消息队列实验报告

202011998088 臧祝利

**实验目的**：

了解和熟悉linux支持的消息通信机制。

**实验任务：**

使用linux系统提供的系统调用msgget(),msgrev(),msgctl()编制一个长度为1K的消息发送和接受的程序。

**实验要求：**

1. 用一个程序作为“引子”，先后fork()两个子进程，SERVER和CLIENT，进行通信。

程序代码如下：

设置了两个进程pid1,pid2，当fork()为0时，为子进程，执行SERVER和CLIENT；

int main(){

    while((pid1 = fork()) == -1);

    if(!pid1){

        SERVER();

    }else{

        while((pid2 = fork()) == -1);

        if(!pid2){

            CLIENT();

        }else wait(0);

    }

    return 0;

}

1. 由SERVER端创建一个Key为75的消息队列，等待CLIENT端进程发来的消息。当遇到类型为1的消息，则作为结束信号，删除该队列，并退出SERVER。SERVER每接受到一个消息后显示一句“(Server)received from + CLIENT端pid+接受消息的类型”,然后发送一个返回消息给CLIENT端，显示一句“(Server)sent”。

程序**代码**如下：

void SERVER(){

    msgid = msgget(MSGKEY, 0777|IPC\_CREAT);

    if (msgid==-1){  //创建失败，报错

        perror("Error!");

        return;

    }

    do{

        msgrcv(msgid, &msg, 1024 ,0, 0);  //接收消息

        printf("%s",msg.message);  //输出所收到的消息

        sprintf(msg.message,"(Client) received from pid %d, type %ld\n", getpid(), msg.mtype);  //将信息传给客户端

        msgsnd(msgid, &msg, 1024 ,0);  //发送消息

        printf("(SERVER)sent\n");

        sleep(1);  //等待客户端接收

    }while (msg.mtype!=1);

    msgctl(msgid, IPC\_RMID, 0);

    exit(0);

}

即先创建一个消息队列，然后等待CLIENT端发送消息，一直循环，直到遇到类型为1的消息，则退出循环并删除队列；

其中当接受到消息时，会输出“(Server)received from + CLIENT端pid+接受消息的类型”；之后使用sprintf()函数对字符串进行写入，再把消息发送给CLIENT端，同时输出“(Server)sent”；

1. CLIENT端使用key为75的消息队列，先后发送类型从10到1的消息，然后退出。发出的最后一个消息是SERVER端需要的结束信号。CLIENT每发送一条消息后显示一句“(Client)sent+发送消息的类型”,然后等待接受SERVER端的返回消息后，显示一句“(Client) received from + SERVER端pid+接受消息的类型”，再在发送下一条消息。

**代码**如下：

void CLIENT(){

    int i;

    msgid = msgget(MSGKEY, 0777);

    for (i=10;i>=1;i--){

        msg.mtype = i;  //设置消息类型

        sprintf(msg.message,"(Server) received from pid: %d, type:%ld\n",getpid(),msg.mtype);  //写入传递信息

        msgsnd(msgid, &msg, 1024, 0);  //发送信息

        printf("(Client)sent %ld\n",msg.mtype);  //发送

        sleep(1);  //等待接收

        msgrcv(msgid, &msg ,1024, 0, 0);  //收到服务端消息

        printf("%s",msg.message);     //输出消息

    }

    exit(0);

}

CLIENT端通过msgget()函数得到消息队列，并从10到1开始不断发送不同类型的消息，首先会将类型值赋值给消息结构体的类型，然后使用sprintf()函数对消息内容进行写入，使用msgsnd()发送给SERVER端，同时输出“(Client)sent+发送消息的类型”，然后再接收SERVER端的消息，输出“(Client) received from + SERVER端pid+接受消息的类型”，待所有类型输出完毕后退出。

1. 父进程在SERVER和CLIENT都退出后结束。

整个程序的**代码以及运行结果**如下：

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/msg.h>

#include <sys/ipc.h>

#include <wait.h>

#define MSGKEY 75

pid\_t pid1,pid2;  //child process

int msgid;

struct msgform{

    long mtype;

    char message[1100];

