Linux Kernel development

Ian McDonald

University of Waikato
ian.mcdonald@jandi.co.nz

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 viewpoint 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

1 Introduction

This paper begins by discussing network protocols and introduces the DCCP protocol.

It then discusses the social aspects of kernel development, followed by tips for building a kernel. The paper then focusses on releasing code and lastly, but not least important, testing and debugging.

The paper highlights some of the lessons learned by the author and the methodology used in developing within the kernel.

2 Network protocols

The author is studying [1] 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 so important as in many cases timeliness is more important than having a perfect picture and sound. TCP implements congestion control [2] 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 [3] 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) [4] 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 datagrams (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 DCCP can be extended in the future.

The author has worked with Arnaldo Carvalho de Melo to have an implemention 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 [5] 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 [6] for CCID3 and code from Patrick McManus [7] for CCID2. The Lulea code was relicensed under the GPL to ensure that it could be merged into the Linux kernel.

Further details on Linux congestion control can be found at [8].

3 Social aspects

One aspect of kernel development which can be quite offputting to new kernel developers is the "flaming" that occurs. Linux kernel development is largely a meritocracy and whether code is accepted or not, depends largely on the strength of the code and the arguments supporting it. This can be intimidating initially as the other interested parties reviewing proposed changes will often express themselves extremely bluntly, including the use of profanities and can be quite dismissive. Once it is understood that this is the sub-culture of Linux development and not a personal attack then it can be tolerated. A positive side of this though is that if the developers ideas are correct then they will be accepted over a large corporation with incorrect code. The author of this paper has several times had code changed over the preference of firms that are in the top five computing firms in the world.

It is often useful to participate in forums that other kernel developers frequent such as IRC chat rooms, conferences or personal e-mail. Establishing a positive rapport with other developers can make a crucial difference to code being accepted or assistance being given. It is useful to give back to the community in the same manner once some experience has been gained.

4 Building a kernel

4.1 Maintaining a source code tree

Keeping in synchronisation with the Linux kernel source code tree takes a non-trivial amount of time. It is necessary to keep synchronised 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 [9] tree. Git is a new source code management tool created by Linus Torvalds. Git stores every change as a patch 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 Linus Torvalds' 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 for the area that is being developed keeps a separate git tree, as most do, then you will 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 synchronisation quicker and uses a fraction of the bandwidth.

It is often useful to use a patch management tool on top of git such as stgit [10] or 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`.

4.2 Use of distcc

To speed up compilation if multiple machines are available then distcc [12] can be 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
```

4.3 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.

5 Releasing code

An often cited mantra in the open source community is "release early and release often" [15]. This concept is useful even prior to code release.

It can productive to talk about what you are planning to implement. This can be useful as other people can inform you if they are working on similar code as is often the case and was with DCCP. It also gives people an opportunity to give feedback on whether the ideas are good or need refining.

Once the ideas have been crystallised it is often useful to release a code as a request for comment (RFC) if the code is not ready to be merged. This, again, allows discussion of the ideas being implemented.

The code will probably go through many, many iterations so if the code is rejected then it is important to work on the areas highlighted rather than becoming disheartened.

When developing code for the kernel it needs to be submitted to the maintainer of the subsystem. The `MAINTAINERS` file in the top level directory of the Linux source code tree contains the list of the maintainers. A posting to the linux-kernel list or other kernel mailing lists [16] will usually produce a respone on how the code is maintained if it is not clear.

Any code released should always be against the latest tree of the maintainer [17]. One of the biggest reasons that code isn't accepted is that the maintainer cannot use it as it applies against an older version of Linux.

When code is submitted it must be able to be applied without problems by the maintainer. Prior to submitting test each patch with:

```
git-apply --check --whitespace=error < mypatch.diff
```

It is also required to choose a mail client to send the patch which does not alter whitespace such as spaces being substituted for tabs, breaking lines up or sending as HTML. The author uses KMail as he has found this is one of the few that will meet these criteria.

The code and patches should also fit into the existing kernel code style. The `Documentation` directory of the Linux source code contains many useful guides to assist with this.

It is recommended for a developer to submit kernel changes in multiple small patches rather than one large patch. This is that so the change can be reviewed in a series of steps which are simpler to read and find bugs in. If multiple patches are provided then part of the change can be accepted, rather than all rejected if one large patch is used.

6 Testing and debugging

It is important that code is well tested before it is released.

There are a number of virtualisation tools available to assist with testing software such as qemu [18], UML [19], VMWare [20] and Xen [21]. These tools can 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[22] and a number of derivatives of gdb. The author has found that these (or any 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 `printk` 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 kprobes 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[23] 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:

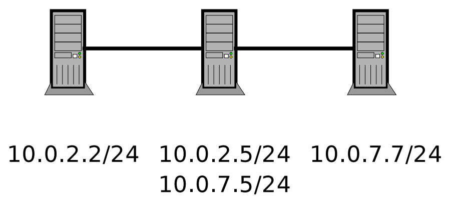
```
#! /bin/bash
# syntax m machine_name directory version
H=$HOME
SRC=$H/linuxsrc/$2
VER=$3
rm $H/tmp/lib/modules/$VER/build
rm $H/tmp/lib/modules/$VER/source
rsync $SRC/System.map root@$1:/boot/System.map-$VER
rsync $SRC/arch/i386/boot/bzImage root@$1:/boot/vmlinuz-$VER
rsync $SRC/vmlinux root@$1:/boot/vmlinux-$VER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

6.1 Network testing

Netem [25] is an open source project that has been used by the author for network testing. Netem allows operations such as loss, reordering, duplication and delay on outbound queues. As Netem works on outbound queues, and also to minimise interaction with other code, it is most usefully placed on a middle box such as shown in figure 1.

Fig. 1. Netem setup



When carrying out network testing it is useful to have a minimal number of tasks running on the computer. It is highly recommend to be running at a text console and shut down the graphical window manager (typically gdm or kdm). Any other services/daemons that are also not needed for the testing should also be shut down.

For performance testing of network code the author has used modified versions of ttcp and iperf [26]. The code has been modified to support TCP congestion control algorithm support and DCCP.

7 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 patrick mcmanus (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–. URL http://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 Linux Showcase. 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/UDP bandwidth measurement tool. Available: http://dast. nlanr. net/Projects/Iperf