Advanced inter-process communications

Suntae Hwang

Kookmin University

레코드 록킹

- problem
  - 두개의 프로세스가 같은 파일을 참조하여 lseek, read, write 함수를 여러 번 병행 할 경우
    - 많은 문제를 표출
- 해결책
  - 파일을 잠글 수 있게 함
    - Record Rocking
- Kernel에 기반 한 레코드 록킹
  - fcntl 함수 사용

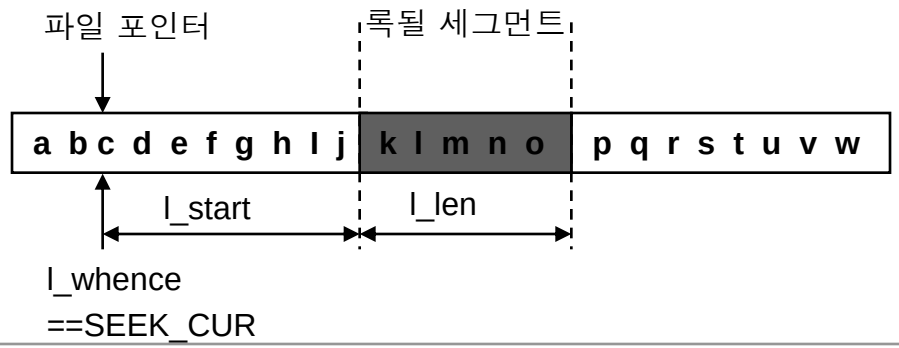
fcntl을 사용한 레코드 록킹

```
#include < fcntl.h>
int fcntl ( int fd,         // file descriptor
            int cmd,      // 동작 지정
            struct flock *ldata );    // cmd의 종류에 따른 추가 옵션
                                                                    return : 0 or –1
```

- cmd
  - F_GETLK : ldata 인수를 통해 전달된 데이터에 기반해서 록 정보를 얻음
  - F_SETLK : 파일에 록을 적용하고, 불가능하면 즉시 되돌아옴, 활동 중인 록 제거에도 사용
  - F_SETLKW : 파일에 록을 적용하고, 만약 다른 프로세스가 소유하는 이전의 록에 의해 봉쇄되면 Sleep

fcntl을 사용한 레코드 록킹(con't)

- ldata structure
  - 록 기술을 보관하는 구조체
  - 구성 요소
    - short l_type; /* 록의 유형을 기술
      - F_RDLCK : 적용된 록이 읽기 록이다.
      - F_WRLCK : 적용된 록이 쓰기 록이다.
      - F_UNLCK : 지정된 세그먼트에 대한 록을 제거해야 한다.
    - short l_whence; /* lseek와 마찬가지로 변위 유형임 */
      - SEEK_SET : 파일의 처음
      - SEEK_CUR : 읽기/쓰기 파일이 가리키는 현재의 위치
      - SEEK_END : 파일의 끝을 변위의 기준으로 함
    - off_t l_start /* 바이트로 표시된 변위 */
    - off_t l_len /* 바이트 단위의 세그먼트 크기 */
    - pid_t l_pid /* 명령에 의해 설정됨 */



fcntl에 의한 록킹

```
#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);
```

- 현재의 읽기/쓰기 포인터의 위치에서 시작하여 512 바이트에 록을 건다
- "만약 지정된 영역의 전체나 일부가 다른 프로세스에 의해 이미 록되었다면, 호출 프로세스는 지정된 영역을 모두 사용 할 수 있을 때까지 sleep 한다."
- 해당 영역이 풀려나면 록을 건다.

Simple example

```
#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.start = 0;
My_lock.l_len = 10;

/* 파일을 개방한다. */
Fd = open("locktest", O_RDWR);

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

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

Switch(fork()) {
Case –1:
        perror ("fork");
        exit (1);
Case 0:
        my_lock.l_len = 5;
        if(fcntl (fd, F_SETLKW, &my_lock))
                == -1)
        {
                peeror("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);
}
```

Fcntl 록 열기

ㅁType을 F_UNLCK으로 설정함

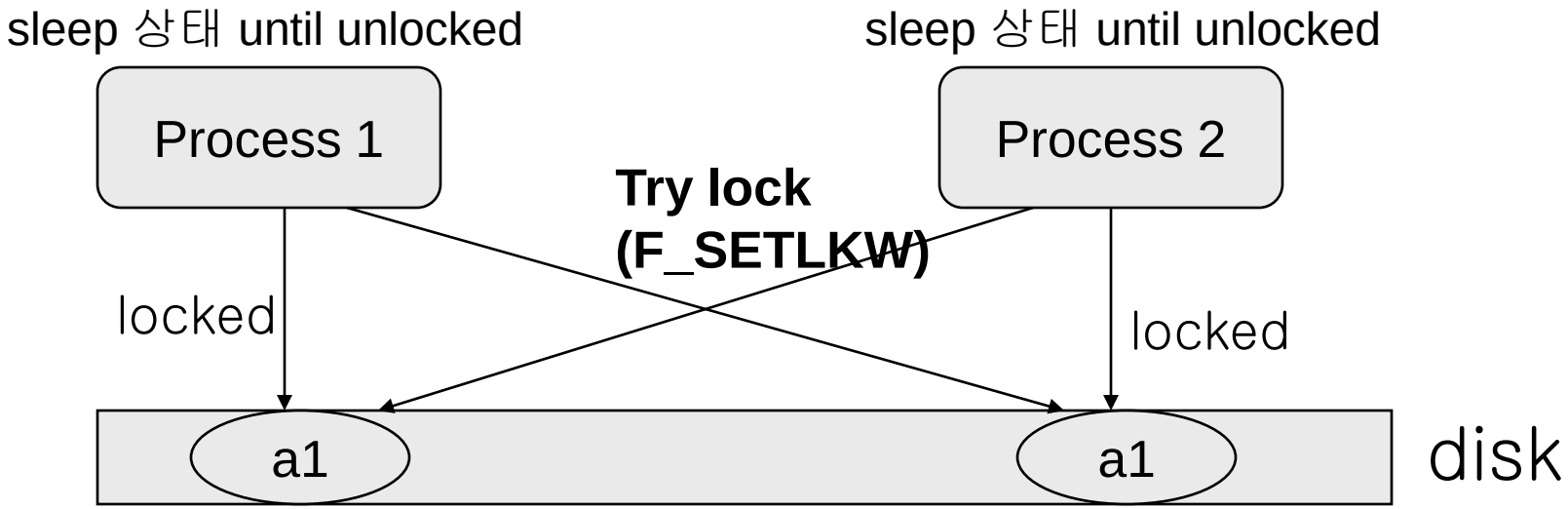
```
#include <unistd.h>
#include <fcntl.h>
…

/* 부모가 퇴장하기 전에 LOCK을 해제한다. */

Printf(“parent : unlocking \n”);
my_lock.l_type = F_UNLCK;

If( fcntl(fd, F_SETLK, &my_lock) == -1)
{
      perror (“parent : unlocking”);
      exit (1);
}
```

