<< Prev | TOC | Front Page | Talkback | FAQ

LINUX GAZETTE ...making Linux just a little more fun!

Exploring The sendfile System Call

By Jeff Tranter

Introduction

The `sendfile` system call is a relatively recent addition to the Linux kernel that offers significant performance benefits to applications such as ftp and web servers that need to efficiently transfer files. In this article I will explore `sendfile`, what it does, and how to use it, illustrated by some example programs.

Background

A server application, such as a web server, spends much of its time transferring files stored on disk to a network connection connected to a client running a web browser. Simple pseudo-code for the data transfer might look like this:

```
open source (disk file)
open destination (network connection)
while there is data to be transferred:
    read data from source to a buffer
    write data from buffer to destination
close source and destination
```

The reading and writing of data would typically use the `read` and `write` system calls respectively, or library functions built on top of them.

If we follow the path of the data from disk to network, it needs to be copied several times. Each time the `read` system call is invoked, data must be transferred from the disk hardware to a kernel buffer (typically using DMA). Then it needs to be copied into the buffer used by the application. When `write` is called, data in the application's buffer needs to be transferred to a kernel buffer and then from the kernel buffer to the hardware device (e.g. network card). Every time a system call is invoked by a user program, there is a *context switch* between user and kernel mode, which is a relatively expensive operation. If there are many calls to `read` and `write` in the program, there will be many context switches required.

This copying of data between kernel and application buffers and back is redundant if the data does not need to be changed. Many operating systems, including Windows NT, FreeBSD, and Solaris, offer what is called a zero-copy system call that can perform a file transfer in a single operation. Early versions of Linux were criticized for lacking this feature, until it was implemented in the 2.2 kernel series. It is now used by popular server applications such as Apache and Samba.

The implementation of `sendfile` varies on different operating systems. For the rest of this article we will just focus on the Linux version. Note that there is a file transfer utility called `sendfile`; this has nothing to do with the kernel system call.

A Detailed Look

To use `sendfile`, include the header file `<sys/sendfile.h>`, which declares a function with the following prototype:

```
ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

The parameters are as follows:

out_fd
: a file descriptor, open for writing, for the data to be written

in_fd
: a file descriptor, open for reading, for the data to be read

offset
: the offset in the input file to start transfer (e.g. a value of 0 indicates the beginning of the file). This is passed into the function and updated when the function returns.

count
: the number of bytes to be transferred

The function returns the number of bytes written or -1 if an error occurred.

On Linux, file descriptors can be true files or devices, such as a network socket. The `sendfile` implementation currently requires that the input file descriptor correspond to a true file or some device which supports `mmap`. This means, for example, it cannot be a network socket. The output file descriptor can correspond to a socket, and this is usually the case when it is used.

Example 1

Let's look at a simple example to illustrate using `sendfile`. Listing 1 shows `fastcp.c`, a simple file copy program that uses `sendfile` to perform a file copy.

The listing here is slightly abbreviated for clarity. The full listing available here has additional error checking and the include directives needed so it will compile.

```
Listing 1: fastcp.c

1     int main(int argc, char **argv) {
2         int src;               /* file descriptor for source file */
3         int dest;              /* file descriptor for destination file */
4         struct stat stat_buf;  /* hold information about input file */
5         off_t offset = 0;      /* byte offset used by sendfile */
6
7         /* check that source file exists and can be opened */
8         src = open(argv[1], O_RDONLY);

9         /* get size and permissions of the source file */
10        fstat(src, &stat_buf);

11        /* open destination file */
12        dest = open(argv[2], O_WRONLY|O_CREAT, stat_buf.st_mode);

13        /* copy file using sendfile */
14        sendfile (dest, src, &offset, stat_buf.st_size);

15        /* clean up and exit */
16        close(dest);
17        close(src);
18    }
```

On line 8 we open the input file, passed as the first command line argument. On line 10 we get information on the file using `fstat`, as we will need the file size and permissions later. On line 12 we open the output for for writing. Line 14 performs the call to sendfile, passing the output and input file descriptors, the offset (zero in this case), and specifying the number of bytes to transfer using the input file size. We then close the files in lines 16 and 17.

Try compiling the program (using the full version here). I suggest experimenting with using it to copy various types of files, such as the following, and see which source and destination devices support `sendfile`:

- from a disk file to another disk file
- using files located on different disks or partitions
- from a mounted CD-ROM to a file
- from a disk file to /dev/null or /dev/full
- from /dev/zero or /dev/null to a disk file
- from a disk file to the floppy device (/dev/fd0)

Example 2

The first example was simple, but not very representative of the typical use of `sendfile` using a network destination. The second example illustrates sending a file over a network socket. This program is longer, mostly due to the setup required to work with sockets, so I don't include it in-line. You can see the full source listing here.

The program, called `server`, does the following:

- Listens on a network socket for a client to connect.
- When a client connects, waits for the client to send it a filename.
- Sends the specified file back to the client using `sendfile`.
- Disconnects the client and listens for another connection.

I assume here you are familiar with the basics of network socket programming. If not, there are many good books on the subject. such as *UNIX Network Programming* by Richard Stevens.

The server arbitrarily uses port 1234 but you can specify it as a command line option. Start the server by running it ("./server"). To act as the client side, you can use the `telnet` program. Run it from another console window while the server is running, specifying the host name and port number (e.g. "telnet localhost 1234"). Once `telnet` indicates it is connected, type the name of a file that exists, such as `/etc/hosts`. The server should send the contents of the file back to the client and then close the connection.

The server should remain running so you can connect again. If you use a filename of "quit" then the server will exit. If you have another machine on a network, try verifying that you can connect to the server and transfer a file from another machine.

Note that this is a very simplistic example of a server: it can only handle one client at a time and does does little error checking, exiting if an error occurs. There are also other performance optimizations that can be done at the TCP layer, that are outside the scope of what can be covered here.

Summary

The `sendfile` system call facilitates high performance network file transfers, a requirement for applications such as ftp and web servers. If you are developing a server application, consider using `sendfile` to give your code a performance boost. Outside of the server arena, it is an interesting feature in it's own right and you may find some other creative uses for it.

Finally, after all this discussion of `sendfile`, I will leave you with this question to ponder: why is there no corresponding `receivefile` system call?

References

1. The sendfile(2) man page.
2. Kernel source for the `sendfile` implementation.

Jeff has been using, writing about, and contributing to Linux since 1992. He works for Xandros Corporation in Ottawa, Canada.

<< Prev | TOC | Front Page | Talkback | FAQ