南京邮电大学

实 验 报 告

（ 2025/ 2026学年 第 2 学期）

课程名称	操作系统基础
实验名称	进程通信
实验时间	2026 年 5 月 8 日
指导单位	计算机学院
指导教师	

学生姓名		班级学号	
学院(系)	计算机学院	专　　业	软件工程

实 验 报 告

实验名称	进程通信			指导教师	吴晓诗
实验类型	验证	实验学时	2	实验时间	2026 5 8

一. 实验目的和要求

1. 熟练使用 Linux 下的 C 语言开发环境;
2. 掌握 Linux 操作系统下并发进程间的同步;
3. 掌握 Linux 操作系统下的进程间通信。

二、实验环境(实验设备)

Windows + VMWare + Ubuntu

三、实验原理及内容

本次实验主要内容如下:

(1) 了解 Linux 下进程通信的几种常用方式:信号机制、消息队列、共享内存以及管道。

(2) 掌握共享内存通信的常用系统调用 shmget / shmat / shmdt / shmctl,完成一对发送 / 接收程序。

(3) 掌握消息队列通信的常用系统调用 msgget / msgsnd / msgrcv / msgctl,在两个终端中观察通信效果。

(4) 使用 pipe、signal、kill 实现父子进程的管道通信和自定义信号处理。

(5) 通过对比父子进程中变量的取值,理解进程地址空间相互独立这一关键概念。

2. 共享内存通信方式。

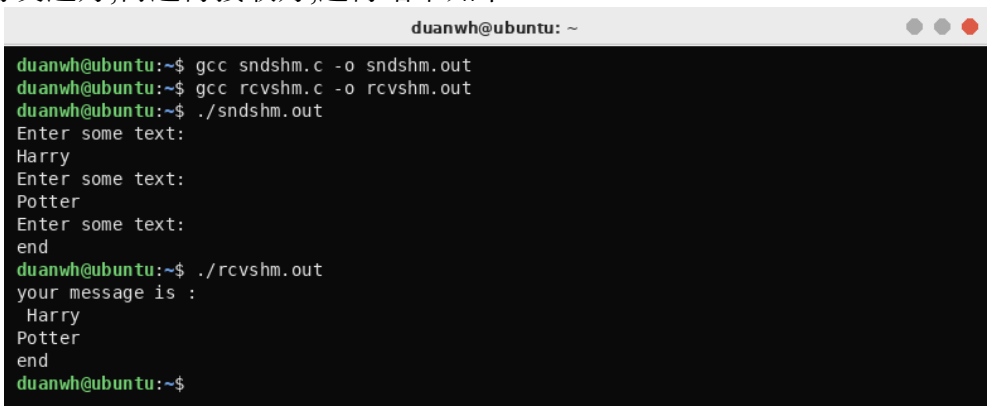
共享内存是让多个进程把同一段物理内存映射进各自的虚拟地址空间。一旦映射建立,谁写谁读都是直接读写内存,速度非常快,但需要自行处理同步。本实验中发送进程把用户输入的字符串追加到共享区域,接收进程一次性把内容读出来。

(1) 发送进程 sndshm.c 的核心代码:

```
/* sndshm.c */
#include <sys/shm.h>
int main() {
    int shmid;
    char *viraddr, buffer[BUFSIZ];
    shmid = shmget(1234, BUFSIZ, 0666 | IPC_CREAT);
    viraddr = (char *)shmat(shmid, 0, 0);
    while (1) {
        puts("Enter some text:");
        fgets(buffer, BUFSIZ, stdin);
        strcat(viraddr, buffer);
        if (strncmp(buffer, "end", 3) == 0) break;
    }
    shmdt(viraddr);
    return 0;
}
```

(2) 接收进程 rcvshm.c 用同样的 key (1234) 调用 shmget 拿到同一段共享内存,把内容打印出来,最后用 shmctl(shmid, IPC_RMID, 0) 删除该共享内存。

(3) 先运行发送方,再运行接收方,运行结果如下:



