- cmd:
  - F_GETLK: ldata 를 통해 록 정보를 획득
  - F_SETLK: 록을 적용하고 즉시 리턴, 록 제거에도 사용
  - F_SETLKW: 록 적용, 타 프로세스에 의해 봉쇄되면 sleep

Record Locking

- struct flock:

```
short l_type;      /* 록의 유형 */
short l_whence;    /* 변위 유형 */
off_t l_start;     /* offset */
off_t l_len;       /* length */
pid_t l_pid;       /* F_GETLK 인 경우에 locking process의 pid값을 저장
   */
```

- 록의 유형
  - F_RDLCK: read lock
  - F_WRLCK: write lock
  - F_UNLCK: unlock



fcntl을 이용한 locking

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

struct flock my_lock;

my_lock.l_type = F_WRLCK;
my_lock.l_whence = SEEK_CUR;
my_lock.l_start = 0;
my_lock.l_len = 512;

fcntl(fd, F_SETLKW, &my_lock);
```

lockit 프로그램

```
/* lockit -- fcntl 록킹의 예시 */
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>

main()
{
int fd;
struct flock my_lock;

/* 쓰기 록의 인수를 지정 */
my_lock.l_type = F_WRLCK;
my_lock.l_whence = SEEK_SET;
my_lock.l_start = 0;
my_lock.l_len = 10;

/* 화일을 개방한다. */
fd = open ("locktest", O_RDWR);

/* 처음 10바이트를 록한다 */
if ( fcntl (fd, F_SETLKW, &my_lock) == -1)
{
        perror ("parent: locking");
        exit (1);
}
```

```
printf ("parent: locked record₩n");


switch (fork()){
case -1:                        /* 오류 */
        perror ("fork");
        exit (1);
case 0:                          /* 자식 */
        my_lock.l_len = 5;
        if ( fcntl (fd, F_SETLKW, &my_lock) == -1)
        {
                perror ("child: locking");
                exit (1);
        }
        printf ("child: locked₩n");
        printf ("child: exiting₩n");
        exit(0);
}

sleep(5);

/* 이제 퇴장(exit)한다. 따라서 록이 해제된다 */
printf ("parent: exiting₩n");
exit (0);
}
```

```
% lockit
parent : locked record
parent : exiting
child : locked
child : exiting
```

fcntl로 lock 해제

```
/* 이제는 부모가 퇴장하기 전에 록을 해제한다. */
printf ("parent: unlocking₩n");
my_lock.l_type = F_UNLCK;
if( fcntl (fd, F_SETLK, &my_lock) == -1)
{
        perror ("parent: unlocking");
        exit (1);
}
```

ACME 항공사 문제

```
/* acmebook 갱신 루틴의 골격 */
struct flock db_lock;
.
.
/* 록을 위한 인수들을 설정한다. */
db_lock.l_type = F_WRLCK;
db_lock.l_whence = SEEK_SET;
db_lock.l_start = recstart;
db_lock.l_len = RECSIZE;
.
.
/*  레코드를 록하라. 레코드가 이미 잠겨 있으면 여기서 수면할 것임. */
if ( fcntl (fd, F_SETLKW, &db_lock) == -1)
fatal("lock failed");

/* 예약 자료를 검사하고 갱신하는 코드 */
.
.
/* 이제 레코드를 자유화하여 다른 프로세스들이 사용할 수 있게 한다. */
db_lock.l_type = F_UNLCK;
fcntl (fd, F_SETLK, &db_lock);
```

lock에 대한 테스트

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

.
.
.

if ( fcntl (fd, F_SETLK, &alock) == -1)
{
if (errno == EACCES ||errno == EAGAIN)
{
        fcntl (fd, F_GETLK, &b_lock);
        fprintf (stderr, "record locked by %d₩n", b_lock.l_pid);
}
else
        perror ("unexpected lock error");
}
```