록에 대한 테스트

□cmd을 F_GETLK로 설정

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

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

교착 상태 (Dead lock)



□Dead lock 해제

– Fcntl 사용

◆ 만약 F_SETLKW 요청 시 Dead lock 이 발생하려 하면 –1 반환, errno는 EDEADLK으로 설정됨

◆ *단, 2개의 process간의 Dead lock 검출*

Dead lock example

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

main()
{
int fd;
struct flock first_lock;
struct flock second_lock;

/* 쓰기 록의 인수를 지정 */
first_ock.l_type = F_WRLCK;
first _lock.l_whence = SEEK_SET;
first _lock.start = 0;
first _lock.l_len = 10;

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


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

if(fcntl (fd, F_SETLW, &first_lock) == -1)
                fatal("A");

printf ( "A : Lock succeeded (proc %d)₩n", getpid()))
```

```
switch(fork()) {
case –1:
        fatal ("fork");
case 0:        /* child process */
        if(fcntl (fd, F_SETLKW, &second_lock)) == -1)
                fatal("b");

        printf ( "B : Lock succeeded (proc %d)₩n",
                                getpid()));

        if(fcntl (fd, F_SETLKW, &first_lock)) == -1)
                fatal("C");

        printf ( "C : Lock succeeded (proc %d)₩n",
                                getpid()))
        exit(0);

default :        /parent process */
        printf("parent sleeping ₩n");
        sleep (10);

        if(fcntl (fd, F_SETLKW, &second_lock)) == -1)
                fatal("D");

        printf ( "D : Lock succeeded (proc %d)₩n",
                                getpid()))

}
}
```

고급 프로세스간 통신

고급 IPC

- 메시지 전달 (message queue)
- 세마포 (semaphore)
- 공유 메모리 (shared memory)
- IPC key
  - Unix system에서 IPC를 구별하기 위한 식별자
  - 파일 경로이름을 키로 변환하는 함수

응용 분야
명명된 화일들을 조작하는 데 IPC기능들이 사용될 때

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

key_t ftok(const char *path, int id);
                                    return : key value if OK, -1 on error
```

고급 IPC 설비

☐ Summary of System V IPC system calls

	Message queue	Semaphore	Shared memory
include file	<sys/msg.h>	<sys/sem.h>	<sys/shm.h>
system call to create or open	msgget	semget	shmget
system call for control operations	msgctl	semctl	shmctl
system calls for IPC operations	msgsnd	semop	shmat
	msgrcv		shmdt

☐ kernel은 각 IPC channel에 대해서도 file을 다루기 위한 정보와 비슷한 정보의 구조를 갖음

- <sys/ipc.h>에 정의
- 이 구조를 사용하고 변경하기 위해서는 세 가지의 ctl system call을 사용
- 하나의 IPC channel을 만들거나 열기위해 사용

```
struct ipc_perm {
   ushort uid;        /*owner's user id */
   ushort gid:        /* owner's group id */
   ushort cuid;       /* creator's user id */
   ushort cgid;       /* creator's group id */
   ushort mode;       /* access modes */
   ushort seq;        /* slot usage sequence number */
   key_t  key;        /* key */
};
```

Message queues

□ **system**내의 각 **message queue**에 대해서 **kernel**은 다음과 같은 정보 구조를 유지

```
#include <sys/types.h>
#include <sys/ipc.h>          /* defines the ipc_perm structure */

struct msqid_ds {
   struct ipc_perm msg_perm;   /* operation permission struct */
   struct msg      *msg_first; /* ptr to first message on q */
   struct msg      *msg_last;  /* ptr to last message on q */
   ushort          msg_cbytes; /* current # bytes on q */
   ushort          msg_qnum;   /* current # of messages on q */
   ushort          msg_qbytes; /* max # of bytes allowed on q */
   ushort          msg_lspid;  /* pid of last msgsnd */
   ushort          msg_lrpid;  /* pid of last msg rcv */
   time_t          msg_stime;  /* time of last msgsnd */
   time_t          msg_rtime;  /* time of last msgrcv */
   time_t          msg_ctime;  /* time of last msgctl (that changed the above) */
};
```

Message queues (con't)

☐ msgget system call

- 새로운 **message queue**을 만들거나 기존의 **message queue**에 접근

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

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

- msgflag : 다음 상수들의 조합

Numeric	Symbolic	Description
0400	MSG_R	Read by owner
0200	MSG_W	Write by owner
0040	MSG_R >> 3	Read by group
0020	MSG_W >> 3	Write by group
0004	MSG_R >> 6	Read by world
0002	MSG_W >> 6	Write by world
	IPC_CREAT	
	IPC_EXCL	

-IPC_CREAT : key에 해당하는 메시지 큐가 존재하지 않는 경우 msgget이 이를 생성하도록 지시(메시지 큐가 있는 경우 덮어쓰지 않음)

