Chapter 8

IPC - II

Chapter Objectives

To understand concepts of IPC used over the duration of the course.

Objectives

For this chapter, the following are the objectives:
- Message Queues,
- Shared Memory,
- Semaphores

Slide #8-1

Notes

In this chapter, we examine System V ("System Five") inter-process communication, mechanisms that are widely used.

The objective of this chapter is to provide an understanding of concepts on System V IPC that are used over the duration of the course.

Chapter Organization

1. **Objective**: Introduction to
   - message queues,
   - shared memory,
   - semaphores

2. **Description**: These are the various tools needed for systems programming
   This chapter provides an introduction to the concepts used in course.

3. **Concepts Covered in Chapter**:
   - Introduction to various tools.
   - LINUX Notes

4. **Prior Knowledge**:
   same as Chapter #1

5. **Teaching & Learning Strategy**:
   Discussion questions are,
   - What are these tools for?

6. **Teaching Format**:
   Theory + Homework Assignments

7. **Study Time**: 120 Minutes (Lecture & Theory)
   + ~45 minutes (Homework Assignments)

8. **Assessment**: Group Homework Assignments

9. **Homework Eval**: Group

10. **Chapter References**:

getopt

getopt

- Parse command line options
- Widely Used .. Portable

Slide #8-2

Notes

getopt – Basic Concepts

UNIX provides a command line parser.

Code example (sourced from http://www.die.net/doc/linux/man/man3/getopt.3.html)

```
#include <stdio.h>     /* for printf */
#include <stdlib.h>    /* for exit */
#include <getopt.h>

int
main (int argc, char **argv) {
    int c;
    int digit_optind = 0;

    while (1) {
        int this_option_optind = optind ? optind : 1;
        int option_index = 0;
        static struct option long_options[] = {
            {"add", 1, 0, 0},
            {"append", 0, 0, 0},
```

```
            {"delete", 1, 0, 0},
            {"verbose", 0, 0, 0},
            {"create", 1, 0, 'c'},
            {"file", 1, 0, 0},
            {0, 0, 0, 0}
        };

        c = getopt_long (argc, argv, "abc:d:012",
                 long_options, &option_index);
        if (c == -1)
            break;

        switch (c) {
        case 0:
            printf ("option %s",
               long_options[option_index].name);
            if (optarg)
                printf (" with arg %s", optarg);
            printf ("\n");
            break;

        case '0':
        case '1':
        case '2':
            if (digit_optind != 0 && digit_optind !=
this_option_optind)
              printf ("digits occur in two different argv-
elements.\n");
            digit_optind = this_option_optind;
            printf ("option %c\n", c);
            break;

        case 'a':
            printf ("option a\n");
            break;

        case 'b':
            printf ("option b\n");
            break;

        case 'c':
            printf ("option c with value '%s'\n", optarg);
            break;

        case 'd':
            printf ("option d with value '%s'\n", optarg);
            break;

        case '?':
```

```
            break;

        default:
            printf ("?? getopt returned character code 
0%o ??\n", c);
        }
    }

    if (optind < argc) {
        printf ("non-option ARGV-elements: ");
        while (optind < argc)
            printf ("%s ", argv[optind++]);
        printf ("\n");
    }

    exit (0);
}
```

System V -- IPC

System V - IPC

- Widely Used .. Portable
- IPC between unrelated processes

Slide #8-3

Notes

System V IPC – Basic Concepts

1. UNIX provides a rich set of features for unrelated processes to communicate with each other. Inter-Process Communication (IPC), can be used to share data between unrelated processes running simultaneously.
    1.1. It is widely used, and is widely available.
    1.2. Portable across most UNIX systems.
2. Linux has System V IPC mechanisms for several years.

