Frequently Asked Questions about lsof

```
**********************************************************************
| The latest release of lsof is always available via anonymous ftp   |
| from lsof.itap.purdue.edu.  Look in pub/lsof.README for its        |
| location.                                                          |
**********************************************************************
```

This file contains frequently asked questions about lsof and answers to them.

Vic Abell <abe@purdue.edu>
July 6, 2004

Table of Contents:

1.0   General Concepts

1.1 Lsof -- what is it?

Lsof is a UNIX-specific tool. Its name stands for LiSt Open Files, and it does just that. It lists information about files that are open by the processes running on a UNIX system.

See the lsof man page, the 00DIST file, the 00QUICKSTART file, and the 00README file of the lsof distribution for more information.

1.2 Where do I get lsof?

Lsof is available via anonymous ftp from lsof.itap.purdue.edu. Look in the pub/tools/unix/lsof sub-directory.

```
ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof
```

Bzip2'd, compressed and gzip'd tar files with GPG certificates are available.

1.2.1 Are there mirror sites?

On September 3, 2003 these sites appeared to mirror lsof:

```
ftp://ftp.cerias.purdue.edu/pub/tools/unix/sysutils/lsof
ftp://ftp.tau.ac.il/pub/unix/admin
ftp://ftp.ayamura.org/pub/lsof/
ftp://ftp.cert.dfn.de/pub/tools/admin/lsof
ftp://ftp.fu-berlin.de/pub/unix/tools/lsof
ftp://ftp.kaizo.org/pub/lsof/
ftp://ftp.tu-darmstadt.de/pub/sysadmin/lsof
ftp://ftp.tux.org/pub/sites/vic.cc.purdue.edu/tools/unix/lsof
ftp://gd.tuwien.ac.at/utils/admin-tools/lsof
ftp://sunsite.ualberta.ca/pub/Mirror/lsof
ftp://the.wiretapped.net/pub/security/host-security/lsof/
```

1.2.2 Are lsof executables available?

Some lsof executables are available in the subdirectory tree pub/tools/unix/lsof/binaries These are neither guaranteed to be current nor cover every dialect and machine architecture.

I don't recommend you use pre-compiled lsof binaries; I recommend you obtain the sources and build your own binary. Even if you're a Sun user without a Sun C compiler, you can use gcc to compile lsof.

If you must use a binary file, please be conscious of the security and configuration implications in using an executable of unknown or different origin. The lsof binaries are accompanied by GPG certificates. Please use them!

Three additional cautions apply to executables:

1. Don't try to use an lsof executable, compiled for one version of a UNIX dialect, on another. Patches can make the dialect version different.

2. If you want to use an lsof binary on multiple systems, they must be running the same dialect OS version and have the same patches and feature support.

1.2.3   Why can't I get the sum(1) result reported in

README.lsof_<revision>?

The "Security" section of the README.lsof_<revision> file of the lsof distribution gives md5, sum, and GPG signature information.

The simplest, the sum(1) signature, seems to be the trickiest. That's because there are different sum(1) methods, BSD systems usually have cksum(1) instead of sum(1), and different systems compute the block size value differently.

First, the lsof sum results are computed with the old, "alternate" algorithm. On newer systems, you can use sum's "-r" option to get that computation result.

Second, on BSD systems you usually must use cksum(1) instead of sum(1), because they have no sum(1). To tell cksum(1) to use the old, "alternate" algorithm, use its "-o1" option.

Third, the second value that sum reports, the block count, may be computed differently on different systems -- usually block count is considered to be 512 or 1,024. The lsof block counts were computed on a system that considers block size to be 1,024. Solaris 8, for example, considers block size to be 512. If your sum(1) or cksum(1) doesn't report a block count that matches the sum(1) signature given in README.lsof_<revision>, check its man page to see what block size it uses, then adjust its block count appropriately.

1.2.4 Why won't gpg accept the lsof-signing PGP public key?

An older PGP key that once signed lsof distributions is included in lsof revisions prior to 4.70. The PGP key is indeed my key, but is incompatible with GPG. It was created about ten years ago and is still acceptable to PGP versions 2.6.2 through 6.5.2.

Lsof revisions 4.70 and above are signed with a copy of my PGP key that has been made acceptable for use with GPG by importing it under GPG's "--allow-non-selfsigned-uid" option.

You can find my GPG compatible key in lsof revisions 4.70 and above and at:

    ftp://lsof.itap.purdue.edu/pub/Victor_A_Abell.gpg

If you have an older lsof revision with my PGP key, there are two possible ways to use it:

* Use it with a PGP version from 2.6.2 through 6.5.2.

* Use GPG's "--allow-non-selfsigned-uid" option when you import my PGP key into your GPG key ring.

```
  $ gpg --allow-non-selfsigned-uid --import Victor_A_Abell.pgp
```

1.3 Where can I get more lsof documentation?

A significant set of documentation may be found in the lsof distribution (See "Where can I get lsof?). There is a manual page, copious documentation in files whose names begin with 00, and a copy of this FAQ in the file 00FAQ (perhaps slightly less recent than this file if you're reading it via a web browser.)

Two URLs provide some documentation that appears in the
lsof distribution:

FAQ: ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/FAQ

man page: ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/lsof_man

1.4 How do I report an lsof bug?

If you believe you have discovered a bug in lsof, you can report it via e-mail to <abe@purdue.edu>. Do NOT report lsof bugs to the UNIX dialect vendor. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

Before you send me a bug report, please do these things:

* Check this file to see if there's a question and answer relevant to your problem.

* Make sure you try the latest lsof revision.

  o Download the latest revision from:

    ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof

  o While connected to lsof.itap.purdue.edu, check for patches:

    ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/patches

  o If patches exist, install them in the latest revision you just downloaded. Then build the latest revision and see if it fixes your bug.

* When you send a bug report, make sure you include output from lsof's -v option. That will tell me what UNIX dialect and lsof revision is involved.

1.5 Where can I get the lsof FAQ?

This lsof FAQ is available in the file 00FAQ in the lsof distribution and at the URL:

    ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/FAQ

1.5.1 How timely is the on-line FAQ?

The on-line FAQ is sometimes too timely. :-)

I update it as soon as new information is available. That may include information about support that won't appear in the lsof source distribution until the next revision. If you encounter something like that, please send me e-mail at <abe@purdue.edu>. I may be able to point you at a pre-release distribution that contains the support of interest. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

1.6 Is there a test suite?

Yes, as of lsof revision 4.63 there's an automated lsof test suite in the tests/ sub-directory of the lsof top-level directory.

More information on using the test suite, what it does, how to use it and how to configure it may be found in the 00TEST file of the lsof distribution. That file also explains where the test suite has been tested.

Frequently asked questions about the test suite will be asked and answered here in the FAQ. (See "Test Suite Problems.")

After lsof has been configured with the Configure script, lsof can be made and tested with:

```
    $ make
    $ cd tests
    $ make
```

Under normal conditions -- i.e., unless the lsof tree has been cleaned or purged severely -- all tests or individual tests may be run by:

```
    $ cd test
    $ make
 or
    $ <run a single test>     (See 00TEST.)
```

1.7   Is lsof vulnerable to the standard I/O descriptor attack?

Lsof revisions 4.63 and above are not vulnerable.

Lsof revisions 4.62 and below are vulnerable, but no damage scenarios have so far been demonstrated.

The standard I/O descriptor attack is a local programmed assault on setuid and setgid programs that tricks them into opening a sensitive file with write access on a standard descriptor, usually stderr (2), and writing error messages to stderr. If the attacker can control the content of the error message, the attacker may gain elevated privileges.

The attack was first described in Pine Internet Advisory PINE-CERT-20020401, available at:

```
    http://www.pine.nl/advisories/pine-cert-20020401.txt
```

If you are using an lsof revision below 4.63, you should remove any setuid or setgid permissions you might have given its executable. Then you should upgrade to lsof revision 4.63.

1.8   Can I alter lsof's make(1) behavior?

Yes. There are at least two ways to do that.

You can put replacements for lsof Makefile strings in your environment. If you specify the -e make option, make will give environment variable values precedence over strings from the Makefile. For example, to change the compiler string CC from the environment, you might do this with the Bourne shell:

```
    $ CC=foobar; export CC
    $ make -e
```

You can also replace lsof Makefile strings in the make command invocation. Here's the previous example done that way:

```
    $ make CC=foobar
```

Changing the CFGF, CFGL, and DEBUG strings used in lsof Makefiles, either from the environment or from the make invocation, can significantly alter lsof make(1) behavior. I commonly use DEBUG to change the -O option to -g so I can build an lsof executable for debugging -- e.g.,

```
    $ make DEBUG=-g
```

(Look for DEBUG in this FAQ for other examples of its use.)

Consult the Makefiles to see what CFGL, CFGL, and other lsof Makefile strings contain, and to see what influence their alteration might have on lsof make(1) behavior.

1.9   Is there an lsof license?

No.

The only restriction on the use or redistribution of lsof is contained in this copyright statement, found in every lsof source file. (The copyright year in or format of the notice may vary slightly.)

```
/*
 * Copyright 2002 Purdue Research Foundation, West Lafayette,
 * Indiana 47907.  All rights reserved.
 *
 * Written by Victor A. Abell
 *
 * This software is not subject to any license of the American
 * Telephone and Telegraph Company or the Regents of the
 * University of California.
 *
 * Permission is granted to anyone to use this software for
 * any purpose on any computer system, and to alter it and
 * redistribute it freely, subject to the following
 * restrictions:
 *
 * 1. Neither the authors nor Purdue University are responsible
 *    for any consequences of the use of this software.
 *
 * 2. The origin of this software must not be misrepresented,
 *    either by explicit claim or by omission.  Credit to the
 *    authors and Purdue University must appear in documentation
 *    and sources.
 *
 * 3. Altered versions must be plainly marked as such, and must
 *    not be misrepresented as being the original software.
 *
 * 4. This notice may not be removed or altered.
 */
```

2.0   Lsof Ports

2.1   What ports exist?

The pub/lsof.README file carries the latest port information:

AIX 4.3.2, 5L, and 5.[12]
Apple Darwin 6.x and 7.x for Power Macintosh systems
BSDI BSD/OS 4.3.1 for x86-based systems
DEC OSF/1, Digital UNIX, Tru64 UNIX 4.0, and 5.[01]
FreeBSD 4.[2-9], 4.10 and 5.[012] for x86-based systems
FreeBSD 5.[012] for Alpha, AMD64 and Sparc64 based systems
HP-UX 11.00 and 11.11
Linux 2.1.72 and above for x86-based systems
NetBSD 1.[456] and 2.0 for Alpha, x86, and SPARC-based systems
NEXTSTEP 3.[13]
OpenBSD 2.[89] and 3.[012345] for x86-based systems
OPENSTEP 4.x
Caldera OpenUNIX 8
SCO OpenServer Release 5.0.[46] for x86-based systems
SCO|Caldera UnixWare 7.1.[134] for x86-based systems
Solaris 2.6, 8, 9 and 10

In the above list the only UNIX dialects present are ones for which I could test the current lsof revision. Lsof may still support unlisted dialect versions -- e.g., HP-UX 10.20, Solris 7, etc. -- but I don't have access to systems where I could test lsof on them, so I can't claim lsof works on them. If your dialect isn't in the list, you should try building lsof on it anyway.

Lsof version 4 predecessors, versions 2 and 3, may support older version of some dialects. Contact me via e-mail if you're interested in their distributions. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

2.2 What about a new port?

The 00PORTING file in the distribution gives hints on doing a port. I will consider doing a port in exchange for permanent access to a test host. I require permanent access so I can test new lsof revisions, because I will not offer distributions of dialect ports I cannot upgrade and test.

2.2.1 User-contributed Ports

Sometimes I receive contributions of ports of lsof to systems where I can't test future revisions of lsof. Hence, I don't incorporate these contributions into my lsof distribution.

However, I do make descriptions of these contributions available. You can find them in the 00INDEX and README files at:

    ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/contrib

Consult the 00INDEX file in the contrib/ directory for a list of the available contributions and consult README there for information on how to obtain them.

2.3 Why isn't there an AT&T SVR4 port?

I haven't produced an AT&T SVR4 port because I haven't seen a UNIX dialect that is strictly limited to the AT&T System V, Release 4 source code. Every one I have seen is a derivative with vendor additions.

The vendor additions are significant to lsof because they affect the internal kernel structures with which lsof does business. While some vendor derivatives of SVR4 are similar, each one I have encountered so far has been different enough from its siblings to require special source code.

If you're interested in an SVR4 version of lsof, here are some existing ports you might consider:

```
    DC/OSx (This obsolete port is only available upon
          special request.)
    Reliant UNIX (This obsolete port is only available
              upon special request.)
    SCO|Caldera UnixWare (This is the most likely choice.)
    Solaris
```

2.4   Why isn't there an SGI IRIX port?

Lsof support for IRIX was terminated at lsof revision 4.36, because it had become increasingly difficult for me to obtain information on the IRIX kernel structures lsof needs to access.

At IRIX 6.5 I decided the obstacles were too large for me to overcome, and I stopped supporting lsof on IRIX. I have sources to the last revision of lsof (4.36) for IRIX, but that version of lsof does not work on IRIX 6.5 and is vulnerable to the standard I/O descriptor attack. (See the "Is lsof vulnerable to the standard I/O descriptor attack?" Q&A for more information.) Contact me to discuss obtaining those sources.

If you wish to pursue the issue, don't contact me, contact SGI. This case was opened with SGI on the subject:

```
    Case ID:      0982584
    Category: Unix
    Priority: 30-Moderate Impact

    Problem Summary:
    kernel structure header files needed for continued lsof
    support

    Problem Description:
    Email In  07/17/98 19:09:23
```

3.0   Lsof Problems

3.1   Configuration Problems

3.1.1 Why can't Configure determine the UNIX dialect version?

The lsof Configure script uses UNIX shell commands, often in a command pipeline, to determine the UNIX dialect version. (Consult the dialect stanza in Configure to determine which commands are used.) If Configure can't determine the dialect version, probably one of the commands is not behaving as Configure expects.

Symptoms of the failure include Configure warning messages and incorrect version definitions in the Makefile CFLAGS.

If you suspect that the lsof Configure script is failing to determine the dialect version correctly, try running the commands from Configure stanza one at a time. That will usually reveal the source of the problem. Be particularly mindful that the PATH environment variable can cause commands to be executed from non-standard directories.

If you can't determine the source of the problem, there is a work-around. You can supply the UNIX dialect version in the LSOF_VSTR environment variable. Use Configure as a guide to forming what it expects in LSOF_VSTR. There is also some information on LSOF_VSTR in the 00XCONFIG documentation file of the lsof distribution.

3.2   Compilation Problems

3.2.1 Why does the compiler complain about missing header files?

When you use make to build lsof, the compiler may complain that it can't find header files -- e.g.,

```
    $ make
    (cd lib; make DEBUG="-O" CFGF="-DAIXA=0 -DAIXV=4330 \
    -DLSOF_VSTR=\"4.3.3.0\"")
    gcc  -DAIXA=0 -DAIXV=4330 -DLSOF_VSTR="4.3.3.0" -O \
    -c ckkv.c
    In file included from ckkv.c:33: ../machine.h:70: \
    sys/types.h: A file or directory in the path name \
    does not exist. \
```

That type of complaint doesn't represent an lsof problem. It represents a problem with a missing system header file that probably should be found in /usr/include or in the system source tree.

As a first step try using find(1) to locate the problem header file. If it's a system header file and can't be found, here are some possible causes:

1. The file set, RPM or package containing the header files has not been installed. instructions for doing that are specific to the UNIX dialect and beyond the scope of this document.

2. If the compiler is gcc, the private gcc header files:

   * May not have been installed;

   * May have been installed incorrectly;

   * May not have been updated properly after the last compiler or system update;

   * Ones from a previous installation may not have been removed.

   A path leading to the gcc private header files can be found with `gcc -v`. Consult the gcc documentation for instructions on proper installation of the private gcc header files.

3. On some dialects -- e.g., FreeBSD, NetBSD, OpenBSD -- lsof may need to use header files that are located in the system source tree -- /sys or /usr/src/sys, for example. Make sure the system source tree has been

installed.

3.2.2 Why does gcc complain about the contents of header files distributed by the system's vendor?

When you use make to build lsof and gcc to compile it, gcc may complain that it finds errors in system header files -- e.g.,

```
    $ make
    (cd lib; make DEBUG="-O" CFGF="-Dsolaris=80000 \
     -DHASPR_GWINDOWS -m64 -DHASIPv6 -DHAS_VSOCK \
     -DLSOF_VSTR=\"5.8\"")
     gcc -Dsolaris=80000  -DHASPR_GWINDOWS -m64 -DHASIPv6 \
     -DHAS_VSOCK -DLSOF_VSTR="5.8"  -O  -c  dvch.c
    In file included from /usr/include/sys/proc.h:31, \
       from /homes/abe/gnu/gcc-3.2.1/lib/gcc-lib/sparcv9-sun-solaris2/ \
     3.2.1/include/sys/user.h:267, from /usr/include/kvm.h:13, \
     from ../dlsof.h:53, from ../lsof.h:172, from dvch.c:43: \
     /homes/abe/gnu/gcc-3.2.1/lib/gcc-lib/sparcv9-sun-solaris2/\
      3.2.1/include/sys/task.h:59: parse error before "uint_t"
```

Errors like the above are most likely not problems in the system's header files, but in the private copies of them that were created when gcc was made or installed. Note the presense of "../gcc-3.2.1/lib/gcc-lib/sparcv9-sun-solaris2/3.2.1/include/..." in the paths for user.h and task.h. It indicates both header files are gcc-specific.

To solve errors like this requires comparing the header files in the vendor's /usr/include tree to the gcc-specific ones in gcc's private gcc-lib/.../include tree. It may be necessary to regenerate gcc-specific header files, correct them or remove them. See the gcc distribution for the appropriate tools.

A possible temporary work-around is to direct gcc to use the vendor's header files instead of its temporary ones by declaring -I/usr/include in the compilation flags.

3.2.3 Other header file problems

Don't overlook any vendor tools that might validate the vendor header files installed on the system -- e.g., the Solaris pkgchk tool can be used to check the header files that were installed from the SUNWhea package.

For other header file problems contact me at <abe@purdue.edu>. Please follow the reporting guidelines in the "How do I report an lsof bug?" section of this FAQ.

3.3 Why doesn't lsof report full path names?

Lsof reports the full path name when it is specified as a search argument for open files that match the argument. However, if the argument is a file system mounted-on directory, and lsof finds additional path name components from the kernel name cache, it will report them.

Lsof reports path name for file system types that have path name lookup features -- e.g., some versions of AdvFS for Digital and Tru64 UNIX. The Linux /proc-based lsof reports full path names, because the Linux /proc file system provides them.

Otherwise, lsof uses the kernel name cache, where it exists
and can be accessed, and reports some or all path name
components (e.g., the sys and proc.h components of
/usr/include/sys/proc.h) for these dialects:

```
      Apple Darwin
      DC/OSx
      DEC OSF/1, Digital UNIX, Tru64 UNIX
      FreeBSD
      HP-UX, /dev/kmem and PSTAT based
      Linux, /dev/kmem-based
      NetBSD
      NEXTSTEP
      OpenBSD
      OPENSTEP
      Reliant UNIX
      Caldera OpenUNIX
      SCO OpenServer
      SCO|Caldera UnixWare
      Solaris 2.x, 7, and 8
```

As far as I can determine, AFS path lookups don't share in
kernel name cache operations, so lsof can't identify open
AFS path name components.

Since the size of the kernel name cache is limited and the
cache is in constant flux, it does not always contain the
names of all components in an open file's path; sometimes
it contains none of them.

Lsof reports the file system directory name and whatever
components of the file's path it finds in the cache, starting
with the last component and working backwards through the
directories that contain it. If lsof finds no path
components, lsof reports the file system device name instead.

When lsof does report some path components in the NAME
column, it prefixes them with the file system directory
name, followed by " -- ", followed by the components --
e.g., /usr -- sys/path.h for /usr/include/sys/path.h. The
" -- " is omitted when lsof finds all the path name components
of a file's name.

The PSTAT-based HP-UX lsof relies on kernel name cache
contents, too, even though its information comes to lsof
via pstat() function calls. Consequently, PSTAT-based
HP-UX lsof won't always report full paths, but may use the
" -- " partial path name notation, or may occasionally
report no path name at all but just the file system mounted-on
directory and device names.

Lsof can't obtain path name components from the kernel name
caches of the following dialects:

```
    AIX
```

Only the Linux kernel records full path names in the
structures it maintains about open files; instead, most
kernels convert path names to device and node number doublets
and use them for subsequent file references once files have
been opened.

To convert the device and node number doublet into a complete path name, lsof would have to start at the root node (root directory) of the file system on which the node resides, and search every branch for the node, building possible path names along the way. That would be a time consuming operation and require access to the raw disk device (usually implying setuid-root permission).

If the prospect of all that local disk activity doesn't concern you, think about the cost when the device is NFS-mounted.

Try using the file system mount point and node number lsof reports as parameters to find -- e.g.,

```
    $ find <mount_point> -inum <node_number> -print
```

and you may get an appreciation of what a file system directory tree search would cost.

3.3.1 Why do lsof -r reports show different path names?

When you run lsof with its repeat (``-r'') option, you may notice that the extent to which it reports path names for the same files may vary from cycle to cycle. That happens because other processes are making kernel calls affecting the cache and causing entries to be removed from and added to it.

3.3.2 Why does lsof report the wrong path names?

Under some circumstances lsof may report an incorrect path name component, especially for files in a rapidly changing directory like /tmp.

In a rapidly changing directory, like /tmp, if the kernel doesn't clear the cache entry when it removes a file, a new file may be given the same keys and lead lsof to believe that the old cache entry with the same keys belongs to the new file.

Lsof tries to avoid this error by purging duplicate entries from its copy of the kernel name cache when they have the same device and inode number, but different names.

This error is less likely to occur in UNIX dialects where the keys to the name cache are node address and possibly a capability ID. The Apple Darwin, BSDI, Digital UNIX, FreeBSD, HP-UX, NEXTSTEP, OPENSTEP, Solaris, Tru64 UNIX, and UnixWare dialects use node address. Apple Darwin, BSDI, FreeBSD, NetBSD, OpenBSD, Tru64 UNIX, and also use a capability ID to further identify name cache entries.

3.3.3 Why doesn't lsof report path names for unlinked (rm'd) files?

Lsof never reports a path names for a file that has been unlinked from its parent directory -- e.g., deleted via rm, or the unlink() system call -- even when some process may still hold the file open. That's because the path name is erased from name caches and the parent directory file when the file is unlinked.

Unlinked open files are sometimes used by applications for temporary, but invisible storage (i.e., ls won't show them, and no other process can open them.) However, they may occasionally consume disk space to excess and cause concern for a system administrator, who will be unable to locate them with find, ls, du, or other tools that rely on finding files by examining the directory tree.

By using lsof's +L option you can see the link count of open files -- in the NLINK column. An unlinked file will have an NLINK value of zero. By using the option +L1 you can tell lsof to display only files whose link count is less than one (i.e., zero).

3.3.4 Why doesn't lsof report the "correct" hard linked file path name?

When lsof reports a rightmost path name component for a file with hard links, the component may come from the kernel's name cache. Since the key which connects an open file to the kernel name cache may be the same for each differently named hard link, lsof may report only one name for all open hard-linked files.   Sometimes that will be "correct" in the eye of the beholder; sometimes it will not. Remember, the file identification keys significant to the kernel are the device and node numbers, and they're the same for all the hard linked names.

3.4 Why is lsof so slow?

Lsof may appear to be slow if network address to host name resolution is slow. This can happen, for example, when the name server is unreachable, or when a Solaris PPP cache daemon is malfunctioning.

To see if name lookup is causing lsof to be slow, turn it off with the ``-n'' option.

Port service name lookup or portmap registration lookup may also be causes of slow-down. To suppress port service name lookup, specify the ``-P'' option.

Lsof doesn't usually make direct portmap calls -- only when +M is specified, or when HASPMAPENABLED is defined during lsof construction. (The lsof help panel, produced with `lsof -h` will display the default portmap registration reporting state.) The quickest first step in checking if lsof is slow because of the portmapper is to use lsof's ``-M'' option.

Lsof may be slow if UID to login name lookups are slow. Suppress them with ``-l''.

On dialects where lsof uses the kernel name cache, try disabling its use with ``-C''.  (You can tell if lsof uses the kernel name cache by looking for ``-C'' in lsof's ``-h'' output.) Of course, disabling kernel name cache use will mean that lsof won't report full or partial path names, just file system and character device names.

Older AIX lsof may be slow to start because of its oslevel identity comparison. (Newer AIX lsof uses uname(2).) See the "Why does AIX lsof start so slowly?" and "Why does lsof warn "compiled for x ... y; this is z.?" sections for more information.

3.5   Why doesn't lsof's setgid or setuid permission work?

If you install lsof on an NFS file system that has been mounted with the nosuid option, lsof may not be able to use the setgid or setuid permission you give it, complaining it can't open the kernel memory device -- e.g., /dev/kmem.

The only solution is to install lsof on a file system that doesn't inhibit setgid or setuid permission.

3.6   Does lsof have security problems?

I don't think so. However, lsof does usually start with setgid permission, and sometimes with setuid-root permission. Any program that has setgid or setuid-root permission, should always be regarded with suspicion.

Lsof drops setgid power, holding it only while it opens access to kernel memory devices (e.g., /dev/kmem, /dev/mem, /dev/swap). That allows lsof to bypass the weaker security of access(2) in favor of the stronger checks the kernel makes when it examines the right of the lsof process to open files declared with -k and -m. Lsof also restricts some device cache file naming options when it senses the process has setuid-root power.

On a few dialects lsof requires setuid-root permission during its full execution in order to access files in the /proc file system. These dialects include:

```
    DC/OSx 1.1 for Pyramid systems
    Reliant UNIX 5.4[34] for Pyramid systems
```

When lsof runs with setuid-root permission it severely restricts all file accesses it might be asked to make with its options.