- IPC_EXCL : IPC_CREAT와 이것이 동시에 설정된 경우 호출은 단지 하나의 메시지 큐를 생성하기 위한것.(만약, key에 대한 큐가 이미 존재하는 경우에는 msgget은 실패하고 –1을 반환, 오류변수 errno :EEXIST

Message queues (con't)

☐ msgsnd system call

- **message queue**에 **message** 하나를 넣음

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

int msgsnd(int msqid, struct msgbuf *ptr, int /ength, int flag);
```

- *ptr* :
```
struct msgbuf {
    long mtype;       /* message type, must be > 0 */
    char mtext[1];    /* message data */
};
```

- */ength* : message의 길이(byte단위)
- *flag* : IPC_NOWAIT나 0으로 지정

  * IPC_NOWAIT
    message queue에 새로운 message를 위한 여유 공간이 없을 경우에 즉시 system call 에서 돌아오도록 함

Message queues (con't)

☐ msgrcv system call

- message queue에서 **message**를 읽음

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

int msgrcv(int msqid, struct msgbuf *ptr, int length, long msgtype, int flag);
```

- *ptr* : msgsnd와에서와 비슷하며 받은 message를 저장할 장소를 명시
- *length* : ptr이 지시하는 구조의 data부분의 크기를 명시
- msgtype : queue로부터 어떤 message를 요구하는지를 명시
  - * msgtype = 0, queue의 첫번째 message를 받음
  - * msgtype > 0, msgtype과 동일한 type을 갖는 첫번째 message를 받음
  - * msgtype < 0, msgtype의 절대치보다 같거나 작은 type중에서 가장 작은 type을 갖는 첫번째 message를 받음
- *flag* : 요구된 type의 message가 queue에 없을 경우에 어떻게 할 것인지를 명시
  - * MSG_NOERROR 설정 시, 받은 message의 data 부분이 length보다 크다면 즉시 data 부분을 자르고 error없이 복귀함

Message queues (con't)

☐ msgctl system call

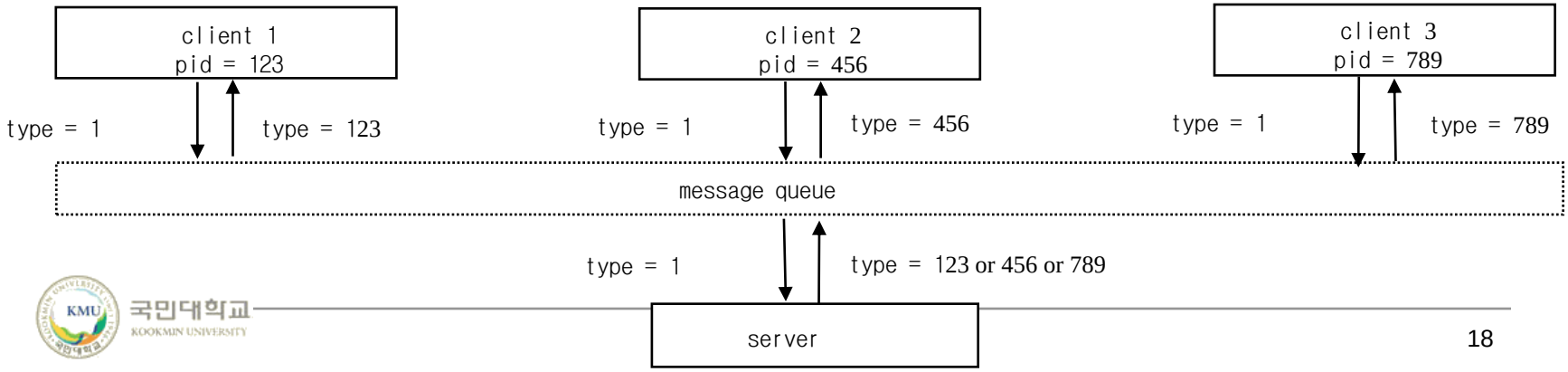
- **message queue**에 대한 다양한 제어 기능을 제공

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

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

- *cmd* :
  - IPC_STAT : 상태정보 GET
  - IPC_SET : 상태정보 SET
  - IPC_RMID는 message queue를 제거하는 것으로 여기서는 이것만 사용

☐ Multiplexing Messages

- 각 **message**마다 관련된 **type**을 갖는 목적
  - : 다수의 **process**들이 하나의 **queue**에서 **message**들을 다중화할 수 있도록 하기위해



두개의 Message queues를 사용한 client-server example

☐ msgq.h

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

#include   <sys/errno.h>
extern int errno;

#define    MKEY1     1234L
#define    MKEY2     2345L

#define    PERMS     0666
```

☐ Server를 위한 main 함수

```
#include   "msgq.h"

main()
{
           int        readid, writeid;

           /*
            * Create the message queues, if required.
            */

           if ( (readid = msgget(MKEY1, PERMS | IPC_CREAT)) < 0)
                      err_sys("server: can't get message queue 1");
           if ( (writeid = msgget(MKEY2, PERMS | IPC_CREAT)) < 0)
                      err_sys("server: can't get message queue 2");

           server(readid, writeid);

           exit(0);
}
```

두개의 Message queues를 사용한 client-server example

□ Client를 위한 main 함수

```
#include  "msgq.h"

main()
{
          int       readid, writeid;

          /*
           * Open the message queues.  The server must have
           * already created them.
           */
          if ( (writeid = msgget(MKEY1, 0)) < 0)
                    err_sys("client: can't msgget message queue 1");
          if ( (readid = msgget(MKEY2, 0)) < 0)
                    err_sys("client: can't msgget message queue 2");

          client(readid, writeid);

          /*
           * Now we can delete the message queues.
           */
          if (msgctl(readid, IPC_RMID, (struct msqid_ds *) 0) < 0)
                    err_sys("client: can't RMID message queue 1");
          if (msgctl(writeid, IPC_RMID, (struct msqid_ds *) 0) < 0)
                    err_sys("client: can't RMID message queue 2");

          exit(0);
}
```

두개의 Message queues를 사용한 client-server example

□ mesg.h

```
/*
 * Definition of "our" message.
 *
 * You may have to change the 4096 to a smaller value, if message queues
 * on your system were configured with "msgmax" less than 4096.
 */