교착상태(deadlock)

```
/* deadlock -- 교착 상태 오류를 시연한다. */
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
main()  {
int fd;
struct flock first_lock;
struct flock second_lock;

first_lock.l_type = F_WRLCK;
first_lock.l_whence = SEEK_SET;
first_lock.l_start = 0;
first_lock.l_len = 10;

second_lock.l_type = F_WRLCK;
second_lock.l_whence = SEEK_SET;
second_lock.l_start = 10;
second_lock.l_len = 5;

fd = open ("locktest", 0_RDWR);
if ( fcntl (fd, F_SETLKW, &first_lock) == -1)        /* A */
        fatal ("A");
printf ("A: lock succeeded (proc %d)₩n", getpid());
switch (fork()){
case -1:
        /* 오류 */
        fatal ("error on fork");
```

```
case 0:
        /* 자식 */
        if ( fcntl (fd, F_SETLKW, &second_lock) == -1)  /* B
        */
                fatal ("B");
        printf ("B: lock succeeded (proc %d)₩n", getpid());
        if ( fcntl (fd, F_SETLKW, &first_lock) == -1)   /* C */
                fatal ("C");
        printf ("C: lock succeeded (proc %d)₩n", getpid());
        exit (0);
default:
        /* 부모 */
        printf ("parent sleeping₩n");
        sleep (10);
        if ( fcntl (fd, F_SETLKW, &second_lock) == -1)   /* D
        */
                fatal ("D");
        printf ("D: lock succeeded (proc %d)₩n", getpid());
}
}
```

Advanced IPC: 기본 개념

- 종류
  - message passing
  - semaphores
  - shared memory
- IPC 설비 키 (facility key)
  - numbers used to identify an IPC object like filenames
  - data type: key_t
  - ftok: 파일 이름을 키로 변환해 주는 함수

    ```
    #include <sys/ipc.h>
    key_t ftok(const char *path, int id);
    ```
- IPC get 연산
  - IPC 객체의 생성
  - 기존의 IPC 객체에 접근
  - create 또는 open 과 유사
  - IPC facility identifier: get 연산의 결과로 획득, fd와 유사하지만 unique 함.
  - msgget, semget, shmget

Advanced IPC: 기본 개념

- Other IPC operations
  - 제어 연산: msgctl, semctl, shmctl
  - specific operations
    - msgsnd, msgrcv …
- Status data structures
  - IPC 객체가 생성될 때 만들어짐
  - 각 IPC facility 마다 하나씩 존재
  - 공통적인 허가 구조: ipc_perm
    - uid_t cuid; /* user-id of creator of IPC object */
    - gid_t cgid; /* group-id of creator */
    - uid_t uid; /* effective user-id */
    - gid_t gid; /* effective group-id */
    - mode_t umode; /* permissions */;

Message Passing

- 메시지: 문자나 바이트의 열
- *message queue*
  - msgget을 통해 만들어짐
  - 메시지 전송의 통로
- msgget

  #include <sys/msg.h>

  int msgget(key_t key, int permflags);
  - return value: message queue identifier
  - key : 메시지 큐를 식별하는 단순한 숫자
  - permflags
    - IPC_CREAT
      - 메시지 큐를 생성
      - 기존의 메시지 큐가 있는 경우 덮어 쓰지 않음
    - IPC_EXCL
      - 단지 한 개의 메시지 큐를 생성 (IPC_CREAT와 동시에 설정된 경우)
      - 큐가 존재하는 경우는 실패, -1을 리턴
  - mqid = msgget((key_t)0100, 0644|IPC_CREATE|IPC_EXCL);

Message Passing

#include <sys/msg.h>

int msgsnd(int mqid, const void *message, size_t size, int flags);

int msgrcv(int mqid, void *message, size_t size, long msg_type, int flags);

- msgsnd: mqid가 가리키는 큐에 메시지를 추가
  - message template

```
struct mymsg{
long mtype;                    /* message type */
char mtext[SOMEVALUE]          /* message text */
};
```

- msgrcv: mqid가 가리키는 큐에서 메시지를 읽음. 이 메시지는 큐에서 제거됨
  - msg_type: mtype 필드에 따라 어떤 메시지를 받아들일지 결정
    - 0: 큐의 첫번째 메시지, 0 < : msg_type과 같은 메시지, 0 > : 가장 작은 msg_type을 갖는 메시지
  - flasgs:
    - IPC_NOWAIT: non-blocking
    - MSG_NOERROR: recv시 크기가 크면 절단

