**北京邮电大学软件学院**

**2018学年第1学期实验报告**

**课程名称： 操作系统**

**实验名称：进程通讯**

**实验完成人：**

**姓名：周子格**

**学号：2016211978**

**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2018 年 3 月 1 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**

Linux centos

1. **实验过程描述**

在这里分别以文字和流程图的方式画出你的实验的几个阶段。

1. 实验内容分析

**Lab2\_1:**

子进程作为Client，父进程作为Server

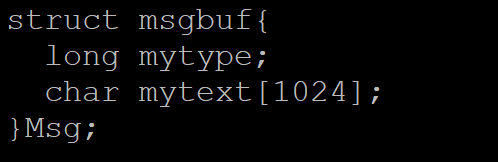
BUG1:

当父进程fork子进程时：子进程复制父进程的资源，说共享实际上是不准确的。

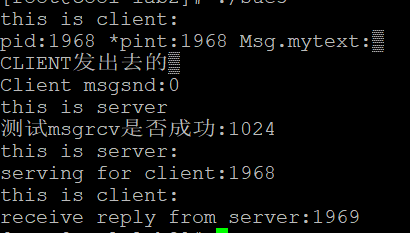
当资源只读时，表面上是共享。当资源被改写时，子进程会重新创建一块资源，来放这个数据。

但是msgbuf的机制应该和子父进程共享资源的机制不一样。Msgbuf是真正的共享资源。A进程更改msgbuf的text，B进程的msgbuf的text就会被改变。

所以在两个进程中都创建msgbuf的方式是错的。（maybe）



BUG2:



Client发送int数据，Server接收int数据。

# 之前尝试的方式是：atoi()和itoa()函数

# 但linux中不能使用itoa函数,这是windows下才有的函数。

# 搜索后改成sprintf函数但出现“Segmentationfault“错误提示。

# 搜索后改成snprintf函数。仍然出现此错误。

# 发现：atoi(),itoa()函数之所以成套出现是有原因的。Int转成string，string转成int的方式就像编码和解码的方式一样，必须成套出现。方式的不同决定了int是怎么存在内存中的。

# 最后，使用指针的方式：

# 

# 

# 编码，解码成套出现。

# 成功解决bug

# End：

**Lab2\_2:**

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。

**Q1:** Shmaddr：用户给定的，将共享存储区附接到进程的虚拟地址空间；

**如何定义？**

Shmat函数返回附加好的共享内存地址。

若想发string类型数据，则将地址强制类型转换为指向char的指针。再往里放string。

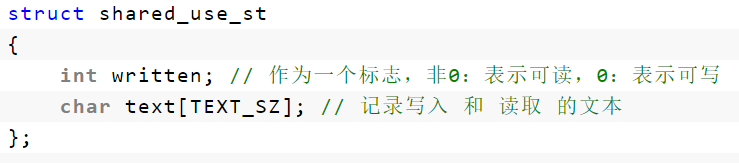
**Shmaddr：**指定共享内存出现在进程内存地址的什么位置，直接指定为NULL让内核自己决定一个合适的地址位置

**Q2:如何将内存空间附加到自己的地址空间？**

shmat(把共享内存区对象映射到调用进程的地址空间)

**Q3:发送消息要不要加锁？A进程怎么知道B进程写完？**

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。所以我们通常需要用其他的机制来同步对共享内存的访问，例如前面说到的信号量



**Written就是锁。**

**0可写，非0可读。**

**若written==0，表示shm可写。ProcessB开始写。写完后将writtem重置为1。因为writtem==1，表示shm可读。ProcessA开始读。读完后将written重置为0.。**

**Q4：**shm有点类似通过 malloc() 获取到的内存，所以这里需要做个 类型强制转换。



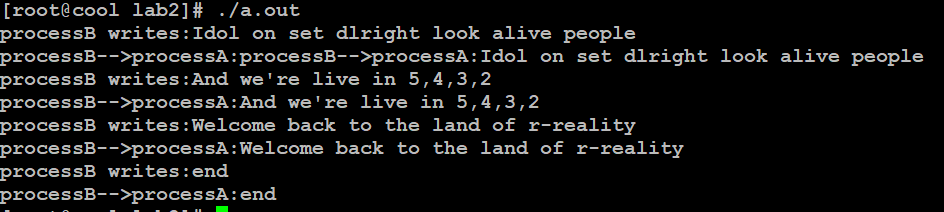
**Q5：shmctl和shmdt有什么不同？**

shmdt是将共享内存从进程空间detach出来.   
shmctl(sid,IPC\_RMID,0)则是将共享内存从系统空间delete.

1. 调用shmctl进程没有退出, 而且没有shmdt, 资源仍然被占用, 先前shmat返回的地址仍然是可用的而且是合法的.
2. 如果其他进程没有shmdt, 仍然可以继续合法的使用.

