Full Code Examples

A

Code Example A-1 Sample Program to Illustrate msgget()

```
/*
 * msgget.c: Illustrate the msgget() function.
 * This is a simple exerciser of the msgget() function. It prompts
 * for the arguments, makes the call, and reports the results.
 */

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

extern void exit();
extern void perror();

main()
{
    key_t   key;    /* key to be passed to msgget() */
    int     msgflg, /* msgflg to be passed to msgget() */
            msqid;  /* return value from msgget() */

    (void) fprintf(stderr,
        "All numeric input is expected to follow C conventions:\n");
    (void) fprintf(stderr,
        "\t0x... is interpreted as hexadecimal,\n");
    (void) fprintf(stderr, "\t0... is interpreted as octal,\n");
    (void) fprintf(stderr, "\totherwise, decimal.\n");
    (void) fprintf(stderr, "IPC_PRIVATE == %#lx\n", IPC_PRIVATE);
    (void) fprintf(stderr, "Enter key: ");
```

```
    (void) scanf("%li", &key);
    (void) fprintf(stderr, "\nExpected flags for msgflg argument
are:\n");
    (void) fprintf(stderr, "\tIPC_EXCL =\t%#8.8o\n", IPC_EXCL);
    (void) fprintf(stderr, "\tIPC_CREAT =\t%#8.8o\n", IPC_CREAT);
    (void) fprintf(stderr, "\towner read =\t%#8.8o\n", 0400);
    (void) fprintf(stderr, "\towner write =\t%#8.8o\n", 0200);
    (void) fprintf(stderr, "\tgroup read =\t%#8.8o\n", 040);
    (void) fprintf(stderr, "\tgroup write =\t%#8.8o\n", 020);
    (void) fprintf(stderr, "\tother read =\t%#8.8o\n", 04);
    (void) fprintf(stderr, "\tother write =\t%#8.8o\n", 02);
    (void) fprintf(stderr, "Enter msgflg value: ");
    (void) scanf("%i", &msgflg);

    (void) fprintf(stderr, "\nmsgget: Calling msgget(%#lx, %#o)\n",
    key, msgflg);
    if ((msqid = msgget(key, msgflg)) == -1)
    {
        perror("msgget: msgget failed");
        exit(1);
    } else {
        (void) fprintf(stderr,
            "msgget: msgget succeeded: msqid = %d\n", msqid);
        exit(0);
    }
}
```

Code Example A-2 Sample Program to Illustrate `msgctl()`

```
/*
 * msgctl.c:  Illustrate the msgctl() function.
 *
 * This is a simple exerciser of the msgctl() function.  It allows
 * you to perform one control operation on one message queue.  It
 * gives up immediately if any control operation fails, so be careful
 * not to set permissions to preclude read permission; you won't be
 * able to reset the permissions with this code if you do.
 */
#include    <stdio.h>
#include    <sys/types.h>
#include    <sys/ipc.h>
#include    <sys/msg.h>
#include    <time.h>

static void do_msgctl();
extern void exit();
extern void perror();
```

```
static char warning_message[] = "If you remove read permission for \
                    yourself, this program will fail frequently!";

main()
{
    struct msqid_dsbuf;    /* queue descriptor buffer for IPC_STAT
                              and IP_SET commands */
    int             cmd,    /* command to be given to msgctl() */
                    msqid;  /* queue ID to be given to msgctl() */

    (void fprintf(stderr,
        "All numeric input is expected to follow C conventions:\n");
    (void) fprintf(stderr,
        "\t0x... is interpreted as hexadecimal,\n");
    (void) fprintf(stderr, "\t0... is interpreted as octal,\n");
    (void) fprintf(stderr, "\totherwise, decimal.\n");

    /* Get the msqid and cmd arguments for the msgctl() call. */
    (void) fprintf(stderr,
        "Please enter arguments for msgctls() as requested.");
    (void) fprintf(stderr, "\nEnter the msqid: ");
    (void) scanf("%i", &msqid);
    (void) fprintf(stderr, "\tIPC_RMID = %d\n", IPC_RMID);
    (void) fprintf(stderr, "\tIPC_SET = %d\n", IPC_SET);
    (void) fprintf(stderr, "\tIPC_STAT = %d\n", IPC_STAT);
    (void) fprintf(stderr, "\nEnter the value for the command: ");
    (void) scanf("%i", &cmd);

    switch (cmd) {
        case IPC_SET:
            /* Modify settings in the message queue control
structure. */
            (void) fprintf(stderr, "Before IPC_SET, get current
values:");
            /* fall through to IPC_STAT processing */
        case IPC_STAT:
            /* Get a copy of the current message queue control
             * structure and show it to the user. */
            do_msgctl(msqid, IPC_STAT, &buf);
            (void) fprintf(stderr, ]
            "msg_perm.uid = %d\n", buf.msg_perm.uid);
            (void) fprintf(stderr,
            "msg_perm.gid = %d\n", buf.msg_perm.gid);
            (void) fprintf(stderr,
            "msg_perm.cuid = %d\n", buf.msg_perm.cuid);
            (void) fprintf(stderr,
```

```
                "msg_perm.cgid = %d\n", buf.msg_perm.cgid);
                (void) fprintf(stderr, "msg_perm.mode = %#o, ",
                buf.msg_perm.mode);
                (void) fprintf(stderr, "access permissions = %#o\n",
                buf.msg_perm.mode & 0777);
                (void) fprintf(stderr, "msg_cbytes = %d\n",
                               buf.msg_cbytes);
                (void) fprintf(stderr, "msg_qbytes = %d\n",
                               buf.msg_qbytes);
                (void) fprintf(stderr, "msg_qnum = %d\n", buf.msg_qnum);
                (void) fprintf(stderr, "msg_lspid = %d\n",
                               buf.msg_lspid);
                (void) fprintf(stderr, "msg_lrpid = %d\n",
                               buf.msg_lrpid);
                (void) fprintf(stderr, "msg_stime = %s", buf.msg_stime ?
                ctime(&buf.msg_stime) : "Not Set\n");
                (void) fprintf(stderr, "msg_rtime = %s", buf.msg_rtime ?
                ctime(&buf.msg_rtime) : "Not Set\n");
                (void) fprintf(stderr, "msg_ctime = %s",
                               ctime(&buf.msg_ctime));
                if (cmd == IPC_STAT)
                    break;
                /*  Now continue with IPC_SET. */
                (void) fprintf(stderr, "Enter msg_perm.uid: ");
                (void) scanf ("%hi", &buf.msg_perm.uid);
                (void) fprintf(stderr, "Enter msg_perm.gid: ");
                (void) scanf("%hi", &buf.msg_perm.gid);
                (void) fprintf(stderr, "%s\n", warning_message);
                (void) fprintf(stderr, "Enter msg_perm.mode: ");
                (void) scanf("%hi", &buf.msg_perm.mode);
                (void) fprintf(stderr, "Enter msg_qbytes: ");
                (void) scanf("%hi", &buf.msg_qbytes);
                do_msgctl(msqid, IPC_SET, &buf);
                break;
            case IPC_RMID:
            default:
                /* Remove the message queue or try an unknown command. */
                do_msgctl(msqid, cmd, (struct msqid_ds *)NULL);
                break;
        }
        exit(0);
}
```

```
/*
 * Print indication of arguments being passed to msgctl(), call
 * msgctl(), and report the results. If msgctl() fails, do not
 * return; this example doesn't deal with errors, it just reports
 * them.
 */
static void
do_msgctl(msqid, cmd, buf)
struct msqid_ds*buf;   /* pointer to queue descriptor buffer */
int             cmd,    /* command code */
                msqid;  /* queue ID */
{
    register intrtrn;/* hold area for return value from msgctl() */

    (void) fprintf(stderr, "\nmsgctl: Calling msgctl(%d, %d, %s)\n",
            msqid, cmd, buf ? "&buf" : "(struct msqid_ds *)NULL");
    rtrn = msgctl(msqid, cmd, buf);
    if (rtrn == -1) {
        perror("msgctl: msgctl failed");
        exit(1);
    } else {
        (void) fprintf(stderr, "msgctl: msgctl returned %d\n",
                        rtrn);
    }
}
```