The device cache file (typically .lsof_hostname in the home directory of the real user ID that executes lsof) has 0600 modes. (The suffix, hostname, is the first component of the host's name returned by gethostname(2).) However, even when lsof runs setuid-root, it makes sure the file's ownerships are changed to that of the real user and group. In addition, lsof checks the file carefully before using it (See the question "How do I disable the device cache file feature or alter it's behavior?" for a description of the checks.); discards the file if it fails the scrutiny; complains about the condition of the file; then rebuilds the file.

See the 00DCACHE file of the lsof distribution for more information about device cache file handling and the risks associated with the file.

3.7   Will lsof show remote hosts using files via NFS?

No. Remember, lsof displays open files for the processes of the host on which it runs. If the host on which lsof is running is an NFS server, the remote NFS client processes that are accessing files on the server leave no process records on the server for lsof to examine.

3.8   Why doesn't lsof report locks held on NFS files?

Generally lock information held by local processes on remote NFS files is not recorded by the UNIX dialect kernel. Hence, lsof can't report it.

One exception is some patch levels of Solaris 2.3, and all versions of Solaris 2.4 and above. Lsof for those dialects does report on locks held by local processes on remotely mounted NFS files.

3.8.1 Why does lsof report a one byte lock on byte zero as a full file lock?

When a process has a lock of length one, starting at byte zero, lsof can't distinguish it from a full file lock. That's because most UNIX dialects represent both locks the same way in their file lock (flock or eflock) structures.

3.9   Why does lsof report different values for open files on the same file system (the automounter phenomenon)?

On UNIX dialects where file systems may be mounted by an automounter with the ``direct'' type, lsof may sometimes report difference DEVICE, SIZE/OFF, INODE and NAME values when asked to report files open on the file system.

This happens because some files open on the file system -- e.g., the current directory of a shell that changed its directory to the file system as the file system's first reference -- may be characterized in the kernel with temporary automounter node information. The cd doesn't cause the file system to be mounted.

A subsequent reference to the file system -- e.g., an ls of any place in it -- will cause the file system to be mounted. Processes with files open to the mounted file system are characterized in the kernel with data that reflects the mounted file system's parameters.

Unfortunately some kernels (e.g., some versions of Solaris 2.x) don't revisit the process that did only a change-directory for the purpose of updating the data associated with the open directory file. The file continues to be characterized with temporary automounter information until it does another directory change, even a trivial ``cd .''.

Lsof will report on both reference types, when supplied the file system name as an argument, but the data lsof reports will reflect what it finds in the kernel. For the different types lsof will display different data, including different major and minor device numbers in the DEVICE column, different lengths in the SIZE/OFF column, different node numbers in the INODE column, and slightly different file system names in the NAME column.

In contrast, fuser, where available, can only report on one reference type when supplied the file system name as an argument. Usually it will report on the one that is associated with the mounted file system information. If the only reference type is the temporary automounter one, fuser will often be silent about it.

3.10 Why don't lsof and netstat output match?

Lsof and netstat output don't match because lsof reports the network information it finds in open file system objects -- e.g., socket files -- while netstat often gets its information from separate kernel tables.

The information available to netstat may describe network activities never or no longer associated with open files, but necessary for proper network state machine operation.

For example, a TCP connection in the FIN_WAIT_[12] state may no longer have an associated open file, because the connection has been closed at the application layer and is now being closed at the TCP/IP protocol layer.

3.10.1 Why can't lsof find accesses to some TCP and UDP ports?

Lsof stands for LiSt Open Files. If there is no open file connected to a TCP or UDP port, lsof won't find it. That's the most common reason why lsof doesn't find a port netstat might report open.

One reason I've found on some UNIX dialects is that their kernels set aside TCP and UDP ports for communicating with support activities, running in application layer servers -- the automounter daemons, and the NFS biod and nfsd daemons are examples. Netstat may report the ports are in use, but lsof doesn't.

Another reason is that netstat may also be able to report a port is open on a particular dialect, because it uses a source of data different from what lsof uses -- e.g., netstat might examine kernel tables or use streams messages to MIB2, while lsof relies on the information it finds in open file structures and their descendants.

Sometimes it's possible to search the data netstat and lsof use. For example, on Linux /proc/tcp and /proc/udp can be examined. There might an entry there for a particular protocol and port, but if the line on which the port appears doesn't have an inode number that matches an inode number of an open file, lsof won't be able to identify the process using the port.

This is a tough question to which there is no easy answer.

3.11 Why does lsof update the device cache file?

At the end of the lsof output you may see the message:

```
    lsof: WARNING: /Homes/abe/.lsof_vic was updated.
```

In this message /Homes/abe/.lsof_vic is the path to the private device cache file for login abe. (See 00DCACHE.)

Lsof issues this message when it finds it necessary to recheck the system device directory (e.g., /dev or /devices) and rebuild the device cache file during the open file scan. Lsof may need to do these things it finds that a device directory node has changed, or if it cannot find a device in the cache.

3.12 Why doesn't lsof report state for UDP socket files?

Lsof reports UDP TPI connection state -- TS_IDLE (Idle), TS_BOUND (Bound), etc. -- for some, but not all dialects.

TPI state is stream-based TCP/IP information that isn't available in many dialects.

A fairly weak general rule is if netstat(1) reports UDP TPI state, lsof may be able to report it, too. But don't be surprised if lsof fails to report UDP TPI state for your dialect. Other factors influence lsof's ability to report UDP TPI state, including the availability of state number data in kernel structures, and state number to state name conversion data.

3.13 I am editing a file with vi; why doesn't lsof find the file?

Classic implementations of vi usually don't keep open the file being edited. (Newer ones may do so in order to maintain an advisory lock.) Instead classic vi opens the file, makes a temporary copy (usually in /tmp or /usr/tmp), and does its work in that file. When you save the file being edited from a classic vi implementation, it reopens and rewrites the file.

During a classic vi session, except for the brief periods when vi is reading or rewriting the file, lsof won't find an open reference to the file from the vi process, because there is none.

3.14 Why doesn't lsof report TCP/TPI window and queue sizes for my dialect?

Lsof only reports TCP/TPI window sizes for Solaris, because only its netstat reports them. The intent of providing TCP/TPI information in lsof NAME column output is to make it easier to match netstat output to lsof output.

In general lsof only reports queue sizes for both TCP and UDP (TPI) connections on BSD-derived UNIX dialects, where both sets of values appear in kernel socket queue structures. SYSV-derived UNIX dialects whose TCP/IP implementations are based on streams generally provide only TCP queue sizes, not UDP (TPI) ones.

While you may find that netstat on some SYSV-derived UNIX dialects with streams TCP/IP may report UDP (TPI) queue sizes, you will probably also find that the sizes are always zero -- netstat supplies a constant zero for UDP (TPI) queue sizes to make its headers align the same for TCP and UDP (TPI) connections. Solaris seems to get it right -- i.e., its netstat does not report UDP (TPI) queue sizes.

When in doubt, I chose to avoid reporting UDP (TPI) queue sizes for UNIX dialects whose netstat-reported values I knew to be a constant zero or whose origin I couldn't determine. OSR is a dialect in this category.

3.14.1   Why doesn't lsof report socket options, socket states, and TCP flags and values for my dialect?

The lsof -T argument, 'f', that selects the reporting of socket options, socket states and TCP flags was implemented at lsof revision 4.71 for the following UNIX dialects, providing the indicated information:

    AIX 4.3.2 and 5.[12]
      All socket options and values, socket states, and TCP flags and values described in lsof(8) are reported.

```
    Apple Darwin 7.2
      All socket options and values, socket states, and TCP
      flags and values described in lsof(8) are reported.
    BSDI BSD/OS 4.3.1
      All socket options and values, socket states, and TCP
      flags and values described in lsof(8) are reported.
    Digital UNIX and Tru64 UNIX 4.0
      All socket options and values, socket states, and TCP
      flags and values described in lsof(8) are reported.
    FreeBSD 4.9, 4.10 and 5.2
      All socket options and values, socket states, and TCP
      flags and values described in lsof(8) are reported.
    HP-UX 11 (/dev/kmem-based lsof)
      Only the TF_NODELAY TCP flag and the TF_MSS value are
      reported.
    HP-UX 11i and 11.11 (PSTAT-based lsof)
      All socket options and values, and socket states are
      reported.  No TCP flags or values are reported.
    Linux
      No socket options and values, socket states, or TCP
      flags and values are reported.  The support for "-Tf"
      could not be added to Linux, because socket options,
      socket states, and TCP flags and values are not
      available via the /proc file system.
    NetBSD 1.6ZH
      All socket options and values, socket states, and TCP
      flags and values described in lsof(8) are reported.
    OpenBSD 3.4
      All socket options and values, socket states, and TCP
      flags and values described in lsof(8) are reported.
    OPENSTEP 4.2
      All socket options and values, socket states, and TCP
      flags and values described in lsof(8) are reported.
    OpenUNIX 8
      All socket options and values, socket states, and TCP
      flags and values described in lsof(8) are reported.
    SCO OpenServer Release 5.0.6
      All socket options and values, socket states, and TCP
      flags and values described in lsof(8) are reported.
    Solaris 2.6, 8, 9 and 10
      The socket option display is limited to BROADCAST,
      DEBUG, DGRAM_ERRIND, DONTROUTE and OOBINLINE.  Socket
      values are limited to KEEPALIVE and LINGER.  No socket
      states are reported.  The TCP DELACK, NODELAY and
      SENTFIN flags are reported.  The TCP MSS value is
      reported.
    UnixWare 7.1.[134]
      All socket options and values, socket states, and TCP
      flags and values described in lsof(8) are reported.
```

3.15 What does "no more information" in the NAME column mean?

When lsof can find no successor structures -- a gnode, inode, socket, or vnode -- connected to the file structure of an open descriptor of a process, it reports "no more information" in the NAME column. The TYPE, DEVICE, SIZE/OFF, and INODE columns will be blank.

Because the file structure is supposed to contain a pointer to the next structure of a file's processing support, if the pointer is NUL, lsof can go no further.

Some UNIX dialects have file structures for system processes

-- e.g., the sched process -- that have no successor structure pointers. The "no more information" NAME will commonly appear for these processes in lsof output.

It may also be the case that lsof has read the file structure while it is being assembled and before a successor structure pointer value has been set. The "no more information" NAME will again result.

Unless lsof output is filled with "no more information" NAME column messages, the appearance of a few should be no cause for alarm.

3.16 Why doesn't lsof find a process that ps finds?

If lsof fails to display open files for a process that ps indicates exists, there may be several reasons for the difference.

The process may be a "zombie" for which ps displays the "(defunct)" state. In that case, the process has exited and has no open file information lsof can display. It does still have a process structure, sufficient for the needs of ps.

Another possible explanation is that kernel tables and structures may have been changing when lsof looked for the process, making lsof unable to find all relevant process structures. Try repeating the lsof request.

3.17 Why doesn't -V report a search failure?

The usual reason that -V won't report a search failure is that lsof located the search item, but was prevented from listing it by an option that doesn't participate in search failure reporting.

For example, this lsof invocation:

```
    $ lsof -V -i TCP@foobar -a -d 999
```

won't report it can't find the Internet address TCP@foobar, even if there is an open file connected to that address, unless the open file also has a file descriptor number of 999 (the ``-a -d 999'' options).

Compile-time options can also affect -V results in much the same way. For example, if HASSECURITY and HASNOSOCKSECURITY are defined at compile time, this lsof invocation, run by a non-root user:

```
    $ lsof -V -c inetd
```

won't report that it can't find the inetd command, even if there is a process running the inetd command, because the HASSECURITY and HASNOSOCKSECURITY options prevent the listing of all but the socket files of another user, and no socket file selector (e.g., "-i") was specified.

3.18 Portmap problems

3.18.1   Why isn't a name displayed for the portmap registration?

When portmap registration reporting is enabled, any time there is a registration for a local TCP or UDP port, lsof displays it in square brackets, following the port number or service name -- e.g., ``:1234[name]'' or ``:name[100083]''.

The TCP or UDP port number or service number (what follows the `:') is displayed under the control of the lsof -P option. The registration identity is held by the portmapper and may be a name or a number, depending on how the registration's owner declared it. Lsof reports what the port map holds and cannot derive a registration name from a registration number.

Lsof can be compiled with registration reporting enabled or disabled by default, under the control of the HASPMAPENABLED #define (usually in machine.h). The lsof help panel (`lsof -h`) will show the default. Lsof is distributed with reporting disabled by default.

3.18.2 How can I display only portmap registrations?

Lsof doesn't have an option that will display only TCP or UDP ports with portmap registrations. The +M option only enables the reporting of registration information when Internet socket files are displayed; +M doesn't select the displaying of Internet socket files -- the -i option does that.

This simple lsof pipe to grep will do the job:

```
$ lsof -i +M | grep "\["
```

This works because -i selects Internet socket files, +M enables portmap registration reporting, and only output lines with opening square brackets will have registrations.

When portmap registration reporting is enabled by default, because the lsof builder constructed it that way, +M is not necessary. (The lsof help panel, produced with `lsof -h` will display the default portmapper registration reporting state.) However, specifying +M when reporting is already enabled is acceptable, as is specifying -M when reporting is already disabled.

Digression: lsof will accept `+' or `-' as a prefix to most options. (That isn't documented in the man page or help panel to reduce confusion and complexity.) The -i option is as acceptable as +i, so the above example could be written a little more tersely as:

```
$ lsof +Mi | grep "\["
```

But be careful to use the ``Mi'' ordering, since ``iM'' implies M is an address argument to `i'.

3.18.3 Why doesn't lsof report portmap registrations for some ports?

Lsof reports portmap registrations for local TCP and UDP ports only. It identifies local ports this way:

* The port appears in the local address section of the kernel structure that contains it.

* The port appears in the foreign address section of a

kernel structure whose local and foreign Internet
   addresses are the same.

* The port appears in the foreign address section of a
   kernel address structure whose Internet address is
   INADDR_LOOPBACK (127.0.0.1).

Following these rules, lsof ignores foreign portmapped ports. That's done for reasons of efficiency and possible security prohibitions. Contacting all remote portmappers could take a long time and be blocked by network difficulties (i.e., be inefficient). Many firewalls block portmapper access for security reasons.

Lsof may occasionally ignore portmap registration information for a legitimate local port by virtue of its local port rules. This can happen when a port appears in the foreign part of its kernel structure and the local and foreign Internet addresses don't match (perhaps because they're on different interfaces), and the foreign Internet address isn't INADDR_LOOPBACK (127.0.0.1).

3.19 Why is `lsof | wc` bigger than my system's open file limit?

There is a strong temptation to count open files by piping lsof output to wc. If your purpose is to compare the number you get to some Unix system parameter that defines the number of open files your system can have, resist the temptation.

One reason is that lsof reports a number of "files" that don't occupy Unix file table space -- current working directories, root directories, jail directories, text files, library files, memory mapped files are some. Another reason is that lsof can report a file shared by more than one process that itself occupies only one file table slot.

If you want to know the number of open files that occupy file table slots, use the +ff option and process the lsof output's FILE_ADDR column information with standard Unix tools like cut, grep, sed, and sort.

You might also consider using use lsof's field output with +ff, selecting the file struct address with -FF, and processing the output with an AWK or Perl script. See the list_fields.awk, list_fields.perl, and shared.perl5 scripts in the scripts/ subdirectory of the lsof distribution for hints on file struct post-processing filters.

3.20 Why doesn't lsof report file offset (position)?

Lsof won't report a file offset (position) value if the -s option has been specified, or if the dialect doesn't support the displaying of file offset (position).

That lsof is reporting only file size is indicated by the fact that the appropriate column header says SIZE instead of SIZE/OFF.

If lsof doesn't support the displaying of file offset (position) -- e.g., for Linux /proc-based lsof -- the -h or -? output panel won't list the -o option.

Sometimes the availability of file offset information

depends on the dialect's kernel. This is particularly true
for socket file offsets.

Maintenance of offsets for pseudo-terminal devices varies
by UNIX dialect and is related to how the dialect kernel
implements pseudo-terminal support. Kernels like AIX, for
example, that short-circuit the transfer of data between
socket and pseudo devices to reduce TCP/IP daemon interrupt
rates won't advance offsets in the TCP/IP daemon socket
files. Instead they will advance offsets in the open
standard I/O files of the shell child precess where the
pseudo-terminal devices are used.

When in doubt about the behavior of lsof in reporting file
offset information, do some carefully measured experiments,
consult the lsof sources, or contact me at <abe@purdue.edu>
to discuss the matter. Please follow the reporting guidelines
in the "How do I report an lsof bug?" section of this FAQ.

3.20.1 What does lsof report for size when the file doesn't really have one?

When a file has no true size -- e.g., it's a socket, a
FIFO, or a pipe -- lsof tries to report the information it
finds in the kernel that describes the contents of associated
kernel buffers.

Thus, for example, size for most TCP/IP files is socket
buffer size. The size of the socket read buffer is reported
for read-only files; the size of the write buffer for
write-only files; and the sum of the buffers sizes for
read-write files.

3.21 Problems with path name arguments

3.21.1 How do I ask lsof to search a file system?

You can ask lsof to search for all open files on a file
system by specifying its mounted path name as an lsof
argument -- e.g.,

```
    $ lsof /
```

Output of the mount command will show file system mounted
path names. It will also show the mounted-on device path
for the file system.

If the mounted-on device is a block device (the permission
field in output of `ls -l <device>` starts with a `b/),
you can specify it's name, too -- e.g.,

```
    $ lsof /dev/sd0a
```

If the mounted-on device isn't a block device -- for example,
some UNIX dialects call a CD-ROM device a character device
(ls output starts with a `c') -- you can force lsof to
assume that the specified device names a file system with
the +f option -- e.g.,

```
    $ lsof +f -- /dev/sd0a
```

(Note: you must use ``--'' after +f or -f if a file name
follows immediately, because +f and -f can be followed by
characters that specify flag output selections.)

When you use +f and lsof can't match the device to a file
system, lsof will issue a complaint.

The +f option may be used in some dialects to ask lsof to
search for an NFS file system by its server name and server
mount point. If the mount application reports an NFS file
system mounted-on value that way, then this sample lsof
request should work.

```
    $ lsof +f -- fleet:/home/fleet/u5
```

Finally, you can use -f if you don't want a mounted file
system path name to be considered a request to report all
open files on the file system. This is useful when you
want to know if anyone is using the file system's mounted
path name. This example directs lsof to report on open
access to the `/' directory, including when it's being used
as a current working or root directory.

```
    $ lsof -f -- /
```

The lsof -f option performs the same function as -f does
in some fuser implementations. However, since the lsof -c
option was chosen for another purpose before the `f' option
was added to lsof, +f was selected as the analogue to the
fuser -c option. (Sorry for the potential confusion.)

3.21.2   Why doesn't lsof find all the open files in a file system?

Lsof may not find all the open files in a file system for
several reasons.

First, some processes with files open on the file system
may have been changing status when lsof examined the process
table, and lsof "missed" them. Remember, the kernel changes
much faster than lsof can respond to the changes.

Second, be sure you have specified the file system correctly.
Perhaps you specified a file instead. You can use lsof's
-V option to have lsof report in detail on what it couldn't
find. Make sure the report for the file system you specified
says "file system." Here's some -V output:

```
    $ /lsof -V /tmp ./lsof.h ./lsof
    COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF  INODE NAME
    lsof    2688  abe  txt   VREG 18,1,7  1428583 226641 ./lsof
    lsof    2689  abe  txt   VREG 18,1,7  1428583 226641 ./lsof
    lsof: no file use located: ./lsof.h
```

You can also use lsof's +f option to force it to consider
a path name as a file system. If lsof can't find a file
system by the specified name, it will issue a complaint --
e.g.,

```
    $ lsof +f -- /usr
    lsof: not a file system: /usr
```

(/usr is a directory in the / file system.)

3.21.3   Why does the lsof exit code report it didn't find open files when some files were listed?

Sometimes lsof will list some open files, yet return a

non-zero exit code, suggesting it hasn't found all the
specified files.

The first thing you should when you suspect lsof is incorrect
is to repeat the request, adding the -V option. In the
resulting report you may find that your file system
specification really wasn't a file system specification,
just a file specification.

Finally, if you specify two files or two file systems twice,
lsof will credit all matches to the first of the two and
believe that there were no matches for the second. It's
possible to specify a single file system twice with different
path names by using both its mounted directory path name
and mounted-one device name.

```
    $ lsof +f -V spcuna:/sysprog /sysprog
    COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF  INODE NAME
    ksh     11092  abe  cwd   VDIR 39,0,1     1536 226562 /sysprog
    (spcuna:/sysprog)
    ...
    lsof: no file system use located: spcuna:/sysprog
```

All matches were credited to /sysprog; none to spcuna:/sysprog.

3.21.4   Why won't lsof find all the open files in a directory?

When you give lsof a simple directory path name argument
(not a file system mounted-on name), you are asking it to
search for processes that have the directory open as a
file, or as a process-specific directory -- e.g., root or
current working directory.

If you want to list instances of open files inside the
directory, you need to specify the individual path names
of those files, or use the lsof +D and +d options.

See the answer to the question "Why are the +D and +d
options so slow?" before you use +D or +d casually.

See the answer to the question "Why do the +D and +d options
produce warning messages?" for an explanation of some
process authority limitations of +D and +d.

3.21.5   Why are the +D and +d options so slow?

The +D and +d options cause lsof to build a path name search
list for a specified directory. +D causes lsof to descend
the directory to its furthest subdirectory, while +d
restricts it to the top level. In both cases, the specified
directory itself is included in the search list. In both
symbolic links are ignored.

Building such a search list can take considerable time,
especially when the specified directory contains many files
and subdirectories -- lsof must call the system readlink()
and stat() functions for each file and directory. Storing
the search list can cause lsof to use more than its normal
amount of dynamic memory -- each file recorded in the search
list consumes dynamic memory for its path name, characteristics,
and search linkages. Using the list means lsof must search
it for every open file in the system.

Building the search list for a directory specified on some

file systems can be slow -- e.g., for an NFS directory with
many files. Some file systems have special logging features
that can introduce additional delays to the building of
the search list -- e.g., NFS logging, or logging on a
Solaris UFS file system. The bottom line is that slow
search list construction may not be so much an lsof problem
as a file system problem. (Hint: if you're using Solaris
UFS logging, consider specifying the "logging,noatime"
option pair to reduce the number of atime writes to the
UFS logging queue and disk.)

A somewhat risky way to speed up lsof's building of the
search list is to use lsof's ``-O'' option. It forces lsof
to do all system calls needed to build the search list
directly, rather than in a child process. While direct
system calls are much faster, they can block in the kernel
-- e.g., when an NFS server stops responding -- stopping
lsof until the kernel operation unblocks.

As an example of the load +D can impose, consider that an
`lsof +D /` on a lightly loaded NeXT '040 cube with a 1GB
root file system disk took 4+ minutes of real time. It
also generated several hundred error messages about files
and directories the lsof process didn't have permission to
access with stat(2).

The bottom line is that +D and +d should be used cautiously.
+D is more costly than +d for deeply nested directory trees,
because of the full directory descent it causes. So use
+d where possible. And you might need to consider the
performance of the file system that holds the directory
you name with +d or +D.

In view of these warnings, when is it appropriate to use
+D or +d? Probably the most appropriate time is when you
would specify the directory's contents to lsof with a shell
globbing construct -- e.g., `lsof *`. If that's what you
need to do, `lsof +d .` is probably more efficient than
having the shell produce a directory list, form it into an
argument vector, and pass the vector to lsof for it to
unravel.

See the answer to the question "Why do the +D and +d options
produce warning messages?" for an explanation of some
process authority limitations of +D and +d.

3.21.6 Why do the +D and +d options produce warning messages?

+D and +d option processing is limited by the authority of
the lsof process -- i.e., lsof can only examine (with
lstat(2) and stat(2)) files the owner of the process can
access.

If the ownership, group membership, or permissions of the
specified directory, file within it, or directory within
it prevents the owner of the lsof process from using lstat(2)
or stat(2) on it, lsof will issue a warning message, naming
the path and giving the system's (lstat(2's or stat(2)'s)
reason (errno explanation text) for refusing access.

As an example, assume user abc has a subdirectory in /tmp,
owned by abc and readable, writable and searchable by only
its owner. If user def asks lsof to search for all /tmp
references with +D or +d, lsof will be unable to lstat(2)

or stat(2) anything in abc's private subdirectory, and will issue an appropriate warning.

Lsof warnings can usually be suppressed with the -w option. However, using -w with +D or +d means that there will be no indication why lsof couldn't find an open reference to a restricted directory or something contained in it.

Hint: if you need to use +D or +d and avoid authority warnings, and if you have super-user power, su and use lsof with +D or +d as root.

3.22 Why can't my C compiler find the rpcent structure definition?

When you try to compile lsof your compiler may complain that the rpcent structure is undefined. The complaints may look like this:

```
    >print.c: In function `fill_portmap':
    >print.c:213: dereferencing pointer to incomplete type
    >...
```

The most likely cause is that someone has allowed a BIND installation to update /usr/include/netdb.h (or perhaps /usr/include/rpc/netdb.h), removing the rpcent structure definition that lsof expects to find there.

Only Solaris has an automatic work-around. (See dlsof.h in dialects/sun.). The Solaris work-around succeeds because there is another header file, <rpc/rpcent.h>, with the rpcent structure definition, and there is a Solaris C pre-processor test that can tell when the BIND <netdb.h> is in place and hence <rpc/rpcent.h> must be included.

Doubtlessly there are similar work-arounds possible in other UNIX dialects whose header files have been "touched" by BIND, but in general I recommend restoration of the vendor's <netdb.h> and any other header files BIND might have replaced. (I think BIND replaces <resolv.h>, <sys/bitypes.h>, <sys/cdefs.h> -- and maybe others.)

3.23 Why doesn't lsof report fully on file "foo" on UNIX dialect "bar?"

Lsof sometimes won't report much information on a given file, or may even report an error message in its NAME column. That's usually because the file is of a special type -- e.g., in a file system specific to the UNIX dialect -- and I haven't used a system where the file appeared during my testing.