可以看到发送方分三次输入了 Harry、Potter、end,接收方一次就把整段内容(包括末尾的 end)都读出来了。

3. 消息队列通信方式。

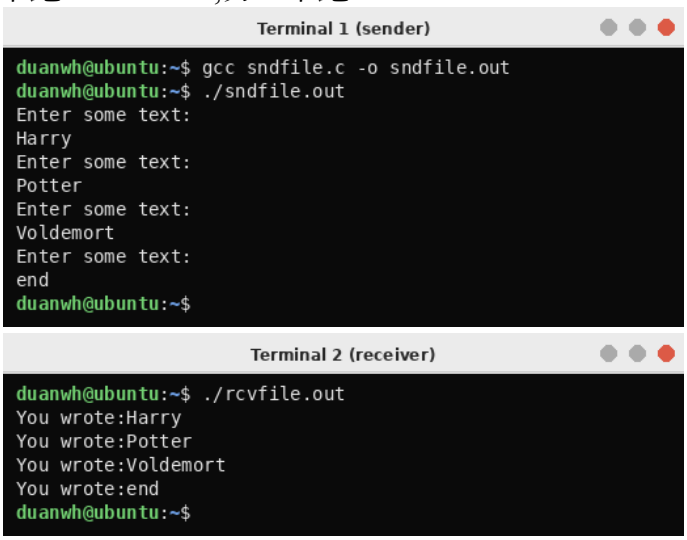
消息队列由内核维护,发送方把消息放进队列,接收方按消息类型从队列中取走。与共享内存不同,消息队列自带同步,不需要发送 / 接收方严格按顺序启动。

(1) 发送进程 sndfile.c 关键代码:

```
struct my_msg { long my_msg_type; char some_text[MAXMSG]; } msg;
msgid = msgget(1234, 0666 | IPC_CREAT);
while (1) {
    fgets(buffer, BUFSIZ, stdin);
    msg.my_msg_type = 1;
    strcpy(msg.some_text, buffer);
    msgsnd(msgid, &msg, MAXMSG, 0);
    if (strncmp(msg.some_text, "end", 3) == 0) break;
}
```

(2) 接收进程 rcvfile.c 调用 msgrcv 从队列中取消息,读到 end 后调用 msgctl 删除队列。

(3) 打开两个终端,一个跑 sndfile.out,另一个跑 rcvfile.out:



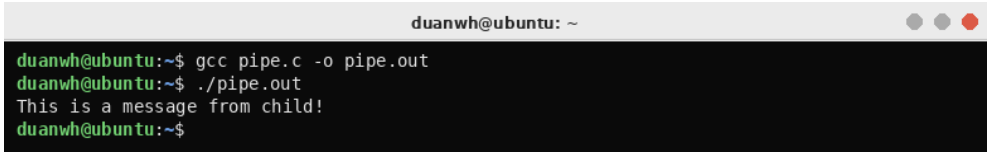
可以看到发送方每输入一行,接收方那边就实时多打印一行,说明消息正确地从内核队列里被取走了。

4. 无名管道(pipe)通信。

pipe 创建一个单向管道,fd[0] 用于读、fd[1] 用于写。在 fork 之前调用 pipe,fork 后父子进程都持有这两个文件描述符,就可以通过它进行通信。

```
/* pipe.c */
int p1, fd[2];
char outpipe[50];
char inpipe[50] = "This is a message from child!";
pipe(fd);
while ((p1 = fork()) == -1);
if (p1 == 0) {                  /* 子进程写 */
    write(fd[1], inpipe, 50);
} else {                         /* 父进程读 */
    wait(0);
    read(fd[0], outpipe, 50);
    printf("%s \n", outpipe);
}
```

运行结果:



父进程通过 wait(0) 等子进程写完再读,所以一次就读到了子进程写入的字符串。

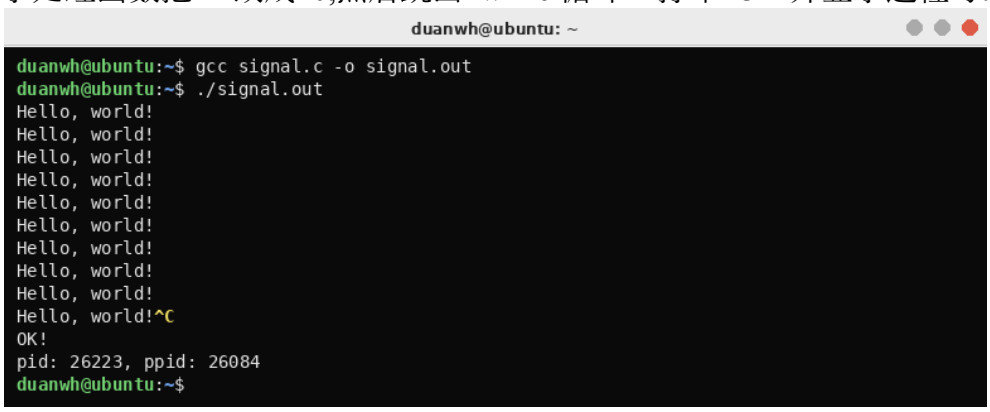
5. 信号机制。

信号是一种异步通信机制,可以由用户键入 (如 Ctrl+C 发送 SIGINT) 或由 kill 命令显式发送。下面的程序用 signal(SIGINT, int_func) 给 SIGINT 注册了一个处理函数,把循环变量 k 置 0,从而优雅退出。

```
/* signal.c */
#include <signal.h>
int k;
void int_func(int sig) { k = 0; }
int main() {
    signal(SIGINT, int_func);
```