Message Passing: a queue with priorities

```
/* q.h -- 메시지 설비 예를 위한 헤더 */


#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <errno.h>


#define QKEY          (key_t)0105    /* 큐의 키를 식별한다
      */
#define QPERM          0660           /* 큐의 허가 */
#define MAXOBN          50             /* 개체 이름의 최대
      길이 */
#define MAXPRIOR       10            /* 최대 우선 순위 수
      준 */


struct q_entry {
long mtype;
char mtext [MAXOBN+1];
};
```

```
/* enter -- 한 객체를 큐에 넣는다. */
#include "q.h"

int enter (char *objname, int priority)
{

int len, s_qid;
struct q_entry s_entry;       /* 메시지를 저장할 구조 */
/* 이름의 길이, 우선순위 수준을 확인한다. */

if ( (len = strlen(objname)) >  MAXOBN)
{
        warn  ("name too long");
        return (-1);
}

if (priority > MAXPRIOR || priority < 0)
{
        warn ("invalid priority level");
        return (-1);
}
```

Message Passing: a queue with priorities

```
/* 필요에 따라 메시지 큐를 초기화한다. */

if ( (s_qid = init_queue()) == -1)
        return (-1);

/* s_entry를 초기화한다. */
s_entry.mtype = (long) priority;
strncpy (s_entry.mtext, objname, MAXOBN);


/* 메시지를 보내고, 필요할 경우 기다린다. */


if (msgsnd (s_qid, &s_entry, len, 0) == -1)
{
        perror ("msgsnd failed");
        return (-1);
}
else
        return (0);
}
```

```
#include <stdio.h>

int warn (char *s)
{
fprintf (stderr, "warning: %s₩n", s);
}



/* init_queue -- 큐 식별자를 획득한다. */

#include "q.h"


int init_queue(void)
{
int queue_id;


/* 메시지 큐를 생성하거나 개방하려고 시도한다 */
if ( (queue_id = msgget (QKEY, IPC_CREAT | QPERM)) == -
        1)
         perror("msgget failed");


return (queue_id);
}
```

Message Passing: a queue with priorities

```
/* serve -- 큐에서 가장 우선 순위가 높은 객체를 처리한다.
       */

#include "q.h"

int serve (void)
{
int mlen, r_qid;
struct q_entry r_entry;

/* 필요에 따라 메시지 큐를 초기화한다. */

if ((r_qid = init_queue()) == -1)
        return (-1);
```

```
/* 다음 메시지를 가져와 처리한다. 필요하면 기다린다. */

for (;;)
{
        if ((mlen = msgrcv (r_qid, &r_entry, MAXOBN, (-1 *
       MAXPRIOR), MSG_NOERROR))== -1)
        {
                perror ("msgrcv failed");
                return (-1);
        }
          else
        {
                /* 우리가 문자열을 가지고 있는지 확인한다. */
                r_entry.mtext[mlen]='₩0';


                /* 객체 이름을 처리한다. */
                proc_obj (&r_entry);
        }
}

}
```

Message Passing: a queue with priorities

```
/* etest -- 큐에 객체 이름을 넣는다. */
#include <stdio.h>
#include <stdlib.h>
#include "q.h"

main (int argc, char **argv)
{
int priority;
if (argc != 3)
{
        fprintf (stderr, "usage: %s objname
       priority₩n", argv[0]);
        exit (1);
}
if ((priority = atoi(arvg[2])) <= 0 || priority > MAXPRIOR)
{
        warn ("invalid priority");
        exit (2);
}
if (enter (argv[1], priority) < 0)
{
        warn ("enter failure");
        exit (3);
}
exit (0);
}
```

```
/* stest -- 큐를 위한 단순한 서버 */
#include <stdio.h>
#include "q.h"
main()
{
pid_pid;

switch (pid = fork()){
case 0:                 /* 자식 */
        serve();
        break;          /* 실제로는, 서버는 결코 퇴장(exit)하지
       않음 */
case -1:
        warn ("fork to start server failed");
        break;
default:
        printf ("server process pid is %d₩n", pid);
}
exit( pid != -1 ? 0 : 1);
}

int proc_obj (struct q_entry *msg)
{
printf ("₩npriority: %ld name: %s₩n", msg>mtype, msg-
        >mtext);
}
```