If you encounter such a situation, send me e-mail at <abe@purdue.edu> and we may be able to devise an addition to lsof that will report on the file in question. Please follow the reporting guidelines in the "How do I report an lsof bug?" section of this FAQ. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

3.24 Why do I get a complaint when I execute lsof that some library file can't be found?

On systems where the LIBPATH (or the equivalent) environment variable is used to record the library search path in executable files when they are built, an incorrect value may make it impossible for the system to find the shared libraries needed to load lsof for execution.

This may be particularly true on systems like AIX >= 4.1.4, where the lsof Makefile takes the precautionary step of using the -bnolibpath loader flag to insure that the path to the private static lsof library is not recorded in the lsof binary. Should LIBPATH be invalid when lsof is built, it will be recorded in the lsof binary as the default library path search order and lead to an inability to find libraries when lsof is executed.

So, if you get missing library complaints when you try to execute lsof, check LIBPATH, or whatever environment variable is used on your system to define library search order in executable files. Use the tools at your disposal to look at the library paths recorded in the lsof binary -- e.g., chatr on HP-UX, dump on AIX, ldd on Solaris.

Make sure, too, that when the correct library search path has been recorded in the executable file, the required library files exist at one or more of the search paths.

3.25 Why does lsof complain it can't open files?

When lsof begins execution, unless it has been asked to report only help or version information, typically it will attempt to access kernel memory and symbol files -- e.g., /unix, /dev/kmem. Even though lsof needs only permission to open these files for reading, read access to them might be restricted by ownerships and permission modes.

So the first step to diagnosing lsof problems with opening files is to use ls(1) to examine the ownerships and permission modes of the files that lsof wants to open. You may find that lsof needs to be installed with some type of special ownership or permission modes to enable it to open the necessary files for reading. See the "Installing Lsof" section of 00README for more information.

3.26 Why does lsof warn "compiled for x ... y; this is z."?

Unless warnings are suppressed (with -w) or the kernel identity check symbol (HASKERNIDCK) definition has been deleted, all but one lsof dialect version (exception: /proc-based Linux lsof) compare the identity of the running kernel to that of the one for which lsof was constructed. If the identities don't match, lsof issues a warning like this:

```
    lsof: WARNING: compiled for Solaris release 5.7; this is 5.6.
```

Two kernel identity differences can generate this warning -- the version number and the release number.

Build and running identity differences are usually significant, because they usually indicate kernels whose structures are different -- kernel structures commonly change at dialect version releases. Since lsof reads data from the kernel in the form of structures, it is sensitive to changes in them. The general rule is that an lsof compiled for one UNIX dialect version will not work correctly when run on a different version.

There are three work-arounds: 1) use -w to suppress the warning -- and risk missing other warnings; 2) permanently disable the identity check by deleting the definition of HASKERNIDCK in the dialect's machine.h header file -- with the same risk; or 3) rebuild lsof on the system where it is to be run. (Deleting HASKERNIDCK can be done with the Customize script or by editing machine.h.)

Generally checking kernel identity is a quick operation for lsof. However, it is potentially slow under AIX, where lsof must run /usr/bin/oslevel. To speed up lsof, use -w to suppress the /usr/bin/oslevel test. See "Why does AIX lsof start so slowly?" for more information.

3.27 How can I disable the kernel identity check?

The kernel identity check is controlled by the HASKERNIDCK definition. When it is defined, most dialects (exclusion: /proc-based Linux lsof) will compare the build-time kernel identity with the run-time one.

To disable the kernel identity check, disable the HASKERNIDCK definition in the dialect's machine.h header file. The Customize script can be used to do that in its section about the kernel identity check.

Caution: while disabling the kernel identity check may result in smaller lsof startup overhead, it comes with the risk of executing an lsof that may produce warning messages, error messages, incorrect output, or no output at all.

3.28 Why don't ps(1) and lsof agree on the owner of a process?

Generally the user ID lsof reports in its USER column is the process effective user ID, as found in the process structure. Sometimes that may not agree with what ps(1) reports for the same process.

There are sundry reasons for the difference. Sometimes ps(1) uses a different source for process information, e.g., the /proc file system or the psinfo structure. Sometimes the kernel is lax or confused (e.g., Solaris 2.5.1) about what ID to report as the effective user ID. Sometimes the system carries only one user ID in its process structure (some BSD derivatives), leaving lsof no choice.

The differences between lsof and ps(1) user identifications should be small and normally it will be apparent that the confusion is over a process whose application has changed to an effective user ID different from the real one.

3.29 Why doesn't lsof find an open socket file whose connection state is past CLOSE_WAIT?

TCP/IP connections in states past CLOSE_WAIT -- e.g., FIN_WAIT_1, CLOSING, LAST_ACK, FIN_WAIT_2, and TIME_WAIT -- don't always have open files associated with them. When they don't, lsof can't identify them. When the connection state advances from CLOSE_WAIT, sometimes the open file associated with the connection is deleted.

3.30 Why don't machine.h definitions work when the surrounding comments are removed?

The machine.h header files in dialect subdirectories have some commented-out definitions like:

```
    /* #define HASSYSDC "/your/choice/of/path */
```

You can't simply remove the comments and expect the definition to work. That's intended to make you think about what value you are assigning to the symbol. The assigned value might have a system-specific convention. HASSYSDC, for example, might be /var/db/lsof.dc for FreeBSD, but it might be /var/adm/lsof.dc for Solaris.

Symbols defined in the lsof documentation are described in 00PORTING, other machine.h comments, and other lsof documentation files. HASSYSDC, for example, is discussed in 00DCACHE. When comments and documentation don't suffice, consult the source code for hints on how the symbol is used.

3.31 What do "can't read inpcb at 0x...", "no protocol control block", "no PCB, CANTSENDMORE, CANTRCVMORE", etc. mean?

Sometimes lsof will report "can't read inpcb at 0x00000000", "no protocol control block", "no PCB, CANTSENDMORE, CANTRCVMORE" or a similar message in the NAME column for open TCP socket files. These messages mean the file's socket structure lacks a pointer to the INternet Protocol Control Block (inpcb) where lsof expects to find connection addresses -- local and foreign ports, local and foreign IP addresses. The socket file has probably been submitted to the shutdown(2) function for processing.

In some implementations lsof issues the "no PCB, CANTSENDMORE, CANTRCVMORE" message, which tries to explain the absence of a protocol control block by showing the socket state settings that have been made by the shutdown(2) function.

If a non-zero address follows the "0x" in the "can't read inpcb" message, it means lsof couldn't read inpcb contents from the indicated address in kernel memory.

3.32 What do the "unknown file system type" warnings mean?

Lsof may report a message similar to"

```
    unknown file system type, v_op: 0x10472f10
```

in the NAME column for some files.

This means that lsof has encountered a vnode for the file whose operation switch address (from v_op) references a file system type for which there is no support in lsof. After lsof identifies the file system type, it uses pre-compiled code to locate the file system specific node for the file where lsof finds information like file size, device number, node number, etc.

To get some idea of what the file system type might be, use nm on your kernel symbol file to locate the symbol name that corresponds to the v_op address -- e.g., on Solaris do:

```
    $ nm -x /dev/ksyms | grep 0x10472f10
    0x10472f10 ... |file_system_name_vnodeops
```

Where "file_system_name" is the clue to the unsupported file system.

Lsof doesn't use the v_op address to identify file system types on all dialects. Sometimes it uses an index number it finds in the vnode. It will translate that symbol to a short name in the warning message -- e.g., "nfs3" -- if possible.

3.33 Installation

3.33.1   How do I install lsof?

There is no "standard" way to install lsof. Too much depends on local conditions for me to be able to provide working install rules in the lsof make files. (The skeleton install rules you will find just give "hints.") See the "Installing Lsof" section of 00README for a fuller explanation.

To install lsof you will need to consider these questions:

* Who should be able to use lsof? (See HASSECURITY and HASNOSOCKSECURITY in the "Security" section of 00README.)

* Where should lsof be installed? This is a decision mostly dictated by local conditions. Somewhere in /usr/local -- etc/ or sbin/ -- is a common choice.

* What permissions should I give the lsof executable? The answer to this varies by dialect. The make files have install rules that give hints. The "Installing Lsof" section of 00README gives information, too.

* What if I want to install lsof in a shared file system for machines that require different lsof configurations? See the next question and answer, "How do I install a common lsof when I have machines that need differently constructed lsof binaries?"

3.33.2   How do I install a common lsof when I have machines that need differently constructed lsof binaries?

A dilemma that faces some system administrators when they install lsof in a shared file system -- e.g., NFS -- is that they must have different lsof executables for different systems.

The answer is to build an lsof wrapper script that is executed in place of lsof. The script can use system commands to determine which lsof binary should be executed.

Consider this example. You have HP-UX machines with 32 and 64 bit kernels that share the /usr/local/sbin directory where you want to install lsof. Consequently, on each system you must use a different lsof executable, built for the system's bit size. (That's because lsof reads kernel structures, sized by the kernel's bit size.)

One answer is to install three things in /usr/local/sbin: 1) a 32 bit lsof as lsof32; 2) a 64 bit lsof as lsof64; and 3) an lsof script. The script might look like this one, based on work by Amir J. Katz:

```
    #!/bin/sh
    x=`/usr/bin/getconf KERNEL_BITS`  # returns 32 or 64
    if /usr/bin/test "X$x" = "X32"
    then
      lsof32 $*
    else
      if /usr/bin/test "X$x" = "X64"
      then
      lsof64 $*
      else
      echo "Can't determine which lsof executable to use;"
      echo "getconf KERNEL_BITS says: $x"
      exit 1
      fi
    fi
```

Solaris users should consult "How do I install lsof for Solaris 7, 8 or 9?" for information on a similar trick using the Solaris isaexec command.

Users of other dialects might be able to use a command like uname(1) that can identify a distinguishing feature of the system to be incorporated in pre-installed lsof executable names. For example, use `uname -r` and install binaries with suffixes that match `uname -r` output.

3.34 Why do lsof 4.53 and above reject device cache files built by earlier lsof revisions?

When lsof revisions 4.53 run and encounter a device cache file built by an earlier revision, it will reject the file and build a new one. The rejection will be advertised with these messages:

```
    lsof: WARNING: no /dev device in <name>: 2 sections
    ...
    lsof: WARNING: created device cache file: <name>
```

This happens because the header line of the device cache file was changed at revision 4.53 to contain the number of the device on which the device directory resides. The old device cache file header line -- the "2 sections" line in the above warning message, node reads "2 sections, dev=600".

This is not a serious problem, since lsof automatically rebuilds the device cache file with the correct header line.

3.35 What do "like block special" and "like character special" mean in the NAME column?

When lsof comes across an open block or character file whose device, raw device and inode place it somewhere other than /dev (or /devices), lsof doesn't report the /dev (or /devices) name in the NAME column. Instead lsof reports the file system name and device or path name in the NAME column and parenthetically adds "like block special <path>" or "like character special <path>".

The value for <path> will point to a block or character device in /dev (or /devices) whose raw device number matches that of the open file being reported, but whose device number or node number (or both) don't match.

Such an open file is connected to a device node that has been created in a directory other than /dev (or /devices.) See mknod(8) for information on how such nodes are created. (Generally one needs root power to create device nodes with mknod.)

3.36 Why does an lsof make fail because of undefined symbols?

When lsof is compiled via the `make` step and the final load step fails because of missing symbols, the problem may not be lsof. The problem may be that ld, called by the compiler as part of the `make` step, can't find some library that lsof needs.

First check the last compiler line of the make operation -- e.g., the last line with cc or gcc in it before the undefined symbol report -- for loader arguments, i.e., ones beginning with "-l". Except for "-llsof" the rest name system libraries. ("-L./lib" precedes "-llsof" to tell the loader its location.)

Check that all the named system libraries exist. Look in /lib and /usr/lib as a start, but that may not be the only place system libraries live. Consult your dialect's documentation, e.g., the compiler and loader man pages, for other possible locations.

If some system library doesn't exist, that may mean it was never installed or was removed. You'll have to re-install the missing library.

You may find that all the system libraries lsof uses exist. Your next step might be to use nm and grep to see if any of them contain the undefined symbols.

```
    $ nm library | grep symbol
```

If the undefined symbol exists in some library named by the lsof make step, then you might have a problem with some environment variable that controls the load step. The most common is LD_LIBRARY_PATH. It may have a setting that causes ld to ignore a directory containing a library lsof names. If this is the case, try unsetting LD_LIBRARY_PATH in the environment of the ld process -- e.g., do:

```
    $ unset LD_LIBRARY_PATH
or
    % unsetenv LD_LIBRARY_PATH
```

Consult your ld man page for other environment variables that might affect library searching -- e.g., LIBPATH, LPATH, SHLIB_PATH, etc.

If the undefined function doesn't exist in any libraries lsof names, check other libraries. See if the function has a man page that names its library. If the latter is true, please let me know, because that is an lsof problem I need to fix.

If none of these solutions work for you, send me some documentation via e-mail at <abe@purdue.edu>. Include `uname -a` output, the output of the lsof `Configure ...` and `make` steps, and the contents of the environment in force when the `make` step was executed -- e.g., `env` or `printenv` output. If you've located the libraries lsof names, send me that information, too. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

3.37 Command Regular Expressions (REs)

3.37.1 What are basic and extended regular expressions?

Lsof's ``-c'' option allows the specification of regular expressions (REs), enclosed in two slash ('/') characters and followed by these modifiers:

```
    b the RE is a basic RE.
    i ignore case.
    x the RE is an extended RE (the default).
```

Note: the characters of the regular expression may need to be quoted to prevent their expansion by the shell.

Example: this RE is an extended RE that matches exactly four characters, whose third may be an upper ('O') or lower case ('o') oh:

```
    -c /^..o.$/i
```

For simplicity's sake, an RE that is acceptable to egrep(1) is usually called an extended RE.

REs suitable for the old line editor, ed(1), are often called basic REs (and sometimes also called obsolete).

These are some ways basic REs usually differ from extended REs. (There are other differences.)

* `|', `+', `?', '{', and '}' are ordinary characters.

* `^' is an ordinary character except at the beginning of the RE.

* `$' is an ordinary character except at the end of the RE.

* `*' is an ordinary character if it appears at the beginning of the RE.

For more information on REs and the distinction between basic and extended REs, consult your dialect's man pages for ed(1), egrep(1), sed(1), and possibly regex(5) or regex(7).

3.37.2 Why can't I put a slash in a command regular expression?

Since a UNIX command name is the last part of a path to the command's executable, the lsof command regular expression (RE) syntax uses slash ('/') to mark the beginning and end of an RE. Slash may not appear in the RE and the `\' back-slash escape is ineffective for "hiding" it.

More likely than not, if you try to put a slash in an lsof command RE, you'll get this response:

```
    $ lsof -s/.\// ...
    lsof: invalid regexp modifier: /
```

Lsof is complaining the the first character it found after
the second slash isn't an lsof command RE modifier -- 'b',
'i', or 'x'.

3.37.3 Why does lsof say my command regular expression wasn't found?

When you use both forms of lsof's -c option --
``-c <command>'' and ``-c /RE/[m]'' -- and ask that lsof
do a verbose search (``-V''), you may be surprised that
lsof will say that the regular expression wasn't found.

This can happen if the ``-c <command>'' form matches first,
because then the ``-c/RE/[m]'' test will never have been
applied. For example:

```
    $ ./lsof -clsof -c/^..o.$/ -V -adcwd
    COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF  NODE NAME
    lsof    7850  abe  cwd   VDIR    6,0     2048 96442 / (/dev/sd0a)
    lsof: no command found for regex: ^..o.$
```

The ``-clsof'' option matched first, so the ``-c/^..o.$/
option wasn't tested.

3.38 Why doesn't lsof report on shared memory segments?

Lsof reports on shared memory segments only if they're
associated with an open file. That's consistent with lsof's
mission -- to LiSt Open Files. Shared memory segments with
no file associations aren't open files.

That's not to say that a report on shared memory segments
and their associated processes wouldn't be useful. But it
calls for a new tool, not more baggage for lsof.

3.39 Why does lsof report two instances of itself?

When you ask lsof to report all open files and it has
permission to do so, you may see two lsof processes in the
output. The processes are connected via pipes -- e.g.,
here's an HP-UX 11 example.

```
    COMMAND     PID USER   FD   TYPE     DEVICE ...
    ...
    lsof      29450  abe    7w  PIPE 0x48732408 ...
    lsof      29450  abe    8r  PIPE 0x48970808 ...
    ...
    lsof      29451  abe    6r  PIPE 0x48732408 ...
    lsof      29451  abe    9w  PIPE 0x48970808 ...
```

The first process will usually be the lsof you initiated;
the second, an lsof child process that is used to isolate
its parent process from kernel functions that can block --
e.g., readlink() or stat().

Information to and from the kernel functions is exchanged
via the two pipes. When the parent process detects that
the child process has become blocked, it attempts to kill
the child. Depending on the UNIX dialect that may succeed
or fail, but the parent won't be blocked in any event.

See the "BLOCKS AND TIMEOUTS" and "AVOIDING KERNEL BLOCKS"
sections of the lsof man page for more information on why

the child process is used and how you can specify lsof
options to avoid it. (Caution: that may be risky.)

3.40 Why does lsof report '\n' in device cache file error messages?

Lsof revisions prior to 4.58 may report '\n' in error
messages it delivers about problems in the device cache
file -- e.g.,

```
lsof: WARNING: no ...: 4 sections\n
```

That's deliberately done to show the exact contents of the
device cache file line about which lsof is complaining,
including its terminating NL (New Line) '\n' character.
In the above example the line in the device cache file
causing the lsof complaint contains "4 sections" and ends
with a '\n'.

At revision 4.58 and above, device cache error messages
like the one in the above example have been changed to
read:

```
lsof: WARNING: no ...: line "4 sections"
```

The terminal '\n' is no longer reported, the line contents
are enclosed in double quote marks ('"'), and the word
"line" has been added as a prefix to denote that what
follows is a line from the device cache file.

3.41 Kernel Symbol and Address Problems

3.41.1   What does "lsof: WARNING: name cache hash size length error: 0" mean?

When run on some systems, lsof may issue this warning:

```
lsof: WARNING: name cache hash size length error: 0
```

That is an example from a FreeBSD system where lsof reads
the kernel's _nchash variable and finds its value is zero.

Similar warnings include:

```
WARNING: kernel name cache size:
WARNING: can't read kernel's name cache:
WARNING: no name cache address
WARNING: name cache hash size length error:
WARNING: unusable name cache size:
```

These warnings are issued when lsof is attempting to read
the kernel's name cache information. They are usually the
result of a mis-match between the addresses for kernel
symbols lsof gets via nlist(2) and the addresses in use by
the kernel.

Lsof usually gets kernel symbol addresses from what it
believes to be the kernel boot file. In FreeBSD, for
example, that's the path returned by getbootfile(3), usually
/kernel. The boot file can have other names in other UNIX
dialects -- /unix, /vmunix, /bsd, /netbsd, /mach, /stand/vmunix,
etc.

Lsof will get incorrect (mismatched) addresses from the
boot file if it has been replaced by a newer one which

hasn't yet been booted -- e.g., if this is done in FreeBSD:

```
# mv /kernel /kernel.OLD
# mv /kernel.NEW /kernel
```

Until the FreeBSD system is rebooted, the booted kernel is /kernel.OLD, but getbootfile() says it is /kernel. If symbol addresses important to lsof in /kernel.OLD and /kernel don't match, the lsof WARNING messages result.

3.41.2   Why does lsof produce "garbage" output?

Kernel name cache warnings may not be the only sign that lsof is using incorrect symbol addresses to read kernel values. If there's no reasonable test lsof can make on what it reads from the kernel, it may issue other warnings or even report nonsensical results.

The warnings may appear on STDERR, such as:

```
lsof: can't read proc table info
```

Or the warnings may appear in the NAME column as messages saying lsof can't read or interpret some kernel structure -- e.g.,

```
... NAME
... can't read file struct from 0x12345
```

One possible work-around is to point lsof's kernel symbol address gathering at the proper boot file. That can be done with lsof's -k option -- e.g.,

```
$ lsof -k /kernel.OLD
```

The best work-around is to make sure the standard boot file is properly sited -- e.g., if you've moved a new /kernel in place, boot it.

3.42   Why does lsof report open files when run as super user that it doesn't report when run with lesser privileges?

The most likely cause is that the HASSECURITY option was selected when the lsof executable was built.

If HASSECURITY is defined when lsof is built, and lsof is run with the privileges of a non-ROOT user, it will only list open files belonging to the user. The same lsof executable, when run with root user privileges, will list all open files.

However, if HASSECURITY and HASNOSOCKSECURITY are both defined when lsof is built, lsof will list open files belonging to the user and will also list anyone else's open socket files, provided their listing is selected with the "-i" option.

So first ask yourself if the process whose open files lsof won't list belong to a user other than the one under which you're running lsof, and are not open socket files. If either is true, use lsof's help (-h or -?) option and look for a line near the bottom of the help panel that says:

```
"... can list all files..."
```

If the leading "..." says "Only root" then HASSECURITY was defined when lsof was built. If the trailing "..." says ", but anyone can list socket files" then HASNOSOCKSECURITY was also defined.

Should you want an lsof not built with HASSECURITY defined, rerun the lsof Configure script. If you let Configure do customization, make sure you answer 'n' when it asks if you want to enable HASSECURITY and HASNOSOCKSECURITY. If you don't need to do customization, you can rebuild lsof with the "-n" option to Configure. Here's an example of such a rebuild sequence:

```
    $ Configure -clean
    $ Configure -n <dialect-abbreviation>
    $ make
```

More information on the HASSECURITY and HASNOSOCKSECURITY options may be found in the "Security" section of the 00README file of the lsof distribution.

3.43 Test Suite Problems

3.43.1   Errors all tests can report:

3.43.1.1 Why do tests complain "ERROR!!! can't execute ../lsof"?

All tests in the test suite expect an executable lsof file to exist in the tests parent directory, ../lsof.

If there's none there, the tests/Makefile has a rule to make it, but there are probably circumstances where that rule may fail.

The work-around is to re-Configure and re-make lsof, then run the test suite.

3.43.1.2 Why do tests complain "ERROR!!! can't find ..." a file?

Many tests create (or use from a supplied environment variable path) a test file and use lsof to find it. When lsof can't file the file, the tests report the error with messages of the form:

```
    ERROR!!!  can't find ... : <some file path>
 or
    ERROR!!!  lsof couldn't find ...
```

These type of error messages mean that the lsof field output delivered to the test didn't contain a file that the test could identify as the one it intended lsof to find. It might also mean that the process information -- command name, PID or parent PID -- didn't match what the test expected.

This could imply a bug in the test or a bug in lsof. Try using lsof to find a known file that is open. For example, while in the tests sub-directory, do this:

```
    $ sleep 30 < Makefile
    $ ../lsof Makefile
```

If lsof doesn't report that Makefile is open, then the fault may be with lsof. If lsof reports the file is open, search further in the test code for the failure cause.

3.43.1.3 Why do some tests fail to compile?

If a test suite program fails to compile, it may be because I've never had an opportunity to compile the test on the particular UNIX version you are using.

See Appendix B in 00TEST for a list of the UNIX dialects where the test suite has been validate.

3.43.1.4 Why do some tests always fail?

There are several tests in the optional group that have conflicting or special requirements:

LTbigf	needs a dialect and file system that support large files.
LTlock	won't work if the tests/ sub-directory is on an NFS file system.
LTnfs	won't work if the tests/ sub-directory is not on an NFS file system.

So for two tests in particular, LTlock and LTnfs, one will generally fail.

Some failing tests can be run successfully by supplying to them a path to the appropriate type of file system with the -p option.

3.43.1.5 Why does the test suite say it hasn't been validated on my dialect?

When you use the default rule of the test suite's Makefile, it may issue this complaint:

```
$ cd tests
$ make
!!!WARNING!!!

This dialect or its particular version may not have
been validated with the lsof test suite.  Consequently
some tests may fail or may not even compile.

!!!WARNING!!!
```

You are then given the opportunity to answer 'y' to have the test suite operation continue.

This message means that the tests/TestDB file in the tests sub-directory doesn't show that the test suite has been run with the combination of compiler flags found in tests/config.cflags. The tests might nor run; they may encounter compiler failures.

See 00TEST for more information on the UNIX dialects where the test suite has been validated and on the workings of TestDB and its supporting scripts.

When the tests/Makefile "auto" rule is used, the message is more terse and the condition is fatal.

This suite has not been validated on:

<dialect_description>

No opportunity to continue is offered.

The tests/Makefile "silent" rule will skip checking for the validation footprint.

3.43.1.6 Why do the tests complain they can't stat() or open() /dev/mem or /dev/kmem?

When the tests detect that lsof for the dialect reads its information from kernel memory (i.e., the LT_KMEM definition is present in tests/config.cflags), and when the lsof executable path is ../lsof, the tests make sure they can stat() and open() for read access the relevant kernel memory devices, /dev/kmem and possibly /dev/mem.

If those stat() or open() operations fail, the tests issue an error message and quit. The message explains why the system rejected the operation in terms of system "errno" symbols and messages. More often than not the explanation will be that the process lacks permission to access the indicated device node.

One work-around is to give the lsof executable being tested the necessary permission -- e.g., via chgrp, chmod, etc. -- and set its path in the LT_LSOF_PATH environment variable. (See 00TEST.)

Another work-around is to make sure the process that runs the tests has the necessary permissions -- e.g., run it as root, or enable the process login to access the resources. For example, I can run the tests on my personal work-station because /dev/kmem and /dev/mem are readable by the "kmem" group and my login is in that group.

3.43.2 LTbigf test issues

3.43.2.1 Why does the LTbigf test say that the dialect doesn't support large files?

Large file support is defined dialect by dialect in the lsof source files and Configure script. If large file support isn't defined there, it isn't defined in the LTbigf test.

