© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR

ISSN: 2278-6848 | Volume: 06 Issue: 03 | July-September 2015

DATA STREAMING USING KERNEL SOCKET MODULE TO OVERCOME LIMITATIONS OF FTP

[1]Pankaj Kumar, [2]Ms. Manisha, Department Of Computer Science & Engineering , RN College Of Engineering & Technology

Abstract: *A socket serves as a communication end point between two processes. One process is known as server and other as client. Server programs create sockets, bind to well-known ports, listen and accept connections from clients. Servers are usually designed to accept multiple connections from clients—they either fork a new process to serve each client request (concurrent servers) or completely serve one request before accepting more connections (iterative servers). Client programs, on the other hand, create sockets to connect to servers and exchange information.*

Keywords: FTP, Port, Socket, Server, Client, Linux, HTTP, TCP, SMTP

[I] Introduction To File Transfer Protocol

Most of the file transfer protocols between two Linux systems are implemented using network data transfer. These all programs are written in user mode and use the system calls provided by the Linux kernel to perform various operations like network read and write. Although this is the traditional method of writing programs, there are performance issues. In specific research and high-performance computing environments, there is a need for achieving data transfers at great speed without any delay. This can be achieved if we implement the file transfer protocol in kernel space instead of user space. I took the FTP protocol for this case and implemented the FTP protocol in kernel space. There are some difficulties to do kernel programming and need to take extreme care for same. FTP transfers involve two TCP connections. The first **control** connection goes from the FTP client to port 21 on the FTP server. This connection is used for logon and to send commands and responses between the endpoints. A data transfer (including the output of "ls" and"dir" commands) requires a second data connection. The **data** connection is dependent on the **mode** that the client is operating.

The File Transfer Protocol (FTP) was one of the first efforts to create a standard means of exchanging files over a TCP/IP network, so the FTP has been around since the 1970's. The FTP was designed with as much flexibility as possible, so it could be used over networks other than TCP/IP, as well as being engineered to have the capability with exchanging files with a broad variety of machines.

The base specification is RFC 959 and is dated October 1985. There are some additional RFCs relating to FTP, but it should be noted that even as of this writing (December 2001) that most of the new additions are not in widespread use. The purpose of this document is to provide general information about how the protocol works without getting into too many technical details. RFC 959 should be consulted for details on the protocol.

[II] Limitation Of Ftp

The disadvantage of anonymous FTP is that you have little control over who accesses your FTP server or how often they do it. If you have particularly popular file downloads, it can place quite a load on the server. For this reason, many organization with limited resources have chosen alternatives like BitTorrent to distribute large files. The scope of this work is to achieve desired speed and highly secured file transfer system so that the following disadvantages of FTP can be eradicated.

It doesn't have a secure protocol.

- It has no username protection for its users.
- There is no packet sniffing for this application.

[III] Objective Of Research

Sending and receiving large files internally and externally has become an integral part of a company's communication system. Companies employ different file sharing solutions for exchanging data and facilitating collaboration between customers, clients, employees and partners. Most prevalent file sharing solutions include email services, FTP, web-based services and in many cases, mailing documents via overnight shipping services. Each of these solutions comes with their own set of advantages and disadvantages - some drawbacks may lead to security risk and limitations, leading to a less successful handshake between the company and its client. The aim of the this work is enhance the efficiency of normal file transfer protocol implementation so that it can be used to achieve maximum speed and provide a kernel based security.

The purpose of work is to design a kernel module equivalent to the features of normal ftp with the added advantage of being fast and secure.

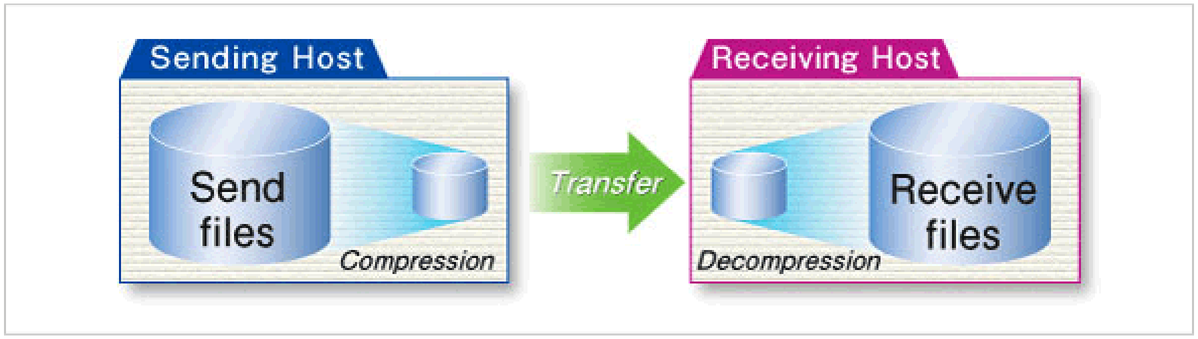

Fig1. File Sending and receiving Host

[IV] User Mode And Kernel Mode

CPU usage is generally represented as a simple percentage of CPU time spent on non-idle tasks. But this is a bit of a simplification. In any modern operating system, the CPU is actually spending time in two very distinct modes:

Kernel Mode

In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire PC. In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire PC.

User Mode

In User mode, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.

It's possible to enable display of Kernel time in Task Manager, as I have in the above screenshot. The green line is total CPU time; the red line is Kernel time. The gap between the two is User time.