#define   MAXMESGDATA         (4096-16)
                                        /* we don't want sizeof(Mesg) > 4096 */

#define   MESGHDRSIZE         (sizeof(Mesg) - MAXMESGDATA)
                                        /* length of mesg_len and mesg_type */

typedef struct {
  int     mesg_len;  /* #bytes in mesg_data, can be 0 or > 0 */
  long    mesg_type; /* message type, must be > 0 */
  char    mesg_data[MAXMESGDATA];
} Mesg;
```

두개의 Message queues를 사용한 client-server example

□ mesg_send 함수

```
#include  "mesg.h"

/*
 * Send a message using the System V message queues.
 * The mesg_len, mesg_type and mesg_data fields must be filled
 * in by the caller.
 */

mesg_send(id, mesgptr)
int       id;                   /* really an msqid from msgget() */
Mesg      *mesgptr;
{
          /*
           * Send the message - the type followed by the optional data.
           */

          if (msgsnd(id, (char *) &(mesgptr->mesg_type), mesgptr->mesg_len, 0) != 0)
                    err_sys("msgsnd error");
}
```

두개의 Message queues를 사용한 client-server example

□ mesg_rcv 함수

```
#include   "mesg.h"

/*
 * Receive a message from a System V message queue.
 * The caller must fill in the mesg_type field with the desired type.
 * Return the number of bytes in the data portion of the message.
 * A 0-length data message implies end-of-file.
 */

int mesg_recv(id, mesgptr)
int        id;                  /* really an msqid from msgget() */
Mesg       *mesgptr;
{
           int        n;

           /*
            * Read the first message on the queue of the specified type.
            */
           n = msgrcv(id, (char *) &(mesgptr->mesg_type), MAXMESGDATA,mesgptr-
>mesg_type, 0);
           if ( (mesgptr->mesg_len = n) < 0)
                      err_dump("msgrcv error");
           return(n);           /* n will be 0 at end of file */
}
```

두개의 Message queues를 사용한 client-server example

☐ Server program

```
#include  <stdio.h>
#include  "mesg.h"
#include  "msgq.h"

Mesg      mesg;

main()
{
          int       id;

          /*
           * Create the message queue, if required.
           */
          if ( (id = msgget(MKEY1, PERMS | IPC_CREAT)) < 0)
                    err_sys("server: can't get message queue 1");
          server(id);
          exit(0);
}
```

두개의 Message queues를 사용한 client-server example

```
server(id)
int        id;
{
           int        n, filefd;
           char       errmesg[256], *sys_err_str();

           /*
            * Read the filename message from the IPC descriptor.
            */
           mesg.mesg_type = 1;              /* receive messages of this type */
           if ( (n = mesg_recv(id, &mesg)) <= 0)
                      err_sys("server: filename read error");
           mesg.mesg_data[n] = '₩0';        /* null terminate filename */

           mesg.mesg_type = 2;              /* send messages of this type */
           if ( (filefd = open(mesg.mesg_data, 0)) < 0) {
                      /*
                       * Error.  Format an error message and send it back
                       * to the client.
                       */
                      sprintf(errmesg, ": can't open, %s₩n", sys_err_str());
                      strcat(mesg.mesg_data, errmesg);
                      mesg.mesg_len = strlen(mesg.mesg_data);
                      mesg_send(id, &mesg);
```

두개의 Message queues를 사용한 client-server example

```
        } else {
                /*
                 * Read the data from the file and send a message to
                 * the IPC descriptor.
                 */
                while ( (n = read(filefd, mesg.mesg_data, MAXMESGDATA)) > 0) {
                        mesg.mesg_len = n;
                        mesg_send(id, &mesg);
                }
                close(filefd);
                if (n < 0)
                        err_sys("server: read error");
        }

        /*
         * Send a message with a length of 0 to signify the end.
         */
        mesg.mesg_len = 0;
        mesg_send(id, &mesg);
}
```

두개의 Message queues를 사용한 client-server example

□ Client program

```
#include  <stdio.h>
#include  "mesg.h"
#include  "msgq.h"
Mesg      mesg;

main()
{
          int       id;
          /*
           * Open the single message queue.  The server must have
           * already created it.
           */
          if ( (id = msgget(MKEY1, 0)) < 0)
                    err_sys("client: can't msgget message queue 1");

          client(id);

          /*
           * Now we can delete the message queue.
           */
          if (msgctl(id, IPC_RMID, (struct msqid_ds *) 0) < 0)
                    err_sys("client: can't RMID message queue 1");
          exit(0);
```

두개의 Message queues를 사용한 client-server example

```
client(id)
int        id;
{
           int        n;
           /*
            * Read the filename from standard input, write it as
            * a message to the IPC descriptor.
            */
           if (fgets(mesg.mesg_data, MAXMESGDATA, stdin) == NULL)
                      err_sys("filename read error");
           n = strlen(mesg.mesg_data);
           if (mesg.mesg_data[n-1] == '₩n')          n--;     /* ignore the newline
from fgets() */
           mesg.mesg_data[n] = '₩0';                /* overwrite newline at end */
           mesg.mesg_len = n;
           mesg.mesg_type = 1;                      /* send messages of this type */
           mesg_send(id, &mesg);
           /*
            * Receive the message from the IPC descriptor and write
            * the data to the standard output.
            */
           mesg.mesg_type = 2;  /* receive messages of this type */
           while( (n = mesg_recv(id, &mesg)) > 0)
                      if (write(1, mesg.mesg_data, n) != n)
                                 err_sys("data write error");
           if (n < 0)err_sys("data read error");
}
```

Message Queue Example: Priority Queue

```
/* q.h */
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<string.h>
#include<errno.h>
#define QKEY (key_t)0105
#define QPERM 0660
#define maxobn 50
#define MAXPRIOR 10
struct q_entry {
long mtype;
char mtext[MAXOBN+1];
};
```