If you think that's wrong for a particular dialect, contact me via e-mail. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

3.43.2.2 Why does LTbigf complain about operations on its config.LTbigf* file?

The LTbigf must be able to write a large file test (size > 32 bits) and seek within it and the process file ulimit size must permit the operation. If the default location for the test file, tests/, isn't on a file system enabled for large file operations or if the process ulimit file block size is too small, lsof will get file operation errors, particularly when seeking

There may be a work-around. Specify the path to a file
LTbigf can write in a file system enabled for large file
operations a the -poption. Make sure that the ulimit file
block size permits writing a large file. For example,
presuming /scratch23 is large-file-enabled, and presuming
you have permission to raise the ulimit file block size,
this shell commands will allow the LTbigf test to run on
AIX:

```
    $ ./LTbigf -p /scratch23/abe/bigfile
```

(Note: syntax for the ulimit command varies by dialect and
by shell. Discovering the proper variant is left to the
reader.)

More information on this subject can be found in the LTbigf
description in the 00TEST file. If course, the LTbigf.c
source file in tests/ is the ultimate source of information,

3.43.2.3 Why does LTbigf warn that lsof doesn't return file offsets?

On some dialects (e.g., Linux) lsof can't report file
offsets, because the data access method underlying lsof
doesn't provide them. If LTbigf knows that lsof can't
report file offsets for the dialect, it issues this warning:

```
    LTbigf ... WARNING!!!  lsof can't return file offsets
            for this dialect, so offset tests have
            been disabled.
```

LTbigf then performs the size test and skips the offset
tests.

For more information see 00TEST and the "Why doesn't
/proc-based lsof report file offsets (positions)?" Q&A of
this file.

3.43.3 Why does the LTbasic test complain "ERROR!!! lsof this ..." and "ERROR!!! lsof that ..."?

The LTbasic test program uses lsof to examine a running
lsof process. It looks for the lsof current working
directory, executable (if possible), and kernel memory file
(if applicable).

Failures to find those things result in error messages.
More information on their production can be found in the
LTbasic.c source file.

3.43.4 NFS test issues

3.43.4.1 Why does the LTnfs test complain "couldn't find NFS file ..."?

The LTnfs test must work with an NFS test file. After it
opens the file it asks lsof to find it on an NFS file system.
If the file isn't on an NFS file system, lsof won't find it,
and the NFS test script complains and fails.

The work-around is to use -p option to supply a path to a
regular NFS file (not a directory) that is on an NFS file
system that LTnfs can read. Presuming /share/bin/file is
such a file and can be opened for reading by the LTnfs
test, this sample shell command could be used to run the

LTnfs test successfully:

```
    $ ./LTnfs -p /share/bin/file
```

(If the NFS file system is enabled for large files, the NFS test will produce the error message described in the following Q&A.)

3.43.5 LTnlink test issues

3.43.5.1 Why does the LTnlink test complain that its test file is on an NFS file system?

The LTnlink test may complain:

```
    LTnlink ... WARNING!!!  test file <path> is NFS mounted.
```

and then issue an explanation and a hint about using the -p option.

The LTnlist test does this because of the way NFS file links are managed when an NFS file is unlinked and the unlinking process still has the file open. Unlike with files on a local file system, when an NFS file that is still open is unlinked, its link count is not reduced.

The file name is changed to a name of the form .nfsxxxx and the link count is left unchanged until the process holding the file open closes it. That's done by NFS so it can keep proper track of the file on NFS clients and servers.

Since the link count isn't reduced when the LTnlink test program closes the NFS test file it still has open, lsof won't find it for LTnlink with a link count of zero. Consequently, LTnlink disables that test section and issues its warning.

The warning suggests that the unlink test section can be run by giving LTnlink a path to a test file with the -p option. That path must name a file LTnlink can write and unlink. Presuming /scratch23/abe/nlinkfile is on a local file system and the LTnlink test can write to it and unlink it, this sample shell command can be used to run the complete LTnlink test successfully:

```
    $ LTnlink -p /scratch23/abe/nlinkfile
```

3.43.5.2 Why does LTnlink delay and report "waiting for link count update: ..."?

On some UNIX dialects and file system combinations the updating of link count after a file has been unlinked can be delayed. Consequently, lsof won't be able to report the updated link count to LTnlink for a while.

When lsof doesn't report the proper link count to LTnlink, it sleeps and repeats the lsof call, using the "waiting for link count update: ..." message as a signal that it is waiting for the expected lsof response. The wait cycle duration is limited to approximately one minute.

3.43.6 LTdnlc test issues

3.43.6.1 Why won't the LTdnlc test run?

Lsof is unable to access the DNLC cache on AIX, because the kernel symbols for the DNLC aren't exported. Contact IBM to learn why that decision was made.

The LTdnlc test won't work on Apple Darwin because lsof can't obtain reliable DNLC information.

The LTdnlc test may fail on other dialects. Failure causes include: a busy system with a DNLC that is changing rapidly; path name components too large for the DNLC; a file system -- e.g., NFS, /tmp, loopback -- which doesn't fully participate in the DNLC; or DNLC limitations (Many DNLC implementations will only store path name components if they are 31 characters or less.)

If you suspect the file system doesn't fully participate in kernel DNLC processing, as a work-around rebuild and test lsof on one that does.

3.43.6.2 What does the LTdnlc test mean by "... <path> found: 100.00%"?

Even when it succeeds the LTdnlc test will report:

```
  LTdnlc ... /export/home/abe/src/lsof4/tests found: 100.00%
```

This message means that the LTdnlc test asked lsof to find the file at the indicated path five times and lsof found the full path name in the indicated percentage of calls. The LTdnlc test considers it a failure if the percentage falls below 50.0%

3.43.6.3 Why does the DNLC test fail?

The DNLC test may fail when some component of the lsof tests/ sub-directory can't be cached by the kernel DNLC. Some kernels have a limit on the length of individual components (typically) 32.

3.43.7 Why hasn't the test suite been qualified for 64 bit HP-UX 11 when lsof is compiled with gcc?

When I attempted to qualify lsof for HP-UX 11, compiled with gcc 3.0, the LTsock test failed. I traced the failure to a gcc compilation error. Because LTsock is an important test, I didn't feel that the test suite was qualified if it failed.

LTsock compiles and runs correctly on 64 bit HP-UX 11 when compiled with HP's ANSI-C.

3.43.8 LTszoff test issues

3.43.8.1 Why does LTszoff warn that lsof doesn't return file offsets?

On some dialects (e.g., Linux) lsof can't report file offsets, because the data access method underlying lsof doesn't provide them. If LTszoff knows that lsof can't report file offsets for the dialect, it issues this warning:

```
    LTszoff ... WARNING!!!  lsof can't return file offsets
                for this dialect, so offset tests have
                been disabled.
```

LTszoff then performs the size test and skips the offset
tests.

For more information see 00TEST and the "Why doesn't
/proc-based lsof report file offsets (positions)?" Q&A of
this file.

3.43.9 LTlock test issues

3.44 File descriptor list (the ``-d'' option) problems

3.44.1 Why does lsof reject a ``-d'' FD list?

Lsof rejects ``-d'' FD lists that contain both exclusions
and inclusions with messages like:

```
    lsof: exclude in an include list: ^1
    lsof: include in an exclude list: 2
```

That's because ``-d'' FD lists are processed as ORed lists,
so it makes no sense for them to contain both exclusions
and inclusions.

I.e.,, if a ``-d'' FD list were to contain ``^cwd,1'', the
``^cwd'' member is useless, because the ``1'' member
dominates by saying "include only FD 1". That effectively
excludes ``cwd'' FD.

Note that lists may have multiple members of the same type,
exclude or include. They are processed as an ORed set.
If an FD isn't excluded by any member of an exclude list,
it is selected. If an FD is included by any member of an
include list, it is selected.

3.44.2 Why are file descriptors other than those in my FD list reported?

The FD list that follows ``-d'' excludes or includes file
descriptors, but unless the ``-a'' (AND) option is specified,
the FD list selections are ORed to the other selections.

For example, the following lsof command will cause all file
descriptors to be listed for the lsof command, and all but
the cwd descriptor for all other commands, probably not
what was intended.

```
    $ lsof -clsof -d^cwd
```

Hint: use ``-a'' -- e.g.,

```
    $ lsof -clsof -a -d^cwd
```

3.45 How can I supply device numbers for inaccessible NFS file systems?

When lsof can't get device numbers for inaccessible NFS file
systems via stat(2) or lstat(2), it attempts to get them from
the mount table's dev=xxx options. Successes are reported with
a warning message that indicates the source of the device
number and that output might be incomplete as a consequence of
the warnings.

Some system mount tables -- e.g., Linux /proc/mounts -- don't
have a dev=xxx option. In that case, and provided lsof for the

dialect supports them, you can use the +m option to create a mount table supplement file and the "+m m" option to use it.

First check the lsof -h (help) output to see if the +m and "+m m" options are supported. If they are, use +m to create a mount table supplement file when all mounted file systems are accessible. Use "+m m" later to make the supplement available when some mounted file systems might not be available.

Here's an example that creates a mount supplement file in $HOME/mnt-sup and later makes it available to lsof.

```
    $ rm -f $HOME/mnt-sup
    $ lsof +m > $HOME/mnt-sup
    ...
    $ lsof +m $HOME/mnt-sup <other lsof options>
```

If lsof has to get the device number from the supplement, it will issue an informative warning message. The warning can be suppressed with lsof's -w option.

Caution! Since the mount table supplement file is static, it is its supplier's responsibility to update it as file system mounts change.

For more information, consult the lsof man page. The "ALTERNATE DEVICE NUMBERS" section has useful information on how lsof acquires device numbers when stat(2) or lstat(2) fail.

3.46 Why won't lsof find open files on over-mounted file systems?

When a file system, /xyz for example, is mounted on the same mount point as another file system, /abc for example, running lsof with an argument of the path of the first file system's mount point -- the over-mounted one, /abc -- probably will not reveal any files open on /abc.

That's because lsof looks for open files on a file system by looking for files with the file system's device number. The two file systems usually have different device numbers and lsof determines the device number search key from the supplied name of the second file system.

A general work-around exists only for Linux. On that UNIX dialect, when you know the over-mounted file system's mount point path, you can ask lsof to report on all open files and grep that output for the path of the over-mounted file system mount point.

3.47 What can be done when lsof reports no more space?

Many lsof methods cache information in memory, using the dialects malloc() library function. When malloc() can't allocate the requested amount of memory, lsof exits with warning messages similar to this AIX message:

```
    lsof: no more dev-ch space at pid 2257750: 0x82a8e600
```

Lsof then exits immediately and produces no more output.

A possible work-around is to increase the memory foot print of the shell that runs lsof. That is often done with the ulimit(1) shell command.

4.0 AIX Problems

4.1 What is the Stale Segment ID bug and why is -X needed?

Kevin Ruderman reports that he has been informed by IBM that processes using the AIX 3.2.x, 4.1[.12345]], 4.2[.1], and 4.3.x kernel's readx() function can cause other AIX processes to hang because of what appears to be file system corruption.

This failure, known as the Stale Segment ID bug, is caused by an error in the AIX kernel's journalled segment memory handler that causes the kernel's dir_search() function erroneously to believe directory entries contain zeroes. The process using the readx() call need not be doing anything wrong. Usually the system must be under such heavy load that the segment ID being used in the readx() call has been freed and then reallocated to another process since it was obtained from kernel memory.

Lsof uses the readx() function to access library entry structures, based on the segment ID it finds in the proc structure of a process. Since IBM probably will never fix the kernel bug, I've added an AIX-specific option to lsof that controls its use of the readx() function.

By default lsof readx() use is disabled; specifying the ``-X'' option enables readx() use.

If you want to change the default readx() behavior of AIX lsof, change the HASXOPT, HASXOPT_ROOT, and HASXOPT_VALUE definitions in dialects/aix/machine.h. You can also use these definitions to enable or disable readx() -- consult the comments in machine.h. You may want to disable readx() use permanently if you plan to make lsof publicly executable.

When HASXOPT_ROOT is defined, lsof will restrict use of the -X option to processes whose real UID is root; if HASXOPT_ROOT isn't defined, any user may specify the -X option. The Customize script offers the option to change HASXOPT_ROOT when HASXOPT is defined and HASXOPT_ROOT is named in any dialect's machine.h header file.

I have never seen lsof cause a problem with its use of readx(), but I believe there is some chance it could, given the right circumstances.

4.1.1 Stale Segment ID APAR

Here are the details of the Stale Segment ID bug and IBM's response, provided by Kevin Ruderman.

```
AIX V3
  APAR=ix49183
      user process hangs forever in kernel due to file
      system corruption
  STAT=closed prs  TID=tx2527 ISEV=2 SEV=2
       (A "closed prs" is one closed with a Permanent
       ReStriction.)
  RCOMP=575603001 aix v3 for rs/6 RREL=r320

AIX V4  (internal defect, no apar #)
```

```
  prefix        p
  name          175671
  abstract      KERMP: loop for ever in dir_search()
```

Problem description:

1. Some user application -- e.g., lsof -- gets the segment ID (SID) for the process private segment of a target process from the process table.

2. The target process exits, deleting the process private segment.

3. The SID is reallocated for use as a persistent segment.

4. The user application runs again and tries to read the user area structure from /dev/mem, using the SID it read from the process table.

5. The loads done by the driver for /dev/mem cause faults in the directory; new blocks are allocated; the size changed; and zero pages created.

6. The next application that looks for a file in the affected directory hangs in the kernel's dir_search() function because of the zero pages. This occurs because the kernel's dir_search() function loops through the variable length entries one at a time, moving from one to the next by adding the length of the current entry to its address to get the address of the next entry. This process should end when the current pointer passes the end of the known directory length.

   However, while the directory length has increased, the entry length data has not, so when dir_search() reaches the zero pages, it loops forever, adding a length of zero to the current pointer, never passing the end of the directory length. The application process is hung; it can't be killed or stopped.

IBM closed the problem with a PRS code (Permanent ReStriction) under AIX Version 3 and had targeted a fix for AIX 4.2. They have recently (I became aware of it September 10, 1996) cancelled the defect report altogether and have indicated they are not going to fix the defect.

4.2 Gcc Work-around for AIX 4.1x

When gcc is used to compile lsof for AIX 4.1x, it doesn't align one element of the user structure correctly. Xlc sees the U_irss element as a type "long long" and aligns it on an 8 byte boundary. That's because the default mode of xlc is -qlonglong; when -qlonglong is enabled, the _LONG_LONG symbol is also defined.

Gcc sees U_irss as a two element array of type long, because _LONG_LONG isn't defined. Hence gcc aligns the U_irss element array on a 4 byte boundary, rather than an 8 byte one, making the gcc incantation of the user structure 4 bytes shorter than xlc's.

When the length of gcc's user structure is supplied as argument 4 to the undocumented getuser() function of the AIX kernel, getuser() rejects it as an incorrect size and returns EINVAL.

Lsof has a work-around for this problem. It involves a special test in the Configure script when the "aixgcc" Configure abbreviation is used -- e.g.,

```
      $ Configure -n aixgcc
```

The test is to compile a small program with gcc and check the alignment of U_irss. If it's not aligned on an 8 byte boundary, the Configure script makes a special copy of <sys/user.h> in ./dialects/aix/aix<AIX_version> whose U_irss will align properly, and generates compile time options to use it.

While I have tested this work-around only with 4.1.4, it should work with earlier versions of AIX 4.1. It does not work for AIX 4.2; a different work-around is employed there. (See the next section.)

If you want to use this technique to compile other AIX 4.1x programs with gcc for using getuser(), check the Configure script.

Stuart D. Gathman identified this gcc AIX alignment problem.

4.3 Gcc and AIX 4.2[.1]

Alignment problems with gcc and AIX 4.2[.1] inside the user structure are more severe, because there are some new 64 bit types in AIX that gcc doesn't yet (as of 2.7.x) support. The <sys/user.h> U_irss element problem, discussed in 4.3 above, doesn't exist in 4.2[.1].

The AIX lsof machine.h header file has a work-around, provided by Henry Grebler, that bypasses gcc alignment problems. Later versions of gcc (e.g., 2.8.x) will probably bypass the problems as well.

4.4 Why won't lsof's Configure allow the use of gcc for AIX below 4.1?

Gcc can't reliably be used to compile lsof for AIX versions below AIX 4.1 because of possible kernel structure element alignment differences between it and xlc.

4.5 What is an AIX SMT file type?

When you run AIX X clients with the DISPLAY environment variable set to ``:0.0'' they communicate with the AIX X server via files whose kernel file structure has an undefined type (f_type == 0xf) -- at least there's no definition for it in <sys/file.h>.

These are Shared Memory Transport (SMT) sockets, an artifact of AIXWindows, designed for more efficient data transfers between the X server and its clients.

Henry Grebler and David J. Wilson alerted me to the existence of these files. Mike Feldman and others helped me identify them as SMT sockets.

The curious reader can find more about SMT sockets in /usr/lpp/X11/README.SMT.

4.6   Why does AIX lsof start so slowly?

When AIX lsof starts it compares the running kernel's identity to the one for which it was built, using /usr/bin/oslevel. That comparison can sometimes take a long time to complete, depending on the system's maintenance level and how recently it was examined with oslevel.

AIX revisions 4.67 and above for AIX 5 and above don't use oslevel to determine the kernel identity. They use uname(2) instead, and it is much faster.

You can skip the oslevel test by suppressing warning messages with lsof's -w option. Doing that carries with it the risk of missing other warning messages, however.

You can also disable the kernel identity check by disabling the definition of the HASKERNIDCK symbol by editing AIX machine.h header file or by using the Customize script to disable it.

See the "Why does lsof warn "compiled for x ... y; this is z.?" section for more information.

4.7   Why does exec complain it can't find libc.a[shr.o]?

When you try to execute lsof you may get this complaint:

```
    exec(): 0509-036 Cannot load program ./lsof because of
                the following errors:
          0509-022 Cannot load library libc.a[shr.o].
          0509-026 System error: A file or directory in
            the path name does not exist.
```

This is probably the result of making lsof when the LIBPATH environment variable contained a directory path that doesn't contain libc.a. You can see what LIBPATH contained when lsof was made by using the dump application on lsof. For example, if LIBPATH contained /foo/bar when lsof was made, you will see this (partial) dump output:

```
    $ dump -H lsof
    ...
            ***Import File Strings***
    INDEX  PATH                         BASE         ...
    0      /foo/bar
```

To correct the problem, revisit the lsof source directory and remake lsof this way:

```
    $ unset LIBPATH; make           (sh or ksh)
or
    % unsetenv LIBPATH; make        (csh or tcsh)
```

4.8   What does lsof mean when it says, "no PCB, CANTSENDMORE, CANTRCVMORE" in a socket file's NAME column?

When an AIX application calls shutdown(2) on an open socket file, but hasn't called close(2) on the file, the file will remain visible to lsof as an open socket file without any extended protocol information.

Lsof reports that state in the NAME column by saying that there is "no PCB" (Protocol Control Block) for the protocol (e.g., TCP in the NODE column). If the open socket file has the state variables SO_CANTSENDMORE and SO_CANTRCVMORE set -- i.e., from the shutdown(2) call -- lsof reports them with the CANTSENDMORE and CANTRCVMORE notes in the NAME column.

4.9 When the -X option is used on AIX 4.3.3, why does lsof disable it, saying "WARNING: user struct mismatch; -X option disabled?"

The -X option causes lsof to read the loader information of the user structure from virtual memory via the readx() system call. It does that with the user structure definition from <sys/user.h> that was compiled into the lsof executable.

On AIX 4.3.3 there are two different user structure definitions in two separate <sys/user.h> header files, distributed at different times by IBM. If lsof was compiled with one and the kernel on which lsof is being run was compiled with the other, lsof normally won't get correct loader information when it calls readx().

In an attempt to compensate for that difference, lsof makes an independent check of the loader information by getting the user structure's open file count via readx() and comparing it to the open file count obtained independently via getprocs(). When the two counts don't match, lsof tries to read the count (and re-read the loader information) with two offsets, based on observed differences between the two user structures.

When one of the three attempts produces a correct open file count, lsof uses its corresponding offset on subsequent readings of the loader information.

When none of the three attempts produces a correct open file count, lsof issues the WARNING message and disables -X processing.

To eliminate this problem, obtain an lsof binary that matches the kernel of the AIX 4.3.3 system where you want to run lsof. Compiling lsof on the target system is the preferred way to get a matching binary.

4.10 Why doesn't the -X option work on my AIX 5L or 5.[12] system?

If your AIX 5L or 5.[12] system uses the ia64 architecture, lsof needs setuid-root permission to be able to do the processing that -X requires.

Check the output of `uname -a` to determine the architecture type.

The work-around is to give lsof setuid-root permission.

4.11 Why doesn't /usr/bin/oslevel report the correct AIX version?

The oslevel man page says, "The oslevel command reports the level of the operating system using a subset of all filesets installed on your system."

You can see which fileset is below the expected level with oslevel's -l option. For example, if you believe your system is at AIX level 4.3.3, but oslevel reports 4.3.2, use this oslevel command to find the filesets below 4.3.3:

```
    $ /usr/bin/oslevel -l 4.3.3.0
```

If you don't know what level argument to supply to oslevel's -l option, use oslevel's -q option first.

4.11.1 Why doesn't /usr/bin/oslevel report the correct AIX version on AIX 5.1?

The subset list for oslevel on AIX 5.1 seems to include at least two filesets, xlsmp.msg.en_US.rte and xlsmp.rte, that do not install from AIX 5.1 media with a 5.1.0.0 level. Hence, oslevel reports 5.0.0.0 instead of the expected 5.1.0.0.

If either xlsmp.msg.en_US.rte or xlsmp.rte is installed, lsof's Configure script and run-time tests will identify the AIX version incorrectly. The run-time test will issue a complaint message of this form:

```
    lsof: WARNING: compiled for AIX version xxx; this is yyy.
```

You can correct the Configure test by pre-defining the oslevel value, setting the correct value in the LSOF_VSTR environment variable before running the Configure script -- e.g., to pre-define AIX 5.1 when using ksh, do this:

```
    $ LSOF_VSTR=5.1.0.0 Configure -n aix
```

You can't affect oslevel output without uninstalling xlsmp.msg.en_US.rte and xlsmp.rte. If you can't do that, you'll have to put up with the run-time complaint.

4.12 Why does lsof for AIX 5.1 or above Power architecture complain about kernel bit size?

When you run an lsof binary on an AIX 5.1 or above Power system, it might complain:

```
    lsof: FATAL: compiled for a 32 bit kernel.
        The bit size of this kernel is 64.
or
    exec: 0509-036 Cannot load program ./lsof because of
               the following errors:
          0509-032 Cannot run a 64-bit program on a 32-bit
               machine.
```

Starting at lsof revision 4.61, lsof binaries for Power architecture systems running AIX 5.1 or above are closely tied to the kernel bit size. Lsof must do that so it can read and understand kernel structures.

Lsof's Configure script tunes the lsof configuration so that the binary built in the make(1) step is adjusted to the kernel bit size.

An lsof binary knows the bit size for which it was constructed, tests the bit size of the kernel under which it is running, and objects if the two sizes don't match. To see the bit size for which lsof was constructed, run it with its -v option and look for these lines in the output:

```
    configuration info: 32 bit kernel
```

```
 or
    configuration info: 64 bit kernel
```

(Note: these lines will appear only in -v output for AIX 5.1 and above lsof binaries, built for Power architecture.)

You can see the kernel bit size test method in the aix stanza of the lsof Configure script and in the get_kernel_access() function of the lsof .../dialects/aix/dproc.c source file.

There is more information on pre-defining the kernel bit size when building lsof in Configure, 00PORTING, and 00XCONFIG.

The only work-around is to use an lsof binary built to match the running kernel bit size.

4.13 What can't gcc be used to compile lsof on the ia64 architecture for AIX 5 and above?

Gcc can't be used to compile lsof on the ia64 architecture for AIX 5 and above because I haven't had access to a system that has a working gcc compiler. The gcc compiler on my one and only ia64 AIX 5.1 test system, provided by IBM, didn't work at all.

4.14 Why does lsof get a segmentation fault when compiled with gcc for a 64 bit Power architecture AIX 5.1 kernel?

When lsof is configured with the lsof "aixgcc" Configure abbreviation, the resulting lsof executable may cause a segmentation violation when it is run. I've observed this with gcc version 2.9-aix43-010414-7.

As far as I have been able to tell, the segmentation fault is the result of a gcc compilation, loading, or library error. Watching lsof run with gcc's companion debugger, gdb, shows no error in the lsof source code that might explain the fault.

The only work-around I know is to use the IBM C compiler in place of gcc -- i.e., use the "aix" lsof Configure abbreviation.

4.15 Why does lsof ignore AFS on my AIX system?

The lsof Configure script quits on AIX when AFS is present, the AIX version is greater than 4.3.3.0 or the AFS version is greater than 3.5. That's because I have no test systems available for those AIX and AFS version combinations.

When the lsof Configure script detects an AIX and AFS version combination that is unsupported, it will report:

```
  !!!FATAL: Lsof does not support AFS on this combination of
          AIX and AFS versions.  To disable AFS, set the
          value of the AIX_HAS_AFS environment variable to
          "no".
```

The only work-around is to set the AIX_HAS_AFS environment variable as explained in the error message:

```
    $ AIX_HAS_NSF=no; export AIX_HAS_NFS
    $ ./Configure -n aix
```

4.16 Why does lsof report "system paging space is low" and exit?

When AIX paging space runs low, the AIX kernel sends a SIGDANGER signal to processes, warning them that they should reduce their memory usage.

When lsof receives that signal, it issues the following fatal error message and exits:

```
    lsof: FATAL: system paging space is low.
```

A possible work-around is to limit the amount of information lsof must cache in its process memory with the "-c", "-g", "-l" and "-p" options.

Also see the answer to the "What can be done when lsof reports no more space?" question.

5.0 Apple Darwin Problems

5.1 Why does Configure ask for a path to the Darwin XNU kernel header files?

When lsof was ported to Apple Darwin by Allan Nathanson at revision 4.53, some kernel header files needed by lsof weren't being exported by the developers. (That's still true at lsof revision 4.70.)