Code Example A-3 Sample Program to Illustrate `msgsnd()` and `msgrcv()`

```
/*
 * msgop.c: Illustrate the msgsnd() and msgrcv() functions.
 *
 * This is a simple exerciser of the message send and receive
 * routines. It allows the user to attempt to send and receive as many
 * messages as wanted to or from one message queue.
 */

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

static intask();
extern voidexit();
extern char*malloc();
extern voidperror();

charfirst_on_queue[] = "-> first message on queue",
```

```
    full_buf[] = "Message buffer overflow. Extra message text\
                 discarded.";

main()
{
    register int   c;      /* message text input */
    int            choice;/* user's selected operation code */
    register int   i;      /* loop control for mtext */
    int            msgflg;/* message flags for the operation */
    struct msgbuf  *msgp;  /* pointer to the message buffer */
    int            msgsz;  /* message size */
    long           msgtyp; /* desired message type */
    int            msqid,  /* message queue ID to be used */
                   maxmsgsz,/* size of allocated message buffer */
                   rtrn;   /* return value from msgrcv or msgsnd */
    (void) fprintf(stderr,
        "All numeric input is expected to follow C conventions:\n");
    (void) fprintf(stderr,
        "\t0x... is interpreted as hexadecimal,\n");
    (void) fprintf(stderr, "\t0... is interpreted as octal,\n");
    (void) fprintf(stderr, "\totherwise, decimal.\n");
    /* Get the message queue ID and set up the message buffer. */
    (void) fprintf(stderr, "Enter msqid: ");
    (void) scanf("%i", &msqid);
    /*
     * Note that <sys/msg.h> includes a definition of struct msgbuf
     * with the mtext field defined as:
     *      charmtext[1];
     * therefore, this definition is only a template, not a structure
     * definition that you can use directly, unless you want only to
     * send and receive messages of 0 or 1 byte. To handle this,
     * malloc an area big enough to contain the template - the size
     * of the mtext template field + the size of the mtext field
     * wanted. Then you can use the pointer returned by malloc as a
     * struct msgbuf with an mtext field of the size you want. Note
     * also that sizeof msgp->mtext is valid even though msgp isn't
     * pointing to anything yet. Sizeof doesn't dereference msgp, but
     * uses its type to figure out what you are asking about.
     */
    (void) fprintf(stderr,
        "Enter the message buffer size you want:");
    (void) scanf("%i", &maxmsgsz);
    if (maxmsgsz < 0) {
        (void) fprintf(stderr, "msgop: %s\n",
                "The message buffer size must be >= 0.");
        exit(1);
```

```
}
msgp = (struct msgbuf *)malloc((unsigned)(sizeof(struct msgbuf)
            - sizeof msgp->mtext + maxmsgsz));
if (msgp == NULL) {
    (void) fprintf(stderr, "msgop: %s %d byte messages.\n",
            "could not allocate message buffer for", maxmsgsz);
    exit(1);
}
/* Loop through message operations until the user is ready to
    quit. */
while (choice = ask()) {
    switch (choice) {
    case 1: /* msgsnd() requested: Get the arguments, make the
            call, and report the results. */
        (void) fprintf(stderr, "Valid msgsnd message %s\n",
            "types are positive integers.");
        (void) fprintf(stderr, "Enter msgp->mtype: ");
        (void) scanf("%li", &msgp->mtype);
        if (maxmsgsz) {
            /* Since you've been using scanf, you need the loop
               below to throw away the rest of the input on the
               line after the entered mtype before you start
               reading the mtext. */
            while ((c = getchar()) != '\n' && c != EOF);
            (void) fprintf(stderr, "Enter a %s:\n",
                        "one line message");
            for (i = 0; ((c = getchar()) != '\n'); i++) {
                if (i >= maxmsgsz) {
                    (void) fprintf(stderr, "\n%s\n", full_buf);
                    while ((c = getchar()) != '\n');
                    break;
                }
                msgp->mtext[i] = c;
            }
            msgsz = i;
        } else
            msgsz = 0;
        (void) fprintf(stderr,"\nMeaningful msgsnd flag is:\n");
        (void) fprintf(stderr, "\tIPC_NOWAIT =\t%#8.8o\n",
            IPC_NOWAIT);
        (void) fprintf(stderr, "Enter msgflg: ");
        (void) scanf("%i", &msgflg);
        (void) fprintf(stderr, "%s(%d, msgp, %d, %#o)\n",
            "msgop: Calling msgsnd", msqid, msgsz, msgflg);
        (void) fprintf(stderr, "msgp->mtype = %ld\n",
                    msgp->mtype);
```

```
        (void) fprintf(stderr, "msgp->mtext = \"");
        for (i = 0; i < msgsz; i++)
            (void) fputc(msgp->mtext[i], stderr);
            (void) fprintf(stderr, "\"\n");
            rtrn = msgsnd(msqid, msgp, msgsz, msgflg);
            if (rtrn == -1)
                perror("msgop: msgsnd failed");
            else
                (void) fprintf(stderr,
                        "msgop: msgsnd returned %d\n", rtrn);
            break;
    case 2: /* msgrcv() requested: Get the arguments, make the
                call, and report the results. */
        for (msgsz = -1; msgsz < 0 || msgsz > maxmsgsz;
                (void) scanf("%i", &msgsz))
            (void) fprintf(stderr, "%s (0 <= msgsz <= %d): ",
                        "Enter msgsz", maxmsgsz);
        (void) fprintf(stderr, "msgtyp meanings:\n");
        (void) fprintf(stderr, "\t 0 %s\n", first_on_queue);
        (void) fprintf(stderr, "\t>0 %s of given type\n",
            first_on_queue);
        (void) fprintf(stderr, "\t<0 %s with type <= |msgtyp|\n",
                    first_on_queue);
        (void) fprintf(stderr, "Enter msgtyp: ");
        (void) scanf("%li", &msgtyp);
        (void) fprintf(stderr,
                    "Meaningful msgrcv flags are:\n");
        (void) fprintf(stderr, "\tMSG_NOERROR =\t%#8.8o\n",
                    MSG_NOERROR);
        (void) fprintf(stderr, "\tIPC_NOWAIT =\t%#8.8o\n",
                    IPC_NOWAIT);
        (void) fprintf(stderr, "Enter msgflg: ");
        (void) scanf("%i", &msgflg);
        (void) fprintf(stderr, "%s(%d, msgp, %d, %ld, %#o);\n",
                    "msgop: Calling msgrcv", msqid, msgsz,
                    msgtyp, msgflg);
        rtrn = msgrcv(msqid, msgp, msgsz, msgtyp, msgflg);
        if (rtrn == -1)
            perror("msgop: msgrcv failed");
        else {
            (void) fprintf(stderr, "msgop: %s %d\n",
                        "msgrcv returned", rtrn);
            (void) fprintf(stderr, "msgp->mtype = %ld\n",
                        msgp->mtype);
            (void) fprintf(stderr, "msgp->mtext is: \"");
            for (i = 0; i < rtrn; i++)
```

```
                    (void) fputc(msgp->mtext[i], stderr);
                (void) fprintf(stderr, "\"\n");
            }
            break;
        default:
            (void) fprintf(stderr, "msgop: operation unknown\n");
            break;
        }
    }
    exit(0);
}

/*
 *  Ask the user what to do next. Return the user's choice code.
 *  Don't return until the user selects a valid choice.
 */
static
ask()
{
    int response;/* User's response. */

    do {
        (void) fprintf(stderr, "Your options are:\n");
        (void) fprintf(stderr, "\tExit =\t0 or Control-D\n");
        (void) fprintf(stderr, "\tmsgsnd =\t1\n");
        (void) fprintf(stderr, "\tmsgrcv =\t2\n");
        (void) fprintf(stderr, "Enter your choice: ");

        /* Preset response so "^D" will be interpreted as exit. */
        response = 0;
        (void) scanf("%i", &response);
    } while (response < 0 || response > 2);

    return(response);
}
```

