Operating Systems Laboratory

Lab 3

Introduction to IPC

1 Introduction

This lab session is designed to

- Message Passing.
- Shared Memory
- Pipe Communication

2. Log Into your virtual machine from last week.

3. Message Passing

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

#define MAX_SEND_SIZE 80

struct mymsgbuf {
        long mtype;
        char mtext[MAX_SEND_SIZE];
};

void send_message(int qid, struct mymsgbuf *qbuf, long type, char *text);
void read_message(int qid, struct mymsgbuf *qbuf, long type);
void remove_queue(int qid);
void change_queue_mode(int qid, char *mode);
void usage(void);


int main(int argc, char *argv[])
{
        key_t key;
        int   msgqueue_id;
        struct mymsgbuf qbuf;

        if(argc == 1)
                usage();


        key = ftok(".", 'm');
        if((msgqueue_id = msgget(key, IPC_CREAT|0660)) == -1) {
                perror("msgget");
                exit(1);
        }

        switch(tolower(argv[1][0]))
        {
                case 's': send_message(msgqueue_id, (struct mymsgbuf
*)&qbuf,
                                          atol(argv[2]), argv[3]);
                          break;
                case 'r': read_message(msgqueue_id, &qbuf, atol(argv[2]));
                          break;
                case 'd': remove_queue(msgqueue_id);
                          break;
                case 'm': change_queue_mode(msgqueue_id, argv[2]);
                          break;

                 default: usage();

        }

        return(0);
      }



void send_message(int qid, struct mymsgbuf *qbuf, long type, char *text)
{
        /* Send a message to the queue */
        printf("Sending a message ...\n");
        qbuf->mtype = type;
        strcpy(qbuf->mtext, text);

        if((msgsnd(qid, (struct msgbuf *)qbuf,
                strlen(qbuf->mtext)+1, 0)) ==-1)
        {
                perror("msgsnd");
                exit(1);
        }
}

void read_message(int qid, struct mymsgbuf *qbuf, long type)
{
        /* Read a message from the queue */
        printf("Reading a message ...\n");
        qbuf->mtype = type;
        msgrcv(qid, (struct msgbuf *)qbuf, MAX_SEND_SIZE, type, 0);

        printf("Type: %ld Text: %s\n", qbuf->mtype, qbuf->mtext);
}

void remove_queue(int qid)
{
        /* Remove the queue */
        msgctl(qid, IPC_RMID, 0);
}

void change_queue_mode(int qid, char *mode)
{
        struct msqid_ds myqueue_ds;
        /* Get current info */
        msgctl(qid, IPC_STAT, &myqueue_ds);

        /* Convert and load the mode */
        sscanf(mode, "%ho", &myqueue_ds.msg_perm.mode);

        /* Update the mode */
        msgctl(qid, IPC_SET, &myqueue_ds);
}

void usage(void)
{
        fprintf(stderr, "msgtool - A utility for tinkering with msg
queues\n");
        fprintf(stderr, "\nUSAGE: msgtool (s)end <type> <messagetext>\n");
        fprintf(stderr, "               (r)ecv <type>\n");
        fprintf(stderr, "               (d)elete\n");
        fprintf(stderr, "               (m)ode <octal mode>\n");
        exit(1);
      }
```

Practical – Can you explain the operation of the code?

4. Shared Memory

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

#define SEGSIZE 100

void usage();
void writeshm(int shmid, char *segptr, char *text);
void changemode(int shmid, char *mode);
void removeshm(int shmid);
void readshm(int shmid, char *segptr);

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

        if(argc == 1)
                usage();


        key = ftok(".", 'S');


        if((shmid = shmget(key, SEGSIZE, IPC_CREAT|IPC_EXCL|0666)) == -1)
        {
                printf("Shared memory segment exists - opening as
client\n");
```

```
                /* Segment probably already exists - try as a client */
                if((shmid = shmget(key, SEGSIZE, 0)) == -1)
                {
                        perror("shmget");
                        exit(1);
                }
        }
        else
        {
                printf("Creating new shared memory segment\n");
        }

        /* Attach (map) the shared memory segment into the current process
*/
        if((segptr = (char *)shmat(shmid, 0, 0)) == (char *)-1)
        {
                perror("shmat");
                exit(1);
        }

        switch(tolower(argv[1][0]))
        {
                case 'w': writeshm(shmid, segptr, argv[2]);
                          break;
                case 'r': readshm(shmid, segptr);
                          break;
                case 'd': removeshm(shmid);
                          break;
                case 'm': changemode(shmid, argv[2]);
                          break;
                 default: usage();

        }
}

writeshm(int shmid, char *segptr, char *text)
{
        strcpy(segptr, text);
        printf("Done...\n");
}

readshm(int shmid, char *segptr)
{
        printf("segptr: %s\n", segptr);
}

removeshm(int shmid)
{
        shmctl(shmid, IPC_RMID, 0);
        printf("Shared memory segment marked for deletion\n");
}

changemode(int shmid, char *mode)
{
        struct shmid_ds myshmds;

        /* Get current values for internal data structure */
        shmctl(shmid, IPC_STAT, &myshmds);

        /* Display old permissions */
        printf("Old permissions were: %o\n", myshmds.shm_perm.mode);

        /* Convert and load the mode */
        sscanf(mode, "%o", &myshmds.shm_perm.mode);

        /* Update the mode */
        shmctl(shmid, IPC_SET, &myshmds);

        printf("New permissions are : %o\n", myshmds.shm_perm.mode);
}

usage()
{
        fprintf(stderr, "shmtool - A utility for tinkering with shared
memory\n");
        fprintf(stderr, "\nUSAGE:  shmtool (w)rite <text>\n");
        fprintf(stderr, "                (r)ead\n");
        fprintf(stderr, "                (d)elete\n");
        fprintf(stderr, "                (m)ode change <octal mode>\n");
        exit(1);
}
```

Practical – Can you explain the operation of the code?

5. Normal PIPE Communciation

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void)
{
        int     fd[2], nbytes;
        pid_t   childpid;
        char    string[] = "Hello, world!\n";
        char    readbuffer[80];

        pipe(fd);

        if((childpid = fork()) == -1)
        {
                perror("fork");
                exit(1);
        }

        if(childpid == 0)
        {

                  close(fd[0]);

                /* Send "string" through the output side of pipe */
                write(fd[1], string, (strlen(string)+1));
                exit(0);
        }
        else
        {



                /* Read in a string from the pipe */
                nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
                printf("Received string: %s", readbuffer);
```

```
          }

          return(0);
     }
```

Practical – Can you explain the operation of the code?

6. Named PIPE Communciation

Server

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>

#include <linux/stat.h>
#define PIPE "fifo"
int main(){

int fd;
char readbuf[20];

mknod(PIPE, S_IFIFO | 0660, 0);  // create pipe
fd = open(PIPE, O_RDONLY, 0);    // open pipe

for (;;)
{

      if (read(fd, &readbuf, sizeof(readbuf)) < 0)

      {

      //read from pipe perror("Error reading pipe");
      exit(1);

      }

      printf("Received string: %s\n", readbuf);

}

exit(0);

}
```

Client

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>


#include <linux/stat.h>
```

```
#define PIPE “fifo”
int main()
{
        int fd;
        char writebuf[20] = “Hello”; // open pipe
        fd = open(PIPE, O_WRONLY, 0);
        // write to pipe
        write(fd, writebuf, sizeof(writebuf));
        exit(0);

}
```

Practical – Can you explain the operation of the code?

Reference:

"Linux Programmer's Guide"