At first a shell script that Allan provided would get the missing header files by checking them out from the CVS root. Although the script was updated from time to time, eventually the re-organization of Darwin sources has made it impossible to update the script to do an automatic dowload of the missing header files.

At lsof revision 4.69 it is necessary for the Darwin lsof builder to download the Darwin XNU kernel headers before attempting to build lsof. The download my be done via a web browser, starting at this URL:

```
    http://www.opensource.apple.com/darwinsource/index.html
```

Once there, select the link to the Mac OS X version that matches the one on the system where lsof is to be built.

Follow that link's "<source>" link. Once there, select the tar.gz link of the xnu* entry near the bottom of the page. That entry should have a name that matches the xnu* name shown by `uname -a` -- e.g., if uname reports:

```
    $ uname -a
    ... root:xnu/xnu-344.49 ...
```

Then the appropriate xnu* entry is xnu-344.49. Clicking its link should lead to an "Apple Open Source" page requesting an Apple ID and password.

Enter them if they're available. If an Apple ID and password aren't available, get them by following the instructions on the page -- e.g., follow the signin.apple.com link.

Once a valid Apple ID and its password have been entered,

the download will begin. Select the saving of the downloaded
xnu*tar.gz file in an appropriate place on the Mac OS X
system.

Once the download completes, gunzip and extract the tar
archive -- e.g.,

```
    $ gunzip -c xnu-344.49.tar.gz | tar xf -
```

Remember the absolute path to the extracted archive. E.g.,
if the xnu-344.49.tar archive was extracted to the lsof
builder's home directory, its full path will be something
like:

```
    ~/xnu-344.49
```

Now run the lsof Configure script. When it asks for the
path to the Darwin XNU kernel header files, supply the path
to the extracted xnu* archive -- e.g., ~/xnu-344.49.

The path to the Darwin XNU kernel headers may also be
supplied to the Configure script in the DARWIN_XNUDIR
environment variable, eliminating the need to enter it
interactively -- e.g.,

```
    $ DARWIN_XNUDIR=~/xnu-344.49 ./Configure -n darwin
```

5.1.1 Why does Configure complain that Darwin XNU kernel header files are missing?

These are some reasons why the lsof Configure script might
claim that Darwin XNU header files are missing:

* The wrong path to them was specified.

* The files and directories in the path are not readable
  and searchable -- i.e., check the modes and ownerships.

* The downloaded archive doesn't match the Mac OS X
  version of the system.

If in doubt, revisit the Darwin XNU kernel header file
download instructions in the answer to the question "Why
does Configure ask for a path to the Darwin XNU kernel
header files?"

If Configure still can't find Darwin XNU kernel header
files, contact me via e-mail at <abe@purdue.edu> for help.
Make sure "lsof" appears in the "Subject:" line so my e-mail
filter won't classify your letter as Spam.

5.2 Why doesn't Apple Darwin lsof report text file information?

At the first port of lsof to Apple Darwin, revision 4.53,
insufficient information was available -- logic and header
files -- to permit the installation of VM space scanning
for text files. As of lsof 4.70 it is sill not available.

Text file support will be added to Apple Darwin lsof after
the necessary information becomes available.

5.3 Why doesn't Apple Darwin lsof support IPv6?

At the first port of lsof to Apple Darwin, revision 4.53,

Apple Darwin lacked IPv6 support. IPv6 became available
in Apple Darwin version 1.5 and support for it was added
to lsof then.

5.4 Why does lsof complain about a mismatch between the release for which lsof was compiled and the booted Mac OS X release?

When lsof is started on the "Gold Master" Darwin release
(aka Mac OS X), it complains:

```
lsof: compiled for 1.0 release; this is 1.3.2.
```

This happens because the lsof binary released with Mac OS
X was built on a system whose release number (1.0) doesn't
match that of the released system -- usually 1.3.x Lsof
makes this check because UNIX dialect OS changes are often
accompanied by header file changes that affect lsof.

In this specific case, this error can be ignored. If you
don't want to do that, get the lsof distribution and build
lsof so its built-on and running-on Mac OS X release numbers
match.

6.0 BSD/OS BSDI Problems

6.1 Why doesn't lsof report on open kernfs files?

Lsof doesn't report on open BSD/OS BSDI kernfs files because
the structures lsof needs aren't defined in the kernfs.h
header file in /sys/misc/kernfs.

7.0 DEC OSF/1, Digital UNIX, and Tru64 UNIX Problems

7.1 Why does lsof complain about non-existent /dev/fd entries?

When you run lsof for Digital UNIX 3.2, lsof may complain:

```
lsof: can't lstat /dev/fd/xxx: No such file or directory
lsof: can't lstat /dev/fd/yyy: No such file or directory
```

(Or it may warn about other missing /dev/fd paths.) When
you do an ``ls /dev/fd'' none of the missing paths are listed.

This is caused by a bug in the DEC library function
getdirentries(). For some reason, when /dev/fd is a file
system mount point, getdirentries() returns an incorrect
size for it to readdir(). (Lsof calls readdir() in its
ddev.c readdev() function.) Because of the incorrect size,
readdir() goes past the end of the /dev/fd directory buffer,
encounters random paths and returns them to lsof. Lsof
then attempts to lstat(2) the random paths, gets error
replies from lstat(2), and complains about the paths.

Duncan McEwan discovered this error and has reported it to
DEC. Duncan also supplied a work- an alternate readdir()
function as a work-around. I've incorporated his readdir()
in dialects/osf/ddev.c (as the static ReadDir() function)
with some slight modifications, and enabled its use when
the USELOCALREADDIR symbol is defined.

The Configure script defines USELOCALREADDIR for Digital
UNIX version and 3.2. If you don't want to use Duncan's

local readdir() function, edit the Makefile and remove
-DUSELOCALREADDIR from the CFGF string. When DEC releases
a corrected getdirentries() function, I'll modify the
Configure script to stop defining USELOCALREADDIR.

7.2 Why does the Digital UNIX V3.2 ld complain about Ots* symbols?

When you compile lsof on your Digital UNIX V3.2 system, ld
may complain:

```
    ld:
    Unresolved:
    knlist
    _OtsRemainder32Unsigned
    _OtsDivide64Unsigned
    _OtsRemainder64Unsigned
    _OtsDivide32Unsigned
    _OtsMove
    _OtsDivide32
    _OtsRemainder32
    *** Exit 1
```

Chris Eleveld reports this happens on Digital UNIX V3.2
systems after the Fortran compiler has been installed.

The best work-around seems to be to remove -lmld from the
CFGL string in the Makefile produced by Configure -- i.e.,
change:

```
    CFGL=    -lmld
to
    CFGL=
```

According to the V3.2 man page for nlist(3), this shouldn't
work, but my testing shows that it does. Although I haven't
been able to test this second work-around, you might try
adding -lots to CFGL, rather than removing -lmld -- i.e.,
change:

```
    CFGL=    -lmld
to
    CFGL=    -lmld -lots
```

WARNING: my testing also shows that the V2.0 nlist(3) man
page means what it says when it calls for -lmld -- lsof
loaded without -mld under V2.0 can't locate the proc
(process) table address.

    DON'T REMOVE -lmld FROM THE DIGITAL UNIX V2.0 MAKEFILE.

If you run into this problem, please let me know what
problem you encountered and how you solved it.

7.3 Why can't lsof locate named pipes (FIFOs) under V3.2?

While lsof for V3.2 can report on named pipes (FIFOs), it
can't find them by name. That appears to happen because
of the way the V3.2 kernel lstat(2) function reports named
pipe device numbers.

The V3.2 kernel reports the device number as 0xfffffff,
while the kernel structures for named pipes that lsof
examines contain the device number of the file system on
which the named pipe resides.

Consequently, lsof can't match the device and inode number pair it receives from applying lstat(2) to the named pipe with any device and inode number pair it finds when scanning kernel structures.

I don't have a work-around. You can, of course, ask for full lsof output and use a post-processing filer (e.g., grep) to locate the named pipe of interest.

This problem doesn't exist under V2.0.

7.4   Why does lsof use the wrong configuration header files? For example, why can't the lsof compilation find cpus.h?

DEC OSF/1, Digital UNIX, and Tru64 UNIX configuration header files describe the hardware and software environment for which your kernel boot file was constructed. For example, /sys/<name>/cpus.h defines the number of CPUs in its NCPUS #define.

Lsof searches for the configuration header file subdirectory in /sys (/usr/sys for Digital UNIX version 4.0 and Tru64 UNIX) by converting the first host name component to capital letters -- e.g., TOMIS is derived from tomis.bio.purdue.edu. If that subdirectory exists, lsof uses header files from it. (Configure reports what subdirectory is being used.)

If Configure doesn't find a host-name derived subdirectory, it prompts you for the entry of a subdirectory name. If you can't find one, quit Configure and run the kernel generation process to create a proper configuration sub-directory. If you don't identify a proper configuration subdirectory and you try to compile lsof, the compiler will complain about missing header files -- e.g., a missing cpus.h.

Once you have located or generated a proper configuration subdirectory, rerun Configure. If you have generated a configuration subdirectory whose name is derived from the host name, Configure will find and use it. If not, you will have to specify its name to Configure.

7.5   Why does lsof indicate incomplete paths with " -- " for Tru64 UNIX 5.1 files?

When lsof can't find a component of a path in the kernel's name cache (aka DNLC), or can't determine that the left-most component has as its parent the file system root, it uses an "incomplete path" notation. That notation begins with the file system root name, followed by " -- ", followed by the consecutive path name components lsof was able to find in the DNLC -- e.g., "/ -- init".

Because the DNLC was significantly redesigned in Tru64 UNIX 5.1, lsof's handling of the cache had to be completely redone. As part of the DNLC redesign a name cache entry parameter lsof formerly used to locate the file system root of a path was removed. With help from Chang Song I've been able to implement an alternate method for detecting the root of these file system types: AdvFS (MSFS), CDFS, DVDFS, FDFS, NFS, NFS3, and UFS.

When lsof doesn't know how to identify the root for a file system type, it will resort to the " -- " incomplete path notation.

7.6 Why doesn't lsof report link count, node number, and size for some Tru64 5.x CFS files?

Lsof reports link count, node number, and size for open CFS files as recorded in their kernel node structure's cached attributes. Sometimes not all attributes are cached on the system where lsof runs, so lsof cannot report them.

7.7 Why does lsof say it can't read the kernel name list or proc table on Digital UNIX 4.x or Tru64 UNIX?

By default on Digital UNIX 4 and Tru64 UNIX lsof reads the addresses for kernel symbols with the knlist(3) function. That function can fail, for example, when the kloadsrv daemon isn't running or is malfunctioning. When that happens, lsof may abort with one of these error messages:

```
    lsof: can't read kernel name list from knlist(3): ...
  or
    lsof: can't read proc table info
```

The first message suggests a complete knlist(3) or kloadsrv failure; the second, a partial one.

If you know the name of the file from which the running system was booted, e.g., /vmunix, you can use lsof's -k option to direct it to read kernel symbol addresses from the name list of that file --

```
    $ lsof -k /vmunix ...
```

If that works, then knlist(3) is malfunctioning and you need to fix it.

8.0 FreeBSD Problems

8.1 Why doesn't lsof report on open kernfs files?

Lsof doesn't report on open FreeBSD kernfs files because the structures lsof needs aren't defined in the kernfs.h header file in /sys/misc/kernfs.

8.2 Why doesn't lsof work on my FreeBSD system?

If lsof doesn't work on your FreeBSD system, first make sure you have the latest lsof revision. See the answer to the "Where do I get lsof?" question for information on how to get the latest lsof revision.

Once you have gotten the latest lsof revision, Configure and make it. If Configure fails -- e.g., it complains about an unknown FreeBSD version -- then lsof probably hasn't been ported to your FreeBSD version yet, and there's no need to go any further. Follow the answer to the "How do I report an lsof bug" to report the Configure complaint to me.

If you are able to Configure and make lsof, run its test suite. (See the answer to the "Is there a test suite?" question for more information on how to use lsof's test suite.)

If lsof still fails, make sure your kernel sources, kernel header files, kernel boot file, standard header files and libraries are synchronized. They should all be built from the same CVS refresh. If they aren't, then the KVM library or lsof may be using kernel structure definitions that don't match the booted kernel.

If you have synchronized your kernel, header files and libraries, and still can't get lsof to work, follow the steps in the answer to the "How do I report an lsof bug" question to report the problem to me.

8.3 Why doesn't lsof work on the RELEASE version of CURRENT?

Lsof tracks the CURRENT release of the current leading edge FreeBSD version, because my access to leading edge FreeBSD is limited to FreeBDSD.org reference systems, all running the CURRENT release.

Sometimes that tracking leads to changes in lsof that won't work on an earlier RELEASE version of the current leading edge version.

When that happens, please send e-mail to me <abe@purdue.edu>. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

9.0 HP-UX Problems

9.1 What do /dev/kmem-based and PSTAT-based mean?

Lsof for HP-UX 11.0 and below uses /dev/kmem to read kernel data structures from which it gathers and reports open file information. That version of lsof is called /dev/kmem-based lsof.

Starting with HP-UX 10.10, finding definitions for the necessary kernel structures became more difficult as HP no longer distributed header files in /usr/include that defined all kernel structures. So I started "inventing" structure definitions by using Q4 to display them.

By HP-UX 11, the process of invention became extremely intensive to support. Following a patch to the ipc_s structure in early 1999, my invented definition of that structure became incorrect. Although I was able to devise a work-around test for the patch with Q4, it was clear that my inventions were bound to cause more problems.

Discussion with HP about the patch led to my proposing that an lsof API in the HP-UX kernel was the proper solution. Much to my surprise, HP agreed. I believe Carl Davidson was the prime mover behind that decision, but I know others participated, among them Louis Huemiller, Rich Rauenzahn, and Sailu Yallapragada. I am indebted to these folks and HP for their willingness to do this work.

The API was added to the PSTAT interface in a project named PEGL, Pstat Enhancements for Glance and Lsof. Louis and Sailu did the bulk of the design and implementation work and testing began in March, 2000

HP-UX 11.11 is the first version that provides PSTAT support for lsof. HP-UX versions in between 11.0 and 11.11 -- all Beta versions as far as I can determine -- have no lsof support.

See the "PSTAT-based HP-UX lsof Questions" section for questions and answers specific to PSTAT-based HP-UX lsof. The next section, "Why doesn't a /dev/kmem-based HP-UX lsof compilation use -O?" covers /dev/kmem-based HP-UX lsof.

9.2   /dev/kmem-based HP-UX lsof Questions

The sources for /dev/kmem-based lsof for HP-UX may be found in lsof_<revision>/dialects/hpux/kmem.

Lsof's Configure shell script decides to use these sources when it finds that the /usr/include/sys/pstat subdirectory doesn't exist.

Lsof can be forced to use the /dev/kmem sources by setting "/dev/kmem" in the HPUX_BASE environment variable. Consult the Configure shell script and 00XPORTING for more information.

9.2.1 Why doesn't a /dev/kmem-based HP-UX lsof compilation use -O?

If you only have the standard (bundled) HP-UX C compiler and haven't purchased and installed the optional one, then you can't use cc's -O option. The HP-UX cc(1) man page says this:

```
  "Options
     Note that in the following list, the cc and c89 options
     -A , -G , -g , -O , -p , -v , -y , +z , and +Z are
     not supported by the C compiler provided as part of
     the standard HP-UX operating system.  They are supported
     by the C compiler sold as an optional separate product."
```

Lsof's Configure script tries to detect what C compiler product you have installed by examining your compiler. If that examination reveals a standard (bundled) compiler, lsof avoids using -O.

If the Configure compiler test fails, the C compiler will complain that it doesn't support -O. You can suppress that complaint with this make invocation:

```
    $ make DEBUG=""
```

9.2.2 Why doesn't the /dev/kmem-based CCITT support work under 10.x?

Pasi Kaara, who originally provided the HP-UX CCITT support, reports that it no longer works under HP-UX 10.x. Consequently, at lsof revision 4.02 it has been disabled.

9.2.3 Why can't /dev/kmem-based lsof be compiled with `cc -Aa` or `gcc -ansi` under HP-UX 10.x?

Some HP-UX 10.x header files, needed by lsof, can't be compiled properly in ANSI_C mode; structure element definition and alignment problems result. The f_offset member of the file structure, for example, is incorrect.

This ANSI-C obstacle extends to using the -Aa option of the HP C compiler and the -ansi option of gcc.

9.2.4 Why does /dev/kmem-based lsof complain about no C compiler?

Lsof's Configure script looks in /bin and /usr/ccs/bin for an HP C compiler, because it needs to know if the compiler is the standard (bundled) one or the optional separate product. If it finds no compiler in either place, Configure quits after complaining:

```
    No executable cc in /bin or /usr/ccs/bin
```

If you don't have a C compiler in either of these standard places, you should consider installing it. If you have gcc installed, you can use it by declaring the ``hpuxgcc'' abbreviation to lsof's Configure script.

If you have a C compiler in a non-standard location, you can use the HPUX_CCDIR[12] environment variables to name the path to it. Consult the 00XCONFIG file of the lsof distribution for more information.

9.2.5 Why does Configure complain about q4 for /dev/kmem-based lsof for HP-UX 11?

When you run Configure on an HP-UX 11 system, it may complain:

```
  !!!ERROR!!!     !!!ERROR!!!     !!!ERROR!!!     !!!ERROR!!!
  Configure can't use /usr/contrib/bin/q4 to examine the ipis_s
  structure.  You must do that yourself, report the result in
  the HPUX_IPC_S_PATCH environment variable, then repeat the
  Configure step.  Consult the Configure script's use of
  /usr/contrib/bin/q4 and the 00XCONFIG file for information
  on ipis_s testing and the setting of HPUX_IPC_S_PATCH.
  !!!ERROR!!!     !!!ERROR!!!     !!!ERROR!!!     !!!ERROR!!!
```

This message states that Configure cannot use q4 from /usr/contrib/bin to examine the kernel's boot image for the ipis_s structure. Maybe q4 hasn't been installed, or perhaps Configure can't execute it.

Lsof needs to gather information about ipis_s to determine if the ipis_s structure is defined in the kernel boot image, if the ipis_s structure of the kernel boot image has an ipis_msgsqueued member, and if the ipc_s structure of the kernel boot image uses has an ipc_ipis member.

The ipis_s structure isn't described in any header file HP-UX releases with HP-UX 11. It appears in the private lsof header file .../dialects/hpux/kmem/hpux11/ipc_s.h. Lsof gets local and remote connection addresses (IP and port numbers) from ipc_s, so an incorrect ipc_s definition may cause incorrect reporting of TCP/IP connection addresses. It definitely will cause incorrect reporting on 32 bit kernels. In any case lsof should be compiled with a correct ipc_s definition no matter the kernel bit size, so the Configure script always tests for it when the HP-UX version is 11.

For lsof's Configure script to gather the necessary ipis_s information q4 needs to be installed in /usr/contrib/bin and the kernel boot image, /stand/vmunix, needs to have been processed with pxdb. If either is untrue, lsof issues the above error message, perhaps preceded by q4 messages.

```
        (Note: lsof's use of q4 may also fail if q4 can't execute
        nm -- e.g., it can't find /usr/bin/nm, or there is a
        conflicting, private version of nm earlier in the path.)

        If /stand/vmunix hasn't been processed by pxdb, the q4
        messages will include:

            q4: (error) vmunix not pxdb'd
        or
            q4: (warning) /stand/vmunix has not been processed by pxdb.

        It's possible to make a suitable private copy of /stand/vmunix
        for configuring lsof.  That requires /opt/langtools/bin/pxdb
        or the q4 version of pxdb from /usr/contrib/bin/q4pxdb.
        The path to the result is supplied to the lsof Configure
        script in the HPUX_BOOTFILE environment variable.  Configure
        still requires /usr/contrib/bin/q4.

        The following sample Bourne shell commands make a private
        copy of /stand/vmunix in /tmp, process it with pxdb or
        q4pxdb, and supply its path to lsof's Configure script in
        HPUX_BOOTFILE.

            $ cp /stand/vmunix /tmp/vmunix.lsof

            $ /opt/langtools/bin/pxdb /tmp/vmunix.lsof
          or
            $ /usr/contrib/bin/q4pxdb /tmp/vmunix.lsof

            ... pxdb messages ...
            $ HPUX_BOOTFILE=/tmp/vmunix.lsof Configure -n hpux

        It may also be necessary to use q4 outside the lsof Configure
        script.  In that case q4 can be to determine the state of
        ipis_s and ipc_s with these q4 commands:

            $ /usr/contrib/bin/q4 /stand/vmunix
            ...
            q4> fields -c struct ipc_s
            ...
            q4> fields -c struct ipis_s

        Look in the q4 output for the ipc_ipis member of the ipc_s
        structure, and look in the q4 output for the ipis_s structure
        for the ipis_msgsqueued member.  If ipc_s has ipc_ipis but
        ipis_s lacks ipis_msgsqueued, set HPUX_IPC_S_PATCH environment
        variable to "1".  If ipc_s has ipc_ipis and ipis_s has
        ipis_msgsqueued, set HPUX_IPC_S_PATCH to "2" -- e.g.,

            $ HPUX_IPC_S_PATCH=1 Configure -n hpux
          or
            $ HPUX_IPC_S_PATCH=2 Configure -n hpux

        If ipc_s has no ipc_ipis member, set HPUX_IPC_S_PATCH to
        "N" -- e.g., use this Configure step:

            $ HPUX_IPC_S_PATCH=N Configure -n hpux

  9.2.6 When compiling /dev/kmem-based lsof for HP-UX 11 what do the
        "aCC runtime: ERROR..." messages mean?

        When the lsof Makefile asks the HP-UX unbundled compiler
        to load lsof, it may complain:
```

```
    /bin/cc -o lsof  -DHPUXV=1100 -DHASVXFS -DHPUXKERNBITS=64 \
      -I/home/abe/src/lsof4/dialects/hpux/kmem/hpux11 +DD64 \
      -DHAS_IPC_S_PATCH=2 -I/home/abe/src/lsof4/dialects/hpux/kmem \
      -DLSOF_VSTR=\"B.11.00\"  -g dfile.o dmnt.o dnode.o dnode1.o \
      dnode2.o dproc.o dsock.o  dstore.o  arg.o main.o misc.o \
      node.o print.o proc.o store.o usage.o -L./lib -llsof  -lelf \
      -lnsl
    aCC runtime: ERROR: Unexpected use of shared libraries
    aCC runtime: ERROR: Read aCC manpage, +A option
    /usr/lib/nls/loc/locales.1//is_IS.iso88591
```

This is a bug in the HP-UX national language support. (Notice the last message with "locales" in it?) Complain to HP -- then use this work-around before executing make:

```
    $ unset LANG
    $ make
```

9.2.7 Why doesn't /dev/kmem-based lsof for HP-UX 11 report VxFS file link counts, node numbers, and sizes correctly?

This is usually the result of running an lsof binary whose revision number is less than 4.57 on a system that has OnlineJFS support installed. It can also happen with lsof 4.57 binaries when the OnlineJFS support with which they were built doesn't match the OnlineJFS status of the system on which they are run.

The OnlineJFS status of lsof 4.57 and higher binaries can be determined by running:

```
    $ lsof -v 2>&1 | grep HASONLINEJFS
```

If that shell pipe produces output, lsof was compiled with OnlineJFS support enabled; no output, disabled.

If OnlineJFS is installed on an HP-UX 11 system the /sbin/fs/vxfs/subtype executable exists and outputs "vxfs3.3" when run.

The problem occurs because the optional OnlineJFS support installation doesn't update <sys/fs/vx_inode.h>. Consequently lsof can be compiled with an incorrect definition of the vx_inode structure and look for for link counts, node numbers, and sizes in the wrong places in the structure.

The current response I have gotten from HP is that no <sys/fs/vx_inode.h> update will be provided for OnlineJFS.

I've addressed this problem temporarily with a work-around (hack) in lsof revision 4.57.

9.2.8 Why can't /dev/kmem-based lsof be built with gcc for 64 bit HP-UX 11?

When Configure is given the "hpuxgcc" abbreviation, the HP-UX version is 11, and the kernel bit size is 64, the lsof Configure script may abort with the messages:

```
    !!!!!!!!!!!!!!!!! FATAL ERROR !!!!!!!!!!!!!!!!!!

    APPARENTLY GCC CANNOT BUILD 64 BIT EXECUTABLES.
    A COMPILER MUST BE USED THAT CAN.  SEE 00FAQ
    FOR MORE INFORMATION.
```

(This is the "more information" in 00FAQ.)

This means the Configure script compiled a test program with gcc the result wasn't an ELF-64 binary. Lsof tries two gcc modes, one with no options and another with the -mlp64 option, before it concludes gcc can't be used.

See the "How can I acquire a gcc for building lsof for 64 bit HP-UX 11?" answer for information on where you might be able to get a gcc for HP-UX 11 that can produce ELF-64 executables.

9.2.8.1   How can I acquire a gcc for building lsof for 64 bit HP-UX 11?

Check this HP URL:

http://h21007.www2.hp.com/dspp/tech/tech_TechSoftwareDetailPage_IDX/1,1703,547,0
0.html

(That's one very long link; be careful you cut 'n paste it all.)

In November 2001 that URL led to a web page whose title was "gcc for hp-ux 11." The page offered a link for downloading a 64 bit gcc 3.0 compiler for HP-UX 11.0 and 11i. Rich Rauenzahn of HP installed that compiler on an HP test system he allows me to use and I successfully built a 64 bit lsof with it.

The HP package may install the 64 bit capable gcc in /usr/local/pa20_64/bin/gcc, so you may have to adjust your path or set the LSOF_CC environment variable to compensate.

9.2.9   Why does /dev/kmem-based lsof for HP-UX 11 report "unknown file system type" for some open files?

The lsof binary being used probably doesn't have support for the VxFS file system.

To confirm that, check `lsof -v` output for "-DHASVXFS". If it's not present, lsof doesn't have VxFS support.

