Lab 5: Inter-Process Communication

1. Objective

- Study the inter-process communication

2. Syllabus

- Understanding the concepts and principle of inter-process communication
- Implementing the inter-process communication using C or C++ in Linux

3. Prerequisite

- C or C++ language
- Computer which can run Linux system (e.g. Ubuntu)
- Understanding the inter-process communication

4. Principles of Inter-Process Communication

Please refer to the chapter 5 of the text book. The main contents have been lectured in the ninth week, 11/07/2016. The slide of this chapter can be found in the course website: http://www.thinkmesh.net/ose/.

5. Experimental Contents

5.1 Pipe Communication

Please study how to implement the ***pipe*** communication in a program firstly. The link: http://users.cs.cf.ac.uk/Dave.Marshall/C/node23.html.

The following codes use the **unnamed pipe** to implement the process communication. Please input these codes into a file manually (e.g. **unnamed_pipe.c**) and run it. **Note:** please modify the program, and consider multiple child process. If it can run, show me your result. If not, tell me why.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define MAXLINE 256
int main(void)
{
    int n;
    int fd[2];
    pid_t pid;
    char line[MAXLINE];
    if (pipe(fd) < 0) {
        printf("pipe error");
        exit(-1);
    }
    if ((pid = fork()) < 0) {
        printf("fork error");
        exit(-1);
    } else if (pid > 0) { /* parent */
        close(fd[0]);
        write(fd[1], "hello pipe\n", 12);
    } else { /* child */
```

```
        close(fd[1]);
        n = read(fd[0], line, MAXLINE);
        write(STDOUT_FILENO, line, n);
    }
    exit(0);
}
```

Run example: gcc -o unnamed_pipe unnamed_pipe.c
./unnamed_pipe

The following codes implement the inter-process communication using a named pipe. Please input these codes into a file manually (e.g. **named_pipe.c**) and run it. **Note:** please modify the program, and consider ten reader threads. If it can run, show me your result. If not, tell me why.

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <pthread.h>
#include <sys/stat.h>
#include <sys/types.h>
#define FIFO_PATH "test_fifo"

void *read_fifo_one(void *dummy)
{
    int fd = open(FIFO_PATH, O_RDONLY);
    int ret;
    char read_buf[1024];
    printf("fifo open for reading ok\n");
    while ((ret = read(fd, read_buf, 1024)) > 0)
    {
        printf("reader got: %s\n", read_buf);
    }
    close(fd);
    return NULL;
}
void *read_fifo_two(void *_fd)
{
    int fd = dup((int)_fd);
    int ret;
    char read_buf[1024];
    if ((ret = read(fd, read_buf, 1024)) > 0)
    {
        printf("reader got: %s\n", read_buf);
    }
    close(fd);
    return NULL;
}
void test1()
{
    pthread_t tid;
    int fd;
    if (access(FIFO_PATH, F_OK) != 0)
    {
        mkfifo(FIFO_PATH, 0666);
    }
    pthread_create(&tid, NULL, read_fifo_one, NULL);
    fd = open(FIFO_PATH, O_WRONLY);
```

```
        printf("fifo open for writing ok\n");
        write(fd, "test1", 4);
        close(fd);

        pthread_join(tid, NULL);
}
void test2()
{
        pthread_t tid;
        int fd;
        fd = open(FIFO_PATH, O_RDWR);
        pthread_create(&tid, NULL, read_fifo_two, (void *)fd);
        write(fd, "test2", 4);
        sleep(1);
        close(fd);
        pthread_join(tid, NULL);
}
void test3()
{
        int fd;
        int ret;

        if (access(FIFO_PATH, F_OK) != 0)
        {
                mkfifo(FIFO_PATH, 0666);
        }

        if (fork() > 0)
        {
                fd = open(FIFO_PATH, O_WRONLY);
                write(fd, "test3", 4);
                close(fd);
        }
        else
        {
                char read_buf[1024];
                fd = open(FIFO_PATH, O_RDONLY);
                if ((ret = read(fd, read_buf, 1024)) > 0)
                {
                        printf("reader got: %s\n", read_buf);
                }
                close(fd);
        }
}
int main(void)
{
        test1();
        test2();
        test3();

        return 0;
}
```

Run example: gcc -o named_pipe named_pipe.c -lpthread
./named_pipe

5.2 Message Queues

Please study how to implement the ***message queue*** communication in a program firstly. The link: http://users.cs.cf.ac.uk/Dave.Marshall/C/node25.html.