Message Passing: msgctl

```
#include <sys/msg.h>
int msgctl(int mqid, int command, struct msqid_ds *msq_stat);
```

- 목적
  - 메시지 큐의 상태정보 획득
  - 메시지 큐에 관련된 제한을 변경
  - 큐의 제거
- *msqid_ds*
  - msg_perm, msg_qnum, msg_qbytes, lsg_lspid, msg_lrpid, msg_stime, msg_rtime, msg_ctime
- *command*
  - IPC_STAT: 메시지 큐의 상태정보를 msq_stat에 저장
  - IPC_SET: 메시지 큐에 대한 제어변수들의 값을 지정
  - IPC_RMID: 메시지 큐의 삭제

Message Passing: msgctl의 예

```
/* showmsg -- 메시지 큐의 자세한 정보를 보인다. */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <time.h>
void mqstat_print (key_t, int, struct msqid_ds *);
main( int argc, char **argv)
{
key_t mkey;
int msq_id;
struct msqid_ds msq_status;
if (argc != 2)
{
        fprintf (stderr, "usage: showmsg keyval₩n");
        exit (1);
}
/* 메시지 큐 식별자를 얻는다. */
mkey = (key_t) atoi (argv[1]);
if (( msq_id = msgget(mkey, 0)) == -1)
{
        perror( "msgget failed");
        exit (2);
}
```

```
/* 상태 정보를 얻는다. */
if (msgctl(msq_id, IPC_STAT, &msq_status) == -1)
{
        perror ("msgctl failed");
        exit (3);
}

/* 상태 정보를 프린트한다. */
mqstat_print (mkey, msq_id, &msq_status);
exit (0);
}

void mqstat_print (key_t mkey, int mqid, struct msqid_ds
        *mstat)
{
printf ("₩nKey %d, msg_qid %d₩n₩n", mkey, mqid);
printf ("%d message(s) on queue₩n₩n", mstat-
        >msg_qnum);
printf ("Last send by proc %d at %s₩n", mstat->msg_lspid,
        ctime(&(mstat->msg_stime)));
printf ("Last recv by proc %d at %s₩n", mstat->msg_lrpid,
        ctime(& (mstat->msg_rtime)));
}
}
```

Semaphore

- *sem*: integer variable
  - operations:

    p(sem) or wait(sem)

    if (sem != 0)

    sem--

    else

    wait until sem becomes non-zero, then decrement

    v(sem) or signal(sem)

    sem++

    if (queue of waiting processes not empty)

    restart first process in wait queue
- critical section

  p(sem);