You also need to establish that lsof really is complaining about VxFS files by checking the kernel boot file for the symbol associated with the hexadecimal address reported in the "unknown file system type" message -- e.g., "v_op: 0x8711c8." Use nm(1) to do that:

```
    $ nm -x /stand/vmunix | grep 8711c8
```

If nm reports the symbol associated with the address is vx_vnodeops, then lsof is complaining about an open VxFS file.

The solution in that case is to build lsof yourself (The bundled C compiler will do it.), making sure that lsof's Configure script detects the presence of VxFS. Configure does that by finding these two header files:

```
    /usr/include/sys/fs/vx_hpux.h
    /usr/include/sys/fs/vx_inode.h
```

If the system where you are building lsof doesn't have those header files, but does have VxFS, you might be able to install the header files by installing the HP JournalFS package from the CoreOS CD -- in particular the file set JournalFS.VXFS-PRG and its associated patch, PHKL_18543. (My thanks to Steve Bonds for that information.)

Finally, if you find that lsof isn't complaining about VxFS when it complains about an unknown file system type, send e-mail to me <abe@purdue.edu> for further assistance. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

9.2.10 Why does the ANSI-C compiler complain about comments in HP-UX 11 header files?

When compiling lsof on HP-UX 11, the HP ANSI-C compiler's pre-processor, cpp, may complain about comments in HP-UX header files -- e.g.,

```
    cpp: "/usr/include/sys/cdfs.h", line 232: warning 2028:
      Found comment inside comment started on line 232.
    cpp: "/usr/include/sys/cdnode.h", line 196: warning 2028:
      Found comment inside comment started on line 196.
    cpp: "/usr/include/nfs/snode.h", line 30: warning 2028:
      Found comment inside comment started on line 30
```

This is not a problem with lsof. It is a problem with the HP-UX header files; they have non-compliant ANSI-C comment sequences in them -- e.g.,

```
    <sys/cdfs.h>: 232
      /* struct  cdfs *cdfs_link;  /* linked list of file systems */
```

The initial "/*" is not terminated by an ending "*/" before the appearance of a second "/*".

9.3 PSTAT-based HP-UX lsof Questions

The sources for PSTAT-based lsof for HP-UX may be found in lsof_<revision>/dialects/hpux/pstat.

Lsof's Configure shell script decides to use these sources when it finds that the /usr/include/sys/pstat subdirectory exists.

Lsof can be forced to use the PSTAT-based sources by setting "pstat" in the HPUX_BASE environment variable. Consult the Configure shell script and 00XPORTING for more information.

9.3.1 Why does PSTAT-based lsof complain about pst_static and other PSTAT structures?

When lsof starts it may issue one of these fatal error messages:

```
    lsof: FATAL: can't determine PSTAT static size
    lsof: FATAL: can't read <n> bytes of pst_static
    lsof: FATAL: pst_static doesn't contain <name>_size
    lsof: FATAL: <name>_size should be <n>
```

These messages indicate that lsof's tests for the proper level of PSTAT support have failed. The structure names, given in <name>, and sizes, given in <n>, identify the support deficiency more precisely.

You may need to upgrade the PSTAT support in your kernel
to be able to use PSTAT-based lsof.

9.3.2 Why does PSTAT-based lsof complain it can't read pst_*
structures?

Lsof may put messages like the following in the NAME
column of its output.

```
    can't read cwd pst_filedetails: Permission denied
    can't read mem pst_filedetails: Permission denied
    can't read rtd pst_filedetails: Permission denied
    can't read txt pst_filedetails: Permission denied
    can't read pst_filedetails: Permission denied
    can't read 3 stream structures: Permission denied
    can't read pst_socket: Permission denied
```

These messages indicate that the lsof binary lacks the
authority to read the name structures for processes other
than ones belonging to the UID under which lsof is running.
Authority to read the structures of other processes is
limited to root processes -- i.e., lsof must have setuid-root
permission if it is to list open files for arbitrary
processes.

If you want to eliminate these errors, you must run lsof
as root or install it with setuid-root permission.

9.3.3 Why does PSTAT-based lsof rebuild the device cache file
after each reboot?

After each HP-UX rebuild, the first time a user runs lsof it
will report:

```
    lsof: WARNING: device cache mismatch: /dev/tun...
    lsof: WARNING: created device cache file: /<user_path>
```

This happens because the device numbers on /dev/tun* device
nodes are recalculated at each reboot. When lsof detects
a change in the device number of a /dev/tun* file, it rebuilds
its local device cache file.

9.3.4 Why doesn't PSTAT-based lsof report TCP addresses for
telnetd's open socket files?

When lsof can't report TCP addresses for telnetd's open
socket files it is because an unpatched PSTAT kernel
interface doesn't report the addresses to lsof.

This has been addressed in PSTAT kernel patch PHKL_24047.
It is available from the HP IT Resource Center at:

    http://itrc.hp.com

In the page's "maintenance / support" box select the
"individual patches" link. Once at its page, select the
"hp-ux" link. On that page select the "Series 800" or
"Series 700" radio button and select "11.11" from the
pull-down list to the right of the button. Under "search
or browse the path list" select "Search by Patch IDs" from
the pull down list, enter PHKL_24047 in the following text
box, and select search. That should lead to information

about PHKL_24047 and a link for downloading it. (You may
have to log in first and you may have to create a login
identity by registering before you can log in.)

9.3.5 Why does PSTAT-based lsof cause an HP-UX 11.11 kernel panic?

When PSTAT-based lsof runs on some HP-UX 11.11 kernels,
the kernel may panic. Symptoms include:

```
  Console message:
    0xFBE000301100EF00 00000000 0000EF00 -
    type 31 = legacy PA HEX chassis-code

  /var/adm/syslog:
    ... vmunix: Trap Type 15 (Data page fault)
    ... vmunix:   Instruction Address (pcsq.pcoq) = 0x...
```

The panic is caused by a bug in the way PSTAT's pstat_getstream()
function obtains module names from streams managed by the
otsam stream driver (part of OSI Transport Services). Lsof
calls pstat_getstream() when it encounters an open otsam
stream file. An HP-UX 11.11 system uses otsam if otsam
appears in /stand/system.

HP-UX 11.11 patch PHKL_24507 (available some time after
July 15, 2001) fixes the pstat_getstream() bug. See the
information in the answer to the "Why doesn't PSTAT-based
lsof report TCP addresses for telnetd's open socket files?"
question for information on how to obtain the patch.

10.0 Linux

10.1 What do /dev/kmem-based and /proc-based lsof mean?

At approximately Linux 2.1.72 and exactly at lsof revision
4.23 support for Linux forks. The first fork, containing
the oldest lsof form is based on access to kernel memory
structures, and is called /dev/kmem-based lsof. A
/dev/kmem-based lsof is heavily intertwined with the Linux
kernel version, its header files, and its system map file.
Typically a /dev/kmem-based lsof needs only setgid permission
to local all open file information.

After approximately Linux 2.1.72 and at revision 4.23 lsof
obtains all its information from the /proc file system.
That lsof is called the /proc-based lsof. A /proc-based
lsof does not read kernel memory, needs neither kernel
header files nor the system map file, and is less likely
to be affected by Linux kernel changes. However, it does
require setuid-root permission to list all open files, and
it can't report file offsets (positions).

After revision 4.52 the /dev/kmem-based Linux sources for
lsof are no longer distributed. Information about them
may be found in the 00INDEX and README files at:

    ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/OLD/src

10.2 /proc-based Linux lsof Questions

10.2.1   Why doesn't /proc-based lsof report file offsets (positions)?

/proc-based lsof can't report file offsets (positions) when

no offset information is available in the /proc/<PID>/fd/
links that describe the files open to a process. During
its initialization /proc-based lsof tests to see if offset
information might be present in the st_size element of the
stat structure returned by the lstat(2) kernel function,
when applied to one of its own open files.

To see if /proc-based lsof thinks your kernel reports
reliable offset information, specify the -o option to it.
If it replies with:

```
    lsof: WARNING: can't report offset; disregarding -o.
```

then its initialization test has indicated that using
lstat(2) on one of its own open files in /proc/<PID>/fd
doesn't deliver offset information. (The /proc-based lsof
offset test may be found in the .../dialects/linux/proc/dproc.c
initialize() function.)

Contact me via e-mail at <abe@purdue.edu> for information on a
possible kernel patch that allows lstat(/proc/<PID>/fd/<FD>) to
deliver offset (position) information. Make sure "lsof"
appears in the "Subject:" line so my e-mail filter won't
classify your letter as Spam.

10.2.2 Why does /proc-based lsof report "can't identify protocol" for
some socket files?

/proc-based lsof may report:

```
    COMMAND PID ... TYPE ... NODE NAME
    pump    226 ... sock ...  309 can't identify protocol
```

This means that it can't identify the protocol (i.e., the
AF_* designation) being used by the open socket file. Lsof
identifies protocols by matching the node number associated
with the /proc/<PID>/fd entry to the node numbers found in
selected files of the /proc/net sub-directory. Currently
/proc-based lsof examines these protocol files:

```
    /proc/net/ax25            (untested)
    /proc/net/ipx       (needs kernel patch)
    /proc/net/raw
    /proc/net/raw6
    /proc/net/tcp
    /proc/net/tcp6
    /proc/net/udp
    /proc/net/udp6
    /proc/net/unix
```

If /proc-based lsof says it can't identify the protocol
for an open socket file, you may be able to identify the
protocol yourself by using grep to look for the specific
node number in the files of /proc/net -- e.g.,

```
    $ grep <node_number> /proc/net/*
```

You may not be able to find the desired node number, because
not all kernel protocol modules fully support /proc/net
information.

If you find a matching node number in a /proc/net file that is
not currently being processed by lsof, contact me via e-mail at
<abe@purdue.edu>. I'll discuss adding support to /proc-based

lsof for the protocol of the /proc/net file with you. Make
sure "lsof" appears in the "Subject:" line so my e-mail filter
won't classify your letter as Spam.

The code that matches node numbers of open IPX protocol
socket files to those in /proc/net/ipx requires Jonathan
Sergent's Linux 2.1.79 patch to /usr/src/linux/net/ipx/af_ipx.c.
The patch, suitable for input to Larry Wall's patch program,
may be found in the lsof distribution file:

```
    .../dialects/linux/proc/patches/net_ipx_af_ipx.c.patch
```

10.2.3 Why does /proc-based lsof warn about unsupported formats?

Lsof may issue the following warning:

```
    lsof: WARNING: unsupported format: /proc/net/<file>
```

if the header line of the indicated <file> in /proc/net --
ax25, ipx, raw, tcp, udp, or unix -- doesn't match what
lsof expects to find.

When the header line of a /proc/net file isn't what lsof
expects, lsof probably can't parse the rest of the file
correctly and doesn't try. As a result, lsof can't report
any NAME column information (e.g., local and remote addresses)
for socket files bound to the indicated network protocol.

If you get this warning, please send me e-mail at
<abe@purdue.edu>. Include the contents of the file lsof claims
has an unsupported format. Make sure "lsof" appears in the
"Subject:" line so my e-mail filter won't classify your letter
as Spam.

10.2.4 Why does /proc-based lsof report "(deleted)" after a path name?

The "(deleted)" notation following a path name in /proc-based
lsof's NAME column comes from the /proc/<PID>/fd/<FD> entry
for the open file. It's the Linux kernel's way of indicating
the file is open but has been unlinked (rm'd).

10.2.5 Why doesn't /proc-based lsof report full open file information for all processes?

/proc-based lsof can only report on processes whose /proc
files it has permission to read. /proc normally grants
permission to read all its files only to root or to the
owning user ID.

Without permission to read most /proc files, lsof can only
report full information for processes belonging to the user
who is running lsof. /proc-based lsof may be able to report
some information for all processes, depending on the
permissions of their associated /proc files, but usually
/proc-based lsof won't be able to access the files in
/proc/<PID>/fd/ that describe regular open files.

If you want /proc-based lsof to report on all processes, you
must install it with setuid-root permission.

10.2.6 Why won't Customize offer to change HASDCACHE or WARNDEVACCESS for /proc-based lsof?

/proc-based lsof doesn't read device information from /dev

or the device cache file, so it makes no sense to change
the state of device cache processing or /dev node accessibility
warnings.

10.2.7 /proc-based lsof Linux NFS questions

10.2.7.1 Why can't lsof find files on an accessible NFS file system?

On occasion lsof may be unable to identify that an open
file is on an NFS file system. This is most likely the
result of a bug in the way the Linux kernel supplies
information to the reader of /proc/mounts (lsof) -- sometimes
that pseudo-file is truncated by the kernel.

One way to see if this is the case is to search for the
NFS file system in /proc/mounts -- e.g.,

```
    $ grep <NFS_file_system_mount_point> /proc/mounts
```

If you get no output or the third word of the output isn't
"nfs", then lsof won't consider the file system an NFS file
system.

A second test is to look at the end of /proc/mounts --
e.g.,

```
    $ tail /proc/mounts
```

If tail reports "# truncated" then /proc/mounts is incomplete
because of a Linux kernel bug. The bug is documented at:

    http://www.xss.co.at/sysinfo/mounts.html

The bug is fixed in Linux kernel 2.4.18, and possibly in
some earlier Linux kernel versions.

10.2.7.2 Why can't lsof find files on an inaccessible NFS file system?

If lsof issues this message about a Linux file system,
mounted from an NFS server:

```
    lsof: WARNING: can't stat() nfs file system /xxx/yyy
```

Then lsof won't be able to find any open files on the file
system.

That's because of an inadequacy in the Linux /proc file
system. Its /proc/mounts file doesn't give the device
doublet (major and minor numbers) of the file system as do
many UNIX systems (e.g., Solaris). The only way lsof can
get the device doublet for a Linux file system is to call
stat(2) on the file system path, which fails if the NFS
server isn't accessible.

When lsof doesn't know the device doublet of a file system,
it can't find open files on the inaccessible file system,
because it can't match the doublets of open files to the
doublet of the inaccessible file system.

This topic is covered extensively in lsof(8) it its ALTERNATE
DEVICE NUMBERS and BLOCKS AND TIMEOUTS sections.

10.2.8 Why doesn't /proc-based Linux lsof report socket options and values, socket state flags, and TCP options and values?

The Linux /proc file system doesn't report socket options and values, socket states, and TCP options and values to lsof.

10.3 Special Linux file types

10.3.1 Why is ``DEL'' reported as a Linux file type?

Lsof usually reports entries from the Linux /proc/<PID>/maps file with ``mem'' in the TYPE column.  However, when lsof can't stat(2) a path in the process' ``maps'' file and the ``maps'' file entry contains ``(deleted)'', indicating the file was deleted after it had been opened, lsof reports the file type as ``DEL''.

10.3.2 Why is ``unknown'' reported as a Linux file type?

Lsof may report a Linux file's type as ``unknown'' in the TYPE column when lsof can't obtain complete stat(2) results for the file.

Usually the NAME column will contain a ``(stat: xxx)'' error message, but that could have been suppressed with the lsof ``-w'' option.

10.4 Linux ``mem'' Entry Problems

10.4.1 What do ``path dev=xxx'' and ``path inode=yyy'' mean in the NAME column of Linux ``mem'' file types?

When the device or inode number in the process' ``maps'' file entry doesn't match the stat(2) results from the file path, lsof reports the inconsistent information from the stat(2) of the path parenthetically after the path in the NAME column in one of these forms:

```
    (path dev=xxx)              only the device number,
                          ``xxx'', from a stat(2) of the
                          ``maps'' file entry path
                          differs from the ``maps'' file
                          entry value reported in the
                          DEVICE column.

    (path inode=yyy)          only the inode number,
                          ``yyy'', from a stat(2) of the
                          ``maps'' file entry path
                          differs from the ``maps'' file
                          entry value reported in the
                          NODE column.

    (path dev=xxx inode=yyy)    Both device and inode numbers
                          differ.
```

Lsof reports the ``maps'' file device number in the DEVICE column and the inode number in the NODE column.

Device and inode inconsistencies can occur when a file at a ``maps'' path is replaced after the process has started, or when a different file system with similar path names is mounted on top of the original file system.

The device inconsistency parenthetical messages can be suppressed with lsof's ``-w'' option.

10.4.2   Why is no size reported for some Linux ``DEL'' and ``mem'' file types?

No size is reported for these entries from the process' ``maps'' file because a stat(2) of the entry file path failed.

10.5 Special Linux NAME column messages

10.5.1 What does ``(stat: xxx)'' mean in the NAME column of Linux files?

When lsof tried to stat(2) the path in the NAME column, the stat(2) system call failed and produced an error message of ``xxx''.

This situation usually occurs if the lsof process lacks permission to stat(2) the path -- e.g., the lsof executable lacks root permission, or lsof is attempting to stat(2) a path on an NFS device mounted with the root_squash option.

The message can be suppressed with lsof's ``-w'' option.

10.5.2 What does ``(readlink: xxx)'' mean in the NAME column of Linux files?

When lsof tried to convert the /proc/<PID>/fd path, reported in the NAME column, to its full and more meaningful path, the readlink(2) system call used to do the conversion failed. The readlink(2) failure message is ``xxx''.

This situation usually occurs if the lsof process lacks permission to readlink(2) some part of the path -- e.g., the lsof executable lacks root permission, or lsof is attempting to stat(2) a path on an NFS device mounted with the root_squash option.

The message can be suppressed with lsof's ``-w'' option.

10.6 Why is ``NOFD'' reported as a Linux file type?

When lsof lacks permission to use opendir() on the fd/ subdirectory of a process' /proc/<PID> directory, it reports a single file of the type ``NOFD'' (for no file descriptors).

Lsof reports the the /proc/<PID>/path in the NAME column, followed by "(opendir: xxx)", where ``xxx'' is the error message returned by opendir().

The ``NOFD'' entry can be suppressed with lsof's ``-w'' option.

10.7   Why does Linux lsof report a NAME column value that begins with ``/proc''?

When lsof has problems processing a ``/proc/<PID>'' entry -- e.g., it can't convert the entry to a full and more meaningful path name, or it can't access the /proc/<PID>/fd subdirectory with opendir() -- it will report the /proc/<PID> path in the NAME column.

11.0 NetBSD Problems

11.1 Why doesn't lsof report on open kernfs files?

Lsof doesn't report on open NetBSD kernfs files because the structures lsof needs aren't defined in the kernfs.h header file in /sys/misc/kernfs.

11.2 Why doesn't lsof report on open files on: file descriptor file systems; /proc file systems; 9660 (CD-ROM) file systems; MS-DOS (floppy disk) file systems; or kernel file systems?

Lsof is not able to report on open files on certain file system if /usr/src/sys/msdosfs didn't exist when the lsof Configure script ran and lsof was made. /usr/src/sys/msdosfs contains header files lsof needs for collecting data on certain file system files.

You can tell if an lsof executable above) lacks support for a file system if the following test of `lsof -v` produces nothing:

```
    $ lsof -v 2>&1 | grep <support_enabled_definition>
```

The <support-enabled_definition> will be:

File System Type	Definition	Note
File descriptor	HASFDESCFS	
/proc	HASPROCFS	
9660	HAS9660FS	
MS-DOS	HASMSDOSFS	(lsof 4.61 and above)
Kernel	HASKERNFS	

The work-around is to install /usr/src/sys, rerun the lsof Configure script, and remake lsof.

11.3 Why does lsof produce confusing results for nullfs file systems?

Consider this report from /sbin/mount:

```
    /usr/home on /home type null (local)
```

(According to /sbin/mount /usr/home is the mounted-on device and /home is the mounted-on directory.)

When lsof is asked to report on open files on /home, it will report them as files on /usr/home instead. That's an artifact of the NetBSD kernel's dynamic name lookup cache (DNLC) and the way the kernel handles nullfs mounted-on directories.

While lsof will report all open files on /home when given /home as a file system directory argument, even though reporting them as located on /usr/home, lsof will not find the same files when asked to report on all open files on /usr/home when given /usr/home as a file system device argument. That's because from the mount perspective /usr/home is equivalent to a device, but from the device perspective it is still a directory.

So, what this lsof command reports:

```
    $ lsof /home
    ... NAME
    ... /usr/home/...
```

Won't be duplicated by this lsof command:

```
    $ lsof /usr/home
```

Another way to look at this confusing /home and /usr/home example is to consider what stat(2) reports. For /home stat(2) reports a device doublet that matches what lsof finds in open file node structures, while the device doublet stat(2) reports for /usr/home won't match what lsof finds. Nor does the mode reported by stat(2) indicate a block devices, as is the expected case.

There is no simple answer to this confusion, nor is there even a simple explanation. Simply be aware that when supplying file system arguments to lsof on NetBSD, use the mounted-on directory name for a nullfs as the lsof argument, and don't be surprised when the NAME column reports the mounted-on device name.

11.4 NetBSD header file problems

11.4.1   Why can't the compiler find some NetBSD header files?

If the compiler's pre-processor complains it can't find some header files when it compiles lsof source files, /usr/include may not have all the header files lsof needs. Some may be elsewhere -- most likely in /sys.

As a work-around use the NETBSD_SYS environment variable to specify to lsof the location of the additional header files -- e.g.,

```
    % setenv NETBSD_SYS /sys
    % ./Configure -n netbsd

 or
     $ NETBSD_SYS=/sys ./Configure -n netbsd
```

Caution: using this work-around may cause the lsof Configure script to activate or omit different features, depending on where it finds the header files that determine the state of the features.

11.4.2   Why does NetBSD lsof produce incorrect output?

If the NetBSD system's kernel was built from header files that don't match those in /usr/include -- e.g., /sys has the ones from which the kernel was built -- lsof may build, but won't produce correct output.

As a work-around, try using the NETBSD_SYS environment variable to request that lsof be compiled with header files in /sys in place of /usr/include, when ones in /sys are available.

Instructions for using that NETBSD_STS work-around may be found in the answer to the "Why can't the compiler find some NetBSD header files?" question.

11.5 Why isn't lsof feature xxx enabled for NetBSD?

Lsof's Configure script enables NetBSD features by locating and examining header files associated with the features,

and based on what it finds, setting compile-time definitions
in Makefiles. (See 00PORTING for a list of the definitions.)

When Configure doesn't find header files or doesn't find
appropriate values in header files, that may mean the header
file tree lsof is searching is incomplete or out of date.

Lsof normally looks for NetBSD header files in /usr/include.
It can also be directed to look in other directories --
e.g., /sys -- if told to do so with the contents of the
LSOF_INCLUDE and NETBSD_SYS environment variables.

To determine what header file enables a missing feature,
check the NetBSD stanza in the Configure script. Then
check the locations it checks for the indicated header
files and contents.

See 00XCONFIG for more information on LSOF_INCLUDE and
and NETBSD_SYS.

12.0 NEXTSTEP and OPENSTEP Problems

12.1 Why can't lsof report on 3.1 lockf() or fcntl(F_SETLK) locks?

Lsof has code to test for locks defined with lockf() or
fcntl(F_SETLK) under NEXTSTEP 3.1, but that code has never
been tested. I couldn't test it, because my NEXTSTEP 3.1
lockf() and fcntl(F_SETLK) functions return "Invalid
argument" every way I have tried to invoke them.

If your NEXTSTEP 3.1 system does allow you to use lockf()
and fcntl(F_SETLK) and lsof doesn't report locks set with
them, then the code in .../dialects/next/dnode.c probably
isn't correct. Please contact me via e-mail at <abe@purdue.edu>
and tell me how you got your lockf() and fcntl(F_SETLK) system
calls to work. Make sure "lsof" appears in the "Subject:" line
so my e-mail filter won't classify your letter as Spam.

12.2 Why doesn't lsof compile for NEXTSTEP with AFS?

I no longer have a NEXTSTEP test system that has AFS.
Changes to lsof since I once had a test system have caused
me to change the AFS code in NEXTSTEP without being able
to test the changes.

If you need AFS support for NEXTSTEP and can't get it to
compile, please contact me. Perhaps we can jointly fix
the problems.

13.0 OpenBSD Problems

13.1 Why doesn't lsof support kernfs on my OpenBSD system?

Lsof supports the kernel file system on OpenBSD versions
whose /sys/miscfs/kernfs/kernfs.h (or <miscfs/kernfs/kernfs.h>
header file correctly defines the kern_target structure.
The lsof Configure script's openbsd stanza checks for the
presence of the structure's kt_name element and activates
kernfs support for the CFLAGS -DHASKERNFS definition only
when it finds kt_name.

The kernfs.h header file is scheduled to be updated in the OpenBSD 2.1 release, according to Kenneth Stailey, who authored its changes.

13.2 Will lsof work on OpenBSD on non-x86-based architectures?

I've not tested lsof on an OpenBSD system that uses a non-x86-based architecture, but I've had one report that lsof 4.33 compiles and works on OpenBSD for the pmax architecture (decstation 3100).

13.3 <sys/pipe.h> problems

13.3.1   Why does the compiler claim nbpg isn't defined?

When compiling lsof on some (older) OpenBSD SPARC versions, the compiler may complain:

```
    In file included from ../dlsof.h:191,
         from ../lsof.h:166,
         from fino.c:52:
    /usr/include/sys/pipe.h:83: `nbpg' undeclared here
                        (not in a function)
    /usr/include/sys/pipe.h:83: size of array `ms' has
                        non-integer type
```

This happens because <sys/pipe.h> uses NBPG from <machine/param.h> to size the `ms' array, and some OpenBSD systems define NBPG in terms of a kernel integer variable, nbpg.

Lsof revisions 4.46 and above have a hack to dlsof.h, developed by Volker Borchert that avoids the compiler problem for SPARC OpenBSD 2.3. The hack might work for other OpenBSD SPARC versions, but hasn't been tested there.

If you want to enable the hack for your OpenBSD SPARC version, modify this code in .../dialects/n+obsd/dlsof.h:

```
    # if    defined(OPENBSDV)
    #  if   OPENBSDV==2030 && defined(__sparc__)
    #   if  defined(nbpg)
    #undef  nbpg
    #   endif       /* defined(nbpg) */
    #define nbpg    4096            /* WARNING!!!  ... */
    #  endif        /* OPENBSDV==2030 && defined(__sparc__) */
    #include <sys/pipe.h>
    #endif  /* defined(OPENBSDV) */
```

You will probably want to change the second #if test to match your OpenBSD version. You may also want to change what value is assigned to nbpg. See the next section, "What value should I assign to nbpg?"

