**北京邮电大学软件学院**

**2017-2018学年第1学期实验报告**

**课程名称： 操作系统**

**实验名称： 实验二 进程通讯**

**实验完成人：**

**姓名：**\_\_朱岩\_\_\_**学号：**\_2016522039\_**成绩：**\_\_\_\_\_\_\_\_

**姓名：**\_\_王子妍\_**学号：**\_2016211980**成绩：**\_\_\_\_\_\_\_\_

**姓名：**\_\_李博\_ **学号：**\_2016212003 **成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_陈晋鹏\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2018 年 3 月 25 日**

1. **实验目的**

（1）消息缓冲队列、共享存储区机制进行进程间的通信；

（2）理解通信机制。

1. **实验内容**

1.使用消息缓冲队列来实现 client 进程和 server 进程之间的通信

server 进程先建立一个关键字为 SVKEY（如 75）的消息队列，然后等待接收类型为 REQ （例如 1）的消息；在收到请求消息后，它便显示字符串“serving for client”和接收到 的 client 进程的进程标识数，表示正在为 client 进程服务；然后再向 client 进程发送应答消息，该消息的类型是 client 进程的进程标识数，而正文则是 server 进程自己的标识 ID。client 进程则向消息队列发送类型为 REQ 的消息（消息的正文为自己的进程标识 ID） 以取得 sever 进程的服务， 并等待 server 进程发来的应答；然后显示字符串“receive reply from”和接收到的 server 进程的标识 ID。

2. 使用共享存储区来实现两个进程之间的进程通信

进程 A 创建一个长度为 512 字节的共享内存，并显示写入该共享内存的数据；进程 B 将共享内存附加到自己的地址空间，并向共享内存中写入数据。

1. **实验环境**

在VMware虚拟机下安装Ubuntu系统作为Linux实验环境。采用以Vi +GCC+GDB为开发环境的C语言实验环境。其中Vi 作为编辑器，GCC作为编译器，GDB作为调试器。

1. **实验过程描述**

一）使用消息缓冲队列来实现 client 进程和 server 进程之间的通信

第一步，分析阶段：建立消息缓冲区存放消息；创建消息队列；客户进程向服务器进程发送给客户；

服务器进行接收消息，完成客户的服务请求后，再将服务结果以消息方式发送给客户。

第二步，用代码实现分析部分。

头文件：

#include<stdio.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

#include<stdlib.h>

然后实现代码，代码的实现在附录里。

第三步，debug和运行。

1. Ctrl+Alt+T调出命令行界面
2. 在命令行中输入命令gcc -o server.c,编译服务端程序，如果没有错误就是编译成功了，有错误就debug一下。
3. server.c编译完成后，输入./server 运行编译server.c文件后生成的可执行文件。
4. 通过Ctrl+Alt+T打开一个新的命令行界面
5. 在命令行中输入命令gcc -o client.c,编译服务端程序，如果没有错误就是编译成功了，有错误就debug一下。
6. client.c编译完成后，输入./client 运行编译client.c文件后生成的可执行文件。

第四步，查看结果

客户端：



服务器端：



二）**使用共享存储区来实现两个进程之间的进程通信**

1.实验内容分析

A需要创建共享内存，用到shmget函数，显示数据printf即可以，B打开共享区，也是用到shmget函数，不过msgflg参数不同而已，用shmat函数获得共享内存地址，写入数据准备就写字符串char\*，另外利用到实验一中的进程控制知识，合理利用exit和wait函数。

2.相关原理介绍

共享内存是允许两个不相关的进程访问同一个逻辑内存。共享内存是在两个正在运行的进程之间共享和传递数据的一种非常有效的方式。不同进程之间共享的内存通常安排为同一段物理内存。进程可以将同一段共享内存连接到它们自己的地址空间中，所有进程都可以访问共享内存中的地址，就好像它们是由用C语言函数malloc分配的内存一样。而如果某个进程向共享内存写入数据，所做的改动将立即影响到可以访问同一段共享内存的任何其他进程。

特别提醒：共享内存并未提供同步机制，也就是说，在第一个进程结束对共享内存的写操作之前，并无自动机制可以阻止第二个进程开始对它进行读取。

void \*memset(void \*s, int ch, [size\_t](https://baike.baidu.com/item/size_t" \t "_blank) n);

函数解释：将s中当前位置后面的n个字节 （typedef unsigned int size\_t ）用 ch 替换并返回 s 。作用是在一段内存块中填充某个给定的值，它是对较大的结构体或数组进行清零操作的一种最快方法

shmget函数

该函数用来创建共享内存，它的原型为：

int shmget(key\_t key, size\_t size, int shmflg)；

第一个参数，程序需要提供一个参数key（非0整数），它有效地为共享内存段命名，shmget函数成功时返回一个与key相关的共享内存标识符（非负整数），用于后续的共享内存函数。调用失败返回-1.

