

南昌大学实验报告8

姓名：谢志彬 学号：6103115112

邮箱地址：siliconx@163.com

专业班级：计算机科学与技术153

实验日期：2018/06/04

课程名称：Linux程序设计实验

实验项目名称

A Hardcore IPC

实验目的

- 1.理解进程间通信(IPC)
- 2.熟悉信号量(semaphore)、消息队列(message queue)、共享内存(shared memory)的使用

实验基础

C语言、Linux、进程间通信、系统调用

实验步骤

T 1: Client-Server Communication using Message Queues

该客户端-服务器通信系统需要两个消息队列:

- Up-消息队列: 用于从客户端发送数据到服务器
- Down-消息队列: 用于从服务器返回数据给客户端

可以看到, 服务器既是Up-消息队列的读者, 又是Down-消息队列的写者; 客户端既是Up-消息队列的写者, 又是Down-消息队列的读者.

同时因为无论是Up-消息队列还是Down-消息队列, 都是一个进程读, 一个进程写, 且读进程会等待写进程写入数据后才开始读取数据, 因此不会产生冲突, 也就不需要使用信号量对消息队列进行读写控制了.

根据以上分析可以开始编码.

I.编写msgq_server.c

负责从Up-消息队列中接收消息，并把接收到的消息进行大小写转换，再把新消息写入Down-消息队列中

```
#include <stdio.h> // for stdin/out
#include <ctype.h> // for islower()/isupper(), tolower()/toupper()
#include <string.h> // for strlen()
#include <sys/ipc.h> // for key_t, ftok, IPC_CREAT/IPC_RMID
#include <sys/msg.h> // for msgget()/msgsnd()/msgrcv()/msgctl()

#define BUFFER_SIZE 256

#define UP_KEY 16 // for Up queue key
#define DOWN_KEY 8 // for Down queue key
#define UP_MSG_TP 1 // for Up message type
#define DOWN_MSG_TP 2 // for Down message type

/**
 * Message Queue server.
 * writer of Down queue
 * reader of Up queue
 */

struct msg_queue { // structure for message queue
    long msg_type;
    char msg_text[BUFFER_SIZE];
} down_queue, up_queue;

int down_queue_writer(char*); // Down queue writer
int up_queue_reader(); // Up queue reader
int reverse(char*, int); // reverse char (upper to lower, lower to upper)

int main(int argc, char const *argv[]) {
    printf("Message Queue Server Running...\n");
    up_queue_reader();
    printf("Done!\n");
    return 0;
}

int down_queue_writer(char* original_msg) { // Down queue writer
    key_t key = DOWN_KEY; // unique key for message
    int msgid;

    // create a message queue with `key` and returns its identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    down_queue.msg_type = DOWN_MSG_TP;

    // copy message to Down queue
    strcpy(down_queue.msg_text, original_msg);
```

```

// reverse message
reverse(down_queue.msg_text, strlen(down_queue.msg_text));

// send message
printf("Sending message...\n");
msgsnd(msgid, &down_queue, sizeof(down_queue), 0);

return 0;
}

int up_queue_reader() { // Up queue reader
    key_t key = UP_KEY; // unique key for message
    int msgid;

    // create a message queue with `key` and returns its identifier
    msgid = msgget(key, 0666 | IPC_CREAT);

    // receive message
    printf("Waiting for message in Up Queue arrive...\n");
    msgrcv(&msgid/msgid, &msgp/&up_queue, &msgsz/sizeof(up_queue),
    &msgtyp*/UP_MSG_TP, &msgflg*/0);

    printf("Received message: %s", up_queue.msg_text);

    // send the received message to Down Queue
    down_queue_writer(up_queue.msg_text);

    // destroy the queue
    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}

int reverse(char* msg, int len) {
    printf("Processing message...\n");
    char c;
    for (int i = 0; i < len; ++i) {
        c = msg[i];
        if (islower(c)) {
            msg[i] = toupper(c);
        } else if (isupper(c)) {
            msg[i] = tolower(c);
        }
    }

    return 0;
}