13.3.2   What value should I assign to nbpg?

If you need to enable the nbpg hack, described in "Why does the compiler claim nbpg isn't defined?", you may also need to assign a value other than 4096 to nbpg. 4096 works for the sun4c processor and should work for sun4m, but 8192 may be needed for sun4.

Check <machine/param.h> and other OpenBSD documentation to determine the correct nbpg assignment.

13.4 Why doesn't lsof report on open MS-DOS file system (floppy disk) files?

Lsof is not able to report on open MS-DOS file system files if /usr/src/sys/msdosfs didn't exist when the lsof Configure script ran and lsof was made. /usr/src/sys/msdosfs contains header files lsof needs for collecting data on MS-DOS file system files.

You can tell if an lsof executable (revisions 4.61 and above) lacks MS-DOS file system support if the following command reports nothing:

```
    $ lsof -v 2>&1 | grep HASMSDOSFS
```

The work-around is to install /usr/src/sys, rerun the lsof Configure script, and remake lsof.

13.5 Why isn't lsof feature xxx enabled for OpenBSD?

Lsof's Configure script enables OpenBSD features by locating and examining header files associated with the features, and based on what if finds, setting compile-time definitions in Makefiles. (See 00PORTING for a list of the definitions.)

When Configure doesn't find header files or doesn't find appropriate values in header files, that may mean the header file tree lsof is searching is incomplete or out of date.

Lsof normally looks for OpenBSD header files in /usr/include and /sys. It can also be directed to look in other directories if told to do so with the contents of the LSOF_INCLUDE and NETBSD_SYS environment variables.

To determine what header file enables a missing feature, check the OpenBSD stanza in the Configure script. Then check the locations it checks for the indicated header files and contents.

See 00XCONFIG for more information on LSOF_INCLUDE and and NETBSD_SYS.

14.0 Output Problems

14.1 Why do the lsof column sizes change?

Lsof dynamically sizes its output columns each time it runs to make sure that each column takes the minimum space. Column parsing -- e.g., with awk -- is possible, because each column is guaranteed to be separated from the preceding one by at lease one space, and no column except the last (NAME) contains embedded spaces.

14.2 Why does the offset have ``0t' and ``0x'' prefixes?

The offset value that appears in the SIZE/OFF column has ``0t' and ``0x'' prefixes to distinguish it from size values that may appear in the same column.

Normally if the offset value is less than 100,000,000 (8 digits), it appears in decimal with a ``0t' prefix; over 99,999,999, in hexadecimal with a ``0x'' prefix.

A decimal offset is handy, for example, when tracking the
progress of an outbound ftp transfer. When lsof reports
on the ftp process, it will report the size of the file
being sent with its open descriptor; it will report the
progress of the transfer via the offset of the outbound
open ftp data socket descriptor.

The ``-o [n]'' option may be used to specify the maximum
number of decimal digits to be printed after ``0t'' before
lsof switches to the hexadecimal digits after `0x''. As
already noted, the default decimal digit count is 8.

14.3 What are the values printed in the FILE_FLAG column and why is 0x<value> sometimes included?

The two comma separated lists, separated by a semicolon,
printed in the FILE-FLAG column (when the "+fg" option is
specified), are short-hand names or hexadecimal values for
the bits lsof finds in the f_flag or f_flags member of file
structures for files (the first list, the one before the
semicolon), and process open files flags found in various
kernel structures, often named "pofile" (the second list,
the one after the semicolon).

Lsof determines the short-hand names from symbols in the
<fcntl.h>, <linux/fs.h>, <sys/fcntl.h>, <sys/fcntlcom.h>,
o<sys/file.h>, and <sys/user.h> header files.

See the discussion of FILE-FLAG in the OUTPUT section of
the lsof man page, and the FF_* and POF_* symbols in lsof.h
for a list of the names.

Bits with no names defined for them are represented by an
0x<value> member of the comma-separated list -- a hexadecimal
integer. When "+fG" is specified (instead of "+fg"), lsof
will list all flag values as two hexadecimal integers,
separated by a semicolon.

When "-FG" is specified to get the flags in an output field,
the format defaults to hexadecimal. You can get names
instead by following "-FG" with "+fg" -- e.g.,

```
    $ lsof -FG +fg ...
```

However, when you precede "-FG" with "+fg" -- e.g.,

```
    $ lsof +fg -FG
```

the format will be hexadecimal; order is important.

14.3.1 Why doesn't lsof display FILE_FLAG values for my dialect?

All versions of lsof except the /proc-based Linux lsof
report FILE-FLAG values. Lsof can't obtain FILE-FLAG
information from the Linux /proc interface.

14.4 Network Addresses

14.4.1 Why does lsof's -n option cause IPv4 addresses, mapped to IPv6, to be displayed in IPv6 notation?

When you use the -n option to tell lsof to display numeric
network addresses, and an IPv4 address has been mapped to

IPv6, lsof displays the address in IPv6 format and puts "ipv4" in the TYPE column. That combination indicates the IPv4 address has been mapped to IPv6.

For example, the IPv4 address 1.2.3.4, when mapped to an IPv6 address, will be displayed by lsof as:

```
[::ffff:1.2.3.4]
```

The enclosing brackets are lsof's signal that this is an IPv6 address. Inside the brackets is a standard IPv6 address, reported by inet_ntop(). The first two colons, signifying zeroes in the first 64 bits of the IPv6 address, and the hexadecimal ffff in the next 32 bits, indicate that the last 32 bits contains a mapped IPv4 address, which is then displayed in IPv4 dot notation.

14.5 Why does lsof output \x, ^x, or \xnn for characters sometimes?

Lsof displays only printable ASCII characters. Lsof considers a character printable if isprint(3) says it is. If isprint(3) says a character isn't printable, the lsof may page explains:

"... Non-printable characters are printed in one of three forms: the C ``\[bfrnt]'' form; the control character `^' form (e.g., ``^@''); or hexadecimal leading ``\x'' form (e.g., ``\xab''). Space is non-printable in the COMMAND column (``\x20'') and printable elsewhere."

14.5.1 Why is space considered a non-printable character in command names?

Space is considered an unprintable character in command names because it is sometimes possible to hide the full command name from scripts that parse ps(1) output by embedding a space in the name.

14.6 Why doesn't lsof print all the characters of a command name?

By default lsof prints the first nine characters of the names of commands associated with processes. If more characters are required, the "w" value of the "+c w" option may be used to specify a larger width.

If "w" is zero ('0') lsof will print all characters of all command names, replacing non-printable characters as discussed in the answer to " Why does lsof output \x, ^x, or \xnn for characters sometimes?"

See the answer to the "Why is space considered a non-printable character in command names?" question for an explanation of why spaces are replaced by the ``\x20'' representation in command names.

15.0 Pyramid Version Problems

15.0.5   Statement of deprecation

As of lsof revision 4.52 support for all Pyramid versions has been dropped. Contact me via e-mail if you're interested in obtaining the last lsof Pyramid distribution. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

15.1 DC/OSx Problems

15.2 Reliant UNIX Problems

15.2.1   Why does lsof complain that it can't find /stand/unix?

When you attempt to run lsof on a Reliant UNIX multi-processor, it may complain that it can't find the kernel boot file, /stand/unix. That's because normally the /stand/unix file is only located on one node's root file system. Lsof needs the file to obtain kernel data addresses.

The work-around is to copy /stand/unix to each node.

15.2.2   Why does lsof complain about bad kernel addresses?

Lsof may complain that some Reliant UNIX kernel addresses aren't usable -- e.g., it may issue a warning like this:

```
    lsof: WARNING: can't read kernel's name cache: 0x00000000
```

This is usually the result of having a /stand/unix file on a Reliant multi-processor that isn't the booted kernel file. Because it doesn't have symbol addresses that match those of the running kernel, lsof has problems reading kernel values.

One work-around is to copy the correct boot file to /stand/unix. If the booted kernel file is available under another name -- e.g., /stand/unix.myboot -- another work-around is to use lsof's -k flag to specify the alternate name as the source of kernel name list values:

```
    $ lsof -k /stand/unix.myboot ...
```

15.2.3   Why does the Reliant C compiler give so many warning messages when compiling lsof?

The Reliant Unix Pyramid C compiler issues warning messages that I haven't found a convenient way to suppress. You can ignore warning messages about casts and conversions that lose bits. The message "warning: undefining __STDC__" is intentionally caused by the lsof MkKernOpts configuration script to suppress warning messages about cast and conversion problems in standard system header files, such as <stdio.h> and <string.h>.

15.2.4   Why does the lsof compilation require -Klp64 for Reliant UNIX 5.44 and why does my compiler reject it?

The -Klp64 flag enables the 64 bit data model lsof requires for handling Reliant Unix 5.44 and above 64 bit kernel pointers. Some compilers don't support -Klp64.

If lsof's Configure script detects that -Kl64 is required, it test-compiles a null program to see if the compiler supports -Klp64. If the compiler doesn't support -Klp64, Configure echoes this message and quits:

```
    /usr/ccs/bin/cc doesn't support -Klp64.  Consult 00FAQ.
```

You can't proceed until you have a compiler that supports -Klp64. Hint: if you have a compiler that does support it, but the compiler is located at a path other than /usr/ccs/bin/cc, supply its path to Configure via the LSOF_CC environment variable -- e.g., if the compiler that supports -Klp64 is in /opt/C/bin/cc, you might use this Configure command:

```
    $ LSOF_CC=/opt/C/bin/cc Configure pyramid
```

I have used both these compilers successfully on Reliant Unix 5.44 and above:

```
    Pyramid C Compiler 06.0A00

    CDS++ Version 02.A00
```

Compilers known to lack support for -Klp64 include C-DS-MI V1.2 and gcc.

16.0 SCO Problems

16.1 SCO OpenServer Problems

16.1.1   How can I avoid segmentation faults when compiling lsof?

If you have an older SCO OpenServer compiler, it may get a segmentation fault when compiling some lsof modules. That appears to happen because of the -Ox optimization action requested in the lsof Makefile.

Try changing -Ox to -O with this make invocation:

```
    $ make DEBUG=-O
```

Bela Lubkin supplied this tip and Steve Williams verified it.

16.1.2   Where is libsocket.a?

If you compile lsof and the loader says it can't find the socket library, libsocket.a, called by the -lsocket option in the lsof compile flags, you probably are running an SCO OpenServer release earlier than 5.0 and don't have the TCP/IP Development System package installed.

You may have the necessary header files, because you have the TCP/IP run-time package installed, but if you don't have the TCP/IP Development System package installed, you won't have libsocket.a.

Your choices are to install the TCP/IP Development System package or upgrade to OpenServer Release 5.0. You will find libsocket.a in 5.0 -- you'll find all the libraries and header files there, in fact -- and you can use gcc to compile lsof if you don't want to install the 5.0 Development System package.

16.1.3   Why do I get "warning C4200" messages when I compile lsof?

When you compile lsof under OSR 3.2v4.2 (and perhaps under earlier versions as well), you may get many compiler warning messages of the form:

```
    node.c(183) : warning C4200: previous declarator is not
    compatible with default argument promotion
```

In my opinion this is a bug in the OSR compiler. Because the compiler cannot handle full ANSI-C prototypes, it assumes default types for function parameters as it encounters untyped in a function prototype -- e.g., in this function declaration from node.c,

```
    readrnode(ra, r)
      KA_T ra;
      struct rnode *r;
    {
    ...
```

the compiler assigns default int types to the ra and r arguments.

Then, when the compiler encounters the fully typed parameters after the function skeleton and sees parameters with types that don't match the assumptions it previously made, it whines about its own assumptions.

You can ignore these messages.

16.2 SCO|Caldera UnixWare Problems

16.2.1 Why doesn't lsof compile on my UnixWare 7.1.1 or above system?

When you Configure lsof with the "uw" abbreviation and try to compile it for UnixWare 7.1.1, you may get compiler error messages like this:

```
    UX:acomp: ERROR: "dproc.c", line 98:
      undefined struct/union member: p_pgidp
```

This suggest that you probably have a non-stop cluster UnixWare 7.1.1 system. Its <sys/proc.h> header file differs from the one on the system where I did the lsof port to UnixWare 7.1.1. I currently don't have access to a non-stop cluster system to be able to develop changes to lsof that would make it compile and work there.

If you have a non-stop cluster UnixWare 7.1.1 system, want lsof for it, and can offer me a test account on the system, please contact me via e-mail at <abe@purdue.edu>. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

If you have a system with nsc_cfs and can offer me a test account on it, please contact me via e-mail at <abe@purdue.edu>. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

16.2.2 Why does lsof complain about node_self() on my UnixWare 7.1.1 or above system?

If lsof exits immediately after issuing this message:

```
    can't identify process NSC node; node_self(): <message>
```

It means that lsof has been built to run on a NonStop Cluster (NSC) UnixWare 7.1.1 or higher system and can't get the number of the node on which it is running. Lsof uses the node number to determine the path to the kernel boot file.

You can tell if lsof has been built for NSC by looking for "-DHAS_UW_NSC" in lsof's "-v" option output.

If the system on which you're trying to run lsof isn't running an NSC kernel, you will need to build a non-NSC lsof.

16.2.3 Why does UnixWare 7.1.1 or above complain about -lcluster, node_self(), or libcluster.so?

When you build, compile, and load lsof for UnixWare 7.1.1 and above, ld may complain that it can't find the -lcluster library or that the node_self symbol is undefined. When you try to run an existing lsof binary it may complain that libcluster.so can't be found.

These messages mean the tests made by Configure on your system led it to believe your system is running a NonStop Cluster (NSC) kernel, or the lsof binary you're trying to use was built on a NonStop Cluster system. If an lsof binary was built for NSC, this shell command produces output:

```
    $ strings <lsof_binary> | grep HAS_UW_NSC
```

If that's not the case, and you can rebuild lsof, set the UW_HAS_NSC environment variable to "N" and do this:

```
   $ Configure -n clean
   $ UW_HAS_NSC=N
   $ export UW_HAS_NSC
   $ Configure -n uw
   $ make
```

You can also edit Makefile and lib/Makefile. Remove -DHAS_UW_NSC from the CFGF strings. Remove -lcluster from the CFGL strings. Then run make again.

If you have an existing NSC lsof binary and you want one for a non-NSC system, you will have to build lsof yourself on the system where you want to use it. (That's always a good idea anyway.)

16.2.4 Why does UnixWare 7.1.1 or above lsof complain it can't read the kernel name list?

If lsof complains:

```
    can't read kernel name list from <path>
```

It means that lsof can't find the booted kernel image file at <path>. On NonStop Cluster (NSC) UnixWare 7.1.1 or higher systems lsof determines the booted file path by examining this file:

```
    /stand/`node_self`/boot
```

If examining that file doesn't lead to an NSC path, lsof uses:

```
    /stand/1/unix
```

On non-NSC systems lsof expects the booted kernel image to be in /stand/unix.

If your booted kernel image is in a different place, use lsof's "-k <path>" option to specify its path.

16.2.5 Why doesn't lsof report link count, node number, and size for some UnixWare 7.1.1 or above CFS files?

Lsof reports link count, node number, and size for open CFS files as recorded in their kernel node structure's cached attributes. Sometimes not all attributes are cached on the node where lsof runs, so lsof cannot report them.

16.2.6 Why doesn't lsof report open files on all UnixWare 7.1.1 NonStop Cluster (NSC) nodes?

Lsof can only report on files open on the node on which it runs, because the information lsof reports comes from the private kernel memory of the node. This may mean that asking lsof to find a specific open file, or use of a specific Internet address or port, may not report all open instances on nodes other than the one used to run lsof.

You can use the NSC onnode(1) command to run lsof on specific nodes, or the onall(1) command to run lsof on all nodes -- e.g.,

```
    $ onall lsof [options] 2>&1 | less
 or
    $ onnode node-number lsof [options] 2>&1 | less
```

Note that, when lsof is run all nodes, the path name component assembly results it reports in its NAME column may vary, because the dynamic name cache from which lsof gets the components is private to the kernel of each node.

Also note the use of shell redirection in the examples to merge the standard error file information from onnode and onall with lsof's standard output file output. That will put the onnode and onall node announcements in proper sequence with lsof's output.

16.2.7 Why doesn't lsof report the UnixWare 7.1.1 NonStop Cluster (NSC) node a process is using?

To induce lsof to report the node on which a process runs would be a significant, non-standard modification to lsof. It has much wider implications than merely the printing of a number in an output column. I'm not currently (April 2001) prepared to undertake such a modification.

If you want node-specific NSC information about open files, run lsof under the control of onall(1) or onnode(1).

```
    $ onall lsof [options] 2>&1 | less
 or
    $ onnode node-number lsof [options] 2>&1 | less
```

17.0 Sun Problems

17.0.5 Statement of deprecation

Lsof support for SunOS 4.1.x was last tested at revision 4.51. Contact me via e-mail if you're interested in obtaining it. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

17.1 My Sun gcc-compiled lsof doesn't work -- why?

Gcc can be used to build lsof successfully. However, an improperly installed Sun gcc compiler will usually not produce a working lsof.

If your Sun gcc-compiled lsof doesn't report anything, or reports ``can't read proc table,'' or gcc refuses to compile lsof without error, check that the gcc step that "fixes" Sun header files was run on the system where you're using gcc to compile lsof. As an alternative, if you have the SunPro C 5.0 compiler or later available, use it to compile lsof -- e.g., use the solariscc Configure abbreviations.

17.2 How can I make lsof compile with gcc under Solaris 2.[456], 2.5.1, 7, 8 or 9?

Presuming your gcc-specific header files are wrong for Solaris, edit the lsof Configure-generated Makefile and lib/Makefile and make this change:

```
      CFGF=   -Dsolaris=20400 ...
to
      CFGF=   -Dsolaris=20400 -D__STDC__=0 -I/usr/include ...
```

or change:

```
      CFGF=   -Dsolaris=20500 ...
to
      CFGF=   -Dsolaris=20500 -D__STDC__=0 -I/usr/include ...
```

or change:

```
      CFGF=   -Dsolaris=20501 ...
to
      CFGF=   -Dsolaris=20501 -D__STDC__=0 -I/usr/include ...
```

This is only a temporary work-around. You really should instruct gcc to to update your gcc-specific header files or install a recent gcc (e.g., 3.2), which has no need for private copies of Solaris include files.

17.3 Why does Solaris Sun C complain about system header files?

You're probably trying to use /usr/ucb/cc if you get compiler complaints like:

```
    cc -O -Dsun -Dsolaris=20300 ...
    "/usr/include/sys/machsig.h", line 81: macro BUS_OBJERR
    redefines previous macro at "/usr/ucbinclude/sys/signal.h",
    line 444
```

Note the reference to "/usr/ucbinclude/sys/signal.h". It reveals that the BSD Compatibility Package C compiler is

in use. Lsof requires the ANSI C version of the Solaris
C compiler, usually found in /usr/opt/bin/cc or
/opt/SUNWspro/bin/cc.

Try adding a CC string to the lsof Makefile that points to
the Sun ANSI C version of the Sun C compiler -- e.g.,

```
    CC= /usr/opt/bin/cc
or
    CC= /opt/SUNWspro/bin/cc.
```

17.4 Why doesn't lsof work under my Solaris 2.4 system?

If lsof doesn't work under your Solaris 2.4 system -- e.g.,
it produces no output, little output, or the output is
missing command names or file descriptors -- you may have
a pair of conflicting Sun patches installed.

Solaris patch 101945-32 installs a kernel that was built
with a <sys/auxv.h> header file whose NUM_*_VECTORS
definitions don't match the ones in the <sys/auxv.h> updated
by Solaris patch 102303-02.

NUM_*_VECTORS in the kernel of patch 101945-32 are smaller
than the ones in the <sys/auxv.h> of patch 102303-02. The
consequence is that when lsof is compiled with the <sys/auxv.h>
whose NUM_*_VECTORS definitions are larger than the ones
used to compile the patched kernel, lsof's user structure
does not align with the one that the kernel employs.

If you have these two patches installed, contact Sun and
complain about the mis-match.

The lsof Configure script attempts to work around the
mis-matched patches by including a modified <sys/auxv.h>
header file from ./dialects/sun/include/sys. That auxv.h
has these alternate definitions:

```
          #define NUM_GEN_VECTORS 4
          #define NUM_SUN_VECTORS 8
```

The Configure script issues a prominent WARNING that it is
putting this work-around into effect. If it doesn't succeed
for you, please contact me.

I thank Leif Hedstrom for identifying the offending patches.

17.5 Where are the Solaris header files?

If you try to compile lsof under Solaris and get a compiler
complaint that it can't find system header files, perhaps
you forgot to add the header file package, SUNWhea.

17.6 Where is the Solaris /usr/src/uts/<architecture>/sys/machparam.h?

When you try to Configure lsof for Solaris 2.[23456], 2.5.1,
and 7 -- e.g., on a `uname -m` == sun4m system -- Configure
complains:

```
    grep: /usr/src/uts/sun4m/sys/machparam.h:
            No such file or directory
    grep: /usr/src/uts/sun4m/sys/machparam.h:
            No such file or directory
```

And when you try to compile the configured lsof, cc or gcc complains:

```
    dproc.c:530: `KERNELBASE' undeclared (first use this function)
```

The explanation is that somehow your Solaris system doesn't have the header files in /usr/src/uts it should have. Perhaps someone removed the directory to save space. Perhaps you're using a gcc installation, copied from another system. In any event, you will have to load the header files from the SUNWhea package of your Solaris distribution.

KERNELBASE is an important symbol to lsof -- it keeps lsof from sending an illegal kernel value to kvm_read() where a segmentation violation might result (a bug in the kvm library). Lsof can get illegal kernel values because it reads kernel values slowly with kvm_read() calls that the kernel is changing rapidly.

Lsof doesn't need KERNELBASE at Solaris 2.5 and above, because it has a KERNELBASE value whose address lsof can find with /dev/ksyms and whose value it can read with kvm_read(). Under Solaris 2.5 /usr/src/uts has moved to /usr/platform.

17.7 Why does Solaris lsof say ``can't read proc table''?

When lsof collects data on processes, using the kvm_*() functions to scan the kernel's proc structure table, it checks to make sure it has identified a reasonable number of them -- a minimum of three. When lsof can't identify three processes during a scan, it repeats the scan.

When five scans fail to yield three processes, lsof issues the fatal message:

```
      lsof: can't read proc table
```

and exits.

Usually lsof fails to identify three processes during a scan because its idea of the form of the proc structure differs from that being used by the kernel. Since the proc structure is defined in <sys/proc.h> and other /usr/include header files, the root cause of a proc structure discrepancy usually can be found in the composition of /usr/include.

One common way that /usr/include header files can be incorrect is that gcc was used to compile lsof, gcc used its special (i.e., "fixed") header files instead of the ones in /usr/include, and the special gcc header files weren't updated when Solaris was. Answers to these questions:

> My Sun gcc-compiled lsof doesn't work -- why?
>
> How can I make lsof compile with gcc under Solaris 2.[456], 2.5.1, 7, 8 or 9?
>
> Why does Solaris Sun C complain about system header files?

discuss the gcc header file problem and offer suggestions on how to fix it or work around it.

It may also be that you are trying to run a version of lsof that was compiled on an older version of Solaris. For example, an lsof executable, compiled for Solaris 2.4, will produce the ``can't read proc table'' message if you try to run it under Solaris 2.5. If you have compiled lsof under Solaris 2.5 and it still won't work, see if the header files in /usr/include have been updated to 2.5, or still represent a previous version of Solaris.

Another source of header file discrepancies to consider is the Solaris patch level and whether a binary kernel patch was not matched with a corresponding header file update. See the "Why doesn't lsof work under my Solaris 2.4 system?" question for an example of one in Solaris 2.4 -- there may be other such patch conflicts I don't know about.

17.8 Why does Solaris lsof complain about a bad cached clone device?

When lsof revisions below 4.04 have been run on a Solaris system and have been allowed to create a device cache file, the running of revisions 4.04 and above on the same systems may produce this complaint:

```
    lsof: bad cached clone device: ...
    lsof: WARNING: created device cache file: ...
```

This is the result of a change in the device cache file that took place at lsof revision 4.04. The change introduced a node number into the clone device lines of the device cache file and was done in such a way that lsof could detect device cache files whose clone lines don't have node numbers (lines created by previous lsof revisions) and recognize the need to regenerate the device cache file.

17.9 Why doesn't Solaris make generate .o files?

Solaris /usr/ccs/bin/make won't generate .o files from .c files if /usr/share/lib/make/make.rules is missing. It may be found in and installed from the SUNWsport package.

17.10 Why does lsof report some Solaris 2.3 and 2.4 lock types as `N'?

For Solaris 2.3 with patch P101318 installed at level 45 or above, and for all versions of Solaris 2.4, NFS locks are represented by a NFS-specific kernel lock structure that sometimes lacks a read or write lock type indicator. When lsof encounters such a lock structure, it reports the lock type as `N'.

17.11 Why does lsof Configure say "WARNING: no cc in ..."?

When lsof's Configure script is executed with the solariscc abbreviation it tries to make sure it's using the Sun C compiler and not the UCB substitute from /usr/ucb/cc. Thus, it looks for cc in the "standard" Sun compiler location, /opt/SUNWspro/bin.

If Configure can't find cc there, it issues the warning:

```
    lsof: WARNING: no cc in /opt/SUNWspro/bin;
        using cc without path.
```

and uses cc for the compiler name, letting the shell find cc with its PATH environment variable.

You can tell Configure where to find your cc with the
SOLARIS_CCDIR cross-configuration environment variable.
(See 00XCONFIG for more information on SOLARIS_CCDIR).
For example, use this Configure shell command:

```
    SOLARIS_CCDIR=/usr/special/bin Configure -n solariscc
```

(SOLARIS_CCDIR should be the full path to the directory
containing your cc.)

