**实验三 进程间通信**

# 姓 名徐昊博 学号 21013134 成绩

实验时间2023年11月9日 指导教师(签名)

**（诚信声明：本实验报告内容，均由本人亲自上机完成。 签名：\_\_\_\_\_\_\_\_\_\_\_\_\_）**

一．实验目的

1.掌握利用管道机制实现进程间的通信的方法

2.掌握利用消息缓冲队列机制实现进程间的通信的方法

3.掌握利用共享存储区机制实现进程间的通信的方法

4.加深对上述三种通信机制的理解。

二．实验工具与设备

已安装 Linux 操作系统的计算机。

三．实验内容

1. **掌握实现进程间通信的系统调用的功能和方法**
2. **编写一段程序，实现进程间的管道通信。 其中，父进程通过管道向子进程发送一个字符串（子进程的进程号），子进程将它显示出来。**

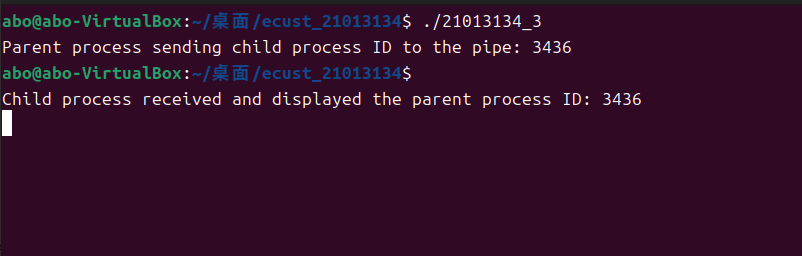
实验代码如下：



根据实验四.1.(3)使用pipe(p)创建了一个管道，使用数组p来存储管道的读写文件描述符。

pid = fork();创建了一个新的进程。如果成功的话，在父进程中pid>0：首先close(p[0])关闭了管道的读取端。使用sprintf()创建包含父进程ID的消息，并通过write()写入管道。然后close(p[1])在写入消息后关闭了管道的写入端。在子进程中当pid=0使用close(p[1])关闭了管道的写入端。使用read()从管道中读取消息（父进程ID）并进行显示。 在读取消息后关闭了管道的读取端。

运行结果如下：



1. **编写一段程序，使用消息缓冲队列来实现 client 进程和 server 进程之间的通信。**

先使用 fork( )建立两个子进程 server 和 client。

server 进程先建立一个关键字为 MSGKEY(如 75)的消息队列，等待其他进程发来的消息。server

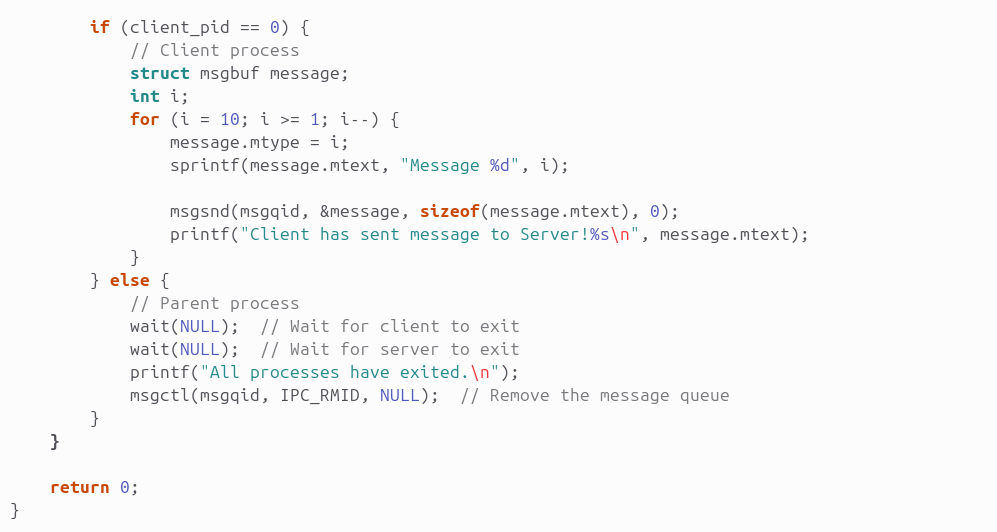
进程每接收到一个消息，便显示字符串“Server has received message from Client!”。当遇到类型为 1

的消息，则作为结束信号，取消该队列，并退出 server。

client 进程则使用关键字为 MSGKEY 的消息队列，先后发送类型从 10 到 1 的消息，然后退出。

client 进程每发送一条消息，便显示字符串“Client has sent message to Server!”。最后一条消息，是 server 进程需要结束的信号。父进程在 server 和 client 均退出后结束。