```

II.编写msgq_client.c

负责从Up-消息队列中写入消息，再等待服务器写入处理过的消息到Down-消息队列中

```
#include <stdio.h> // for stdin/out
#include <sys/ipc.h> // for key_t, ftok, IPC_CREAT/IPC_RMID
#include <sys/msg.h> // for msgget/msgsnd/msgrcv/msgctl

#define BUFFER_SIZE 256

#define UP_KEY 16 // for Up queue key
#define DOWN_KEY 8 // for Down queue key
#define UP_MSG_TP 1 // for Up message type
#define DOWN_MSG_TP 2 // for Down message type

/**
 * Message Queue Client.
 * writer of Up queue
 * reader of Down queue
 */

struct msg_queue { // structure for message queue
    long msg_type;
    char msg_text[BUFFER_SIZE];
} down_queue, up_queue;

int up_queue_writer(); // Up queue writer
int down_queue_reader(); // Down queue reader

int main(int argc, char const *argv[]) {
    printf("Message Queue Client Running...\n");
    up_queue_writer();
    return 0;
}

int up_queue_writer() { // Up queue writer
    key_t key = UP_KEY; // unique key for message
    int msgid;

    // create a message queue with `key` and returns its identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    up_queue.msg_type = UP_MSG_TP;

    // input message
    printf("Insert message to send to server: ");
    fgets(up_queue.msg_text, sizeof(up_queue.msg_text), stdin);

    // send message
    msgsnd(msgid, &up_queue, sizeof(up_queue), 0);

    // now waiting for the message send from server in Down queue
    down_queue_reader();

    return 0;
}
```

```

int down_queue_reader() { // Down Queue reader
    key_t key = DOWN_KEY; // unique key for message
    int msgid;

    // create a message queue with `key` and returns its identifier
    msgid = msgget(key, 0666 | IPC_CREAT);

    // receive message
    printf("Waiting for message in Down Queue arrive...\n");
    msgrcv(/*msgid*/msgid, /*msgp*/&down_queue, /*msgsz*/sizeof(down_queue),
    /*msgtyp*/DOWN_MSG_TP, /*msgflg*/0);

    // show the message get from Down queue
    printf("Msg processed: %s \n", down_queue.msg_text);

    // destroy the queue
    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}

```

III.编写makefile(msgq_makefile)

```

#main/Makefile

all: ipc_msgq

ipc_msgq: msgq_server.c msgq_client.c
    gcc msgq_server.c -o msgq_server
    gcc msgq_client.c -o msgq_client

```

IV.编译运行

```
siliconx@Lenovo: ~/code/LinuxProgramming/c/ipc
File Edit View Search Terminal Help

siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ ls
consumer.c  #include <sys/msg.h>  producer.c  shm_makefile  C RMID
msgq_client.c  msgq_server.c  reset.c  get/msgsnd/msgrcv/msgctl

siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ make -f msgq_makefile
gcc msgq_server.c -o msgq_server
gcc msgq_client.c -o msgq_client

siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ ./msgq_client
Message Queue Client Running...
Insert message to send to server: lower case message
Waitting for message in Down Queue arrive...
Msg processed: LOWER CASE MESSAGE

siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ ./msgq_client
Message Queue Client Running...
Insert message to send to server: UPPER CASE MESSAGE
Waitting for message in Down Queue arrive...
Msg processed: upper case message

siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ |
23 int up_queue_writer(); // Up queue writer
24 int down_queue_reader(); // Down queue reader
25
26 int main(int argc, char const *argv[]) {
27     printf("Message Queue Client Running...\n");
28     up_queue_writer();
29     return 0;
30 }

siliconx@Lenovo: ~/code/LinuxProgramming/c/ipc
File Edit View Search Terminal Help

siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ cd ~/code/LinuxProgramming/c/ipc/
siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ ./msgq_server
Message Queue Server Running...
Waitting for message in Up Queue arrive...
Received message: lower case message
Processing message...
Sending message...
Done!
siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ ./msgq_server
Message Queue Server Running...
Waitting for message in Up Queue arrive...
Received message: UPPER CASE MESSAGE
Processing message...
Sending message...
Done!
siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ |
```

T 2: The Consumer-Producer Problem using Shared Memory

根据题目要求, shelf的实质是一个数值. 由于需要被两个进程共享, 因此可以用共享内存实现.