```
/* init_queue */
#include "q.h"

int init_queue(void)
{
int queue_id;

if ((queue_id=msgget(QKEY,IPC_CREAT|QPERM))==-1)
          perror("msgget failed");

return(queue_id);
}
```

Message Queue Example: Priority Queue(cont)

```
/* enter */
int enter (char *objname, int priority)
{
        int len, s_qid;
        struct q_entry s_entry;
        if((len=strlen(objname))>MAXOBN)
        {
                warn ("name too long");
                return (-1);
        }
        if(priority>MAXPRIOR||priority<0)
        {
                warn("invalid priority level");
                return (-1);
        }
        if((s_qid=init_queue())==-1)
                return(-1);
        s_entry. mtype=(long)priority;
        strncpy(s_entry.mtext, objname,MAXOBN);
        if(msgsnd(s_qid,s_entry, len,0)==-1)
        {
                perror("msgnd failed");
                return(-1);
        }
        else return(0);
}
```

Message Queue Example: Priority Queue(cont)

```
/* server */
int serve(void)
{
int mlen, r_qid;
Struct q_entry r_entry;

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

For(;;)
{
          if ((mlen = msgrcv(r_qid, &r_entry, MAXOBN,
                    (-1*MAXPRIOR), MSG_NOERROR))==-1)
          {
                    perror("msgrcv failed");
                    return(-1);
          }
          else
          {
                    r_entry.mtext[mlen]='\0';
                    proc_obj(&r_entry);
          }
}
}
```

Message Queue Example: Priority Queue(cont)

```
/* etest */
#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(argv[2])) <= 0 || priority > MAXPRIOR)
    {
       warn("invalid priority");
       exit(2);
    }
    if(enter(argv[1], priority) < 0)
    {
       warn("enter failure");
       exit(3);
    }
    exit(0);
}
```

Message Queue Example: Priority Queue(cont)

```
/* stest */
#include <stdio.h>
#include "q.h"

main()
{
  pid_t pid;

  switch(pid = fork())
  {
     case 0:
          serve();
          break;
     case -1:
          warn("fork to start serve failed");
          break;
     default:
          printf("serve 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 Queue Example: Priority Queue(cont)

```
$ etest objname1 3
$ etest objname2 4
$ etest objname3 1
$ etest objname4 9
$ stest
Server process pid is 2545
$
priority: 1 name: objname3

priority: 3 name: objname1

priority: 4 name: objname2

priority: 9 name: objname4
```

Semaphore

☐ semaphore

- 동기화의 기본
- 다수의 **process**들의 작업을 동기화하기위해서 사용
- 주로 사용하는 곳은 **shared memory segment**에의 접근을 동기화하기 위함

☐ 시스템내의 모든 **semaphore**의 집합에 대해 **kernel**은 다음과 같은 구조를 유지

```
#include <sys/types.h>
#include <sys/ipc.h> /* defines the ipc_perm structure */

struct semid_ds {
   struct ipc_perm sem_perm;   /* operation permission struct */
   struct sem      *sem_base;  /* ptr to first semaphore in set */
   ushort             sem_nsems;         /* # of semaphores in semop */
   time_t             sem_otime;         /* time of last semop */
   time_t             sem_ctime;         /* time of last change */
};
```

```
struct sem {
    ushort semval;      /* semaphore value, nonnegative */
    short   sempid;     /* pid of last operation */
    ushort semncnt;     /* # awaiting semval > cval */
    ushort semzcnt;     /* # awaiting semval = 0 */
};
```

Semaphore(con't)

☐ semget system call

- **semaphore**를 만들거나 기존의 **semaphore**에 접근

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

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

- **nsems : semaphore**의 수
- **semflag :** 다음 상수들의 조합

Numeric	Symbolic	Description
0400	SEM_R	Read by owner
0200	SEM_W	Write by owner
0040	SEM_R >> 3	Read by group
0020	SEM_W >> 3	Write by group
0004	SEM_R >> 6	Read by world
0002	SEM_W >> 6	Write by world
	IPC_CREAT	
	IPC_EXCL	

Semaphore(con't)

☐ semop system call

- 집합내의 하나 또는 그 이상의 **semaphore** 값들에 대해 연산을 수행

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

int semop(int semid, struct sembuf **opsptr, unsigned int nops);
```

- *opsptr* :
```
struct sembuf {
    ushort sem_num;      /* semapore # */
    short  sem_op;       /* semaphore operation */
    short  sem_flg;      /* operation flags */
};
```

- sem_op :
  - ① sem_op > 0 : sem_val의 값이 semaphore의 현재 값에 더해짐
    이것은 semaphore가 제어하는 자원들의 해제에 해당
  - ② sem_op = 0 : 호출한 process는 semaphore의 값이 0이 될 때까지 기다리기를 원함
  - ③ sem_op < 0 : 호출한 process는 semaphore의 값이 sem_op의 절대값보다 크거나 같아질 때까지 기다린다. 이것은 자원의 할당에 해당

Semaphore(con't)

☐ semctl system call

- **semaphore**에 대한 다양한 제어 연산들을 제공

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

int semctl(int semid, int semnum, int cmd, union semun arg);

 union semun {
        int                 val;      /* used for SETVAL only */
        struct semid_ds    *buff;     /* used for IPC_STAT and IPC_SET */
        ushort             *array;    /* used for IPC_GETALL & IPC_SETALL */
 } arg;
```

- ***cmd*** : **IPC_RMID**는 **semaphore**를 제거하기 위해 사용
  GETVAL은 **semaphore** 값을 가져오기 위해 사용
  SETVAL은 명시된 **semaphore** 값으로 바꾸기 위해 사용