Code Example A-4 Sample Program to Illustrate semget()

```
/*
 * semget.c: Illustrate the semget() function.
 *
 * This is a simple exerciser of the semget() function. It prompts
 * for the arguments, makes the call, and reports the results.
*/

#include    <stdio.h>
```

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

extern void    exit();
extern void    perror();

main()
{
    key_t   key;        /* key to pass to semget() */
    int     semflg;     /* semflg to pass to semget() */
    int     nsems;      /* nsems to pass to semget() */
    int     semid;      /* return value from semget() */

    (void) fprintf(stderr,
        "All numeric input must follow C conventions:\n");
    (void) fprintf(stderr,
        "\t0x... is interpreted as hexadecimal,\n");
    (void) fprintf(stderr, "\t0... is interpreted as octal,\n");
    (void) fprintf(stderr, "\totherwise, decimal.\n");
    (void) fprintf(stderr, "IPC_PRIVATE == %#lx\n", IPC_PRIVATE);
    (void) fprintf(stderr, "Enter key: ");
    (void) scanf("%li", &key);

    (void) fprintf(stderr, "Enter nsems value: ");
    (void) scanf("%i", &nsems);
    (void) fprintf(stderr, "\nExpected flags for semflg are:\n");
    (void) fprintf(stderr, "\tIPC_EXCL = \t%#8.8o\n", IPC_EXCL);
    (void) fprintf(stderr, "\tIPC_CREAT = \t%#8.8o\n", IPC_CREAT);
    (void) fprintf(stderr, "\towner read = \t%#8.8o\n", 0400);
    (void) fprintf(stderr, "\towner alter = \t%#8.8o\n", 0200);
    (void) fprintf(stderr, "\tgroup read = \t%#8.8o\n", 040);
    (void) fprintf(stderr, "\tgroup alter = \t%#8.8o\n", 020);
    (void) fprintf(stderr, "\tother read = \t%#8.8o\n", 04);
    (void) fprintf(stderr, "\tother alter = \t%#8.8o\n", 02);
    (void) fprintf(stderr, "Enter semflg value: ");
    (void) scanf("%i", &semflg);
    (void) fprintf(stderr, "\nsemget: Calling semget(%#lx, %
                %#o)\n",key, nsems, semflg);
    if ((semid = semget(key, nsems, semflg)) == -1) {
        perror("semget: semget failed");
        exit(1);
    } else {
        (void) fprintf(stderr, "semget: semget succeeded: semid =
```

```
%d\n",
            semid);
        exit(0);
    }
}
```

Code Example A-5 Sample Program to Illustrate semctl()

```
/*
 * semctl.c:Illustrate the semctl() function.
 *
 * This is a simple exerciser of the semctl() function. It lets you
 * perform one control operation on one semaphore set. It gives up
 * immediately if any control operation fails, so be careful not to
 * set permissions to preclude read permission; you won't be able to
 * reset the permissions with this code if you do.
 */

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

struct semid_ds semid_ds;

static void do_semctl();
static void do_stat();
extern char *malloc();
extern void exit();
extern void perror();

char            warning_message[] = "If you remove read permission\
                for yourself, this program will fail frequently!";

main()
{
    union semun     arg;            /* union to pass to semctl() */
    int             cmd,            /* command to give to semctl() */
                    i,              /* work area */
                    semid,          /* semid to pass to semctl() */
                    semnum;         /* semnum to pass to semctl() */

    (void) fprintf(stderr,
                "All numeric input must follow C conventions:\n");
    (void) fprintf(stderr,
                "\t0x... is interpreted as hexadecimal,\n");
```

```
(void) fprintf(stderr, "\t0... is interpreted as octal,\n");
(void) fprintf(stderr, "\totherwise, decimal.\n");
(void) fprintf(stderr, "Enter semid value: ");
(void) scanf("%i", &semid);

(void) fprintf(stderr, "Valid semctl cmd values are:\n");
(void) fprintf(stderr, "\tGETALL = %d\n", GETALL);
(void) fprintf(stderr, "\tGETNCNT = %d\n", GETNCNT);
(void) fprintf(stderr, "\tGETPID = %d\n", GETPID);
(void) fprintf(stderr, "\tGETVAL = %d\n", GETVAL);
(void) fprintf(stderr, "\tGETZCNT = %d\n", GETZCNT);
(void) fprintf(stderr, "\tIPC_RMID = %d\n", IPC_RMID);
(void) fprintf(stderr, "\tIPC_SET = %d\n", IPC_SET);
(void) fprintf(stderr, "\tIPC_STAT = %d\n", IPC_STAT);
(void) fprintf(stderr, "\tSETALL = %d\n", SETALL);
(void) fprintf(stderr, "\tSETVAL = %d\n", SETVAL);
(void) fprintf(stderr, "\nEnter cmd: ");
(void) scanf("%i", &cmd);