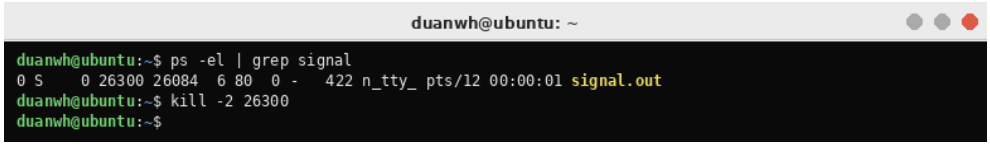
```
    k = 1;
    while (k == 1) printf("Hello, world!\n");
    printf("OK!\n");
    printf("pid: %d, ppid: %d \n", getpid(), getppid());
}
```

运行后程序不停打印 Hello world,按下 Ctrl+C (在终端中显示为 ^C) ,程序并未直接退出,而是执行了处理函数把 k 改成 0,然后跳出 while 循环、打印 OK 并显示进程号:



如果把 signal(SIGINT, int_func) 一行注释掉,再次运行后按 Ctrl+C,程序就会直接被中断,不会打印 OK,说明这时使用的是 SIGINT 的默认处理 (终止进程) 。

(2) 用 kill 命令发送信号。打开两个终端,一个跑 signal.out,另一个用 ps 找到它的进程号,然后用 kill -2 PID 发送 SIGINT,效果与 Ctrl+C 完全相同:



6. 验证:简单变量和指针都不能跨进程通信。

初学者容易误以为父进程定义的全局变量,fork 出来的子进程也可以"共享",其实不是的。fork 之后两个进程各自拥有独立的虚拟地址空间。下面的程序在子进程中把 year 改成 2018,父进程 sleep 4 秒后再打印,会发现父进程那边的 year 仍是 1368。

```
int year = 1368;
pid = fork();
if (pid == 0) {
    year = 2018;
    printf("child year after: %d\n", year);
} else {
    sleep(4);
    printf("parent year after: %d\n", year);
}
```

运行结果:

即便父子进程中 &year 打印出来的虚拟地址完全相同,它们指向的也是两段不同的物理内存,子进程修改不影响父进程。所以共享变量并不存在,进程间通信必须依靠操作系统提供的 IPC 机制。

实 验 报 告

实 验 报 告

四、实验小结（包括问题和解决方法、心得体会、意见与建议等）

本次实验主要目的是理解操作系统中进程通信的几种典型机制并实际跑通它们。整个实验下来,对"进程间通信不同于同一个进程内函数之间的数据传递"这件事有了非常直观的认识。主要收获如下:

1. 完成了共享内存的发送 / 接收程序,熟悉了 shmget → shmat → 读写 → shmdt → shmctl 这一组流程。共享内存速度最快但需要自己处理同步。

2. 完成了消息队列通信,体会到与共享内存最大的不同:消息队列自带同步,发送和接收方启动的先后顺序没有限制,msgrcv 在队列空时会自动阻塞。

3. 用 pipe 实现了父子进程通信,理解了管道是半双工的,fd[0] 读 fd[1] 写。

4. 信号机制是异步的。对比有 / 无 signal(SIGINT, ...) 两种情况,清楚地看到:有处理函数时是 "接到信号→执行处理→继续主流程",没有则是默认的 "终止进程"。

5. 通过对比父子进程中变量的取值,真正理解了进程的虚拟地址空间是相互独立的,即使虚拟地址相同物理上也是两份拷贝。这就是为什么进程通信非用 IPC 不可。

问题及解决:

1. 一开始共享内存程序的接收方先启动,屏幕上什么也不打印。后来意识到这是顺序问题:发送方还没写入,接收方读到的当然是空,改成先跑发送方就好了。

2. 消息队列实验做完后如果没有 msgctl(IPC_RMID),队列会一直留在系统里。可以用 ipcs -q 查看遗留队列、ipcrm -q msqid 手动清理。

3. signal 测试中,kill -9 PID 发送的 SIGKILL 是不能被捕获或忽略的,自定义处理函数不会被调用。

意见与建议:

1. 希望后续能再做一个同步原语 (信号量 / 互斥锁) 的实验,把共享内存的同步问题真正写一遍。

2. 实验文档可以补充 ipcs / ipcrm 等命令的简单说明。

五、指导教师评语

成 绩		批阅人		日 期	