

《操作系统》lab3

|  |  |
| --- | --- |
| 学 院： | 计算机与信息技术学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 计科1602 |
| 姓 名： | 麻锦涛 |
| 学 号： | 16281262 |

**日期：2019.4.22**

1. **实验目的**

该项目的目标是实践各种IPC方法（用于数据传递和同步）并学习map-reduce（并行计算）。两者都是工业中经常使用的非常重要的技术

1. **实验要求**

该项目由三个独立的子项目组成，每个子项目执行相同的任务：给定一个文本文件，程序输出包含给定单词的行。例如，给出一行“Hello World！”并假设感兴趣的词是“世界”，那么应该输出这一行。 （注意，如果一行是“Hello worlds”，则不应输出该行）。您的程序执行（即父进程）将创建子进程。父进程可以打开文本文件，读取内容，并使用以下方法之一将其传递给子进程，但无法检查单词，而子进程无法打开该文件但可以检查单词。最后，父进程（而不是子进程）应按字母顺序输出行。以下是三个子项目的要求：

1.使用Pipe作为传递文件内容和结果的方法。

2.使用Unix Domain Socket作为传递文件内容和结果的方法。

3.使用共享内存作为传递文件内容和结果的方法。此外，子进程创建4个线程，每个线程充当Mapper;并且子进程的主线程作为单个reducer运行。只有这个子项目需要Map-reduce。不能使用Hadoop作为map-reduce基础架构;相反，你必须使用Posix线程编程来实现map-reduce。

1. **实现过程**

**进程间使用管道通信**

管道本身是一种数据结构，遵循先进先出原则。先进入管道的数据，也能先从管道中读出。数据一旦读取后，就会在管道中自动删除。管道通信以管道数据结构作为内部数据存储方式，以文件系统作为数据存储媒体。Linux系统中有两种管道，分别是无名管道和命名管道。pipe系统调用可创建无名管道，open系统调用可创建命名管道。下面介绍这两种管道的实现方式。

**1 。1 pipe系统调用**

系统调用pipe用来建立管道。与之相关的函数只有一个，即pipe()函数，该函数被定义在头文件unistd.h中，它的一般形式是：

int pipe(int filedes[2]);

pipe系统调用需要打开两个文件，文件标识符通过参数传递给pipe()函数。文件描述符filedes[0]用来读数据，filedes[1]用来写数据。调用成功时，返回值为0，错误时返回-1。管道的工作方式可以总结为以下3个步骤。

**1．将数据写入管道**

将数据写入管道使用的是write()函数，与写入普通文件的操作方法一样。与文件不同的是，管道的长度受到限制，管道满时写入操作会被阻塞。执行写操作的进程进入睡眠状态，直到管道中的数据被读取。管道满时，write()函数的返回值为0。如果写入数据长度小于管道长度，则要求一次写入完成。如果写入数据长度大于管道长度，在写完管道长度的数据时，write()函数将被阻塞。

**2．从管道读取数据**

读取数据使用read()函数实现，读取的顺序与写入顺序相同。当数据被读取后，这些数据将自动被管道清除。因此，使用管道通信的方式只能是一对一，不能由一个进程同时向多个进程传递同一数据。如果读取的管道为空，并且管道写入端口是打开的，read()函数将被阻塞。读取操作的进程进入睡眠状态，直到有数据写入管道为止。

**3．关闭管道**

管道虽然有2个端口，但只有一个端口能被打开，这样避免了同时对管道进行读和写的操作。关闭端口使用的是close()函数，关闭读端口时，在管道上进行写操作的进程将收到SIGPIPE信号。关闭写端口时，进行读操作的read()函数将返回0。

1. #include **<unistd.h>**                     // 标准函数库
2. #include **<sys**/types.h**>**                  // 该头文件提供系统调用的标志
3. #include **<sys**/wait.h**>**                   // wait系统调用相关函数库
4. #include **<stdio.h>**                      // 基本输入输出函数库
5. #include **<string.h>**                     // 字符串处理函数库
6. int main()
7. {
8. int fd[2], cld\_pid, status;          // 创建文件标识符数组
9. char buf[200], len;                  // 创建缓冲区
10. if (pipe(fd) == -1) {                // 创建管道
11. perror("创建管道出错");
12. exit(1);
13. }
14. if ((cld\_pid=fork()) == 0) {         // 创建子进程，  
    判断进程自身是否是子进程
15. close(fd[1]);                     // 关闭写端口
16. len = read(fd[0], buf, sizeof(buf));      // 从读  
    端口中读取管道内数据
17. buf[len]=0;                               // 为缓  
    冲区内的数据加入字符串
18. // 结束符
19. printf("子进程从管道中读取的数据是：%s ",buf);  //   
    输出管道中的数据
20. exit(0);                                  // 结束子进程
21. }
22. else {
23. close(fd[0]);                             // 关闭读端口
24. sprintf(buf, "父进程为子进程（PID=%d）创建该数据", cld\_pid);
25. // 在缓  
    冲区创建字符串信息
26. write(fd[1], buf, strlen(buf));           // 通过  
    写端口向管道写入数据
27. exit(0);                                  // 结束父进程
28. }
29. return 0;
30. }