/* Do some setup operations needed by multiple commands. */
switch (cmd) {
    case GETVAL:
    case SETVAL:
    case GETNCNT:
    case GETZCNT:
        /* Get the semaphore number for these commands. */
        (void) fprintf(stderr, "\nEnter semnum value: ");
        (void) scanf("%i", &semnum);
        break;
    case GETALL:
    case SETALL:
        /* Allocate a buffer for the semaphore values. */
        (void) fprintf(stderr,
            "Get number of semaphores in the set.\n");
        arg.buf = &semid_ds;
        do_semctl(semid, 0, IPC_STAT, arg);
        if (arg.array =
            (ushort *)malloc((unsigned)
                (semid_ds.sem_nsems * sizeof(ushort)))) {
            /* Break out if you got what you needed. */
            break;
        }
        (void) fprintf(stderr,
            "semctl: unable to allocate space for %d values\n",
            semid_ds.sem_nsems);
        exit(2);
```

```
}

/* Get the rest of the arguments needed for the specified
   command. */
switch (cmd) {
    case SETVAL:
        /* Set value of one semaphore. */
        (void) fprintf(stderr, "\nEnter semaphore value: ");
        (void) scanf("%i", &arg.val);
        do_semctl(semid, semnum, SETVAL, arg);
        /* Fall through to verify the result. */
        (void) fprintf(stderr,
            "Do semctl GETVAL command to verify results.\n");
    case GETVAL:
        /* Get value of one semaphore. */
        arg.val = 0;
        do_semctl(semid, semnum, GETVAL, arg);
        break;
    case GETPID:
        /* Get PID of last process to successfully complete a
           semctl(SETVAL), semctl(SETALL), or semop() on the
           semaphore. */
        arg.val = 0;
        do_semctl(semid, 0, GETPID, arg);
        break;
    case GETNCNT:
        /* Get number of processes waiting for semaphore value to
           increase. */
        arg.val = 0;
        do_semctl(semid, semnum, GETNCNT, arg);
        break;
    case GETZCNT:
        /* Get number of processes waiting for semaphore value to
           become zero. */
        arg.val = 0;
        do_semctl(semid, semnum, GETZCNT, arg);
        break;
    case SETALL:
        /* Set the values of all semaphores in the set. */
        (void) fprintf(stderr,
                       "There are %d semaphores in the set.\n",
                       semid_ds.sem_nsems);
        (void) fprintf(stderr, "Enter semaphore values:\n");
        for (i = 0; i < semid_ds.sem_nsems; i++) {
            (void) fprintf(stderr, "Semaphore %d: ", i);
            (void) scanf("%hi", &arg.array[i]);
```

```
    }
    do_semctl(semid, 0, SETALL, arg);
    /* Fall through to verify the results. */
    (void) fprintf(stderr,
        "Do semctl GETALL command to verify results.\n");
case GETALL:
    /* Get and print the values of all semaphores in the
       set.*/
    do_semctl(semid, 0, GETALL, arg);
    (void) fprintf(stderr,
                "The values of the %d semaphores are:\n",
                semid_ds.sem_nsems);
    for (i = 0; i < semid_ds.sem_nsems; i++)
        (void) fprintf(stderr, "%d ", arg.array[i]);
    (void) fprintf(stderr, "\n");
    break;
case IPC_SET:
    /* Modify mode and/or ownership. */
    arg.buf = &semid_ds;
    do_semctl(semid, 0, IPC_STAT, arg);
    (void) fprintf(stderr, "Status before IPC_SET:\n");
    do_stat();
    (void) fprintf(stderr, "Enter sem_perm.uid value: ");
    (void) scanf("%hi", &semid_ds.sem_perm.uid);
    (void) fprintf(stderr, "Enter sem_perm.gid value: ");
    (void) scanf("%hi", &semid_ds.sem_perm.gid);
    (void) fprintf(stderr, "%s\n", warning_message);
    (void) fprintf(stderr, "Enter sem_perm.mode value: ");
    (void) scanf("%hi", &semid_ds.sem_perm.mode);
    do_semctl(semid, 0, IPC_SET, arg);
    /* Fall through to verify changes. */
    (void) fprintf(stderr, "Status after IPC_SET:\n");
case IPC_STAT:
    /* Get and print current status. */
    arg.buf = &semid_ds;
    do_semctl(semid, 0, IPC_STAT, arg);
    do_stat();
    break;
case IPC_RMID:
    /* Remove the semaphore set. */
    arg.val = 0;
    do_semctl(semid, 0, IPC_RMID, arg);
    break;
default:
    /* Pass unknown command to semctl. */
    arg.val = 0;
```

```
            do_semctl(semid, 0, cmd, arg);
            break;
    }
    exit(0);
}

/*
 * Print indication of arguments being passed to semctl(), call
 * semctl(), and report the results. If semctl() fails, do not
 * return; this example doesn't deal with errors, it just reports
 * them.
 */
static void
do_semctl(semid, semnum, cmd, arg)
union semun arg;
int     cmd,
        semid,
        semnum;
{
    register int          i;         /* work area */

    void) fprintf(stderr, "\nsemctl: Calling semctl(%d, %d, %d, ",
            semid, semnum, cmd);
    switch (cmd) {
        case GETALL:
            (void) fprintf(stderr, "arg.array = %#x)\n",
                           arg.array);
            break;
        case IPC_STAT:
        case IPC_SET:
            (void) fprintf(stderr, "arg.buf = %#x)\n", arg.buf);
            break;
        case SETALL:
            (void) fprintf(stderr, "arg.array = [", arg.buf);
            for (i = 0;i < semid_ds.sem_nsems;) {
                (void) fprintf(stderr, "%d", arg.array[i++]);
                if (i < semid_ds.sem_nsems)
                        (void) fprintf(stderr, ", ");
            }
            (void) fprintf(stderr, "])\n");
            break;
        case SETVAL:
        default:
            (void) fprintf(stderr, "arg.val = %d)\n", arg.val);
            break;
    }
```

```
    i = semctl(semid, semnum, cmd, arg);
    if (i == -1) {
        perror("semctl: semctl failed");
        exit(1);
    }
    (void) fprintf(stderr, "semctl: semctl returned %d\n", i);
    return;
}