System V - IPC

- Shared Memory
- Message Queues
- Semaphores

Slide #8-4

System V IPC – Basic Concepts

1. The System V (SysV) UNIX specification describes three mechanisms for IPC, collectively referred to as SysV IPC:
   1.1. Message queues
   1.2. Semaphores
   1.3. Shared memory

System V - IPC

The Key

- Can use a hard coded value, or,
- Generate a key using `ftok()`

Slide #8-5

System V IPC – Basic Concepts

1. Hard Coded value, or,
2. Creating the key

```
key = ftok("./ch7.c", 'R');
```

Shared Memory

System V - Shared Memory

Memory
- Shared by multiple processes
- Persists till removed

Slide #8-6

Notes

Basic Concepts – Shared Memory Segments

1. Shared memory segments are segments of memory that is shared between processes.
2. The steps:
   2.1. Create and connect to the shared memory segment, and get a pointer
   2.2. Attach to segment
   2.3. Read and Write to this pointer and all changes are visible to anyone else connected to the segment.
   2.4. Creating the segment and connecting.
3. Creating and connecting to the shared segment

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

where `shmget()` returns an identifier for the shared memory segment,

key	is hardcoded or created using ftok()
size	size in bytes of shared memory segment.
shmflg	permissions of segment - OR'd with `IPC_CREAT` to create segment. - can be 0.

- `IPC_CREAT` can always be specified.
  It will connect, if the segment already exists.

Here's an example to create a 4K segment with `644` permissions (`rw-r--r--`):

```
key_t key = 0xFEED4BBC;

int shmid;

shmid = shmget( key,  1024,  0644  |  IPC_CREAT);
```

4. Attaching to the shared segment

   `shmat()` gets a pointer to that data from the `shmid` handle.

   Before using a shared memory segment, it needs to be attached using the `shmat()` call:

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

where		
	`shmget()`	returns an identifier for the shared memory segment,
	key	is hardcoded or created using ftok()
	size	size in bytes of shared memory segment.
	shmflg	permissions of segment - OR'd with `IPC_CREAT` to create segment. - can be 0.

5. Detaching and deleting shared segments

```
int shmdt(void *shmaddr);
```

   shmaddr, is the address you got from `shmat()`.
   The function returns `-1` on error, `0` on success.

6. Sample Code (Sourced From http://www.ecst.csuchico.edu/~beej/guide/ipc/shmem.html)

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

#define SHM_SIZE 1024  /* make it a 1K shared memory segment */

int main(int argc, char *argv[])
{
    key_t key;
    int shmid;
    char *data;
```

```
    int mode;

    if (argc > 2) {
        fprintf(stderr, "usage: shmdemo [data_to_write]\n");
        exit(1);
    }

    /* make the key: */
    if ((key = ftok("shmdemo.c", 'R')) == -1) {
        perror("ftok");
        exit(1);
    }

    /* connect to (and possibly create) the segment: */
    if ((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT)) == -1) {
        perror("shmget");
        exit(1);
    }

    /* attach to the segment to get a pointer to it: */
    data = shmat(shmid, (void *)0, 0);
    if (data == (char *)(-1)) {
        perror("shmat");
        exit(1);
    }

    /* read or modify the segment, based on the command line: */
    if (argc == 2) {
        printf("writing to segment: \"%s\"\n", argv[1]);
        strncpy(data, argv[1], SHM_SIZE);
    } else
        printf("segment contains: \"%s\"\n", data);

    /* detach from the segment: */
    if (shmdt(data) == -1) {
        perror("shmdt");
        exit(1);
    }

    return 0;
}
```

semaphores

semaphore

- Concurrency control mechanism

Slide #8-7

Notes

Basic Concepts – Semaphores

1. Semaphore is a lock.
   It is used for concurrency control.
   1.1. Create semaphore set using `semget()`
   1.2. and connect to the semaphore set, and get a pointer
   1.3. Lock, and access
   or, Wait, till semaphore lock is released
   1.4. Unlock

2. The steps:
   http://www.ecst.csuchico.edu/~beej/guide/ipc/semaphores.html

semget()

2.1. Creates a semaphore set

2.2. Returns an *id* to a semaphore set.
A semaphore set can contain one or many semaphores.

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

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

`key` is a unique identifier used by different processes to identify this semaphore set.

`nsems`, is the number of semaphores in this semaphore set.

`semflg` is the permissions on the new semaphore set

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

key_t key;
int semid;

key = ftok("somefile", 'E');
semid = semget(key, 10, 0666 | IPC_CREAT);
```

semop()