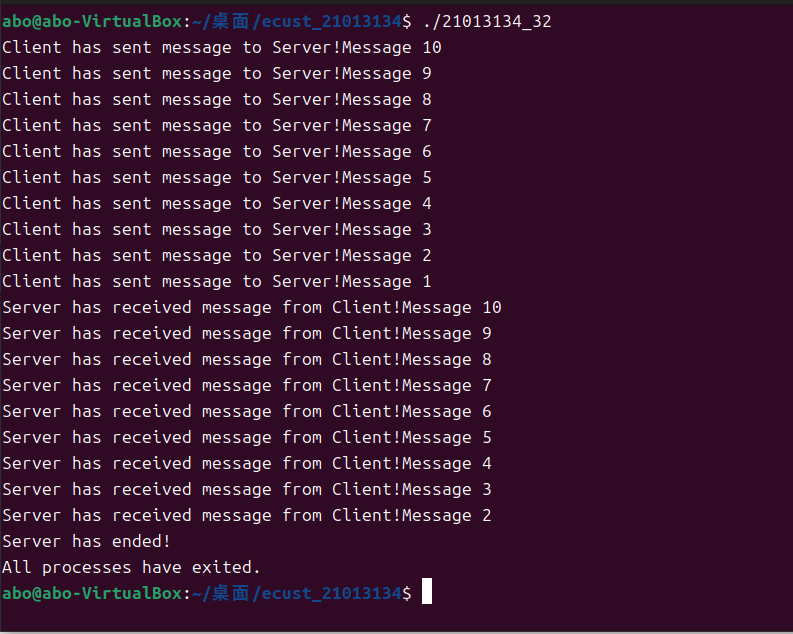


首先进行消息队列的创建：使用msgqid=msgget(MSGKEY, 0666|IPC\_CREAT) 。如果队列已存在，它会使用现有队列的标识符。否则，它会新建一个队列并返回其标识符。标识符由MSGKEY指定，并且使用权限为0666，表示可读可写权限。

当server\_pid==0时，进程处于服务器模式。在一个无限循环中，服务器通过msgrcv()函数接收消息队列中的消息。如果收到的消息类型（message.mtype）为1，设定为退出信号，服务器会通过msgctl()函数删除消息队列并结束自己的执行。

当 server\_pid!=0时，进程为父进程。它创建了一个客户端进程（client\_pid）。客户端进程（client\_pid==0）会向消息队列中发送10条不同类型的消息（从10到1），每条消息都包含一个消息类型和文本内容。父进程等待客户端进程和服务器进程结束通过wait(NULL)实现。一旦两个子进程都结束，父进程输出 "All processes have exited." 表示所有进程都已退出，并使用 msgctl()删除消息队列。

运行结果如下：



1. **编写一个与上述功能相同的程序，使其用共享存储区来实现两个进程之间的通信。**

先使用 fork( )建立两个子进程 server 和 client。

server 进程先建立一个关键字为 SHMKEY(如 75)的共享区，并将第一个字节置为-1，作为数据

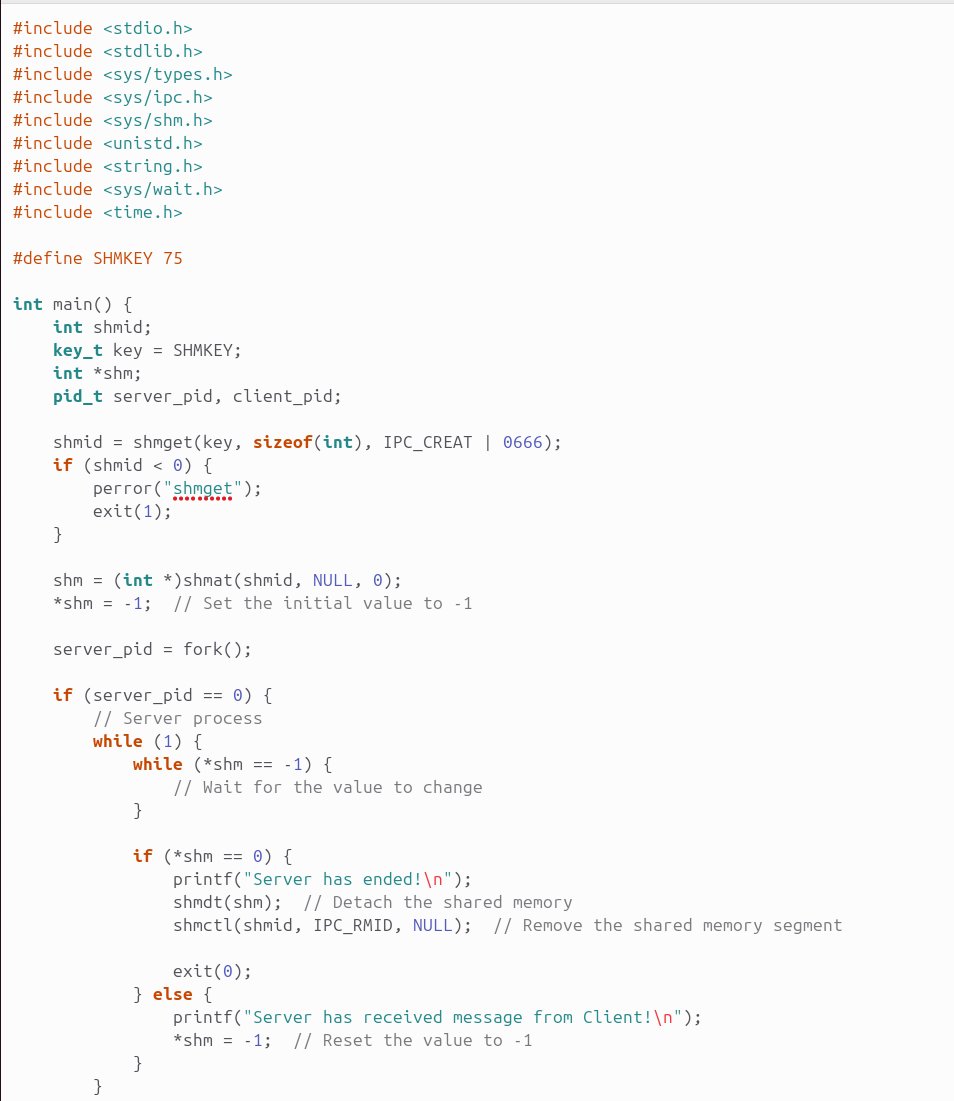
空的标志。等待其他进程发来的消息，当该字节的值发生变换时，表示收到了信息，进行处理，然