/*
 *  Display contents of commonly used pieces of the status structure.
 */
static void
do_stat()
{
    (void) fprintf(stderr, "sem_perm.uid = %d\n",
                        semid_ds.sem_perm.uid);
    (void) fprintf(stderr, "sem_perm.gid = %d\n",
                        semid_ds.sem_perm.gid);
    (void) fprintf(stderr, "sem_perm.cuid = %d\n",
                        semid_ds.sem_perm.cuid);
    (void) fprintf(stderr, "sem_perm.cgid = %d\n",
                        semid_ds.sem_perm.cgid);
    (void) fprintf(stderr, "sem_perm.mode = %#o, ",
                        semid_ds.sem_perm.mode);
    (void) fprintf(stderr, "access permissions = %#o\n",
                        semid_ds.sem_perm.mode & 0777);
    (void) fprintf(stderr, "sem_nsems = %d\n", semid_ds.sem_nsems);
    (void) fprintf(stderr, "sem_otime = %s", semid_ds.sem_otime ?
                    ctime(&semid_ds.sem_otime) : "Not Set\n");
    (void) fprintf(stderr, "sem_ctime = %s",
                        ctime(&semid_ds.sem_ctime));
}
```

Code Example A-6 Sample Program to Illustrate semop()

```
/*
 * semop.c: Illustrate the semop() function.
 *
 * This is a simple exerciser of the semop() function. It lets you
 * to set up arguments for semop() and make the call. It then reports
 * the results repeatedly on one semaphore set. You must have read
 * permission on the semaphore set or this exerciser will fail. (It
 * needs read permission to get the number of semaphores in the set
 * and to report the values before and after calls to semop().)
 */
```

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

static int          ask();
extern void    exit();
extern void     free();
extern char     *malloc();
extern void    perror();

static struct semid_dssemid_ds;      /* status of semaphore set */

static char    error_mesg1[] = "semop: Can't allocate space for %d\
                              semaphore values. Giving up.\n";
static charerror_mesg2[] = "semop: Can't allocate space for %d\
                              sembuf structures. Giving up.\n";

main()
{
    register int   i;          /* work area */
    int            nsops;      /* number of operations to do */
    int            semid;      /* semid of semaphore set */
    struct sembuf  *sops;      /* ptr to operations to perform */

    (void) fprintf(stderr,
               "All numeric input must follow C conventions:\n");
    (void) fprintf(stderr,
               "\t0x... is interpreted as hexadecimal,\n");
    (void) fprintf(stderr, "\t0... is interpreted as octal,\n");
    (void) fprintf(stderr, "\totherwise, decimal.\n");
    /* Loop until the invoker doesn't want to do anymore. */
    while (nsops = ask(&semid, &sops)) {
        /* Initialize the array of operations to be performed.*/
        for (i = 0; i < nsops; i++) {
            (void) fprintf(stderr,
                   "\nEnter values for operation %d of %d.\n",
                           i + 1, nsops);
            (void) fprintf(stderr,
                   "sem_num(valid values are 0 <= sem_num < %d): ",
                   semid_ds.sem_nsems);
            (void) scanf("%hi", &sops[i].sem_num);
            (void) fprintf(stderr, "sem_op: ");
            (void) scanf("%hi", &sops[i].sem_op);
            (void) fprintf(stderr,
                   "Expected flags in sem_flg are:\n");
```

```
            (void) fprintf(stderr, "\tIPC_NOWAIT =\t%#6.6o\n",
                    IPC_NOWAIT);
            (void) fprintf(stderr, "\tSEM_UNDO =\t%#6.6o\n",
                    SEM_UNDO);
            (void) fprintf(stderr, "sem_flg: ");
            (void) scanf("%hi", &sops[i].sem_flg);
        }

        /* Recap the call to be made. */
        (void) fprintf(stderr,
                    "\nsemop: Calling semop(%d, &sops, %d) with:",
                    semid, nsops);
        for (i = 0; i < nsops; i++)
        {
            (void) fprintf(stderr, "\nsops[%d].sem_num = %d, ", i,
                        sops[i].sem_num);
            (void) fprintf(stderr, "sem_op = %d, ", sops[i].sem_op);
            (void) fprintf(stderr, "sem_flg = %#o\n",
                        sops[i].sem_flg);
        }

        /* Make the semop() call and report the results. */
        if ((i = semop(semid, sops, nsops)) == -1) {
            perror("semop: semop failed");
        } else {
            (void) fprintf(stderr, "semop: semop returned %d\n", i);
        }
    }
}

/*
 * Ask if user wants to continue.
 *
 * On the first call:
 *  Get the semid to be processed and supply it to the caller.
 * On each call:
 *  1. Print current semaphore values.
 *  2. Ask user how many operations are to be performed on the next
 *     call to semop. Allocate an array of sembuf structures
 *    sufficient for the job and set caller-supplied pointer to that
 *     array. (The array is reused on subsequent calls if it is big
 *     enough. If it isn't, it is freed and a larger array is
 *     allocated.)
 */
static
ask(semidp, sopsp)
```

```
int         *semidp;/* pointer to semid (used only the first time) */
struct sembuf**sopsp;
{
    static union semun arg;    /* argument to semctl */
    int                 i;     /* work area */
    static int          nsops = 0;/* size of currently allocated
                                   sembuf array */
    static int          semid = -1;/* semid supplied by user */
    static struct sembuf*sops;    /* pointer to allocated array */

    if (semid < 0) {
        /* First call; get semid from user and the current state of
           the semaphore set. */
        (void) fprintf(stderr,
               "Enter semid of the semaphore set you want to use: ");
        (void) scanf("%i", &semid);
        *semidp = semid;
        arg.buf = &semid_ds;
        if (semctl(semid, 0, IPC_STAT, arg) == -1) {
            perror("semop: semctl(IPC_STAT) failed");
            /* Note that if semctl fails, semid_ds remains filled
              with zeros, so later test for number of semaphores will
               be zero. */
            (void) fprintf(stderr,
                   "Before and after values are not printed.\n");
        } else {
            if ((arg.array = (ushort *)malloc(
                (unsigned)(sizeof(ushort) * semid_ds.sem_nsems)))
                           == NULL) {
                (void) fprintf(stderr, error_mesg1,
                           semid_ds.sem_nsems);
                exit(1);
            }
        }
    }
    /* Print current semaphore values. */
    if (semid_ds.sem_nsems) {
        (void) fprintf(stderr,
                       "There are %d semaphores in the set.\n",
                       semid_ds.sem_nsems);
        if (semctl(semid, 0, GETALL, arg) == -1) {
            perror("semop: semctl(GETALL) failed");
        } else {
            (void) fprintf(stderr, "Current semaphore values are:");
            for (i = 0; i < semid_ds.sem_nsems;
                (void) fprintf(stderr, " %d", arg.array[i++]));
```

```
            (void) fprintf(stderr, "\n");
        }
    }
    /* Find out how many operations are going to be done in the next
       call and allocate enough space to do it. */
    (void) fprintf(stderr,
                "How many semaphore operations do you want %s\n",
                "on the next call to semop()?");
    (void) fprintf(stderr, "Enter 0 or control-D to quit: ");
    i = 0;
    if (scanf("%i", &i) == EOF || i == 0)
        exit(0);
    if (i > nsops) {
        if (nsops)
            free((char *)sops);
        nsops = i;
        if ((sops = (struct sembuf *)malloc((unsigned)(nsops *
            sizeof(struct sembuf)))) == NULL) {
            (void) fprintf(stderr, error_mesg2, nsops);
            exit(2);
        }
    }
    *sopsp = sops;
    return (i);
}
```