当shelf未满($shelf < 5$)时, Producer在shelf中添加一件物品($shelf + 1$);

当shelf非空(shelf > 0)时, Consumer从shelf中取走一件物品(shelf - 1);

可以看到Producer和Consumer都要对shelf进行读写, 因此需要通过信号量来避免同时使用shelf而产生冲突.

I.编写producer.c

负责检查shelf上物品的数量, 并在数量小于MAX(根据题目, 这里取5)时添加一个物品.

Producer循环工作N次后退出

```
#include <stdio.h> // for stdin/out
#include <stdlib.h> // for srand()/rand()
#include <time.h> // for time()
#include <unistd.h> // for sleep()
#include <sys/ipc.h> // for IPC
#include <sys/shm.h> // for shared memory
#include <fcntl.h> // for O_CREAT
#include <semaphore.h> // for semaphore

#define KEY 1 // unique key for shared memory
#define MAX 5 // max item on shelf
#define N 20 // times of producer to run
#define shelf_sem_name "shelf_semaphore" // `shelf` semaphore name

int main() {
    printf("Producer running...\n");

    // unique key
    key_t key = KEY;

    // semaphore for shelf
    sem_t *shelf_avail = sem_open(shelf_sem_name, O_CREAT, 0666, 3);

    // returns an identifier in shmid
    int shmid = shmget(key, /*size_tsize*/sizeof(int), 0666 | IPC_CREAT);

    // attach to shared memory
    int *shelf = (int*) shmat(shmid, /*shmaddr*/(void*)0, /*shmflg*/0);
    *shelf = MAX; // initial itmes on shelf is `MAX`

    // use current time as seed for random generator
    srand(time(0));

    for (int i = 0; i < N; ++i) { // loop N times
        printf("\n===== No.%d =====\n", i);
        printf("Waitting until shelf available...\n");
        sem_wait(shelf_avail);

        printf("Shelf available, now checking the number of items on the shelf...\n");
        if ((*shelf) < MAX) { // not full
            printf("Shelf has %d item(s) currently, not full.\n", (*shelf));
```

```

        printf("Putting an item to the shelf...\n");
        ++(*shelf); // add an item
        printf("Now, there are %d item(s) on the shelf.\n", (*shelf));
    } else { // full
        printf("Shelf is full, let it go!\n");
    }

    sem_post(shelf_avail);
    sleep(rand() % 3 + 1);
}

printf("\n===== End =====\n");
printf("I'm tired, bye bye!\n");

// detach from shared memory
shmdt(shelf);

return 0;
}

```

II.编写consumer.c

Consumer负责检查shelf上物品的数量，并在数量大于MIN(根据题目，这里取0)时取出一个物品。

当连续M次发现shelf为空时，可以推测Producer已经退出，此时Consumer也应该退出。

根据以上分析，可以看到Producer总是先于Consumer退出，因此关于共享内存的销毁、信号量的关闭等善后工作都应由Consumer承担。

```

#include <stdio.h> // for stdin/out
#include <stdlib.h> // for srand()/rand()
#include <unistd.h> // for sleep()
#include <time.h> // for time()
#include <sys/ipc.h> // for IPC
#include <sys/shm.h> // for shared memory
#include <fcntl.h> // for O_CREAT
#include <semaphore.h> // for semaphore

#define KEY 1 // unique key for shared memory
#define MIN 0 // min item on shelf
#define M 3 // max times of shelf in empty status continuously
#define shelf_sem_name "shelf_semaphore" // `shelf` semaphore name

int main() {
    printf("Consumer running...\n");

    // unique key
    key_t key = KEY;

    // semaphore for shelf