第二个参数，size以字节为单位指定需要共享的内存容量

第三个参数，shmflg是权限标志，它的作用与open函数的mode参数一样，如果要想在key标识的共享内存不存在时，创建它的话，可以与IPC\_CREAT做或操作。共享内存的权限标志与文件的读写权限一样，举例来说，0644,它表示允许一个进程创建的共享内存被内存创建者所拥有的进程向共享内存读取和写入数据，同时其他用户创建的进程只能读取共享内存，0777表示所有人都有读写执行的权限。IPC\_CREAT 如果共享内存不存在，则创建一个共享内存，否则打开操作。IPC\_EXC 只有在共享内存不存在的时候，新的共享内存才建立，否则就产生错误。如果单独使用IPC\_CREAT，shmget()函数要么返回一个已经存在的共享内存的操作符，要么返回一个新建的共享内存的标识符。如果将IPC\_CREAT和IPC\_EXCL标志一起使用，shmget()将返回一个新建的共享内存的标识符；如果该共享内存已存在，或者返回-1。IPC\_EXEL标志本身并没有太大的意义，但是和IPC\_CREAT标志一起使用可以用来保证所得的对象是新建的，而不是打开已有的对象。

shmat函数

第一次创建完共享内存时，它还不能被任何进程访问，shmat函数的作用就是用来启动对该共享内存的访问，并把共享内存连接到当前进程的地址空间。它的原型如下：

void \*shmat(int shm\_id, const void \*shm\_addr, int shmflg);

第一个参数，shm\_id是由shmget函数返回的共享内存id。

第二个参数，shm\_addr指定共享内存连接到当前进程中的地址位置，通常为空，表示让系统来选择共享内存的地址。

第三个参数，shm\_flg是一组标志位，通常为0。

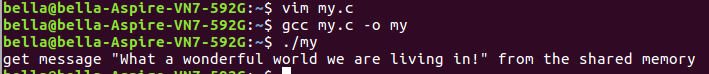
调用成功时返回一个指向共享内存第一个字节的指针，如果调用失败返回-1.

3.程序调试、运行、跟踪和成果收集

实验中遇到的问题：主要是刚开始想得过于复杂，程序很不简洁，后来缕清思路后，对代码进行simplify。

调试过程中结果都是一样，主要是对代码的整理，增强可读性

结果截图：



1. **实验结果**

所有实验均已认真完成，思路与分析在实验过程描述中，代码添加在了附件中。

1. **附件**

**6.1 附件1：源代码**

**1.使用消息缓冲队列来实现 client 进程和 server 进程之间的通信**

服务器端：#include<stdio.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

#include<stdlib.h>

struct msgtype

{

long mtype; //type

char mtext[256]; //text

}msg;

int msgqid;

void server()

{

int pid;

int i,\*pint;

msgqid=msgget(75,0777|IPC\_CREAT);//set up a message queue SVKEY=75

for(;;)

{

msgrcv(msgqid,&msg,sizeof(struct msgtype),1,0);//receive message REQ=1

pint=(int\*)msg.mtext;

pid=\*pint; //get client's pid

printf("serving for client,client's pid:'%d'\n",pid);

msg.mtype=pid;

pid=getpid();

msgsnd(msgqid,&msg,sizeof(pid),0);//向 client 进程发送应答消息

}

//msgctl(msgqid,IPC\_RMID,0);

exit(0);

}

int main()

{

server();

return 0;

}

客户端：

#include<stdio.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

#include<stdlib.h>

struct msgtype

{

long mtype; //type

char mtext[256]; //text

}msg;

int msgqid;

void client()

{

int i,pid;

msgqid=msgget(75,0777);//SVKEY=75 set up a message queue

pid=getpid();//get the current process id

int \*pint;

pint=(int\*)msg.mtext;

\*pint=pid;

msg.mtype=1;

msgsnd(msgqid,&msg,sizeof(int),0);//send

msgrcv(msgqid,&msg,sizeof(struct msgtype),pid,0);

printf("receive reply from,server's pid:%d \n",pid);

exit(0);

}

int main()

{

client();

return 0;

}

2：使用共享存储区来实现两个进程之间的进程通信

#include <sys/shm.h>

#include <sys/ipc.h>

#include <string.h>//memset(),strncpy()

#include <wait.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#define SHMKEY 75

int shmid; //共享内存标识符

char \*addr;//共享内存物理首地址

char\* argv[] = {"What a wonderful world we are living in!"};//准备写入的数据信息

void B() {

shmid = shmget(SHMKEY, 512, 0777); //打开共享区，内存512字节，所有人都有全部

的读写以及执行权限

addr = shmat(shmid, NULL, 0);//获得共享区首地址

memset(addr, '\0', 512);//将addr的后512字节设置成字符'\0'

strncpy(addr, argv[0], 512);//将argv[0]的前512个字节（不足补'\0')存入共享区addr

exit(0);//进程结束

}

void A() {

shmid = shmget(SHMKEY, 512, 0777|IPC\_CREAT);//创建共享存储区

addr = shmat(shmid, NULL, 0);

printf("get message \"%s\" from the shared memory\n", addr);//读取共享区的数

据

exit(0);

}

int main() {

pid\_t pid1 = fork();

if (pid1 == 0) A();

if (pid1 < 0) printf("A进程创建失败");

pid\_t pid2 = fork();

if (pid2 == 0) B();

if (pid2 < 0) printf("B进程创建失败");

wait(NULL);//等待子进程执行完毕

return 0;

}

分工：

实验一：王子妍

实验二：

1. 使用消息缓冲队列来实现 client 进程和 server 进程之间的通信 **----**朱岩

2.使用共享存储区来实现两个进程之间的进程通信 ----李博