Code Example A-7 Sample Program to Illustrate `shmget()`

```
/*
 * shmget.c: Illustrate the shmget() function.
 *
 *  This is a simple exerciser of the shmget() function. It prompts
 * for the arguments, makes the call, and reports the results.
 */

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

extern void    exit();
extern void    perror();

main()
{
    key_t   key;        /* key to be passed to shmget() */
    int     shmflg;     /* shmflg to be passed to shmget() */
```

```
    int     shmid;      /* return value from shmget() */
    int     size;       /* size to be passed to shmget() */

    (void) fprintf(stderr,
        "All numeric input is expected to follow C conventions:\n");
    (void) fprintf(stderr,
                "\t0x... is interpreted as hexadecimal,\n");
    (void) fprintf(stderr, "\t0... is interpreted as octal,\n");
    (void) fprintf(stderr, "\totherwise, decimal.\n");

    /* Get the key. */
    (void) fprintf(stderr, "IPC_PRIVATE == %#lx\n", IPC_PRIVATE);
    (void) fprintf(stderr, "Enter key: ");
    (void) scanf("%li", &key);

    /* Get the size of the segment. */
    (void) fprintf(stderr, "Enter size: ");
    (void) scanf("%i", &size);

    /* Get the shmflg value. */
    (void) fprintf(stderr,
                "Expected flags for the shmflg argument are:\n");
    (void) fprintf(stderr, "\tIPC_CREAT = \t%#8.8o\n", IPC_CREAT);
    (void) fprintf(stderr, "\tIPC_EXCL = \t%#8.8o\n", IPC_EXCL);
    (void) fprintf(stderr, "\towner read =\t%#8.8o\n", 0400);
    (void) fprintf(stderr, "\towner write =\t%#8.8o\n", 0200);
    (void) fprintf(stderr, "\tgroup read =\t%#8.8o\n", 040);
    (void) fprintf(stderr, "\tgroup write =\t%#8.8o\n", 020);
    (void) fprintf(stderr, "\tother read =\t%#8.8o\n", 04);
    (void) fprintf(stderr, "\tother write =\t%#8.8o\n", 02);
    (void) fprintf(stderr, "Enter shmflg: ");
    (void) scanf("%i", &shmflg);

    /* Make the call and report the results. */
    (void) fprintf(stderr,
                    "shmget: Calling shmget(%#lx, %d, %#o)\n",
                    key, size, shmflg);
    if ((shmid = shmget (key, size, shmflg)) == -1) {
        perror("shmget: shmget failed");
        exit(1);
    } else {
        (void) fprintf(stderr,
                    "shmget: shmget returned %d\n", shmid);
        exit(0);
    }
}
```

Code Example A-8 Sample Program to Illustrate shmctl()

```
/*
 * shmctl.c: Illustrate the shmctl() function.
 *
 * This is a simple exerciser of the shmctl() function. It lets you
 * to perform one control operation on one shared memory segment.
 * (Some operations are done for the user whether requested or not.
 * It gives up immediately if any control operation fails. Be careful
 * not to set permissions to preclude read permission; you won't be
 *able to reset the permissions with this code if you do.)
*/

#include    <stdio.h>
#include    <sys/types.h>
#include    <sys/ipc.h>
#include    <sys/shm.h>
#include    <time.h>
static voiddo_shmctl();
extern voidexit();
extern voidperror();

main()
{
    int     cmd;    /* command code for shmctl() */
    int     shmid;  /* segment ID */
    struct shmid_dsshmid_ds;   /* shared memory data structure to
                                  hold results */

    (void) fprintf(stderr,
        "All numeric input is expected to follow C conventions:\n");
    (void) fprintf(stderr,
                "\t0x... is interpreted as hexadecimal,\n");
    (void) fprintf(stderr, "\t0... is interpreted as octal,\n");
    (void) fprintf(stderr, "\totherwise, decimal.\n");

    /* Get shmid and cmd. */
    (void) fprintf(stderr,
                "Enter the shmid for the desired segment: ");
    (void) scanf("%i", &shmid);
    (void) fprintf(stderr, "Valid shmctl cmd values are:\n");
    (void) fprintf(stderr, "\tIPC_RMID =\t%d\n", IPC_RMID);
    (void) fprintf(stderr, "\tIPC_SET =\t%d\n", IPC_SET);
    (void) fprintf(stderr, "\tIPC_STAT =\t%d\n", IPC_STAT);
    (void) fprintf(stderr, "\tSHM_LOCK =\t%d\n", SHM_LOCK);
    (void) fprintf(stderr, "\tSHM_UNLOCK =\t%d\n", SHM_UNLOCK);
```

```
    (void) fprintf(stderr, "Enter the desired cmd value: ");
    (void) scanf("%i", &cmd);

    switch (cmd) {
        case IPC_STAT:
            /* Get shared memory segment status. */
            break;
        case IPC_SET:
            /* Set owner UID and GID and permissions. */
            /* Get and print current values. */
            do_shmctl(shmid, IPC_STAT, &shmid_ds);
            /* Set UID, GID, and permissions to be loaded. */
            (void) fprintf(stderr, "\nEnter shm_perm.uid: ");
            (void) scanf("%hi", &shmid_ds.shm_perm.uid);
            (void) fprintf(stderr, "Enter shm_perm.gid: ");
            (void) scanf("%hi", &shmid_ds.shm_perm.gid);
            (void) fprintf(stderr,
                "Note: Keep read permission for yourself.\n");
            (void) fprintf(stderr, "Enter shm_perm.mode: ");
            (void) scanf("%hi", &shmid_ds.shm_perm.mode);
            break;
        case IPC_RMID:
            /* Remove the segment when the last attach point is
               detached. */
            break;
        case SHM_LOCK:
            /* Lock the shared memory segment. */
            break;
        case SHM_UNLOCK:
            /* Unlock the shared memory segment. */
            break;
        default:
            /* Unknown command will be passed to shmctl. */
            break;
    }
    do_shmctl(shmid, cmd, &shmid_ds);
    exit(0);
}

/*
 * Display the arguments being passed to shmctl(), call shmctl(),
 * and report the results. If shmctl() fails, do not return; this
 * example doesn't deal with errors, it just reports them.
 */
static void
do_shmctl(shmid, cmd, buf)
```

```
int           shmid,      /* attach point */
              cmd;        /* command code */
struct shmid_ds*buf;      /* pointer to shared memory data structure */
{
    register int    rtrn;   /* hold area */

    (void) fprintf(stderr, "shmctl: Calling shmctl(%d, %d, buf)\n",
        shmid, cmd);
    if (cmd == IPC_SET) {
        (void) fprintf(stderr, "\tbuf->shm_perm.uid == %d\n",
                buf->shm_perm.uid);
        (void) fprintf(stderr, "\tbuf->shm_perm.gid == %d\n",
                buf->shm_perm.gid);
        (void) fprintf(stderr, "\tbuf->shm_perm.mode == %#o\n",
                buf->shm_perm.mode);
    }
    if ((rtrn = shmctl(shmid, cmd, buf)) == -1) {
        perror("shmctl: shmctl failed");
        exit(1);
    } else {
        (void) fprintf(stderr,
                    "shmctl: shmctl returned %d\n", rtrn);
    }
    if (cmd != IPC_STAT && cmd != IPC_SET)
        return;

    /* Print the current status. */
    (void) fprintf(stderr, "\nCurrent status:\n");
    (void) fprintf(stderr, "\tshm_perm.uid = %d\n",
                    buf->shm_perm.uid);
    (void) fprintf(stderr, "\tshm_perm.gid = %d\n",
                    buf->shm_perm.gid);
    (void) fprintf(stderr, "\tshm_perm.cuid = %d\n",
                    buf->shm_perm.cuid);
    (void) fprintf(stderr, "\tshm_perm.cgid = %d\n",
                    buf->shm_perm.cgid);
    (void) fprintf(stderr, "\tshm_perm.mode = %#o\n",
                    buf->shm_perm.mode);
    (void) fprintf(stderr, "\tshm_perm.key = %#x\n",
                    buf->shm_perm.key);
    (void) fprintf(stderr, "\tshm_segsz = %d\n", buf->shm_segsz);
    (void) fprintf(stderr, "\tshm_lpid = %d\n", buf->shm_lpid);
    (void) fprintf(stderr, "\tshm_cpid = %d\n", buf->shm_cpid);
    (void) fprintf(stderr, "\tshm_nattch = %d\n", buf->shm_nattch);
    (void) fprintf(stderr, "\tshm_atime = %s",
        buf->shm_atime ? ctime(&buf->shm_atime) : "Not Set\n");
```

```
    (void) fprintf(stderr, "\tshm_dtime = %s",
           buf->shm_dtime ? ctime(&buf->shm_dtime) : "Not Set\n");
    (void) fprintf(stderr, "\tshm_ctime = %s",
                    ctime(&buf->shm_ctime));
}
```