These two modes aren't mere labels; they're enforced by the CPU hardware. If code executing in User mode attempts to do something outside its purview-- like, say, accessing a privileged CPU instruction or modifying memory that it has no access to -- a trappable exception is thrown. Instead of your entire system crashing, only that particular application crashes. That's the value of User mode.

x86 CPU hardware actually provides four protection rings: 0, 1, 2, and 3. Only rings 0 (Kernel) and 3 (User) are typically used. In User mode, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.

[V] DATA TRANSFER USING SOCKET IN LINUX

Except few data transfer protocols all protocols implementation's in Linux are done in user space which uses the system calls provided by Linux kernel to perform various operations like read and write on network port. But this has limitation in regard of performance. So in this implementation, Client and server programs are implemented in Linux kernel mode.

To execute user programs in kernel mode, Kernel Mode Linux has a special start_thread (start_kernel_thread) routine, which is called in processing execve(2) and sets registers of a user process to specified initial values. The original start_thread routine sets CS segment register to __USER_CS. The start_kernel_thread routine sets the CS register to __KERNEL_CS. Thus, a user program is started as a user process executed in kernel mode.

The biggest problem of implementing Kernel Mode Linux is a stack starvation problem. Let's assume that a user program is executed in kernel mode and it causes a page fault on its user stack. To generate a page fault exception, an IA-32 CPU tries to push several registers (EIP, CS, and so on) to the same user stack because the program is executed in kernel mode and the IA-32 CPU doesn't switch its stack to a kernel stack. Therefore, the IA-32 CPU cannot push the registers and generate a double fault exception and fail again. Finally, the IA-32 CPU gives up and reset itself. This is the stack starvation problem.

[VI] Results And Discussions

We have examined the performance of Linux under various load conditions and havealso analyzed how it reacts to changes in load. According to our experiments we have obtained the following results.

• Under moderate loads a 10Mbps ethernet poses a performance bottleneck.

• For the case of transferring different files we find that the 8MB RAM becomes a bigger bottleneck than the ethernet with just 4 connections. The rate of transfer reduces by less than 30% when number of connections are doubled.

This means that Linux scales quite well with load.

• For the case of transferring same files we find that the memory is sufficient for around 16 connections. Memory would pose a bottleneck for more number of connections.

• Transferring different files reduces the performance but the fall in performance reduces as the files become very large.

• In the loop back experiment we discovered that there was a very large fall in transfer rate from 2 to 4 connections when transferring different files.

From these observations we can conclude that Linux does scale quite well, and will be able to handle heavy loads. This implementation of FTP protocol is in kernel space but only basic command of FTP protocol is implemented. It can be used for high-performance content serving and are well suited for environments that demand data transfer at high rate. Scientific data sets are growing at exponential rates, and new data movement protocols and system interfaces are needed to keep up. TCP and UDP using traditional UNIX sockets use too much CPU to be able to scale to the data rates needed for tomorrows scientific workflows. Our experience on cluster test bed have shown that the kernel mode FTP implementation has a better transfer rate .A graph showing the quantitative comparison of the normal FTP and kernel mode FTP is depicted in the following diagram.

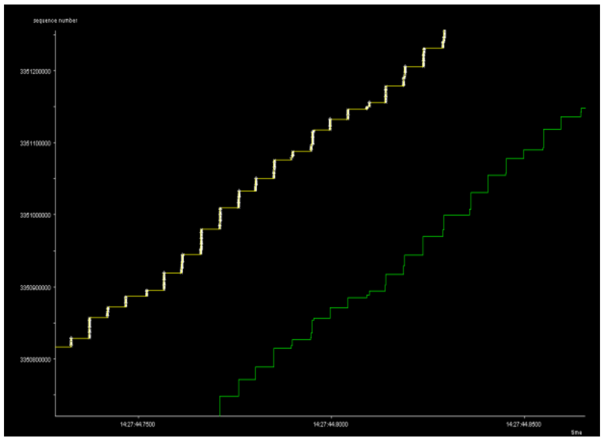

Fig 2: The white line shows the efficiency of kernel mode FTP

[VII] Future Work

a. Implement the other FTP commands in FTP server and FTP client.

b. Measure the performance after comparison of user space FTP and our program.

c. Run the program in very high performance environment

d. Run the program in the environments that demand very high data transfer.

Reference

1. For FTP RFC

http://tools.ietf.org/html/rfc959

2. Wiki for FTP

http://en.wikipedia.org/wiki/File_Transfer_Protocol#cite_note-for-1

3. Wiki of full FTP Client command http://en.wikipedia.org/wiki/List_of_FTP_commands

4. Complete List of Server return code. http://en.wikipedia.org/wiki/List_of_FTP_server_return_codes

5. Network programming book. Unix Network Programming, Volume 1 by W. Richard Stevens, Bill Fenner, Andrew M. Rudoff

6. To download the Linux Kernel Source code. https://www.kernel.org/

7. For network internal of Linux. Understanding Linux Network Internals by Christian Benvenuti.

8. [Bac86] Maurice J. Bach. The Design of the UNIX Operating System. Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1986.

9. [PR85] J. Postel and J. Reynolds. File transfer protocol (ftp). Technical Report RFC-959, Network Working Group, 1985.

10.[Ste92] Richard Stevens. Advanced Programming in the UNIX Environment. Addison-Wesley,Reading, MA, USA, 1992.

11. [TCT] Theodore Ts´o, Remy Card, and Stephen Tweedie. Design and implementation of the second extended filesystem. In Proceedings of the First Dutch International Symposium on Linux.

12.[WS91] Larry Wall and Randal L. Schwartz. Programming Perl. Nutshell Handbooks. O'Reilly and Associates, Inc., 632 Petuluma Avenue, Sebastopol,