Linux下管道的原理

7.1.1 Linux管道的实现机制

在Linux中，管道是一种使用非常频繁的通信机制。从本质上说，管道也是一种文件，但它又和一般的文件有所不同，管道可以克服使用文件进行通信的两个问题，具体表现为：

· 限制管道的大小。实际上，管道是一个固定大小的缓冲区。在Linux中，该缓冲区的大小为1页，即4K字节，使得它的大小不象文件那样不加检验地增长。使用单个固定缓冲区也会带来问题，比如在写管道时可能变满，当这种情况发生时，随后对管道的write()调用将默认地被阻塞，等待某些数据被读取，以便腾出足够的空间供write()调用写。

· 读取进程也可能工作得比写进程快。当所有当前进程数据已被读取时，管道变空。当这种情况发生时，一个随后的read()调用将默认地被阻塞，等待某些数据被写入，这解决了read()调用返回文件结束的问题。

注意：从管道读数据是一次性操作，数据一旦被读，它就从管道中被抛弃，释放空间以便写更多的数据。

1. 管道的结构

在 Linux 中，管道的实现并没有使用专门的数据结构，而是借助了文件系统的file结构和VFS的索引节点inode。通过将两个 file 结构指向同一个临时的 VFS 索引节点，而这个 VFS 索引节点又指向一个物理页面而实现的。如图 7.1所示。

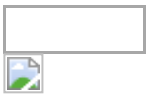

图7.1 管道结构示意图

图7.1中有两个 file 数据结构，但它们定义文件操作例程地址是不同的，其中一个是向管道中写入数据的例程地址，而另一个是从管道中读出数据的例程地址。这样，用户程序的系统调用仍然是通常的文件操作，而内核却利用这种**抽象机制**实现了管道这一特殊操作。

2.管道的读写

管道实现的源代码在fs/pipe.c中，在pipe.c中有很多函数，其中有两个函数比较重要，即管道读函数pipe_read()和管道写函数pipe_wrtie()。管道写函数通过将字节复制到 VFS 索引节点指向的物理内存而写入数据，而管道读函数则通过复制物理内存中的字节而读出数据。当然，内核必须利用一定的机制同步对管道的访问，为此，内核使用了锁、等待队列和信号。

当写进程向管道中写入时，它利用标准的库函数write()，系统根据库函数传递的文件描述符，可找到该文件的 file 结构。file 结构中指定了用来进行写操作的函数（即写入函数）地址，于是，内核调用该函数完成写操作。写入函数在向内存中写入数据之前，必须首先检查 VFS 索引节点中的信息，同时满足如下条件时，才能进行实际的内存复制工作：

·内存中有足够的空间可容纳所有要写入的数据；

·内存没有被读程序锁定。

如果同时满足上述条件，写入函数首先锁定内存，然后从写进程的地址空间中复制数据到内存。否则，写入进程就休眠在 VFS 索引节点的等待队列中，接下来，内核将调用调度程序，而调度程序会选择其他进程运行。写入进程实际处于可中断的等待状态，当内存中有足够的空间可以容纳写入数据，或内存被解锁时，读取进程会唤醒写入进程，这时，写入进程将接收到信号。当数据写入内存之后，内存被解锁，而所有休眠在索引节点的读取进程会被唤醒。

管道的读取过程和写入过程类似。但是，进程可以在没有数据或内存被锁定时立即返回错误信息，而不是阻塞该进程，这依赖于文件或管道的打开模式。反之，进程可以休眠在索引节点的等待队列中等待写入进程写入数据。当所有的进程完成了管道操作之后，管道的索引节点被丢弃，而共享数据页也被释放。

因为管道的实现涉及很多文件的操作,因此,当读者学完有关文件系统的内容后来读pipe.c中的代码，你会觉得并不难理解。

测试：【环境：Linux hgc-VirtualBox 3.5.0-26-generic #42~precise1-Ubuntu SMP Mon Mar 11 22:19:42 UTC 2013 i686 i686 i386 GNU/Linux】

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>


int main(int argc, char* argv[])
{
    int pipefds[2]; //[0] for read, [1] for write
    pipe(pipefds);

    char buf[4096];
    for (int i = 0; i < sizeof(buf); ++i)
    {
        buf[i] = 0x7f;
    }

    ssize_t ret = -1;

    int loop = 100;
    if (argc > 1)
    {
        loop = atoi(argv[1]);
    }

    for (int i = 0; i < loop; ++i)
```

```
    {
        printf("loop: %d\n", i);
        ret = write(pipefds[1], buf, sizeof(buf));
        if (ret < 0)
        {
            perror(NULL);
        }
        else
        {
            printf("%d\n", ret);
        }
    } // 当i=16的时候会阻塞，可知管道大小为64k

    close(pipefds[0]);
    close(pipefds[1]);

    return 0;
}
```

获取Linux 内存页（基页）大小的命令：getconf PAGE_SIZE ， 一般的输出是4096，即4KB。