暨南大学本科实验报告专用纸

课程名称 操作系统原理 成绩评定

实验项目名称 进程间的通信 指导教师 赵阔

实验项目编号 0806002903 实验项目类型 综合性 实验地点

学生姓名 陈旭天 学号 2021100733

学院 智能科学与工程学院 系 人工智能 专业

实验时间 2023 年 4 月 13 日 上午～4 月 13 日 下 午 温度 ℃湿度

暨南大学本科实验报告专用纸(附页)

一、实验目的和要求

1. 实验目的：

学习如何利用管道机制或消息缓冲队列进行进程间通信，加深对上述通信机制的理解。提高学生分析问题和解决问题的能力，并撰写规范的科学研究报告

2. 实验要求：

了解系统 pipe(),msgsnd(),msgrcv()的功能和实现过程

二、实验原理和主要内容

1. 实验内容：

（1）编写一段程序，使用管道来实现父子间进程通信。子进程项父进程发送自己的进程表示符，以及某字符串。父进程则通过管道读出子进程发来的消息，将消息显示在屏幕上，然后终止。或者，编写一段程序，使其用消息缓冲队列来实现 client 和 server 进程之间的通信。

2. 实验原理：

（1）pipe 函数：

```
int pipe(int fd[2]);
```

pipe 函数定义中的 fd 参数是一个大小为 2 的一个数组类型的指针。该函数成功时返回 0，并将一对打开的文件描述符值填入 fd 参数指向的数组。失败时返回 -1 并设置 errno。通过 pipe 函数创建的这两个文件描述符 fd[0] 和 fd[1] 分别构成管道的两端，往 fd[1] 写入的数据可以从 fd[0] 读出。并且 fd[1] 一端只能进行写操作，fd[0] 一端只能进行读操作，不能反过来使用。要实现双向数据传输，可以使用两个管道。

（2）msgsnd 函数：

int msgsnd(int msqid, const void *ptr, size_t length, int flag);

msqid: 由消息队列的标识符

ptr: 消息缓冲区指针。

追加一条新消息到消息队列的系统调用语法

（3）Msgrcv 函数：

int msgrcv(int msqid, void *ptr, size_t length, long type, int flag);

msqid: 由消息队列的标识符

ptr :消息缓冲区指针。

从消息队列中读出一条新消息、

三、C 程序源码

1. 程序运行结果

```
root@LAPTOP-40J2BMTQ:~/OS2023/pro_communication# vim pipe.c
root@LAPTOP-40J2BMTQ:~/OS2023/pro_communication# gcc pipe.c && ./a.out
296is sending a message to parent!
root@LAPTOP-40J2BMTQ:~/OS2023/pro_communication#
```

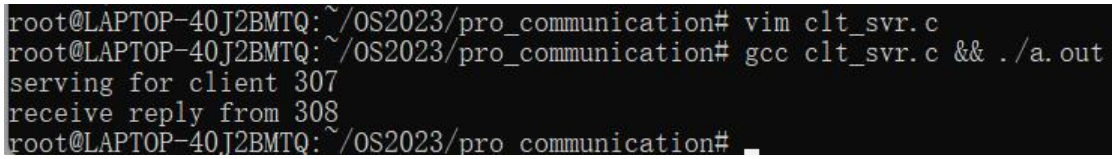
2. 实验源码：

```
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#define MAXSIZE 100

int main(){
        int n;
        pid_t pid;
        int fd[2];
        char W[MAXSIZE];
        char R[MAXSIZE];
        if(pipe(fd)<0)
        {
                perror("Error occured in pipe");
                exit(1);
        }
        if((pid=fork())<0)
        {
                perror("Error occured in fork");
                exit(1);
        }
        else
        {
                if(pid==0)
                {
                        n=getpid();
                        printf("%d",n);
                        strcpy(W,"is sending a message to parent!\n");
                        close(fd[0]);
                        write(fd[1],W,sizeof(W));
                        close(fd[1]);
                }
                else
                {
                        close(fd[1]);
                        read(fd[0],R,sizeof(R));
```

```
                        printf("%s",R);
                        close(fd[0]);
                }
        }
        return 0;
}
```

3. 实验结果



4. 实验源码

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

#define MSGKEY 75
#define BUFSIZE 100

struct msgform
{
        int mtype;
        char mtext[BUFSIZE];
}msg;

int msgqid;
int main()
{
        int pid, serpid, cltpid;
        char str[20];
        if((pid=fork())!=0)
        {
                sleep(1);
                cltpid=getpid();
                msgqid=msgget(MSGKEY,0777);
                msg.mtype=1;
                sprintf(str,"%d",cltpid);
                strcpy(msg.mtext,str);
                msgsnd(msgqid,&msg,BUFSIZE,0);
                sleep(3);
                msgrcv(msgqid,&msg,BUFSIZE,0,0);
                serpid=atoi(msg.mtext);
                printf("receive reply from %d\n",serpid);
```

```
                msgctl(msgqid,IPC_RMID,0);
        }
        else
        {
                msgqid=msgget(MSGKEY,0777|IPC_CREAT);
                while(msg.mtype!=1)
                        msgrcv(msgqid,&msg,BUFSIZE,0,0);
                cltpid=atoi(msg.mtext);
                printf("serving for client %d\n",cltpid);
                serpid=getpid();
                msg.mtype=cltpid;
                sprintf(str,"%d",serpid);
                strcpy(msg.mtext,str);
                msgsnd(msgqid,&msg,BUFSIZE,0);
        }
        return 0;
}
```

四、实验总结

进程间的同步和通信是并发和多线程的重要一环，程序运行是本质是一个状态机，特别是在并发的时候我们需要保证一些操作的原子性以防止程序运行到我们事先没预想到的结果，通信可以让不同进程间，不同栈帧间除了全局变量，还有其他可以共享数据的方式