2.3. set, get, or test-n-set a semaphore

2.4. functionality depends on `struct sembuf`

```
unsigned short sem_num;  /* semaphore number */
short          sem_op;   /* semaphore operation */
short          sem_flg;  /* operation flags */
```

sem_op	Description
Positive	used to mark a resource as allocated. value of sem_op is added to semaphore's value. This operation can always proceed. It never forces a process to wait. The calling process must have alter permission on the semaphore set.
Negative	If the absolute value of sem_op is greater than the value of the semaphore, the calling process will block until the value of the semaphore reaches that of the absolute value of sem_op. Finally, the absolute value of sem_op will be subtracted from the semaphore's value. This is how a process releases a resource guarded by the semaphore.
Zero	This process will wait until the semaphore in question reaches 0.

http://www.ecst.csuchico.edu/~beej/guide/ipc/semaphores.html

`seminit.c` creates the semaphore. Try using `ipcs` from the command line to verify that it exists. Them run `semdemo.c` in a couple of windows and see how they interact. Finally, use `semrm.c` to remove the semaphore. You could also try removing the semaphore while running `semdemo.c` just to see what kinds of errors are generated.

Here's `seminit.c` (run this first!):

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

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

int main(void)
{
    key_t key;
    int semid;
    union semun arg;

    if ((key = ftok("semdemo.c", 'J')) == -1) {
        perror("ftok");
        exit(1);
    }

    /* create a semaphore set with 1 semaphore: */
    if ((semid = semget(key, 1, 0666 | IPC_CREAT)) == -1) {
        perror("semget");
        exit(1);
    }

    /* initialize semaphore #0 to 1: */
    arg.val = 1;
    if (semctl(semid, 0, SETVAL, arg) == -1) {
        perror("semctl");
        exit(1);
    }

    return 0;
}
```

Here's `semdemo.c`:

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

int main(void)
{
    key_t key;
    int semid;
    struct sembuf sb = {0, -1, 0};  /* set to allocate resource */

    if ((key = ftok("semdemo.c", 'J')) == -1) {
        perror("ftok");
        exit(1);
    }

    /* grab the semaphore set created by seminit.c: */
    if ((semid = semget(key, 1, 0)) == -1) {
        perror("semget");
        exit(1);
    }
    printf("Press return to lock: ");
    getchar();
```

```
        printf("Trying to lock...\n");
        if (semop(semid, &sb, 1) == -1) {
            perror("semop");
            exit(1);
        }
        printf("Locked.\n");
        printf("Press return to unlock: ");
        getchar();
        sb.sem_op = 1; /* free resource */
        if (semop(semid, &sb, 1) == -1) {
            perror("semop");
            exit(1);
        }
        printf("Unlocked\n");
        return 0;
    }
```

Here's semrm.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int main(void)
{
    key_t key;
    int semid;
    union semun arg;

    if ((key = ftok("semdemo.c", 'J')) == -1) {
        perror("ftok");
        exit(1);
    }

    /* grab the semaphore set created by seminit.c: */
    if ((semid = semget(key, 1, 0)) == -1) {
        perror("semget");
        exit(1);
    }

    /* remove it: */
    if (semctl(semid, 0, IPC_RMID, arg) == -1) {
        perror("semctl");
        exit(1);
    }

    return 0;
}
```

Message Queues

Message queues

Slide #8-8

Notes

Basic Concepts – Message Queues

1. Message Queues are a queue for allowing multiple processes to communicate with each other
2. The steps:
   2.1. Create or connect to the message queue, and get a pointer

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

`msgget()` returns the message queue ID on success, or -1 on failure

   2.2. Send to Queue

Each message is made up of two parts, which are defined in the template structure `struct msgbuf`, as defined in `sys/msg.h`:

