International Journal For Technological Research In Engineering
Volume 2, Issue 9, May-2015 *ISSN (Online): 2347 - 4718*

LINUX KERNEL

Sanjana Singh[1], Surbhi Bhardwaj[2], Shivansh Mudgil[3]
Computer Science Department, Dronacharya College of Engineering
Khentawas, Farukhnagar, Gurgaon

Abstract: This paper gives a brief overview of the author's research into network protocols and then focuses on the process of development for the Linux kernel. This topic is approached both from a technical view point and also the interactions with the open source community. The aim of the paper is for readers not to have the same difficulties the author experienced!
Keywords: Network Research, Operating Systems

I. INTRODUCTION

How does a distributed group of individuals, who are dispersed across space, time, and organizational boundaries, organize themselves to create a useful product? The advent of the Internet and web-based technologies has enabled specialized communities to convene, interact, and share resources extensively via electronic interfaces. One prominent example is the Open Source community, which shares the source code of the software that its members have developed collectively with anyone who wishes to download via the Internet free-of-charge. The Linux operating system development project is among the first attempts to make a deliberate effort to use the globally connected software developers as its main source of talent and input to create a widely shared and used software product. As a natural experiment, the Linux project has demonstrated the feasibility of a large-scale on-line collaboration effort where developers and users can be one and the same - though over time, the proportion of "non-developer users" grows more rapidly than that of "developer users." Linux type of phenomena are increasingly popular beyond the Open Source community as the use of the Internet increases. For example, the Open Directory Project provides the technologies for Internet users to develop an alternative directory service web site to commercial portals, such as Yahoo!. Internet users can volunteer as editors by choosing an area of expertise and collecting high quality Internet links for that category. More than 20,000 Open Directory editors have identified links to more than 1,200,000 web sites, with more than 3,000 new sites added daily. 2 Many Internet search services such as HotBot, Lycos, and Netscape Communications have recognized the Open Directory as a source of quality information.

II. NETWORK PROTOCOLS

The author is studying the interaction of real time media applications with network protocols. In this research the existing protocols TCP and UDP have been reviewed and compared to a newer protocol DCCP. A reliable protocol such as TCP gives overhead because any dropped frames must be retransmitted. With multimedia applications dropped frames are not as important as in many cases timeliness is more important than having a perfect picture and sound. TCP implements congestion control] which reduces the risk of congestion collapse when there is more traffic than bottlenecks in the path. The preference therefore has been to use a protocol such as UDP which gives unreliable delivery meaning it is not guaranteed that packets will reach the other end and they will not be retransmitted. UDP however does not establish a session and, as such, every packet is sent individually and without congestion control. Recent research has shown that most streaming multimedia traffic uses TCP instead of UDP. This is because the lack of a session causes UDP difficulty in traversing NAT devices such as home routers or company firewalls. Datagram Congestion Control Protocol (DCCP) is a newer protocol which aims to address the issues of TCP and UDP for many applications. DCCP is a session based, unreliable protocol with congestion control. Data is transmitted in data grams (packets) and delivery is not guaranteed. DCCP has Congestion Control IDs (CCIDs) which specify different congestion control mechanisms which can be specified by the application using DCCP and are designed so that DCCPcan be extended in the future. The author has worked with Arnaldo Carvalho de Melo to have an implementation of DCCP merged into the Linux kernel. The Linux kernel implementation was based on the existing Linux TCP code for CCID2, and the WAND group from University of Waikato's code for CCID3. The overall framework was based on new code, existing Linux code and the WAND group. The WAND code was based on code from the University of Lulea for CCID3 and code from Patrick McManus for CCID2. The Lulea code was relicensed under the GPL to ensure that it could be merged into the Linux kernel.

III. X-LOADER AND U-BOOT

X-loader is a small first stage boot loader derived from the u-boot base code to be loaded into the internal static ram. Because the internal static ram is very small (64k-32k), x-loader is stripped down to the essentials and is used to initialize memory and enough of the peripheral devices to access and load the second stage loader (U-Boot) into main memory. U-Boot is designed to be the "Universal Boot loader" and certainly goes a long ways to support multiple processors, boards, and OS's. It's primary purpose is to setup the kernel environment; and load and boot the kernel. The sources of u-boot and x-loader specific to the ARM board can be obtained and built using make utility on terminal. The procedure for building it is same as it is for linux kernel. The output of the build process is u-boot.bin and MLO.

