Inessential kernel ______

John Hawkinson

Student Information Processing Board
Massachusetts Institute of Technology
84 Massachusetts Avenue
Cambridge, Massachusetts 02139

ABSTRACT

This document provides a general guide to common operations you might want to perform on kernels. This is targeted at the platforms I use, including SunOS, IRIX, NetBSD, and Solaris, with a smattering of IOS. "Common operations" include patching variables, patching code, and debugging crashes.

Todo-type notes. This document probably needs to finish being written before it gets too much commentary. I've incorporated my responses to aurora & katyking's comments scribbled on the 3 October edition. I've ignored some of them in the interest of maintaing conversational jhawk-style documentation, but implemented some others.

1 December 1996

Inessential kernel ______

John Hawkinson

Student Information Processing Board
Massachusetts Institute of Technology
84 Massachusetts Avenue
Cambridge, Massachusetts 02139

1. Introduction

The kernel is the inner core of the UNIX® operating system. The kernel is responsible for loading executable programs, maintaing processes, context-switching between them, and providing services to those programs through interfaces such as *system*calls and **traps**.

Because of these things, if you experience a fault or crash while the processor is executing in the kernel (often a "kernel panic"), the machine will require a reboot. Contast this with a user-level program which will exit (perhaps core dumping) and may be re-run without major disruption to the entire operating system.

Kernels come in different shapes and sizes. Under classic BSD-derived systems like SunOS and NetBSD, the kernel exists as one file in the root directory of the filesystem -- */vmunix /netbsd*, **respectively. Under Solaris, the kernel consists of a large number of dynamically loaded[1] files.** ***/kernel/unix*** **is the principal part of the kernel, and it loads various different modules from specified locations, including subdirectories of** ***/kernel*** **and** ***/usr/kernel*** **IRIX has its own way of doing things that I'm not 100% clear on, so that section is a bit more to-be-written than the rest (but there's something in** ***/unix*****).**

So why is this document named *Inessential kernel* ______ instead of some more sensible name? Well, I couldn't decide what the ______ should represent[2], so it has been left up to your imagination. When the document title is pronounced, the ______ should be accompanied by expansive hand-waving motions.

2. Examining your kernel

Before you start too far down the path of kernel hacking, you should take a moment to acquaint yourself with your surroundings. How large is your kernel? (Kernels typically range from 1 to 3 megabytes in size[3]) Do you have space for more than one copy of the kernel on your root partition (typically the part of the operating system responsible for loading the kernel, usually the "loader", can only load the kernel from the root partition of the system)?

What version of your kernel is installed? Sometimes this is relatively easy to figure out, sometimes it is not. When the operating system boots this informaton is usually displayed; often it is logged by **syslogd(8)**, or is available through a command that displays the kernel message buffer, such as **dmesg(8)**. It is also often useful to know who built your kernel. While you can't always run crying to them for help, chances are they are less than totally clueless about goings on.

Under NetBSD, you can obtain the version of the running kernel and information about by whom and where the kernel was built with the **sysctl(8)** command:

---

[1] NetBSD and SunOS have support for dynamically loadable kernel modules, however they are not an integral part of the operating system, and typically require a user-level program to actually load the modules.

[2] Possible suggestions included "crap", "crud", and "stuff". None of these were particularly appealing.

[3] Kernels are typically built without debugging symbols. An unstripped kernel built with debugging symbols may be significantly larger, perhaps 13 megabytes in size.

```
[planet-zorp!jhawk] ˜> sysctl kern.version
kern.version = NetBSD 1.1 (ATHENAOTHER) #4: Thu Jan 18 03:29:37 EST 1996
  ghudson@zygorthian-space-raiders:/u1/build/sys/arch/i386/compile/ATHENAOTHER
```

Of course, it's often useful to find out information about a particular kernel image on disk, rather than the running kernel. You might expect the kernel build process to provide this information using the RCS Tag convention, so it could be extracted with the **ident(1)** command, however this is not the case. All is not lost, as the predecesor to the **ident(1)** command -- the SCCS **what(1)** command -- may be used instead. **what(1)** searches for a string beginning with **@(#)** and displays the rest of the text. Again, under NetBSD:

```
[planet-zorp!jhawk] ˜> what /netbsd
/netbsd
     NetBSD 1.1 (ATHENAOTHER) #4: Thu Jan 18 03:29:37 EST 1996
```