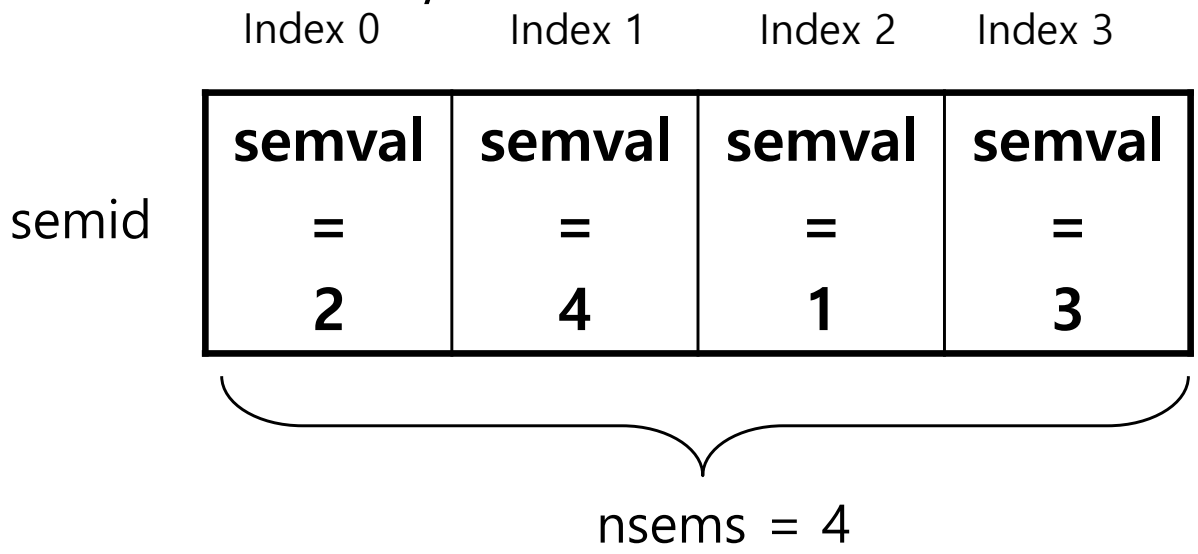
  something interesting …

  v(sem);

semget

```
#include <sys/sem.h>
int semget(key_t key, int nsems, int permflags);
```

- return value: *semaphore set identifier*



semctl

```
#include <sys/sem.h>
int semctl(int semid, int sem_num, int command,
            union semun ctl_arg);
```

- command
  - 표준 IPC 기능
  - 단일 세마포 연산
  - 전체 세마포 연산
- sem_num: 특정 세마포를 식별하기 위한 값
- ctl_arg

```
union semun{
    int val;
    struct semid_ds *buf;
    unsigned short *array;
};
```

semctl

- Command
  - 표준 IPC 기능
    - IPC_STAT: 상태정보를 ctl_arg.stat에 저장한다.
    - IPC_SET: ctl_arg.stat에 저장된 형태로 소유권과 허가를 지정한다.
    - IPC_RMID: 시스템에서 해당 semaphore의 집합을 삭제한다.
  - 단일 세마포 연산
    - GETVAL: semaphore 값 semval을 return한다.
    - SETVAL: semaphore 값을 ctl_arg.val로 지정한다.
    - GETPID: sempid(semaphore에 가장 최근에 접근한 pid) 값을 return한다.
    - GETNCNT: semncnt(semaphore 값이 현재의 값보다 더 큰 값을 갖기를 기다리는 프로세스의 수)를 return한다.
    - GETZCNT: semxcnt(semaphore의 값이 0이 되기를 기다리는 프로세스의 수)를 retrun한다.
  - 전체 세마포 연산
    - GETALL: 모든 semvals의 값을 ctl_arg.array로 저장한다.
    - SETALL: ctl_arg.array의 값을 이용하여 모든 semvals 값을 지정한다.

semctl의 예

```
/* initsem -- 세마포 초기화 */

#include "pv.h"

int initsem (key_t semkey)
{
int status = 0, semid;

if ((semid = semget (semkey, 1, SEMPERM | IPC_CREAT |
       IPC_EXCL)) == -1)
{
        if (errno == EEXIST)
                semid = semget (semkey, 1, 0);
}
else            /* 만일 생성되었으면 ... */
{
        semun arg;
        arg.val = 1;
        status = semctl(semid, 0, SETVAL, arg);
}
if (semid == -1 || status == -1)
{       perror ("initsem failed");
        return (-1); }
/* 모든 것이 잘되었음. */
return (semid);
}
```

```
/* 세마포 예의 헤더 화일 */


#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>


#include <errno.h>


#define SEMPERM 0600
#define TRUE     1
#define FALSE    0
typedef union semun {
        int val;
        struct semid_ds *buf;
        ushort *array;
} semun;
```

semop

```
#include <sys/sem.h>
int semop(int semid, struct sembuf *op_array, size_t num_ops);
```

- num_ops: sembuf 구조의 개수
- sembuf
  - sem_num: 세마포 집합내의 index
  - sem_op: semop이 수행하는 기능을 나타냄
    - 음수일 때: p()
    - 양수일 때: v()
    - 0일 때: semval의 값이 0이 될 때까지 기다림
  - sem_flg
    - IPC_NOWAIT
    - SEM_UNDO