File locking example

```
#include  <sys/types.h>
#include  <sys/ipc.h>
#include  <sys/sem.h>
#define SEMKEY       123456L    /* key value for semget() */
#define PERMS        0666

static struct sembuf op_lock[2] = {
          0, 0, 0,   /* wait for sem#0 to become 0 */
          0, 1, 0    /* then increment sem#0 by 1 */ };

static struct sembuf op_unlock[1] = {
          0, -1, IPC_NOWAIT    /* decrement sem#0 by 1 (sets it to 0) */ };
int       semid = -1;          /* semaphore */

my_lock(fd)
int       fd;
{         if (semid < 0)
                    if ( (semid = semget(SEMKEY, 1, IPC_CREAT | PERMS)) < 0)
                              err_sys("semget error");
          if (semop(semid, &op_lock[0], 2) < 0)
                    err_sys("semop lock error");
}

my_unlock(fd)
int       fd;
{         if (semop(semid, &op_unlock[0], 1)< 0)
                    err_sys("semop unlock error);
}
```

Semaphore Example: P() & V()

```
/* pv.h */
#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
```

Semaphore Example: P() & V() (cont)

```
#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
                    {
                              semun arg;
                              arg.val = 1;
                              status = semctl(semid, 0, SETVAL, arg);
                    }

                    if (semid == -1 || status == -1)
                    {
                              perror("initsem failed");
                              return (-1);
                    }

                    return (semid);
}
```

Semaphore Example: P() & V() (cont)

```
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);
}
```

```
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);
}
```

Semaphore Example: P() & V() (cont)

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

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

        prinff("\nprocess %d before critical section\n", pid);

        p(semid);

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

        /* 실제로 무언가 한다 */
        sleep(10);

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

        v(semid);

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

Semaphore Example: P() & V() (cont)

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

        for (1=0; i<3; i++)
        {
                if (fork() == 0)
                        handlesem(semkey);
        }
}
```

Shared memory

□ **Client-Server** 화일 복제 프로그램에 사용되는 통상적인 몇 가지 단계를 고려해보자.

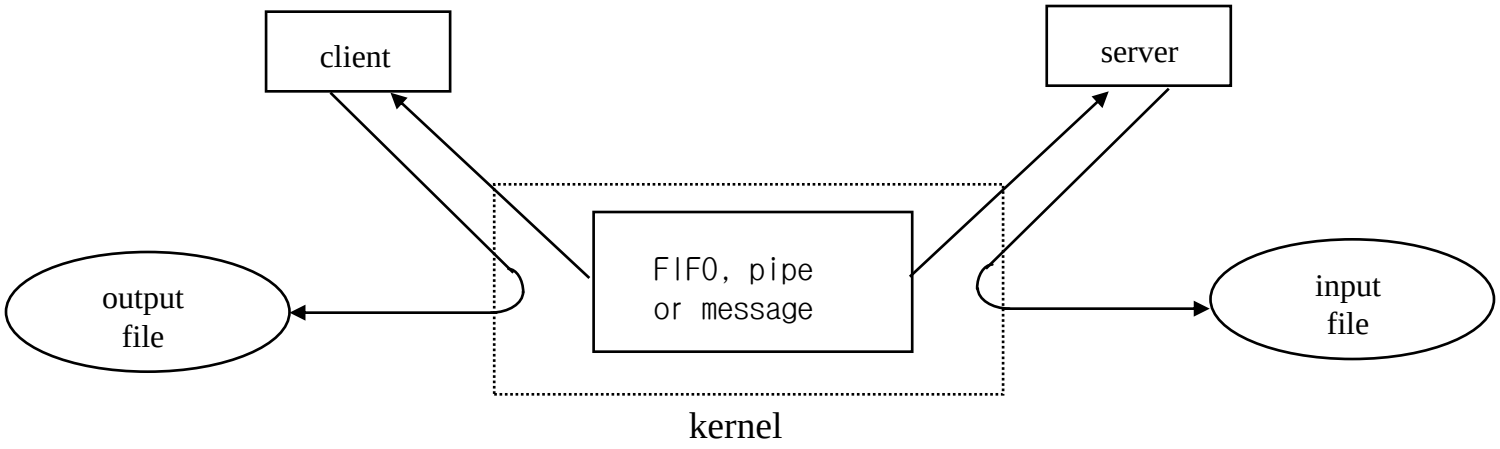
-**server**가 입력 화일에서 읽는다. 이러한 동작은 보통 **kernel**이 데이타를 자신의 내부 **block buffer**중의 하나로 읽어들이고, 이것을 **server**의 **buffer**로 복제하는 두 가지 동작으로 이루어진다.
-**server**는 **pipe, FIFO, message queue** 등의 기법중에 하나를 사용하여 이러한 **data**를 **message**안에 쓸 수 있다. 이러한 세가지 **IPC** 형태의 어느 것이나 데이터가 **user**의 **buffer**로 복제하여야 한다.
-**client**는 **IPC channel**에서 데이타를 읽는데 이때, 다시 데이타를 **kernel**의 **IPC buffer**에서 **client**의 **buffer**로 복제하여야 한다.
-마지막으로 데이타는 **system call write**의 두번째 독립 변수인 **client**의 **buffer**에서 출력 화일로 복제된다.

□ 전체적으로 데이타를 네번 복제하는 것이 필요

- 이러한 네번의 복제는 **kernel**과 사용자 **process**간에서 행해진다.
  : 이러한 복제를 **intercontext copy** 라고 한다.

Shared memory (con't)

☐ **client**와 **server**사이에서의 전형적인 데이터 이동



☐ 이러한 **IPC** 형태**(pipe, FIFO, message queue)**와 관련된 문제

- 두 프로세스가 정보를 교환하기 위해서 정보가 **kernel**을 거쳐가야 한다**.**

☐ **shared memory**

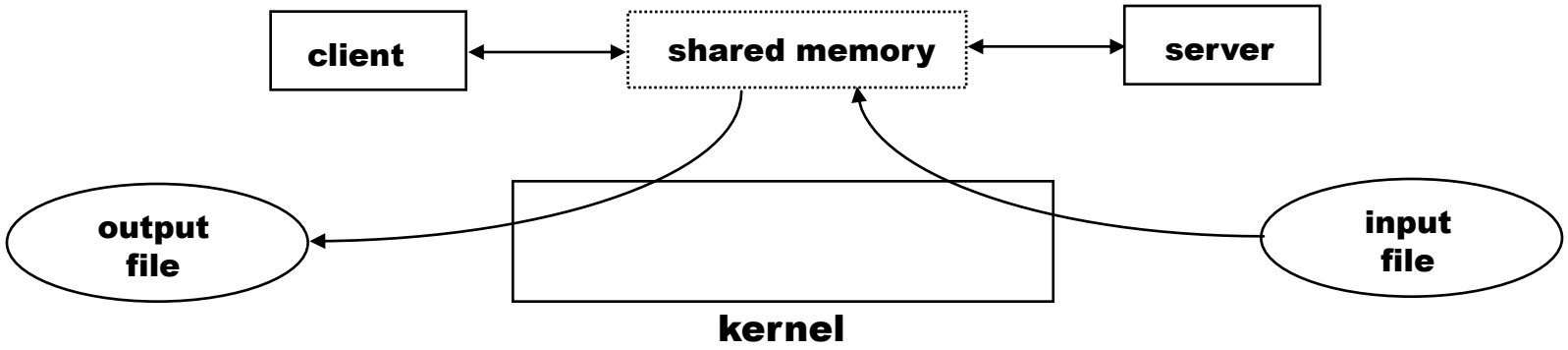
- 두개 이상의 프로세스가 기억장소의 일부를 공유하도록 해서 이러한 문제를 해결

Shared memory (con't)

☐ **shared memory** 를 이용한 **Client-Server** 파일 복제 프로그램에서의 몇 가지 단계

- server는 semaphore를 사용하여 shared memory segment로 접근한다.
- server는 입력화일을 shared memory segment로 읽어 들인다. 읽어들일 주소는 shared memory를 가리킨다.
- 읽기가 완료되면 일꾼은 다시 semaphore를 사용하여 client에게 알린다.
- client는 shared memory segment에서 출력 화일로 데이터를 쓴다.

☐ **shared memory**를 이용한 **client**와 **server**간의 데이터 이동



Shared memory (con't)

- 모든 shared memory segment에 대해서 kernel은 다음과 같은 정보 구조를 유지