Code Example A-9 Sample Program to Illustrate `shmat()` and `shmdt()`

```
/*
 * shmop.c: Illustrate the shmat() and shmdt() functions.
 *
 * This is a simple exerciser for the shmat() and shmdt() system
 * calls. It allows you to attach and detach segments and to
 * write strings into and read strings from attached segments.
 */

#include    <stdio.h>
#include    <setjmp.h>
#include    <signal.h>
#include    <sys/types.h>
#include    <sys/ipc.h>
#include    <sys/shm.h>

#define     MAXnap  4   /* Maximum number of concurrent attaches. */

static      ask();
static voidcatcher();
extern voidexit();
static      good_addr();
extern voidperror();
extern char*shmat();

static struct state{   /* Internal record of currently attached
segments. */
    int     shmid;      /* shmid of attached segment */
    char    *shmaddr;   /* attach point */
    int     shmflg;     /* flags used on attach */
}   ap[MAXnap];         /* State of current attached segments. */

static int     nap;    /* Number of currently attached segments. */
static jmp_bufsegvbuf; /* Process state save area for SIGSEGV
                          catching. */

main()
{
    register int   action;     /* action to be performed */
    char           *addr;      /* address work area */
```

```
register int   i;          /* work area */
register struct state*p;   /* ptr to current state entry */
void       (*savefunc)();/* SIGSEGV state hold area */
(void) fprintf(stderr,
    "All numeric input is expected to follow C conventions:\n");
(void) fprintf(stderr,
    "\t0x... is interpreted as hexadecimal,\n");
(void) fprintf(stderr, "\t0... is interpreted as octal,\n");
(void) fprintf(stderr, "\totherwise, decimal.\n");
while (action = ask()) {
    if (nap) {
        (void) fprintf(stderr,
                    "\nCurrently attached segment(s):\n");
        (void) fprintf(stderr, " shmid address\n");
        (void) fprintf(stderr, "------ ----------\n");
        p = &ap[nap];
        while (p-- != ap) {
            (void) fprintf(stderr, "%6d", p->shmid);
            (void) fprintf(stderr, "%#11x", p->shmaddr);
            (void) fprintf(stderr, " Read%s\n",
                (p->shmflg & SHM_RDONLY) ?
                "-Only" : "/Write");
        }
    } else
        (void) fprintf(stderr,
            "\nNo segments are currently attached.\n");
    switch (action) {
    case 1:     /* Shmat requested. */
        /* Verify that there is space for another attach. */
        if (nap == MAXnap) {
            (void) fprintf(stderr, "%s %d %s\n",
                    "This simple example will only allow",
                    MAXnap, "attached segments.");
            break;
        }
        p = &ap[nap++];
        /* Get the arguments, make the call, report the
            results, and update the current state array. */
        (void) fprintf(stderr,
            "Enter shmid of segment to attach: ");
        (void) scanf("%i", &p->shmid);

        (void) fprintf(stderr, "Enter shmaddr: ");
        (void) scanf("%i", &p->shmaddr);
        (void) fprintf(stderr,
            "Meaningful shmflg values are:\n");
```

```
    (void) fprintf(stderr, "\tSHM_RDONLY = \t%#8.8o\n",
        SHM_RDONLY);
    (void) fprintf(stderr, "\tSHM_RND = \t%#8.8o\n",
        SHM_RND);
    (void) fprintf(stderr, "Enter shmflg value: ");
    (void) scanf("%i", &p->shmflg);

    (void) fprintf(stderr,
        "shmop: Calling shmat(%d, %#x, %#o)\n",
        p->shmid, p->shmaddr, p->shmflg);
    p->shmaddr = shmat(p->shmid, p->shmaddr, p->shmflg);
    if(p->shmaddr == (char *)-1) {
        perror("shmop: shmat failed");
        nap--;
    } else {
        (void) fprintf(stderr,
            "shmop: shmat returned %#8.8x\n",
            p->shmaddr);
    }
    break;

case 2:     /* Shmdt requested. */
    /* Get the address, make the call, report the results,
        and make the internal state match. */
    (void) fprintf(stderr,
        "Enter detach shmaddr: ");
    (void) scanf("%i", &addr);

    i = shmdt(addr);
    if(i == -1) {
        perror("shmop: shmdt failed");
    } else {
        (void) fprintf(stderr,
            "shmop: shmdt returned %d\n", i);
        for (p = ap, i = nap; i--; p++) {
            if (p->shmaddr == addr)
                *p = ap[--nap];
        }
    }
    break;
case 3:/* Read from segment requested. */
    if (nap == 0)
        break;

    (void) fprintf(stderr, "Enter address of an %s",
        "attached segment: ");
```

```
                (void) scanf("%i", &addr);

                if (good_addr(addr))
                    (void) fprintf(stderr, "String @ %#x is `%s'\n",
                        addr, addr);
                break;

            case 4:/* Write to segment requested. */
                if (nap == 0)
                    break;

                (void) fprintf(stderr, "Enter address of an %s",
                    "attached segment: ");
                (void) scanf("%i", &addr);

                /* Set up SIGSEGV catch routine to trap attempts to
                    write into a read-only attached segment. */
                savefunc = signal(SIGSEGV, catcher);

                if (setjmp(segvbuf)) {
                    (void) fprintf(stderr, "shmop: %s: %s\n",
                        "SIGSEGV signal caught",
                        "Write aborted.");
                } else {
                    if (good_addr(addr)) {
                        (void) fflush(stdin);
                        (void) fprintf(stderr, "%s %s %#x:\n",
                            "Enter one line to be copied",
                            "to shared segment attached @",
                            addr);
                        (void) gets(addr);
                    }
                }
                (void) fflush(stdin);

                /* Restore SIGSEGV to previous condition. */
                (void) signal(SIGSEGV, savefunc);
                break;
            }
        }
        exit(0);
        /*NOTREACHED*/
    }
    /*
    **  Ask for next action.
    */
```

```
static
ask()
{
    int     response;   /* user response */
    do {
            (void) fprintf(stderr, "Your options are:\n");
            (void) fprintf(stderr, "\t^D = exit\n");
            (void) fprintf(stderr, "\t 0 = exit\n");
            (void) fprintf(stderr, "\t 1 = shmat\n");
            (void) fprintf(stderr, "\t 2 = shmdt\n");
            (void) fprintf(stderr, "\t 3 = read from segment\n");
            (void) fprintf(stderr, "\t 4 = write to segment\n");
            (void) fprintf(stderr,
                "Enter the number corresponding to your choice: ");

            /* Preset response so "^D" will be interpreted as exit.
*/
            response = 0;
            (void) scanf("%i", &response);
    } while (response < 0 || response > 4);
    return (response);
}
/*
** Catch signal caused by attempt to write into shared memory segment
** attached with SHM_RDONLY flag set.
*/
/*ARGSUSED*/
static void
catcher(sig)
{
    longjmp(segvbuf, 1);
    /*NOTREACHED*/
}
/*
** Verify that given address is the address of an attached segment.
** Return 1 if address is valid; 0 if not.
*/
static
good_addr(address)
char*address;
{
    register struct state            *p;/* ptr to state of attached
segment */

    for (p = ap; p != &ap[nap]; p++)
            if (p->shmaddr == address)
```

```
                return(1);
    return(0);
}
```