semaphore의 예

```
/* p.c -- 세마포 p 연산 */


#include "pv.h"


int p (int semid)
{
struct sembuf p_buf;


p_buf.sem_num = 0;
p_buf.sem_op = -1;
p_buf.sem_flg = SEM_UNDO;


if (semop(semid, &p_buf, 1) == -1)
{
        perror ("p(semid) failed");
        exit (1);
}
return (0);
}
```

```
/* v.c -- 세마포 v 연산 */


#include "pv.h"


int v (int semid)
{
struct sembuf v_buf;


v_buf.sem_num = 0;
v_buf.sem_op = 1;
v_buf.sem_flg = SEM_UNDO;


if (semop (semid, &v_buf, 1) == -1)
{
        perror ("v (semid) failed");
        exit (1);
}
return (0);
}
```

semaphore의 예

```
/* testsem -- 세마포 루틴을 테스트한다. */

#include "pv.h"

void handlesem (key_t skey);

main()
{
key_t semkey = 0x200;
int i;


for(i = 0; i < 3; i++) {
        if (fork() == 0)
                handlesem (semkey);
}
}

void handlesem (key_t skey)
{
int semid;
pid_t pid = getpid();

if ((semid = initsem(skey)) < 0)
         exit (1);
```

```
printf( "₩nprocess %d before critical section₩n", pid);

p(semid);

printf ("process %d in critical section₩n", pid);


/* 실제로는 무언가 흥미있는 일을 한다. */
sleep (10);


printf ("process %d leaving critical section₩n", pid);


v(semid);


printf ("process %d exiting₩n", pid);
exit (0);
}
```

semaphore의 예

수행시킨 결과

```
Process 799 before critical section
Process 799 in critical section
Process 800 before critical section
Process 801 before critical section
Process 799 leaving critical section
Process 801 in critical section
Process 799 exiting
Process 801 leaving critical section
Process 801 exiting
Process 800 in critical section
Process 800 leaving critical section
Process 800 exiting
```

Shared Memory

- 두개 이상의 프로세스가 물리적 메모리의 일부를 공유
- 세가지 IPC 기법 중에서 가장 효율적
- shmget → shmat → shmdt
- 공유 메모리의 생성: shmget

```
#include <sys/shm.h>
int shmget (key_t key, size_t size, int permflags);
```

- 공유 메모리 연산: shmat

```
#include <sys/shm.h>
void *shmat (int shmid, const void *daddr, int shmflags);
```

  - shmget에 의해 생성된 메모리 영역을 자신의 논리적 자료 공간에 부착시킴
  - daddr
    - if NULL, the segment is attached at the first available address
    - if not NULL, the segment is attached at, or near, the address held within it
  - shmflags: SHM_RDONLY, SHM_RND(페이지 경계에 맞춤)

Shared Memory

- 공유 메모리 연산: shmdt

retval = shmdt(memptr);

  - detaches a shared memory segment from the process' logical address space

- shmctl

#include <sys/shm.h>

int shmctl (int shmid, int command, struct shmid_ds *shm_stat);

  - *command*: IPC_STAT, IPC_SET, IPC_RMID

Shmcopy

```
/* 공유 메모리 예를 위한 헤더 화일 */
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>

#define SHMKEY1         (key_t) 0x10    /* 공유 메모리 키 */
#define SHMKEY2         (key_t) 0x15    /* 공유 메모리 키 */
#define SEMKEY          (key_t) 0x20    /* 세마포 키 */

/* 읽기와 쓰기를 위한 버퍼의 크기 */
#define SIZ 5*BUFSIZ

/* 데이터와 읽은 계수를 저장한다.  */
struct databuf {
int d_nread;
char d_buf[SIZ]; };

typedef union _semun {
        int val;
         struct semid_ds *buf;
         ushort *array; } semun;
```

```
/* 초기화 루틴들 */
#include "share_ex.h"

#define IFLAGS (IPC_CREAT |IPC_EXCL)
#define ERR     ((struct databuf *)-1)

static int shmid1, shmid2, semid;

void getseg(struct databuf **p1, struct databuf **p2)
{
/* 공유 메모리 영역을 생성한다 */
if ((shmid1 = shmget (SHMKEY1, sizeof(struct databuf),
        0600 |IFLAGS)) == -1)
           fatal ("shmget");
if ((shmid2 = shmget (SHMKEY2, sizeof(struct databuf),
        0600 |IFLAGS)) == -1)
          fatal ("shmget");

/* 공유 메모리 영역을 부착한다. */
if ((*p1 = (struct databuf *) shmat (shmid1, 0, 0)) == ERR)
         fatal ("shmat");
if ((*p2 = (struct databuf *) shmat (shmid2, 0, 0)) == ERR)
         fatal ("shmat");
}
```