程序中，首先创建了一个管道，并且将管道的文件标识符传递给fd[]数组。该数组有2个元素，fd[0]是读取管道的端口，fd[1]是写入管道的端口。然后，通过fork()系统调用创建了一个子进程。父进程的操作是向管道写入数据，子进程的操作是读取管道内的数据，最后子进程将所读取的数据显示到终端上。

1. **实验总结与收获**

**1.linux下进程间通信的几种主要手段：**

1）管道（Pipe）及有名管道（named pipe，FIFO）

匿名管道是Linux支持的最初Unix IPC形式之一，具有以下特点：

管道是半双工的，数据只能向一个方向流动；需要双方通信时，需要建立起两个管道；STREAMS管道是一个双向（全双工）管道，单个STREAMS管道就能向父、子进程提供双向数据流。Solaris支持STREAMS管道，Linux的可选附加包也提供了STREAMS管道。

只能用于父子进程或者兄弟进程之间（具有亲缘关系的进程）；

单独构成一种独立的文件系统：管道对于管道两端的进程而言，就是一个文件，但它不是普通的文件，它不属于某种文件系统，而是自立门户，单独构成一种文件系统，并且只存在与内存中。

数据的读出和写入：一个进程向管道中写的内容被管道另一端的进程读出。写入的内容每次都添加在管道缓冲区的末尾，并且每次都是从缓冲区的头部读出数据。

两者的不同点：

（1）匿名管道它没有名字，只能用于具有亲缘关系进程间的通信。有名管道FIFO克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信。

（2）匿名管道对于管道两端的进程而言是一个文件，但不是Linux某种类型的文件、不属于某个文件系统，而且仅存在于内存中；pipe()函数打开了两个文件描述符，分别用于读写。有名管道提供一个路径名与之关联，是linux文件类型的一种，因FIFO文件形式存在于文件系统中。

相同点：

（1）管道和FIFO的数据是字节流，应用程序之间必须事先确定特定的传输"协议"，采用传播具有特定意义的消息。

（2）单向（半双工）数据流。

（3）系统对管道和FIFO的两个限制OPEN\_MAX（一个进程任意时刻打开的最大描述符数）、PIPI\_BUF（可原子地写往一个管道或FIFO的最大数据量，posix要求至少512）。

4）都是随进程持续的IPC（IPC对象一直存在到打开该对象的最后一个进程关闭该对象为止。）

管道常用于两个方面：

（1）在shell中时常会用到管道（作为输入输入的重定向），在这种应用方式下，管道的创建对于用户来说是透明的；

（2）用于具有亲缘关系的进程间通信，用户自己创建管道，并完成读写操作。

2)Unix域协议

Unix域协议并不是一个实际的协议族，而是在单个主机上执行客户/服务器通信的一种方法，所使用的API就是在不同主机上执行客户/服务器通信所用的API（套接字API）。

Unix域套接字仅仅复制数据，并不执行协议处理，不需要添加或删除网络报头，无需计算校验和，不要产生顺序号，无需发送确认报文。Unix域套接字提供流和数据报两种接口。Unix域数据报服务是可靠的，既不会丢失消息也不会传递出错。它是套接字和管道之间的混合物。

使用Unix域套接字的理由：

（1）Unix域套接字往往比通信两端位于同一主机的TCP套接字快一倍（TCPv3）。Unix域套接字仅仅复制数据，并不执行协议处理，不需要添加或删除网络报头，无需计算校验和，不要产生顺序号，无需发送确认报文。

（2）可用于在同一台主机的不同进程之间传递描述符。

（3）Unix域套接字较新的实现把客户的凭证（用户ID和组ID）提供给服务器，从而提供了额外的安全检查措施。

为了创建一对非命名的、相互连接的UNIX域套接字，用户可以使用它们面向网络的域套接字接口，也可以使用socketpair函数。

3）信号（Signal）

信号是比较复杂的通信方式，用于通知接受进程有某种事件发生，除了用于进程间通信外，进程还可以发送信号给进程本身；linux除了支持Unix早期信号语义函数signal外，还支持语义符合Posix.1标准的信号函数sigaction（实际上，该函数是基于BSD的，BSD为了实现可靠信号机制，又能够统一对外接口，用sigaction函数重新实现了signal函数）。sigaction包含了信号产生的相关信息。

4）消息队列

消息队列是消息的链接表，包括Posix消息队列和system V消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺点。而且消息队列是随内核持续的（IPC对象会一直存在，直到内核重启或显示删除该对象为止）。

5）共享内存

6）信号量（semaphore）

7）套接口（Socket）

更为一般的进程间通信机制，可用于不同机器之间的进程间通信。起初是由Unix系统的BSD分支开发出来的，但现在一般可以移植到其它类Unix系统上：Linux和System V的变种都支持套接字。