}msg;

void CLIENT(){

    int i;

    msgid = msgget(MSGKEY, 0777);

    for (i=10;i>=1;i--){

        msg.mtype = i;  //设置消息类型

        sprintf(msg.message,"(Server) received from pid: %d, type:%ld\n",getpid(),msg.mtype);  //写入传递信息

        msgsnd(msgid, &msg, 1024, 0);  //发送信息

        printf("(Client)sent %ld\n",msg.mtype);  //发送

        sleep(1);  //等待接收

        msgrcv(msgid, &msg ,1024, 0, 0);  //收到服务端消息

        printf("%s",msg.message);     //输出消息

    }

    exit(0);

}

void SERVER(){

    msgid = msgget(MSGKEY, 0777|IPC\_CREAT);

    if (msgid==-1){  //创建失败，报错

        perror("Error!");

        return;

    }

    do{

        msgrcv(msgid, &msg, 1024 ,0, 0);  //接收消息

        printf("%s",msg.message);  //输出所收到的消息

        sprintf(msg.message,"(Client) received from pid %d, type %ld\n", getpid(), msg.mtype);  //将信息传给客户端

        msgsnd(msgid, &msg, 1024 ,0);  //发送消息

        printf("(SERVER)sent\n");

        sleep(1);  //等待客户端接收

    }while (msg.mtype!=1);

    msgctl(msgid, IPC\_RMID, 0);

    exit(0);

}

int main(){

    while((pid1 = fork()) == -1);

    if(!pid1){

        SERVER();

    }else{

        while((pid2 = fork()) == -1);

        if(!pid2){

            CLIENT();

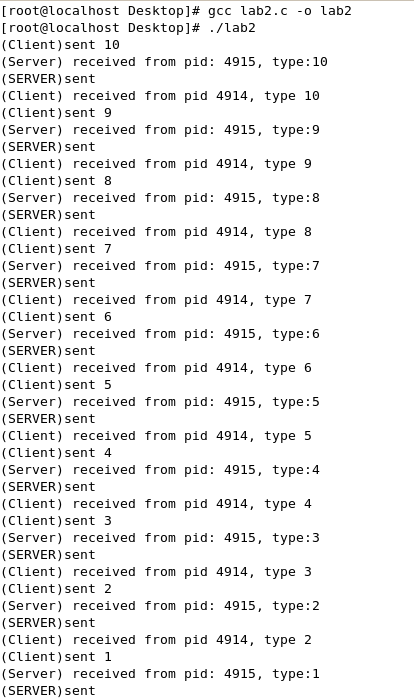
        }else wait(0);

    }

    return 0;

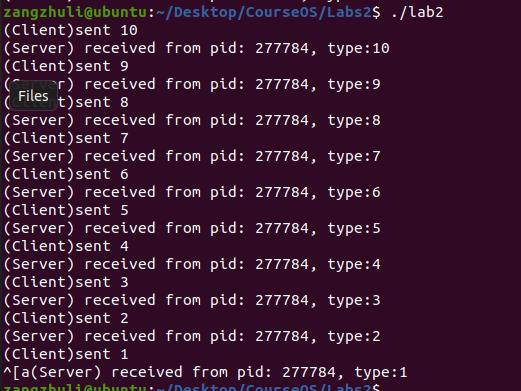
}

**结果**如下：



**总结：**

在这个实验中通过使用msgget()等相关函数，对消息队列的运行过程有了进一步的了解，也学会了如何书写相关的代码，但是在运行过程中也存在着一些问题，比如在Ubuntu和CentOS上运行相同的程序结果略有不同，只在Ubuntu上运行程序时也会出现一些问题，比如整个过程只进行了“一半”；或者type为10的消息服务端会输出两遍；



同时sleep是很重要的，如果没有sleep很容易出现死循环或其他很乱的情况；