Code Example A-10 Example of Record Locking With Lock Promotion

The next example demonstrates inserting an entry into a doubly linked list that is stored in a file of list element records. For the example, assume that the record after which the new record is to be inserted has a read lock on it already. The lock on this record must be changed or promoted to a write lock so that the record can be edited.

Promoting a lock (generally from read lock to write lock) is permitted if no other process is holding a read lock in the same section of the file. When processes with pending write locks are sleeping on the same section of the file, the lock promotion succeeds and the other (sleeping) locks wait. Changing a write lock to a read lock carries no restrictions. In either case, the lock is merely reset with the new lock type. Because the `lockf` function does not have read locks, lock promotion does not apply to that call.

The locks on these three records were all set to wait (sleep) if another process was blocking them from being set. This was done with the `F_SETLKW` command. If the `F_SETLK` command were used instead, the `fcntl` functions would fail if blocked. The program would then have to be changed to handle the blocked condition in each of the error-return sections.

```
struct record {
    ...             /* data portion of record */
    off_t prev;     /* index to previous record in the list */
    off_t next;     /* index to next record in the list */
};

/* Lock promotion using fcntl(2): When this routine is entered it is
 * assumed that there are read locks on "here" and "next." If write
 * locks on "here" and "next" are obtained;
 *      Set a write lock on "this."
 *      Return index to "this" record.
 * If any write lock is not obtained;
 *      Restore read locks on "here" and "next."
 *      Remove all other locks.
 *      Return a -1.
 */

off_t
set3lock (this, here, next)
off_t this, here, next;
```

```
{
    struct flock lck;
    lck.l_type = F_WRLCK;/* setting a write lock */
    lck.l_whence = 0;/* offset l_start from beginning of file */
    lck.l_start = here;
    lck.l_len = sizeof(struct record);

    /* promote lock on "here" to write lock */
    if (fcntl(fd, F_SETLKW, &lck) < 0) {
            return (-1);
    }
    /* lock "this" with write lock */
    lck.l_start = this;
    if (fcntl(fd, F_SETLKW, &lck) < 0) {
        /* "this" lock failed; demote "here" lock to read lock. */
        lck.l_type = F_RDLCK;
        lck.l_start = here;
        (void) fcntl(fd, F_SETLKW, &lck);
        return (-1);
    /* promote lock on "next" to write lock */
    lck.l_start = next;
    if (fcntl(fd, F_SETLKW, &lck) < 0) {
        /* "next" lock failed; demote lock on "here" to read lock, */
        lck.l_type = F_RDLCK;
        lck.l_start = here;
        (void) fcntl(fd, F_SETLK, &lck);
        /* and remove lock on "this". */
        lck.l_type = F_UNLCK;
        lck.l_start = this;
        (void) fcntl(fd, F_SETLK, &lck);
        return (-1);/* cannot set lock, try again or quit */
    }

    return (this);
}
```

Code Example A-11 Record Write Locks With `lockf()`

```
/* lockf(3C)
 * When this routine is entered, it is assumed that there are no
 * locks on "here" and "next". If locks are obtained: set a lock
 * on "this"; return index to "this" record. If any lock is not
 * obtained: remove all other locks; return a -1.
 */
#include <unistd.h>

long
```

```
set3lock (this, here, next)
long this, here, next;
{
    /* lock "here" */
    (void) lseek(fd, here, 0);
    if (lockf(fd, F_LOCK, sizeof(struct record)) < 0) {
        return (-1);
    }
    /* lock "this" */
    (void) lseek(fd, this, SEEK_SET);
    if (lockf(fd, F_LOCK, sizeof(struct record)) < 0) {
        /* Lock on "this" failed. Clear lock on "here". */
        (void) lseek(fd, here, 0);
        (void) lockf(fd, F_ULOCK, sizeof(struct record));
        return (-1);
    }
    /* lock "next" */
    (void) lseek(fd, next, 0);
    if (lockf(fd, F_LOCK, sizeof(struct record)) < 0) {
        /* Lock on "next" failed. Clear lock on "here". */
        (void) lseek(fd, here, 0);
        (void) lockf(fd, F_ULOCK, sizeof(struct record));
        /* and remove lock on "this". */
        (void) lseek(fd, this, 0);
        (void) lockf(fd, F_ULOCK, sizeof(struct record));
        return (-1);   /* cannot set lock, try again or quit */
    }
    return (this);
}
```