The program includes two processes. Please input these codes into two files manually (e.g. **process1.c** and **process2.c**) and run them. **Note:** please modify the program, and consider three processes communication through message queue. The format: process1->process2->process3. The process1 generates a message (i.e. "I am process 1") and sends to process2. The process2 receives it and resends to process 3.

The following codes is in "**process1.c**".

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

#define MSG_KEY 1097
#define MSG_LEN_MAX 32
#define MSG_TYPE 19

struct msg_struct
{
    long msg_type;
    char data[MSG_LEN_MAX];
};

int main()
{
    int ret;
    char buff[MSG_LEN_MAX];
    struct msg_struct message1;
    int msg_type = MSG_TYPE;
    int msg_id;
    int loop = 1;

    message1.msg_type = MSG_TYPE;

    msg_id = msgget((key_t)MSG_KEY, 0666 | IPC_CREAT);
    if (msg_id == -1) {
        perror("error in creating message queue:");
        exit(EXIT_FAILURE);
    }

    while(loop) {
        printf("enter some text\n");
        fgets(buff, MSG_LEN_MAX, stdin);
        strncpy(message1.data, buff, MSG_LEN_MAX-1);

        ret = msgsnd(msg_id, &message1, MSG_LEN_MAX, msg_type, 0);
        if (ret == -1) {
            perror("error in reading from msgq\n");
            exit(EXIT_FAILURE);
        }
        printf("process1 writes = %32s\n",message1.data);
        if (strncmp(message1.data, "end", 3) == 0)
```

```
                loop = 0;
        }

        return EXIT_SUCCESS;
}
```

The following codes is in “**process2.c**”.

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

#define MSG_KEY 1097
#define MSG_LEN_MAX 32
#define MSG_TYPE 19

struct msg_struct
{
        long msg_type;
        char data[MSG_LEN_MAX];
};

int main()
{
        int ret;
        struct msg_struct message1;
        int msg_type = MSG_TYPE;
        int msg_id;
        int loop = 1;

        message1.msg_type = MSG_TYPE;

        msg_id = msgget((key_t)MSG_KEY, 0666 | IPC_CREAT);
        if (msg_id == -1) {
                perror("error in creating message queue:");
                exit(EXIT_FAILURE);
        }

        while(loop) {
                ret = msgrcv(msg_id, &message1, MSG_LEN_MAX, msg_type, 0);
                if (ret == -1) {
                        perror("error in reading from msgq\n");
                        exit(EXIT_FAILURE);
                }
                printf("process2 reads = %32s\n",message1.data);
                if (strncmp(message1.data, "end", 3) == 0)
                        loop = 0;
        }

        return EXIT_SUCCESS;
}
```

Run example:

```
gcc -o process1 process1.c
gcc -o process2 process2.c
./process2 &
./process1
```

5.3 Shared Memory

Please study how to implement the ***Shared Memory*** communication in a program firstly. The link: http://users.cs.cf.ac.uk/Dave.Marshall/C/node27.html.

The program includes two processes, parent and child processes. Please input these codes into a file manually (e.g. **shm.c**) and run it. **Note:** please modify the program, and consider one parent and two child processes. If it can run, show me your result. If not, tell me why.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define SHMKEY 1

void parent()
{
    int shmid;
    int ret;
    char *addr;

    shmid = shmget(SHMKEY, 0, 0666);
    if (shmid < 0)
    {
        shmid = shmget(SHMKEY, 4096, IPC_CREAT | IPC_EXCL | 0666);
        if (shmid < 0)
        {
            perror("parent shmget");
            return;
        }
    }

    addr = shmat(shmid, 0, 0);
    if (addr == (void *)-1)
    {
        perror("shmat");
        return;
    }

    sprintf(addr, "I am parent");

    usleep(500);
    ret = shmctl(shmid, IPC_RMID, NULL);
    if (ret < 0)
    {
        perror("shmctl");
    }
}

void child()
{
    int shmid;
    char *addr;
    char buf[1024];

    shmid = shmget(SHMKEY, 0, 0666);
    if (shmid < 0)
    {
        perror("child shmget");
        return;
    }

    addr = shmat(shmid, 0, 0);

    memcpy(buf, addr, 1024);
    printf("buf is: %s\n", buf);
}

int main(void)
{
    pid_t pid;
    int status;

    if ((pid = fork()) < 0)
    {
        printf("fork error");
        exit(-1);
    }
    else if (pid > 0)
    { /* parent */
        parent();
        wait(&status);
    }
    else
    { /* child */
        child();
    }

    exit(0);
}
```

How to run:

```
gcc -o shm shm.c
./shm
```

6. Conclusion

In this chapter, we have completed some experiments to understand inter-process communication.