Shmcopy

```
int getsem (void)  /* 세마포 집합을 얻는다 */
{
semun x;
x.val = 0;

/* 두 개의 세마포를 갖는 집합을 생성한다 */
if ((semid = semget (SEMKEY, 2, 0600|IFLAGS)) == -1)
        fatal ("semget");
/* 초기값을 지정한다. */
if (semctl (semid, 0, SETVAL, x) == -1)
        fatal ("semctl");
if (semctl (semid, 1, SETVAL, x) == -1)
        fatal ("semctl");
return (semid);
}

/* 공유 메모리 식별자와 세마포 집합 식별자를 제거한다. */
void remobj (void)
{
if (shmctl (shmid1, IPC_RMID, NULL) == -1)
        fatal ("shmctl");
if (shmctl (shmid2, IPC_RMID, NULL) == -1)
        fatal ("shmctl");
if (semctl (semid, IPC_RMID, NULL) == -1)
        fatal ("semctl"); }
```

```
/* shmcopy -- main 함수 */
#include "share_ex.h"
main()
{
int semid;
pid_t pid;
struct databuf *buf1, *buf2;

/* 세마포 집합을 초기화한다. */
semid = getsem();
/* 공유 메모리 영역을 생성하고 부착한다. */
getseg (&buf1, &buf2);

switch (pid = fork()){
case -1:
        fatal("fork");
case 0: /* 자식 */
        writer (semid, buf1, buf2);
        remobj();
        break;
default: /* 부모 */
        reader (semid, buf1, buf2);
        break;
}
exit(0);
}
```

Shmcopy

```
/* reader -- 화일 읽기를 처리한다. */
#include "share_ex.h"
/*이들은 두 세마포를 위해 p()와 v()를 정의한다.*/
struct sembuf p1={0,-1,0}, p2={1,-1,0};
struct sembuf v1={0, 1, 0}, v2={1, 1, 0};

void reader (int semid, struct databuf *buf1, struct
      databuf *buf2)
{
   for(;;) {
   /* 버퍼 buf1으로 읽어들인다. */
   buf1->d_nread = read(0, buf1->d_buf,SIZ);

   /* 동기화 지점 */
   semop (semid, &v1, 1);
   semop (semid, &p2, 1);

/* writer가 수면하는 것을 피하기 위해 검사한다.*/
   if (buf1->d_nread <=0) return;

   buf2->d_nread = read(0, buf2->d_buf,SIZ);

   semop(semid, &v1, 1);
   semop(semid, &p2, 1);

   if(buf2->d_nread <=0) return; }
}
```

```
/* writer -- 쓰기를 처리한다. */
#include "share_ex.h"

extern struct sembuf p1, p2; /* reader.c에 정의되어 있음.
      */
extern struct sembuf v1, v2; /* reader.c에 정의되어 있음. */

void writer (int semid, struct databuf *buf1, struct databuf
      *buf2)
{
  for(;;) {
    semop (semid, &p1, 1);
    semop (semid, &v2, 1);

    if (buf1->d_nread <= 0) return;

    write (1, buf1->d_buf, buf1->d_nread);

    semop (semid, &p1, 1);
    semop (semid, &v2, 1);

    if (buf2->d_nread <= 0) return;

    write (1, buf2->d_buf, buf2->d_nread); }

}
```

ipcs와 ipcrm

- ipcs
  - IPC 설비의 현재 상태정보를 출력

    IPC status from <running system> as of 2007년 11월 21일 수요일 오후 08시 09분 34초

    T ID KEY MODE OWNER GROUP

    Message Queues:

    Shared Memory:

    Semaphores:

    s 10 0x00000200 –ra------ keith users
- ipcrm
  - 시스템으로 부터 IPC 설비를 제거
  - Ipcrm –q *msgid*
  - ipcrm –m *shmid*
  - ipcrm -s *semid*