**若直接shmctl，**addr就成野指针了；所以，要先shmdt(addr)，再shmctl。

**End：**



**原理解析：**

**1消息队列**

消息队列是消息的链接表，包括 Posix 消息队列 systemV 消息队列。它克服了前两种通信方式中信息量有限的缺点，具有写权限的进程可以向消息队列中按照一定的规则添加新消息；对消息队列有读权限的进程则可以从消息队列中读取消息。

消息队列通过系统调用 msgget()、msgrcv()来实现，函数的功能和实现过程分别如下。 原型：int msgget（key\_t key，int msgflg）

参数：

key：消息队列关联的键。

msgflg：消息队列的建立标志和存取权限。

返回值：成功执行时，返回消息队列标识值。失败返回-1，errno 被设为以下的某个值：

EACCES ： 指定的消息队列已存在 ，但调用进程没有权限访问它 ，而且不拥有

CAP\_IPC\_OWNER 权能。

EEXIST：key 指定的消息队列已存在，而 msgflg 中同时指定 IPC\_CREAT 和 IPC\_EXCL 标志。

ENOENT：key 指定的消息队列不存在同时 msgflg 中不指定 IPC\_CREAT 标志。

ENOMEM：需要建立消息队列，但内存不足。

ENOSPC：需要建立消息队列，但已达到系统的限制

原型：ssize\_t msgrcv(int msqid, void \*msgp, size\_t msgsz, long msgtyp, int msgflg)

参数：

msqid：消息队列的识别码。

msgp：指向消息缓冲区的指针，此位置用来暂时存储发送和接收的消息，是一个用户 可定义的通用结构。

msgsz：消息的大小。

msgtyp：从消息队列内读取的消息形态。如果值为零，则表示消息队列中的所有消息 都会被读取。

msgflg：用来指明核心程序在队列没有数据的情况下所应采取的行动。如果 msgflg 和常数 IPC\_NOWAIT 合用，则在消息队列呈空时，不做等待马上返回-1，并设定错误码为 ENOMSG。当 msgflg 为 0 时， msgrcv()在队列呈满或呈空的情形时，采取阻塞等待的处理模式。

返回值：当成功执行时， msgrcv()返回拷贝到 mtext 数组的实际字节数。失败返回-1， errno 被设为以下的某个值：

E2BIG：消息文本长度大于 msgsz，并且 msgflg 中没有指定 MSG\_NOERROR。 EACCES：调用进程没有读权能，同时没具有 CAP\_IPC\_OWNER 权能。

EAGAIN：消息队列为空，并且 msgflg 中没有指定 IPC\_NOWAIT。

EFAULT：msgp 指向的空间不可访问。

EIDRM：当进程睡眠等待接收消息时，消息已被删除。

EINTR：当进程睡眠等待接收消息时，被信号中断。

EINVAL：参数无效。

ENOMSG：msgflg 中指定了 IPC\_NOWAIT，同时所请求类型的消息不存在。

**2共享内存**

共享内存是操作系统常采用的进程间通信方式。它使得多个进程可以访问同一块内存空间，不同进程可以及时看到对方进程中对共享内存中数据的更新。这种通信方式需要依靠某种同步机制，如互斥锁和信号量等。

共享内存通过系统调用 shmget()、shmat()、shmdt()、shmctl()来实现，函数的功能和实现过程分别如下**。**

系统调用：shmget()

原型：int shmget(key\_t key, int size, int shmflg)

参数：

Key:共享存储区的名字；

Size：共享存储区的大小（以字节计）；

Shmflag：用户设置的标志，如 IPC\_CREAT，IPC\_CREAT 表示若系统中尚无指名的共 享存储区，则由核心建立一个共享存储区；如系统中已有共享存储区，便忽略 IPC\_CREAT。

返回值：

成功返回共享内存的标识符；不成功返回-1，errno 储存错误原因。

EINVAL 参数 size 小于 SHMMIN 或大于 SHMMAX。

EXIST 预建立 key 所致的共享内存，但已经存在。

EIDRM 参数 key 所致的共享内存已经删除。

ENOSPC 超过了系统允许建立的共享内存的最大值(SHMALL )。

ENOENT 参数 key 所指的共享内存不存在，参数 shmflg 也未设 IPC\_CREAT 位。

EACCES 没有权限。

ENOMEM 核心内存不足。

系统调用：shmat()

原型：void\* shmat(int shmid, const void \*shmaddr, int shmflg);

参数：

Shmid：共享存储区的标识符；

Shmaddr：用户给定的，将共享存储区附接到进程的虚拟地址空间；

Shmflg：规定共享存储区的读、写权限，以及系统是否应对用户规定的地址做舍 入操作。其值为 SHM\_RDONLY 时，表示只能读，其值为 0 时，表示可读、可写，其值为 SHM\_RND （取整）时，表示操作系统在必要时舍去这个地址。

返回值：

该系统调用的返回值为共享存储区所附接到的进程虚地址。

系统调用：shmdt()

原型：int shmdt(void \*shmaddr)

参数：

Shmaddr：要断开连接的虚地址，亦即以前由连接的系统调用 shmat（）所返回的虚地址。

返回值：

调用成功时，返回 0 值，调用不成功，返回-1 值。

系统调用：shmctl()

原型：int shmctl(int shmid, int cmd, struct shmid\_ds \*buf);

参数：

Shmid：由 shmget 所返回的标识符； Cmd：是操作命令，可以分多种类型：

（1）用于查询有关共享存储区的情况。如其长度、当前连接的进程数、共享 区的创建者标识符。

（2）用于设置或改变共享存储区的属性，如共享存储区的许可权、当前连接 的进程计数等。

（3）对共享存储区的加锁和解锁命令。

（4）删除共享存储区标识符等。

Buf：用户缓冲区地址。

返回值：

调用成功则返回 0，如果失败则返回-1。

6 程序设计、调试

7 程序运行、跟踪和成果收集

8 实验总结

若是实验分组一定注明每个人具体工作量。

1. **实验结果**
2. **附件**

**6.1 附件1：源代码**

**6.2 附件2：XXX**