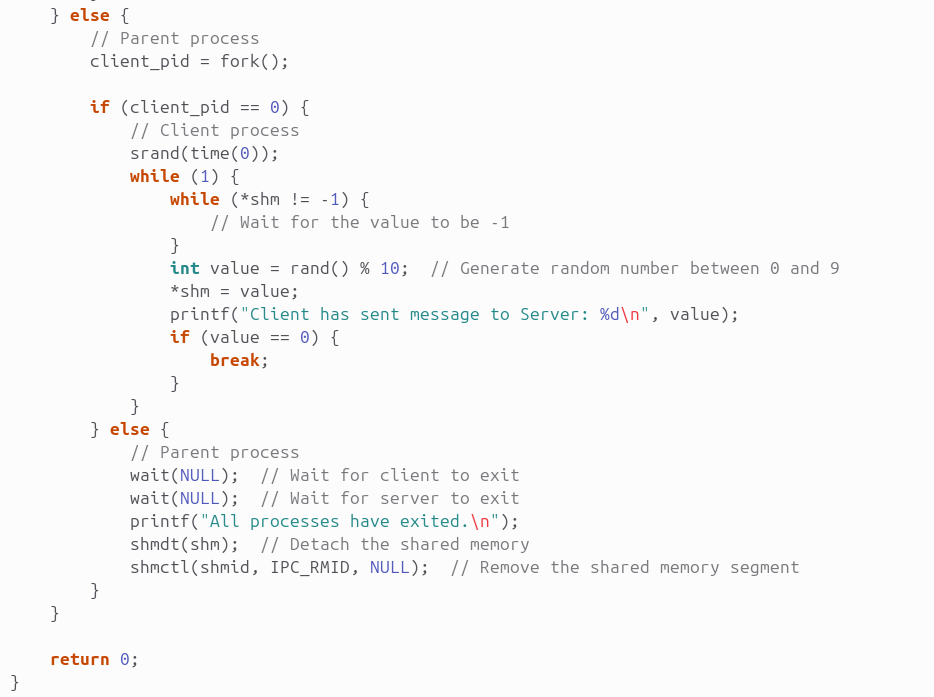
后再次把它的值置为-1。server 进程每接收到一个消息，便显示字符串“Server has received message

from Client!”。当遇到的值为 0，则视为结束信号，取消该存储区，并退出 server。

client 进程则建立一个关键字为 SHMKEY(如 75)的共享区，当共享取得第一个字节为-1 时，server 端空闲，可发送请求。client 随机填入 9 到 0。期间等待 server 端的再次空闲。进行完这些操作后， client 退出。client 进程每发送一条消息，便显示字符串“Client has sent message to Server!”。