```
struct msgbuf {
    long mtype;
    char mtext[1];
};
```

The field `mtype` is used later when retrieving messages from the queue, and can be set to any positive number. `mtext` is the data this will be added to the queue.

int msgsnd(int *msqid*, const void **msgp*, size_t *msgsz*, int *msgflg*);

msqid is the message queue identifier returned by `msgget()`. The pointer *msgp* is a pointer to the data you want to put on the queue. *msgsz* is the size in bytes of the data to add to the queue. Finally, *msgflg* allows you to set some optional flag parameters,

2.3. Receive from Queue

```
int msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);
```

the behavior of `msgrcv()` can be modified by choosing a *msgtyp* that is positive, negative, or zero:

msgtyp	**Effect on `msgrcv()`**
Zero	Retrieve the next message on the queue, regardless of its `mtype`.
Positive	Get the next message with an `mtype` **equal to** the specified *msgtyp*.
Negative	Retrieve the first message on the queue whose `mtype` field is less than or equal to the absolute value of the *msgtyp* argument.

2.4. Destroy Queue

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

cmd => IPC_RMID, which is used to remove the message queue

```
msgctl(msqid, IPC_RMID, NULL);
```

http://www.ecst.csuchico.edu/~beej/guide/ipc/mq.html

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

    struct my_msgbuf {
        long mtype;
        char mtext[200];
    };

    int main(void)
    {
        struct my_msgbuf buf;
        int msqid;
        key_t key;

        if ((key = ftok("kirk.c", 'B')) == -1) {
            perror("ftok");
            exit(1);
        }

        if ((msqid = msgget(key, 0644 | IPC_CREAT)) == -1) {
            perror("msgget");
            exit(1);
        }

        printf("Enter lines of text, ^D to quit:\n");

        buf.mtype = 1; /* we don't really care in this case */
```

```
        while(gets(buf.mtext), !feof(stdin)) {
            if (msgsnd(msqid, (struct msgbuf *)&buf, sizeof(buf), 0) == -1)
                perror("msgsnd");
        }

        if (msgctl(msqid, IPC_RMID, NULL) == -1) {
            perror("msgctl");
            exit(1);
        }

        return 0;
    }
```

and

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

    struct my_msgbuf {
        long mtype;
        char mtext[200];
    };

    int main(void)
    {
        struct my_msgbuf buf;
        int msqid;
        key_t key;

        if ((key = ftok("kirk.c", 'B')) == -1) {  /* same key as kirk.c */
            perror("ftok");
            exit(1);
        }

        if ((msqid = msgget(key, 0644)) == -1) { /* connect to the queue */
            perror("msgget");
            exit(1);
        }

        printf("spock: ready to receive messages, captain.\n");

        for(;;) { /* Spock never quits! */
            if (msgrcv(msqid, (struct msgbuf *)&buf, sizeof(buf), 0, 0) ==
-1) {
                perror("msgrcv");
                exit(1);
            }
            printf("spock: \"%s\"\n", buf.mtext);
        }

        return 0;
    }
```

Chapter Summary

For this chapter, the following are the objectives:
- getopts,
- shared memory segments
- semaphores,
- message queues

Slide #8-9

Chapter 8

IPC - II

Assignment Questions

Questions:

8.1 Modify "mysh" (from chapter 7) to include parsing of command line arguments, such that

8.1.1 Using the '-pipe' argument will use the PIPE IPC mechanism

8.1.2 Using the '-sysv' argument will use the IPC mechanism of message queues to pass commands; the shared memory segment is used for any communication back to the parent. And, semaphores to regulate the access to the shared memory segments.

Useful links

http://www.ecst.csuchico.edu/~beej/guide/ipc/shmem.html

http://www.ecst.csuchico.edu/~beej/guide/ipc/mq.html#ftok

http://www.ecst.csuchico.edu/~beej/guide/ipc/semaphores.html