Not all operating systems are as agreeable. More often than not, **what(1)** or **ident(1)** will merely produce identification strings associated with modules linked into the kernel, some of which may not even be significant. Far more reliable are various permutations of the **strings(1)** command. For instance:

NetBSD:

```
[planet-zorp!jhawk] ˜> strings - /netbsd | grep -A1 '@(#)NetBSD'
@(#)NetBSD 1.1 (ATHENAOTHER) #4: Thu Jan 18 03:29:37 EST 1996
  ghudson@zygorthian-space-raiders:/u1/build/sys/arch/i386/compile/ATHENAOTHER
[planet-zorp!jhawk] ˜>
```

SunOS:

```
[all-purpose-gunk!jhawk] ˜> strings - /vmunix | grep SunOS
SunOS Release 4.1.4 (ALL-PURPOSE-GUNK) #4: Thu Apr 11 19:16:05 EDT 1996
```

Solaris (yes, this is pretty lame):

```
[bart-savagewood!jhawk] ˜> /usr/ccs/bin/what /kernel/unix
/kernel/unix:
     SunOS 5.4 Generic 101945-37 December 1995
```

Of course, there are other things to examine than just versioning information. **nm(1)** lists symbols inside an executable (the kernel is normally in a standard file-format for operating system executables). The start of the kernel typically contains a symbol table[4] which contains function and variable names and addresses. **nm(1)** basically just dumps this information.

For instance, here's some symbols under SunOS:

```
[all-purpose-gunk!jhawk] ˜> nm /vmunix | grep _ip_f
f001cfe4 T _ip_forward
f01c7a50 D _ip_forwarding
f001c69c T _ip_freef
f0022310 T _ip_freemoptions
```

3. Booting the kernel

Under recent-vintage Sun 4 machines (i.e. SPARC machines) the OpenBOOT PROM monitor permits the following syntax for booting:

boot *device filename*

Where *device* defaults to **disk** and filename to **/vmunix** under for SunOS.

[4] Under most kernels, you can hear the symbol table quite clearly at the start of the file if you listen to the kernel, eg: *cat /vmunix > /dev/audio*. Be prepared to adjust the volume down (consider using headphones) and to have others look at you strangely.

Under NetBSD/i386, the boot loader displays a prompt as the machine is booting, where you're given a short period of time to respond before producing a spinning combination of |, /, -, and \, characters. It will default to booting /netbsd, but if you're quick you can type in another name (eg: /netbsd.new).

Under NetBSD/SPARC you should just follow the previous instructions.

4. Building a kernel

Building a kernel isn't necessary on all operating systems (well, it may be *necessary*, but in some cases you may not have that option without purchasing a source license, since some vendors don't ship enough source code...), but on some of them it really is, particularly SunOS and NetBSD (and other classic BSD systems). Don't even try it under Solaris, and the faint-of-heart may attempt IRIX.

In the classic BSD system, the kernel source tree lives in */sys*. Typically this is a symlink. Under SunOS, it leads to */usr/kvm/sys* and under NetBSD it leads to */usr/src/sys*. To compile a kernel, just cd to the appropriate directory (*/sys/arch/{i386,sparc,etc.}/conf* for NetBSD, */sys/{sun4m,sun4c}/conf*) edit the kernel configuration file (kernel configuration files are tradionally named in all caps, like GENERIC and TELEGENIC), and type **config KERNELNAME**.

So why would you actually want to rebuild your kernel? If you change the device configuration associated with your machine (port numbers, IRQs, slot numbers, etc.) you may need to rebuild your kernel. Perhaps more importantly, many operating systems come with stupid or insufficient default values. Under SunOS, you might want to apply important security patches distributed by Sun (usually there are a boatload...).

Under SunOS, even though you don't have full kernel sources, you can still rebuild the kernel. This is because Sun ships object files ready-to-be linked. This is why building a kernel under SunOS may take 3 minutes but building a similar kernel under NetBSD may take an hour.