父进程在 server 和 client 均退出后结束。





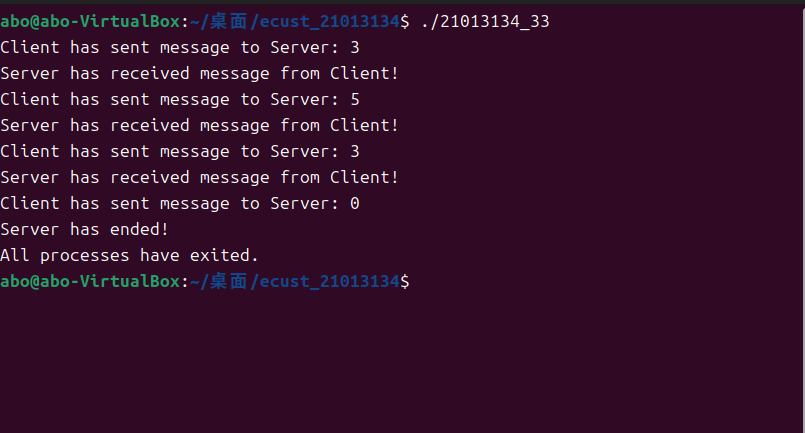
首先进行共享内存的创建同上一题通过方法：shmat(shmid, NULL, 0)将共享内存连接到进程的地址空间中，返回一个指向共享内存起始地址的指针。

当 server\_pid=0时，进程处于服务器模式。在一个无限循环中，服务器等待共享内存中的值改变。当共享内存中的值为0时，服务器打印消息并清理共享内存，并结束自己的执行。

当 server\_pid != 0时，进程为父进程。它创建了一个客户端进程（client\_pid）。客户端进程不断在共享内存中写入随机生成的0到9之间的整数，并在写入0后退出。

父进程等待客户端进程和服务器进程结束通过wait(NULL)实现。一旦两个子进程都结束，父进程输出 "All processes have exited." 表示所有进程都已退出，并使用shmdt()分离共享内存，然后使用shmctl()删除共享内存段。

运行结果如下：



# 四．思考题

1. **上述哪些通信机制提供了发送进程和接收进程之间的同步功能？这些同步是如何进行的？**

管道通信机制提供了同步功能。

在管道通信中，主要使用两个文件描述符，一个用于读取，另一个用于写入。父进程创建管道后，通过`fork()`创建子进程后，它们共享同一个管道。

实现同步的关键在于：

1. 阻塞特性：管道在读取端和写入端的操作上都具有阻塞特性。如果读取端没有数据可读，读取操作将会阻塞，直到有数据可读；如果写入端写入数据时，如果管道已满，写入操作也会阻塞，直到管道有空间可以写入数据。

2. 父子进程之间的通信顺序：在父子进程通信时，由于管道是单向的，父进程向管道写入数据，子进程从管道读取数据。因此，在这种场景下，子进程会阻塞等待父进程向管道写入数据。

基于这些特性，可以通过以下方式实现一定程度的同步：

父子进程之间的同步：

父进程和子进程在通过管道进行通信时，由于管道的阻塞特性，父进程写入数据到管道时，如果子进程暂时没有进行读取操作，父进程的写入操作会阻塞，直到子进程进行读取操作，才能继续执行。

这样保证了父进程写入操作的同步，即父进程写入数据后，会等待子进程读取数据之后才能继续执行，实现了一定程度的同步机制。

虽然管道在读写操作上具有阻塞特性，但管道本身并未提供对多个进程之间的显式同步控制。因此，在多个无关联进程之间的通信中，管道本身并不能提供灵活的同步控制机制，开发者可能需要使用其他同步原语（如信号量、互斥锁等）来确保进程间通信的同步和互斥。

1. **上述通信机制各有什么特点，它们分别适合于何种场合？**

1. 管道（Pipes）

特点：单向通信，常用于父子进程之间或者相关性较强的进程间通信。 基于文件描述符，在文件系统中没有对应的文件，只存在于内存中。 仅支持单向数据流，可以是匿名管道（只在相关进程间使用）或命名管道（允许无关联进程间通信）。

适用场合：适用于父子进程之间或者有相关性的进程间通信。在需要简单、轻量级的数据交换时，尤其适合串行数据流的场景。

2. 消息队列（Message Queues）

特点：提供了异步、可靠的进程间通信方式。消息的发送和接收是通过消息队列标识符控制的。具备消息类型，支持不同类型的消息传递。

适用场合：在需要不同类型的消息传递，或者多个进程之间进行解耦的场景下很有用。适合多个进程之间大量数据交换，尤其是异步且无需直接进行数据共享的情况。

3. 共享内存（Shared Memory）

特点：允许多个进程访问同一块物理内存，因此效率高。数据直接存储在共享内存中，进程间直接进行读写，无需数据拷贝。不提供同步机制，需结合其他同步原语控制访问。

适用场合：

适用于大量数据交换或频繁数据访问的场景，因为共享内存效率高。当进程需要高效共享大块数据时，但需要谨慎处理同步和互斥问题。

在实际应用中，根据具体的需求和通信特点选择合适的通信机制很重要。通常会根据数据交换量、进程间关系、同步要求等因素来决定选择哪种进程间通信机制。