Building X-loader
In order to compile the X-loader, you need to set the CROSS_COMPILE environment variable and specify the pathto tool chain.
#export
PATH=/usr/local/xtools/arm--linux-uclibcgnueabi/bi
n:$PATH
#exportCROSS_COMPILE=arm-none-linux-
gnueabi-
To set the configuration for the target board and build the image, give the command:
□□ # make BOARDNAME_config
□ □ # make
The resulting file is stored in the x-loader main directory asx--load.bin. This file must be "signed" in order to be executed by the processor. Using the sign GP tool in the tools/subdirectory of the main directory of the lab, sign thex-load.bin file. This produces an x-load.bin.ift file. You can copy it to the MMC card, renaming it as MLO.

Building U-boot
U-Boot is a typical free software project. It is freely available at http://www.denx.de/wiki/U-Boot.Get the source code from the website, and uncompress it. The include/configs/ directory contains one configuration file for each supported board. It defines the CPU type, the peripherals and their configuration, the memory mapping, the U-Boot features that should be compiled in, etc. Assuming that your board is already supported by U-Boot, there should be one file corresponding to your board, for example include/configs/omap2420h4.h.U-Boot must be configured before being compiled.

#make BOARDNAME_config
Make sure that the cross-compiler is available in PATH
#export
PATH=/usr/local/uclibc-0.9.29-2/arm/bin/:$PATH
Compile U-Boot, by specifying the cross-compiler prefix.
Example, if your cross-compiler executable is arm-linux-gcc:
□ □ #make CROSS_COMPILE=arm-linux
This command would result in u-boot.bin file in working directory

IV. BUILDING A KERNEL

A. Maintaining a source code tree

Keeping in synchronization with the Linux kernel source code tree takes a nontrivial amount of time. It is necessary to keep synchronized if it is intended to have the code merged into the tree or released as an ongoing codebase. The Linux source code is maintained in a git tree. Git is a new source code management tool created by Linus Torvalds. Git stores every change as apatch addressed by a SHA1 hash. Developers can make copies of git trees and because each patch is unique it is relatively simple for developers to synchronise their code base with other developers. For developing in the kernel it is recommended to make a copy of LinusTorvalds' source code tree by issuing a statement similar to: git-clone \
git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git \
~/linuxsrc/linus and then using git-pull to keep the git tree up to date. If the maintainer forthe area that is being developed keeps a separate git tree, as most do, then youwill need to create a copy of it by issuing a statement similar to:
git-clone --reference ~/linuxsrc/linus \
git://git.kernel.org/pub/scm/linux/kernel/git/davem/net-2.6.git \ ~/linuxsrc/davem
When the above syntax is used it uses the local copy where possible making synchronization quicker and uses a fraction of the bandwidth. It is often useful to use a patch management tool on top of git such as stgitor quilt [11]. These tools allow a series of patches to be maintained as a"stack" of patches which can be applied against different or updated trees. Often a maintainer will "rebase" their tree which means they delete their original tree, make a fresh clone of Linus' tree and then they will reapply any outstanding patches to that tree. If a maintainer rebases their tree then it is usually not possible to update it using the git-pull or stg pull command. Presuming the use of stgit, issue the command stg export to export the patches(this should also be done before any pull in case of a problem), save the files that are under patches-branch in a temporary directory, clone the git tree again to the latest tree and then import the patches into the new tree by stg import.

B. Use of distcc

To speed up compilation if multiple machines are available then distcc [12] canbe used which is a distributed C compiler, and then specifying to make how many parallel threads to run using the -j parameter. For kernel development there are a couple of caveats to be aware of. The same version of the C compiler gcc must be installed on the machines that are being used as a compile pool. It is also important to specify only one target on the command line or else the kernel will continue to rebuild from scratch each time the make command is issued. For example to build a kernel and prepare for it to be installed the following commands could be issued:
make -j6 CC=distcc all
make modules_install INSTALL_MOD_PATH=~/tmp

C. Other resources

For an introduction to Linux development there are excellent resources available such as [13] and [14]. The use of simple tools should also not be underestimated such as the use of grep -n -r phrase to find a symbol in the kernel. The author also maintains a Wiki page at http://wlug.org.nz/KernelDevelopment which contains other tips. There are also an abundance of resources available for development in the Linux kernel on the Internet as well which can be used to assist.