Under SunOS, I typically apply the multicast patches (available from *ftp://parcftp.xerox.com/pub/net-research/ipmulti/*), as well as support for the Berkeley Packet Filter (*ftp://ftp.ee.lbl.gov/bpf.tar.Z*). I'm also careful to turn on UDP checksums[5], increase the size of *tcpsendspace* and *tcprecvspace*, and increase the TCP MSS (Maximum Segment Size) from the lame default of 512 to 1460[6]. To accomplish this, I add these lines to my config file[7]:

```
# Various parameters in in_proto.c:
options "TCPSENDSPACE=\(1024*64\)"    # We live in a world of high bw*delay,
options "TCPRECVSPACE=\(1024*64\)"    #   so lets try and deal.
options "TCPDEFAULTMSS=1460"          # Tcp max. segment size.
options "UDPCKSUM=-1"                 # Udp checksums
```

And then apply the following patch to /sys/netinet/in_proto.c:

```
*** in_proto.c     1996/02/21 04:33:25     1.2
--- in_proto.c     1996/02/21 05:01:39     1.3
```

[5] It used to be thought that UDP checksums weren't very important because packets wouldn't get corrupted very often and they were expensive to compute. At this point computation isn't very expensive at all, and not having UDP checksums turned on can lead to corrupted data in surprising places, like the Domain Name System (DNS). You really want them on.

[6] Because SunOS doesn't support Path MTU Discovery (RFC1191), this may result in some packets being fragmented by routers just before low-MTU links. IMHO, folks maintaining links with MTUs beneath 1500 deserve to lose, so this result is just fine.

[7] note. this whole section needs reformatting

```
***************
*** 153,165 ****
   * Default TCP Maximum Segment Size - 512 to be conservative,
   * Higher for high-performance routers
   */
! int	tcp_default_mss = 512;

  /*
   * Default TCP buffer sizes (in bytes)
   */
! int	tcp_sendspace = 1024*4;
! int	tcp_recvspace = 1024*4;

  /*
   * size of "keep alive" probes.
--- 153,174 ----

   * Default TCP Maximum Segment Size - 512 to be conservative,
   * Higher for high-performance routers
   */
! #ifndef TCPDEFAULTMSS
! #define TCPDEFAULTMSS 512
! #endif
! int	tcp_default_mss = TCPDEFAULTMSS;

  /*
   * Default TCP buffer sizes (in bytes)
   */
! #ifndef TCPSENDSPACE
! #define TCPSENDSPACE (1024*4)
! #endif
! #ifndef TCPRECVSPACE
! #define TCPRECVSPACE (1024*4)
! #endif
! int	tcp_sendspace = TCPSENDSPACE;
! int	tcp_recvspace = TCPRECVSPACE;

  /*
   * size of "keep alive" probes.
```

```
***************
*** 170,176 ****
  int    tcp_keepidle = TCPTV_KEEP_IDLE;        /* for Keep-alives */
  int    tcp_keepintvl = TCPTV_KEEPINTVL;

! int    udp_cksum = 0;             /* turn on to check & generate udp checksums */
  int    udp_ttl = 60;              /* default time to live for UDPs */

  /*
--- 179,188 ----
  int    tcp_keepidle = TCPTV_KEEP_IDLE;        /* for Keep-alives */
  int    tcp_keepintvl = TCPTV_KEEPINTVL;

! #ifndef UDPCKSUM
! #define UDPCKSUM 0
! #endif
! int    udp_cksum = UDPCKSUM;    /* turn on to check & generate udp checksums */
  int    udp_ttl = 60;              /* default time to live for UDPs */

  /*
```

5. Interactively debugging a kernel

kadb, kgdb, ddb, symmon

5.1. Using kadb

kadb is the Sun kernel debugger. It may be booted with **boot kadb** at which point it the boot loader loads the kernel debugger, and the kernel debugger then loads the SunOS or Solaris kernel.

kadb accepts standard *adb* commands, including **$c** to continue (resume exection of the OS) and **$q** to quit (exit to the ROM monitor).

6. Configuring kernel crash dumps

7. Forcing a kernel crash dump

On an OpenBOOT SPARC:

```
0 set-pc
go
```

8. Debugging a kernel crash dump

8.1. Using adb or dbx

8.2. Using crash

9. Patching the kernel

9.1. Patching variables

9.2. Patching code

9.2.1. Patching the running kernel

Dungeon would ask you, "Do you wish me to try to patch you?", and if you said "No", it would report:

> What? You don't trust me? Why, only last week I patched a running
> RSX system and it survived for over thirty seconds. Oh, well.

10. Acknowledgements

Without the inspiration of *Inessential 'roff* this document would never have been approached.

11. Possible appendex A?

ps, kill, etc. /afs/sipb/user/jhawk/src/openprom

12. Possible appendix B?

/afs/sipb/user/jhawk/src/sunostos

13. References