```
#include <sys/types.h>
#include <sys/ipc.h>                    /* defines the ipc_perm structure */

struct shmid_ds {
   struct ipc_perm shm_perm;            /* operation permission struct */
   int             shm_segsz;           /* segment size */
   struct XXX      shm_YYY;             /* implementation dependent info */
   ushort          shm_lpid;            /* pid of last operation */
   ushort          shm_cpid;            /* creator pid */
   ushort          shm_nattch;          /* current # attached */
   ushort          shm_cnattch;         /* in-core # attached */
   time_t          shm_atime;           /* last attach time */
   time_t          shm_dtime;           /* last detach time */
   time_t          shm_ctime;           /* last change time */
};
```

Shared memory (con't)

☐ shmget system call

- **shared memory segment**를 생성하거나 기존의 것에 접근

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

int shmget(key_t key, int size, int shmflag);
```

- **return value : shared memory** 식별자 ***shmid***
- **nsems : semaphore**의 수
- **semflag :** 다음 상수들의 조합

Numeric	Symbolic	Description
0400	SHM_R	Read by owner
0200	SHM_W	Write by owner
0040	SHM_R >> 3	Read by group
0020	SHM_W >> 3	Write by group
0004	SHM_R >> 6	Read by world
0002	SHM_W >> 6	Write by world
	IPC_CREAT	
	IPC_EXCL	

Shared memory (con't)

☐ shmat system call

- shared memory segment를 attach시킴

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

int shmat(int shmid, char *shmaddr, int shmflag);
```

- return value :shared memory segment의 address
- shared memory의 address를 결정하는 규칙 :
  - ① *shmaddr* = 0 : system은 호출한 프로세스를 위한 주소를 선정
  - ② *shmaddr* <> 0 : 돌려주는 주소는 호출 프로세스가 *shmflag* 독립변수로 SHM_RND값을 명시했는지 여부에 따라 다르다.

☐ shmdt system call

- shared memory segment를 detach시킴

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

int shmdt(char *shmaddr);
```

Shared memory (con't)

☐ shmctl system call

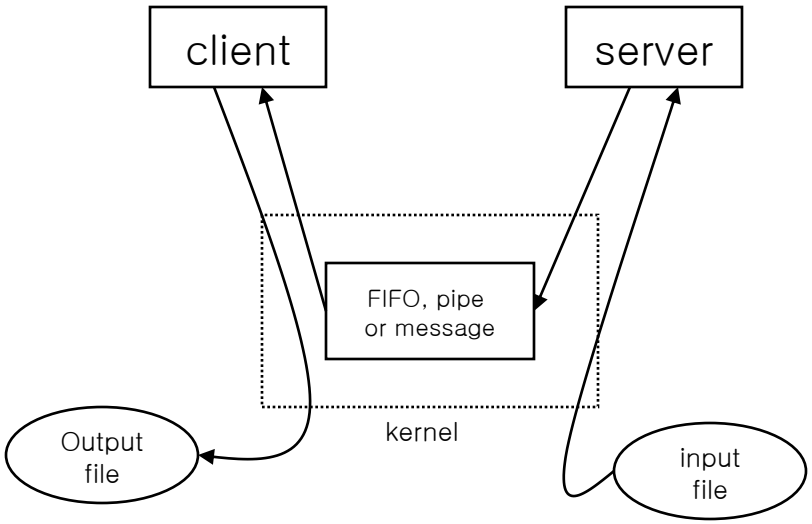
- shared memory segment를 제거시킴(*cmd*값으로 IPC_RMID를 명시)