```



```

sem_t *shelf_avail = sem_open(shelf_sem_name, O_CREAT, 0666, 3);

// returns an identifier in shmid
int shmid = shmget(key, sizeof(int), 0666 | IPC_CREAT);

// attach to shared memory
int *shelf = (int*) shmat(shmid, /*shmaddr*/(void*)0, /*shmflg*/0);

// use current time as seed for random generator
srand(time(0));

// counter
int count = 0;

// times of shelf in empty status continuously
int times = 0;

while (times < M) { // exists when shelf empty in M times continuously
    printf("\n===== No.%d =====\n", count);
    printf("Waitting until shelf available...\n");
    sem_wait(shelf_avail);

    printf("Shelf available, now checking the number of items on the shelf...\n");
    if ((*shelf) > MIN) { // not empty
        printf("Shelf has %d item(s) currently, not empty.\n", (*shelf));
        printf("Getting an item from the shelf...\n");
        --(*shelf); // remove an itme
        printf("Now, there are %d item(s) on the shelf.\n", (*shelf));
        times = 0;
    } else { // empty
        ++times;
        printf("Shelf is empty, let it go!\n");
    }

    sem_post(shelf_avail);
    sleep(rand() % 3 + 1);

    ++count;
}

printf("\n===== End =====\n");
printf("Times up, Shelf is empty in %d times continuously.\n", M);
printf("Maybe the producer was tired and exited.\nI have to exit, too. Bye!!!\n");

// detach from shared memory
shmdt(shelf);

// destroy the shared memory
shmctl(shmid, IPC_RMID, NULL);

// close and unlink semaphore
sem_close(shelf_avail);
sem_unlink(shelf_sem_name);

```

```
    return 0;
}
```

III.编写makefile(shm_makefile)

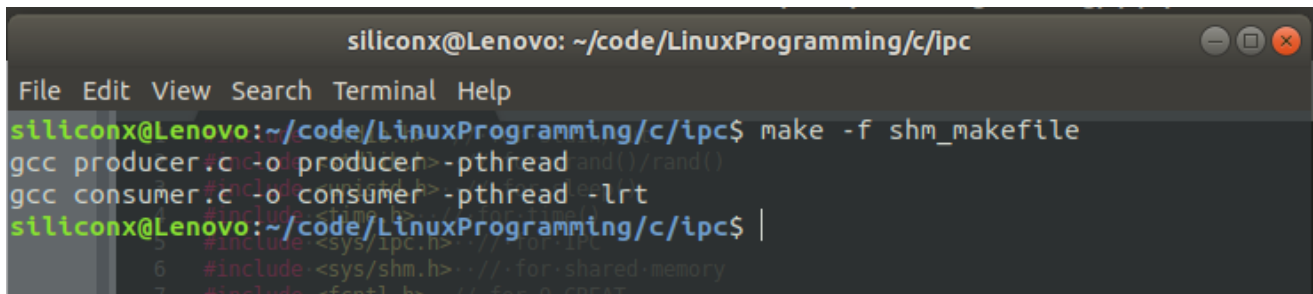
```
#main/Makefile

all: ipc_shm

ipc_shm: producer.c consumer.c
    gcc producer.c -o producer -pthread
    gcc consumer.c -o consumer -pthread -lrt
```

IV.编译运行

make

A terminal window titled 'siliconx@Lenovo: ~/code/LinuxProgramming/c/ipc' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'make -f shm_makefile' being executed. The output shows the compilation of 'producer.c' and 'consumer.c' into 'producer' and 'consumer' executables using 'gcc' with '-pthread' and '-lrt' flags. The prompt then returns to the shell.

```
siliconx@Lenovo: ~/code/LinuxProgramming/c/ipc
File Edit View Search Terminal Help
siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ make -f shm_makefile
gcc producer.c -o producer -pthread
gcc consumer.c -o consumer -pthread -lrt
siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ |
```