V. TESTING AND DEBUGGING

It is important that code is well tested before it is released. There are a number of virtualization tools available to assist with testing software such as qemul, UML, VMWare and

Xen . These tool scan make testing easier if computers with sufficient processing power are used. Any software developed should also be tested on computers natively as well as this can uncover separate bugs. If at all possible code should be tested on a variety of machines to uncover more subtle bugs such as endian or processor issues. There are debugging tools available that can be used on the kernel such as gdb and a number of derivatives of gdb. The author has found that these (orany debugger) have issues with any timing dependent code such as networking as they cause expiry timers and round trip time (rtt) to become invalid. The simplest debugging tool to use for timing dependent code is the print k statement in the code which acts like a printf statement, except the output goes to the kernel logging daemon. If it is desired to find which patch caused a bug then git-bisect can be used to "divide and conquer" the code base. In the kernel there is also a mechanism for tracing code execution and capturing data called k probes which allows hooks to be added into any function entry or exit dynamically without any changes being made to the original function. In our research we have taken an implementation of this for monitoring TCP and implemented it for DCCP. With testing it is often necessary to transfer newly built kernels to multiple machines. This can be automated through a script such as:

```
#! /bin/bash
# syntax m machine_name directory version
H=$HOME
SRC=$H/linuxsrc/$2
VER=$3
rm $H/tmp/lib/modules/$VER/build
rm $H/tmp/lib/modules/$VER/source
rsync $SRC/System.map root@$1:/boot/System.map-$VER
rsync $SRC/arch/i386/boot/bzImage root@$1:/boot/vmlinuz-$VER
rsync $SRC/vmlinux root@$1:/boot/vmlinux-$VER
rsync -av $H/tmp/lib/modules/$VER root@$1:/lib/modules
```

The line for copying vmlinux is only needed if a profiler such as oprofile or another tool needs it.

VI. CONCLUSION

Development in the Linux kernel is more than simply editing code and typing make all. It is the hope of the author that this paper helps more people develop in the Linux kernel by taking into account other considerations and applying these.

REFERENCES

[1] McDonald, I.: Phd research proposal: Congestion control for real time media applications (2005)
[2] Jacobson, V.: Congestion avoidance and control. In: ACM SIGCOMM '88, Stanford, CA (August 1988) 314–329
[3] Guo, L., Tan, E., Chen, S., Xiao, Z., Spatscheck, O., Zhang, X.: Delving into internet streaming media delivery: a quality and resource utilization perspective. In: IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement New York, NY, USA, ACM Press (2006) 217–230
[4] Kohler, E., Handley, M., Floyd, S.: Designing DCCP: Congestion Control Without Reliability. Submitted to ICNP (2003)
[5] WAND: Wand implementation of dccp (Accessed 2006)
[6] Lulea: Dccp projects (Accessed 2005)
[7] McManus, P.: Dccp implementation by patrickmcmanus (Accessed 2006)
[8] McDonald, I., Nelson, R.: Congestion control advancements in Linux
[9] Web: git. http://git.or.cz/ (Accessed 2006)
[10] Web: Stacked git. http://www.procode.org/stgit/ (Accessed 2006)
[11] Web: Quilt patch management tools. http://savannah.nongnu.org/projects/quilt/(Accessed 2006)
[12] Pool, M., et al.: Distcc: a fast, free distributed c/c++ compiler, 2002–. URLhttp://distcc. samba. org
[13] Love, R.: Linux Kernel Development. Second edn. Novell Press (2005)
[14] Web: Lxr. http://lxr.linux.no/ (Accessed 2006)
[15] Raymond, E.: Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. (2001)
[16] Web: Kernel.org mailing lists. http://vger.kernel.org/vger-lists.html (Accessed 2006)
[17] Web: kernel.org git trees. http://kernel.org/git/ (Accessed 2006)
[18] QEMU, C.: Emulator. URL: http://fabrice. bellard. free. fr/qemu
[19] Dike, J.: A user-mode port of the Linux kernel. Proceedings of the Annual LinuxShowcase. Atlanta, GA, Oct (2000)
[20] VMware, I.: The VMWare software package. See http://www. vmware. com
[21] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer,R., Pratt, I., Warfield, A.: Xen and the art of virtualization. Proceedings of the nineteenth ACM symposium on Operating systems principles (2003) 164–177
[22] Stallman, R., Pesch, R.: The GDB Manual: The GNU Source-level Debugger. Free Software Foundation (1992)
[23] Hemminger, S.: Tcp probe. http://linux-net.osdl.org/index.php/TcpProbe (Accessed 2006)
[24] Levon, J.: Oprofile-a system profiler for linux. Web site: http://oprofile. sourceforge. net (2005)
[25] Hemminger, S.: Netem emulating real networks in the lab. Proc. Linux Conference Australia (2005)
[26] Tirumala, A., Qin, F., Dugan, J., Ferguson, J., Gibbs, K.: Iperf-The TCP/UD bandwidth measurement tool. Available: http://dast. nlanr. net/Projects/Iperf