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

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

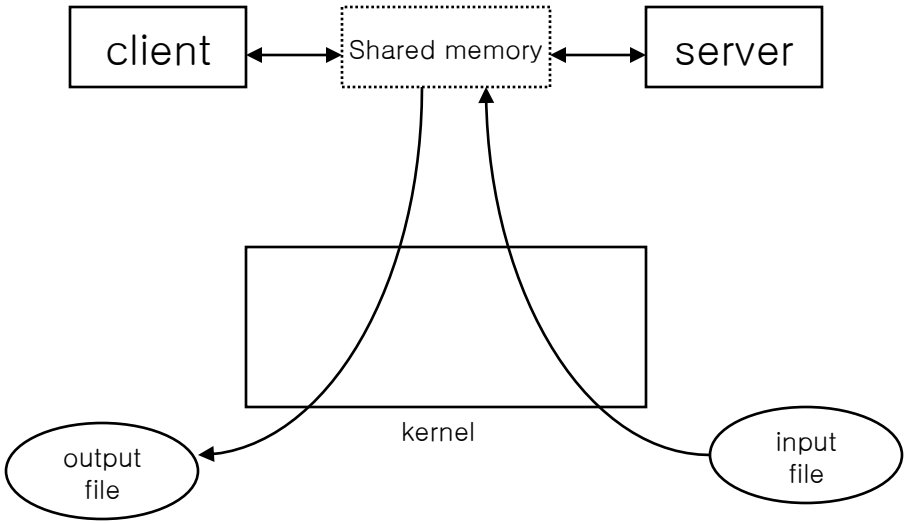
☐ shm.h

```
#include      "mesg.h"

#define NBUFF          4          /* number of buffers in shared memory */
                                  /* (for multiple buffer version */
#define SHMKEY         ((key_t) 7890)          /* base value for shmem key */

#define SEMKEY1        ((key_t) 7891)          /* client semaphore key */
#define SEMKEY2        ((key_t) 7892)          /* server semaphore key */

#define PERMS          0666
```

Shared memory (con't)



C/S사이의 전형적인 데이터 이동



공유 기억장소를 이용한 C/S간의 데이터 이동

Shared memory structure

```
#include <sys/types.h>
#include <sys/ipc.h>      // defines the ipc_perm structure

struct shmid_ds {
        struct  ipc_perm  shm_perm;     /* operation permission struct */
        int               shm_segsz;    /* segment size */
        struct  XXX       shm_YYY;      /* implementation dependent info */
        ushort            shm_lpid;     /* pid of last operation */
        ushort            shm_cpid;     /* creator pid */
        ushort            shm_nattch;   /* current # attached */
        ushort            shm_cnattch;  /* in-core # attached */
        time_t            shm_atime;    /* last attach time */
        time_t            shm_dtime;    /* last detach time */
        time_t            shm_ctime;    /* last change time */
};
```

Shared memory function(1)

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

int shmget (key_t  key,  int  size,  int  shmflag);
```

Numeric	Symbolic	Description
0400	SHM_R	Read by owner
0200	SHM_W	Write by owner
0040	SHM_R >> 3	Read by group
0020	SHM_W >> 3	Write by group
0004	SHM_R >> 6	Read by world
0002	SHM_W >> 6	Write by world
	IPC_CREAT	
	IPC_EXCL	

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

int shmat (int shmid, char *shmaddr, int shmflag);
```

0 or SHM_RND 명시여부

Shared memory function(2)

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

int shmdt (char *shmaddr);
```

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

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

IPC_RMID

Shared memory example: shmcopy

```
#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 /* shared mem key */
#define SHMKEY2                 (key_t) 0x15 /* shared mem key */
#define SEMKEY                  (key_t) 0x20 /* semphore key */

/* buffer size for reads and write */
#define SIZ 5*BUFSIZ

/* will hold data and read count */
struct databuf {
          int d_nread;
          char d_buf[SIZ];
};

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

Shared memory example: shmcopy(cont)

```
#include "shared_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)
{
          /* create shared memory segment */
          if ((shmid1 = shmget(SHMKEY1, sizeof(struct databuf),
                    0600 | IFLAGS)) == -1)
                    fatal("shmget");
          if ((shmid2 = shmget(SHMKEY2, sizeof(struct databuf),
                    0600 | IFLAGS)) == -1)
                    fatal("shmget");

          /* attach shared memory segments */
          if ((*p1 = (struct databuf *)shmat(shmid1,0,0)) == ERR)
                    fatal("shmat");
          if ((*p2 = (struct databuf *)shmat(shmid2,0,0)) == ERR)
                    fatal("shmat");
}
```

Shared memory example: shmcopy(cont)

```
int getsem(void)                    /* get semaphore set */
{
        semun x;
        x.val = 0;

        /* create two semaphore set */
        if ((semid = semget(SEMKEY, 2, 0600,| IFLAGS)) == -1)
                fatal("semget");

        /* set initial values */
        if (semctl(semid, 0, SETVAL, x) == -1)
                fatal("semctl");
        if (semctl(semid, 1, SETVAL, x) == -1)
                fatal("semctl");
        return (semid);

/* remove shared memory identifiers + sem set id */
void remobj(void)
{
        if (shmctl(shmid1, IPC_RMID, NULL) == -1)
                fatal("shmctl");
        if (shmctl(shmid2, IPC_RMID, NULL) == -1)
                fatal("shmctl");
        if (shmctl(semid, IPC_RMID, NULL) == -1)
                fatal("semctl");
}
```

Shared memory example: shmcopy(cont)

```
#include "share_ex.h"
/* these define p() and v() for two semaphores */
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 (;;) {
                    /* read into buffer buf1 */
                    buf1->d_nread = read(0, buf1->d_buf, SIZ);

                    /* synchronization point */
                    semop(semid, &v1, 1);
                    semop(semid, &p2, 1);

                    /* test here to avoid writer sleeping */
                    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;
          }
```

Shared memory example: shmcopy(cont)

```
#include "share_ex.h"

extern struct sembuf p1, p2 /* defined in reader.c */
extern struct sembuf v1, v2 /* defined in reader.c */

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);
        }
}
```