开始运行

```
siliconx@Lenovo: ~/code/LinuxProgramming/c/ipc
File Edit View Search Terminal Help
siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ ./consumer
Consumer running...
===== No.0 =====
Waiting until shelf available...
Shelf available, now checking the number of items on the shelf...
Shelf has 5 item(s) currently, not empty.
Getting an item from the shelf...
Now, there are 4 item(s) on the shelf.
===== No.1 =====
Waiting until shelf available...
Shelf available, now checking the number of items on the shelf...
Shelf has 5 item(s) currently, not empty.
Getting an item from the shelf...
Now, there are 4 item(s) on the shelf.
===== No.2 =====
Waiting until shelf available...
Shelf available, now checking the number of items on the shelf...
Shelf has 5 item(s) currently, not empty.
Getting an item from the shelf...
Now, there are 4 item(s) on the shelf.

siliconx@Lenovo: ~/code/LinuxProgramming/c/ipc
File Edit View Search Terminal Help
siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ ./producer
Producer running...
===== No.0 =====
Waiting until shelf available...
Shelf available, now checking the number of items on the shelf...
Shelf is full, let it go!
===== No.1 =====
Waiting until shelf available...
Shelf available, now checking the number of items on the shelf...
Shelf has 4 item(s) currently, not full.
Putting an item to the shelf...
Now, there are 5 item(s) on the shelf.
===== No.2 =====
Waiting until shelf available...
Shelf available, now checking the number of items on the shelf...
Shelf has 4 item(s) currently, not full.
Putting an item to the shelf...
Now, there are 5 item(s) on the shelf.
===== No.3 =====
Waiting until shelf available...
```

运行结束

```
siliconx@Lenovo: ~/code/LinuxProgramming/c/ipc
File Edit View Search Terminal Help
Shelf has 1 item(s) currently, not empty.
Getting an item from the shelf...
Now, there are 0 item(s) on the shelf.
===== No.24 =====
Waiting until shelf available...
Shelf available, now checking the number of items on the shelf...
Shelf is empty, let it go!
===== No.25 =====
Waiting until shelf available...
Shelf available, now checking the number of items on the shelf...
Shelf is empty, let it go!
===== No.26 =====
Waiting until shelf available...
Shelf available, now checking the number of items on the shelf...
Shelf is empty, let it go!
===== End =====
Times up, Shelf is empty in 3 times continuously,
Maybe the producer was tired and exited.
I have to exit, too! Bye!!!
siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$ ./consumer

siliconx@Lenovo: ~/code/LinuxProgramming/c/ipc
File Edit View Search Terminal Help
===== No.17 =====
Waiting until shelf available...
Shelf available, now checking the number of items on the shelf...
Shelf has 4 item(s) currently, not full.
Putting an item to the shelf...
Now, there are 5 item(s) on the shelf.
===== No.18 =====
Waiting until shelf available...
Shelf available, now checking the number of items on the shelf...
Shelf has 4 item(s) currently, not full.
Putting an item to the shelf...
Now, there are 5 item(s) on the shelf.
===== No.19 =====
Waiting until shelf available...
Shelf available, now checking the number of items on the shelf...
Shelf has 4 item(s) currently, not full.
Putting an item to the shelf...
Now, there are 5 item(s) on the shelf.
===== End =====
I'm tired, bye bye!
siliconx@Lenovo:~/code/LinuxProgramming/c/ipc$
```

实验思考

- 理解进程间通信
- 理解并使用信号量、消息队列、共享内存等
- 将理论知识运用到实践中
- 在课程网站上的示例代码中看到可以在函数参数中添加注释以改善代码的可读性, 遂取之