17.12 Solaris 7, 8 and 9 Problems

17.12.1 Why does lsof say the compiler isn't adequate for Solaris 7, 8 or 9?

Solaris 7, 8 and 9 kernels come in two flavors, 32 and 64
bit. 64 bit kernels run on machines that support the SPARC
v9 instruction set architecture. Separate executables for
some programs, -- e.g., ones using libkvm like lsof -- must
be built for 32 and 64 bit kernels.

Previous Sun (e.g., SC4.0) and earlier gcc compilers will
build lsof for 32 bit kernels, but they won't build it for
64 bit kernels. Compilers that will build lsof for 64 bit
Solaris 7, 8 and 9 kernels are the Sun WorkShop Compilers
C 5.0 and above, and recent gcc versions, e.g., 3.2.

When given the ``-xarch=v9'' flag, the C 5.0 compiler and
above, and associated loader and 64 bit libraries will
build a 64 bit lsof executable; when given the "-m64" or
"-mcpu=v9" (deprecated) flags, an appropriate gcc compiler
will build a 64 bit lsof executable.

When the lsof Configure script detects a 64 bit kernel is
in use (e.g., by executing `/bin/isainfo -kv`), and when
it finds that the specified compiler is inappropriate,
it complains with these messages:

For gcc:

```
    "!!!WARNING!!!=========!!!WARNING!!!=========!!!WARNING!!!"
    "!                                                       !"
    "! LSOF NEEDS TO BE CONFIGURED FOR A 64 BIT KERNEL, BUT  !"
    "! THIS GCC DOESN'T SUPPORT THE BUILDING OF 64 BIT       !"
    "! SOLARIS EXECUTABLES.  LSOF WILL BE CONFIGURED FOR A   !"
    "! 32 BIT echo KERNEL.                                   !"
    "!                                                       !"
    "!!!WARNING!!!=========!!!WARNING!!!=========!!!WARNING!!!"
```

For Sun C:

```
  !!!WARNING!!!==========!!!WARNING!!!==========!!!WARNING!!!
  !                                                         !
  ! LSOF NEEDS TO BE CONFIGURED FOR A 64 BIT KERNEL, BUT     |
  ! THE VERSION OF SUN C AVAILABLE DOESN'T SUPPORT THE      !
  ! -xarch=v9 FLAG.  LSOF WILL BE CONFIGURED FOR A 32 BIT   !
  ! KERNEL.                                                 !
  !                                                         !
  !!!WARNING!!!==========!!!WARNING!!!==========!!!WARNING!!!
```

17.12.2 Why does Solaris 7, 8 or 9 lsof say "FATAL: lsof was compiled for..."?

Solaris 7, 8 or 9 lsof may say:

```
    lsof: FATAL: lsof was compiled for a xx bit kernel,
        but this machine has booted a yy bit kernel.

    Where: xx = 32 or 64
           yy = 64 or 32

    (xx and yy won't match.)
```

This message indicates that lsof was compiled for one size kernel and is being asked to execute on a different size one. That's not possible for programs like lsof that use libkvm.

Depending on the instruction sets for which you need Solaris 7, 8 or 9 lsof, you may need two or more versions of lsof, compiled for each kernel size, installed for use with /usr/lib/isaexec. See the "How do I install lsof for Solaris 7, 8 or 9?" section of this document for more information on that.

17.12.3   How do I build lsof for a 64 bit Solaris kernel under a 32 bit Solaris kernel?

If your Solaris system has an appropriate compiler (e.g., WorkShop Compilers C 5.0 and above, or a recent gcc like 3.2) and the 64 bit libraries have been installed, you can force lsof's Configure script to build a 64 bit version of lsof with:

```
    $ SOLARIS_KERNBITS=64 Configure -n solariscc
```

The SOLARIS_KERNBITS environment variable is part of the lsof cross-configuration support, described in the 00XCONFIG file of the lsof distribution.

17.12.4   How do I install lsof for Solaris 7, 8 or 9?

If you are installing lsof where it will be used only under the bit size kernel for which it was built, no special installation is required.

If, however, you are installing different versions of lsof for different bit sizes -- e.g., for use on a 64 bit NFS server and from its 32 bit clients -- you should read the man page for isaexec(3C) and install lsof according to its instructions.

The executable at the directory where lsof is to be found should be a hard link to /usr/lib/isaexec or a copy of it. In the directory there must be instruction architecture subdirectories -- e.g., .../sparc/ and .../sparcv9/. The lsof for 64 bit size kernels is installed in the .../sparcv9/ subdirectory; the one for 32 bit size kernels, in .../sparc/.

For example, if you're installing 32 and 64 bit lsof executables in /usr/local/etc, you would:

```
      # cd /usr/local/etc
      # ln /usr/lib/isaexec lsof
      # mkdir sparc sparcv9
      # install the 32 bit lsof as sparc/lsof
      # install the 64 bit lsof as sparcv9/lsof
      # chmod, chown, and chgrp sparc/lsof and
```

```
                sparcv9/lsof appropriately

        Lsof permissions and ownerships are the same whether one
        or more lsof executables are being installed, with or
        without the /usr/lib/isaexec hard link.

17.12.5 Why does my Solaris 7, 8 or 9 system say it cannot execute
        lsof?

        When you attempt to execute lsof, your Solaris 7, 8 or 9
        shell may complain:

            ksh: ./lsof: cannot execute

        If the lsof executable exists and has the proper execution
        permissions, this error may be the result of trying to
        execute an lsof, built for a 64 bit kernel, on a 32 bit
        kernel.

        This will tell you about the lsof executable:

            $ file lsof
            lsof: ELF 64-bit MSB executable SPARCV9 Version 1,
                dynamically linked, not stripped

        The "64-bit" notation indicates the binary was built for
        a 64 bit kernel.  To see the running kernel bit size, use
        this command:

            $ isainfo -kv
            32-bit sparc kernel modules

        The "32-bit" notation indicates a 32 bit kernel has been
        booted.

        The only work-around is to obtain, or Configure and make,
        an lsof for the appropriate kernel bit size.  If you
        Configure and make lsof on the kernel where you wish to
        run it the proper compiler, the lsof Configure step will
        generate Makefiles that can be used with make to build an
        appropriate lsof executable.

        To compile a 64 bit lsof, you must have an appropriate
        compiler -- i.e., Sun WorkShop Compilers C 5.0 or higher
        or a recent gcc like 3.2.

17.12.6 What gcc will produce 64 bit Solaris 7, 8 and 9 executables?
        8 and 9 executables?

        Properly built and installed recent gcc versions -- e.g.,
        3.2 -- will build lsof for 64 bit Solaris kernels.

        If you update your gcc version to 3.2 or later, make sure
        the private gcc header files become current -- i.e., clear
        out any private header files from a previous gcc or Solaris
        installation before installing the new ones, or build to
        a new --prefix root and replace the old root with it after
        the build and installation are complete.

17.12.7 Why does lsof on my Solaris 7, 8 or 9 system say, "can't
        read namelist from /dev/ksyms?"

        You're probably trying to use an lsof executable built for
        an earlier Solaris release on a 64 bit Solaris 7, 8 or 9
```

kernel. The output from `lsof -v` will tell you the build environment of your lsof executable. You should also have gotten a warning message that lsof is compiled for a different Solaris version than the one under which it is running -- something like this:

```
    lsof: WARNING: compiled for Solaris release X; this is Y
```

You need to build lsof on the system where you want to use it. For 64 bit Solaris 7, 8 and 9 you need a compiler that can generate 64 bit Solaris executables -- e.g., the Sun Workshop 5 C compiler or later, or a recent gcc version like 3.2. See the "Why does lsof say the compiler isn't adequate for Solaris 7, 8 or 9?" section and the ones following it for a discussion of building lsof for 64 bit Solaris 7, 8 or 9.

17.13 Solaris and COMMON

17.13.1 What does COMMON mean in the NAME column for a Solaris VCHR file?

When lsof puts COMMON or (COMMON) in the NAME column of a Solaris VCHR file, it means that the file is handled by the special file system functions of the kernel through a common vnode.

17.13.2 Why does a COMMON Solaris VCHR file sometimes seem to have an incorrect minor device number?

When lsof reports on an open file in a Solaris special file system that uses a COMMON vnode, and the file is a VCHR file, lsof tries to locate the associated device node by looking for matches on the major and minor device numbers first.

If no major and minor match results, lsof then looks for a match on pseudo and clone device files. (See /devices/pseudo.) Those device nodes are matched specially by either their major or minor device numbers, but not both. Hence, when lsof finds a match under those special conditions, it may report a value in its output DEVICE column that differs from one of the major and minor numbers of the device node.

Here's an example from a sun4m Solaris 7 system:

```
    $ ls -li /devices/pseudo/pm@0:pm
    151261 crw-rw-rw-   1 root     sys      117,  0 ...
    $ lsof /devices/pseudo/pm@0:pm
    COMMAND ... DEVICE ...   NODE NAME
    powerd       117,1 ... 151261 /devices/pseudo/pm@0:pm (COMMON)
    Xsun    ...  117,0 ... 151261 /devices/pseudo/pm@0:pm
```

Note that the DEVICE value for the file with (COMMON) in its name field has a different minor device number (1) from what ls reports (0), while the DEVICE value for the file without (COMMON) matches the ls output exactly. Both match on the major device number, 117. The minor device number mis-match is a result of the way the Solaris kernel handles special file system common vnodes, and it's the reason lsof puts (COMMON) after the name to signal that a mis-match is possible.

17.14 Why don't lsof and Solaris pfiles reports always match?

/usr/proc/bin/pfiles for Solaris 2.6, 7, 8, and 9 also reports information on open files for processes. Sometimes the information it reports differs from what lsof reports.

There are several reasons why this might be true. First, because pfiles is a Sun product, based on Sun kernel features, its developers have a better chance of knowing exactly how open file information is organized. I sometimes have to guess at how kernel file structure linkages are constructed by gleaning hints from header files.

Second, lsof is aimed at providing information, specifically device and node numbers, that can be used to identify named file system objects -- i.e., path names. Thus, lsof tries to make sure its device and node numbers match those reported by stat(2). Pfiles doesn't always report numbers that match stat(2) -- e.g., for files using clone and pseudo devices via common vnodes like the nlist() /dev/ksyms usage.

Here's the Solaris 7 COMMON VCHR example again with additional pfiles output:

```
    $ ls -li /devices/pseudo/pm@0:pm
    151261 crw-rw-rw-   1 root     sys      117,  0 ...
    $ lsof /devices/pseudo/pm@0:pm
    vic1: 10 = lsof /dev/pm
    COMMAND ... DEVICE ...   NODE NAME
    powerd  ...  117,1 ... 151261 /devices/pseudo/pm@0:pm (COMMON)
    Xsun    ...  117,0 ... 151261 /devices/pseudo/pm@0:pm
    $ pfiles ...
    0: S_IFCHR ... dev:32,24 ino:61945 ... rdev:117,1
    ...
    14: S_IFCHR ... dev:32,24 ino:151261 ... rdev:117,0
```

Note that the NODE number, reported by lsof, matches what ls(1) and stat(2) report, while the ino value pfiles reports doesn't. Lsof also indicates with the (COMMON) notation that the DEVICE number is a pseudo one, derived from the character device's value. The lsof DEVICE value matches the pfiles rdev value, correct behavior for a character device, but pfiles gives no sign that it's not possible to find that character device number in /devices with ls(1) or stat(2).

17.15 Why does lsof say, "kvm_open (namelist=default, core=default): Permission denied?"

Lsof needs permission to read from the /dev/kmem and /dev/mem memory devices. Access to them is opened via a call to the kvm_open() library function and it reports the indicated message.

You must give lsof permission to read the memory devices. The super user can almost always do that, but other lsof users can do it if some group -- e.g., sys -- has permission to read the memory devices, and the lsof binary is installed with the group's ownership and with the setgid permission bit enabled.

17.16 Why is lsof slow on my busy Solaris UFS file system?

Lsof may be slow on a busy Solaris UFS file system when UFS logging has been enabled with the "logging" mount option. That option can significantly increase disk operations under certain conditions -- e.g., when a lot of files are accessed quickly.

When only the "logging" option is specified to mount, all file accesses (atime updates) are logged to the UFS logging queue. Each atime update requires two writes to the disk to complete it.

If you want to do UFS logging -- and there are reliability advantages to it -- consider using the "logging,noatime" mount options instead. That will shift atime updates from the logging queue to fewer and independent asynchronous operations, consequently making the UFS logging queue a smaller bottleneck.

Consult mount_ufs(1M) for more information on the logging and noatime options.

(My thanks to Casper Dik for this tip on improving the performance of UFS logging.)

17.17 Why is lsof so slow on my Solaris 8 or 9 system?

Solaris 8 has a post-release feature upgrade modifying kernel name cache (DNLC) handling that can slow lsof throughput dramatically. The feature, sometimes called negative DNLC caching, is standard in Solaris 9.

As best I can tell, when you install the Solaris 8 MU1 package, you get negative DNLC caching. If this pipe produces any output, your system has negative DNLC caching.

```
    $ nm /dev/ksyms | grep negative_cache_vnode
```

The reason negative DNLC caching perturbs lsof is that a single vnode address (found in the negative_cache_vnode kernel variable) is used to mark entries in the DNLC that are not (the negative part) found on disk.

Since a single vnode address (the DNLC key lsof uses) can represent many (I've seen upwards of 30,000.) DNLC entries, their presence overloads lsof's internal DNLC hashing function. An overloaded hash function is a slow hash function, and lsof's slows to a crawl when it encounters thousands of keys that produce the same value when the lsof DNLC hash function is applied to them.

The solution is simple -- ignore negative DNLC cache keys. They don't represent path name components lsof can use. Lsof revisions 4.50 and above have an addition that ignores them and the performance of those lsof revisions improves significantly when presented with negative DNLC cache keys.

If you don't have an lsof revision at 4.51 or later, there's a work-around. Use lsof's ``-C'' option.  It disables lsof's DNLC caching.  Of course, that also inhibits the reporting of any path name components from the kernel DNLC. When ``-c'' is used, lsof will continue to report file system and character device paths.

17.18 Why doesn't lsof support VxFS 3.4 on Solaris 2.6, 7, and 8?

Lsof will not support VxFS version 3.4 on Solaris 2.6, 7,

or 8 unless some files from VxFS Update 2 have been installed.
VxFS 3.4 FCS and VxFS 3.4 update 1 lack the header files
lsof normally uses to obtain information from the VxFS 3.4
kernel node structure, vx_inode. VxFS 3.4 Update 2 provides
a method whereby lsof can obtain the necessary vx_inode
information from the vxfsu_get_ioffsets() function in
Veritas utility libraries.

The utility libraries (32 bit and 64 bit versions) may be
found in /opt/VRTSvxfs/lib. An ancillary header file may
be found in /opt/VRTSvxfs/include/sys/fs/vx_libutil.h.
Documentation of the vxfsu_get_ioffsets(3) function may be
found in /opt/VRTS/man/man3/vxfsu_get_ioffsets.3.

Those files of VxFS 3.4 Update 2 may be downloaded from:

```
    ftp://ftp.veritas.com/pub/support/vxfs_34.i64243.tar
```

The vxfs_34.i64243.tar archive will unpack into an i64243
directory containing these files:

```
    $ ls i64243
    README
    libvxfsutil.sol26.sums
    libvxfsutil.sol26.tar.Z
    libvxfsutil.sol27.sums
    libvxfsutil.sol27.tar.Z
    libvxfsutil.sol28.sums
    libvxfsutil.sol28.tar.Z
```

Read README. Select the *.tar.Z file appropriate for your
Solaris version. Its contents will unpack into /opt/VRTS
and /opt/VRTSvxfs, so you will need sufficient permission
-- e.g., do it as root -- to unpack the uncompressed archive.
Once you've done that, it's a good idea to compare the
checksums of the archive you unpacked with the ones recorded
in the appropriate *.sums file. Use `sum -r` to verify
the checksums.

For example, if you want the Solaris 8 version, uncompress
and unpack libvxfsutil.sol28.tar.Z -- e.g.,

```
    $ su
    ...
    # cd i6423
    # zcat libvxfsutil.sol28.tar.Z | tar xf -
```

That should create these new files and subdirectories with
the indicated checksums:

```
    File or subdirectory                 sum -r

    /opt/VRTSvxfs/include/vxfsutil.h            03938
    /opt/VRTSvxfs/lib/libvxfsutil.a       51794
    /opt/VRTSvxfs/lib/sparcv9/
    /opt/VRTSvxfs/lib/sparcv9/libvxfsutil.a     07420
    /opt/VRTS/man/man3/
    /opt/VRTS/man/man3/vxfsu_get_ioffsets.3     62480
```

Once these files are in place, run lsof's Configure script
for the solaris or solariscc abbreviation. Configure will
locate the appropriate VxFS 3.4 Update 2 files and set up
for the making of an lsof that will properly display open
VxFS 3.4 file information.

17.18.1 Why does lsof report "vx_inode: vxfsu_get_ioffsets error" for open Solaris 2.6, 7, and 8 VxFS 3.4 files?

Even when lsof supports VxFS 3.4 on Solaris 2.6, 7, or 8, it may report "vx_inode: vxfsu_get_ioffsets error" in the NAME column for all VxFS files.

The usual cause is that lsof doesn't have permission to read the file at the end of the /dev/vxportal symbolic link. If, for example, lsof has been installed setgid(sys), then the /dev/vxportal symbolic link destination should be owned by the sys group and readable by it.

Update 2 for VxFS 3.4 sets the modes of the /dev/vxportal symbolic link destination to 0640 and the group ownership to sys.

17.19 Large file problems

17.19.1 Why does lsof complain it can't stat(2) a Solaris 2.5.1 large file?

When given an argument that is the path to a Solaris 2.5.1 file, enable for large file operations with the O_LARGEFILE open(2) option, lsof complains that it can't stat(2) the file. That's because lsof isn't using a stat(2) call and associated structure enabled for large files.

This error has been fixed, starting at lsof revision 4.58 for Solaris 2.6 and above. That fix won't work on Solaris 2.5.1 and I no longer have access to a Solaris 2.5.1 test system to develop a separate fix.

The work-around is to avoid specifying a O_LARGEFILE path as an argument to lsof on Solaris 2.5.1. Instead use a combination of lsof and grep to achieve the same results, albeit more clumsily.

17.20 Why does lsof get a segmentation fault on 64 bit Solaris 8 using NIS+?

I have received a report from Gary Craig that lsof produces a segmentation fault on his 64 bit Solaris 8 system using NIS+. Via an independent test program we have exonerated lsof and tracked the fault to the NIS+ __nis_server_name() function in the C name server library, -lnsl.

Lsof causes the __nis_server_name() NIS+ function to be called by calling getservent() to read entries of the port number to service name map.

The only Sun bug ID that appears to describe the problem is 4304244, although its text is unclear enough to leave room for doubt.

Until Sun eliminates the __nis_server_name() segmentation fault cause, a work-around for lsof is to use its "-P" option, causing lsof to avoid port to service name lookups.

17.21 Will lsof crash the Solaris kernel?

I've received and investigated one report that it has when the Sun hardware (a QME interface) was faulty. Today (May

23, 2002) I've learned that Sun has reports of kernel crashes caused by adb, lsof, and mdb.

The Sun investigation pinpointed a problem in the /dev/kmem kernel driver and there is a Sun bug report, 4344513, about the problem. There is a fix in Solaris 9, and patches for Solaris 7 and 8 (SPARC and x86).

To see if your Solaris system is fixed, look for a /devices/pseudo/*allkmem node.

Extensive address filtering was added to lsof revision 4.50 to forestall what I then (July 2001) believed to be only the possibility that lsof might crash Solaris. However, the filtering isn't perfect, since a filtered address might become invalid after lsof has filtered it but before lsof has delivered it to /dev/kmem. That filtering work is described in .../dialects/sun/solaris_kaddr_filters, also available at:

ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/solaris_kaddr_filters

The best and safest work-around is to upgrade to Solaris 9 or install an appropriate patch or its equivalent from this list:

```
    Solaris SPARC       x86
    Version Patch       Patch
    ======= =====       =====
       7    106541-20   106542-20
       8    108528-14   108529-14
```

17.22   Why does lsof on Solaris 7, 8, or 9 report a kvm_open() failure?

When lsof is started on some Solaris 7, 8, and 9 systems it may report:

```
    lsof: kvm_open (namelist=default, core=default): \
        No such file or directory
```

Lsof revisions 4.65 and later will first report:

```
    lsof: cannot stat /dev/allkmem
```

The second message, not delivered in lsof revisions below 4.65, explains the cause of the kvm_open() failure; it can't find /dev/allkmem.

/dev/allkmem is a device added to Solaris 7 and 8 in patches and in the Solaris 9 FCS. See the preceding "Will lsof crash the Solaris kernel?" section for more information on /dev/allkmem and the patches.

The kvm_open(3KVM) function in the KVM library of patched Solaris 7 and 8 systems and in Solaris 9 expects to find /dev/allkmem and exits on error when it does not.

If you have installed the patch that updated your KVM library to a version that expects /dev/allkmem to be present and it is not, you may need to reconfigure your system's devices with devfsadm(1M) or enter "boot -r" to the OpenBoot monitor's prompt (usually "ok").

17.23 Solaris and SAM-FS

17.23.1 Why does Solaris lsof report "(limited SAM-FS info)"?

Lsof 4.68 and above report "(limited SAM-FS info)" on Solaris in the NAME column after the path or file system name for all files it finds on SAM-FS file systems.

That's because no more information is known about the composition of the nodes that follow SAM-FS vnodes. If you can provide that information, please contact me via e-mail at <abe@purdue.edu>. Make sure "lsof" appears in the "Subject:" line so my e-mail filter won't classify your letter as Spam.

17.23.2 Why can't lsof locate named SAM-GS files?

Solaris lsof 4.68 and above can't locate files on SAM-FS file systems when the files are named as lsof arguments because lsof doesn't know how to locate open SAM-FS file device and node number information. (See also 'Why does Solaris lsof report "(limited SAM-FS info)?')

17.24 Lsof and Solaris 10 zones

17.24.1 How can I make lsof list the Solaris zone?

Use the lsof "-z [z]" option.

17.24.2 Why doesn't lsof work in a Solaris 10 zone?

When run from within a Solaris 10 zone, lsof will usually report:

```
    lsof: can't stat(/devices): No such file or directory
```

That's because a Solaris zone usually has no /devices subdirectory, a restriction of the zone implementation intended to limit the ability of zone processes to control global system resources, including physical devices.

While a zone may have a /dev subdirectory, that subdirectory usually lacks the /dev/allkmem, /dev/mem and /dev/kmem devices lsof and the KVM library it uses require.

The work-around is to run lsof in the global zone. When it is run in a global zone lsof will be able to report on processes running in any zone, including the global zone.

17.24.3 Why does lsof complain it can't stat() Solaris 10 zone file systems?

When run from the global zone on Solaris 10 lsof may complain:

```
    lsof: WARNING: can't stat() 15 zone file systems;
               using dev= options
```

The warning message means lsof found the reported number of file system entries in the mount table for which it didn't have permission to get stat(2) results, but which had "zone=" and "dev=" mount table options.

That is a normal restriction of Solaris 10 zones. Since the lsof warning message indicates it was able to find "dev="

options for the file systems, lsof will probably work correctly.

One work-around is to relax the restrictions on zone mount points, so that lsof can stat() them. While that may be possible by changing directory modes or group ownerships, it is probably not a good idea, because it weakens the restrictions zones are intended to provide.

Another work-around is to suppress the warning message with lsof's "-w" option. The down side of that is that it causes the suppression of all warning messages, leading to the possibility that some non-stat() warning messages will be suppressed.

18.0 Lsof Features

18.1 Why doesn't lsof doesn't report on /proc entries on my system?

/proc file system support is generally available only for BSD, SYSV R4 dialects, and Tru64 UNIX (Digital UNIX, DEC OSF/1). It's also available for Linux, and Pyramid DC/OSx and Reliant UNIX.

Even on some SYSV R4 dialects I encountered many problems while trying to incorporate /proc file system support. The chief problem is that some vendors don't distribute the header file that describes the /proc file system node -- usually called prdata.h.

18.2 How do I disable the device cache file feature or alter it's behavior?

To disable the device cache file feature for a dialect, remove the HASDCACHE definition from the machine.h file of the dialect's machine.h header file. You can also use HASDCACHE to change the default prefix (``.lsof'') of the device cache file.

Be sure you consider disabling the device cache file feature carefully. Having a device cache file significantly reduces lsof startup overhead by eliminating a full scan of /dev (or /devices) once the device cache file has been created. That full scan also overloads the kernel's name cache with the names of the /dev (or /devices) nodes, reducing the opportunity for lsof to find path name components of open files.

If you're worried about the presence of mode 0600 device cache files in the home directories of the real user IDs that execute lsof, consider these checks that lsof makes on the file before using it:

1. To read the device cache file, lsof must gain permission from access(2).

2. The device cache file's modes must be 0600 (0644 if lsof is reading a system-wide device cache file) and its size non-zero.

3. There must be a correctly formatted section count line at the beginning of the file.

4. Each section must have a header line with a count that properly numbers the lines in the section. Legal sections are device, clone, pseudo-device, and CRC.

5. The lines of a section must have the proper format.

6. All lines are included in a 16 bit CRC, and it is recorded in a non-checksummed section line at the end of the file.

7. The checksum computed when the file is read must match the checksum recorded when the file was written.

8. The checksum section line must be followed by end-of-information.

9. Lsof must be able to get matching results from stat(2) on a randomly chosen entry of the device section.

For more information on the device cache file, read the 00DCACHE file of the lsof distribution.

18.2.1 What's the risk with a perverted device cache file?

Even with the checks that lsof makes on the device cache file, it's conceivable that an intruder could modify it so it would pass lsof's tests.

The only serious consequence I know of this change is the removal of a file whose major device number identifies a socket from some user ID's device cache file. When such a device has been removed from the device cache file, and when lsof doesn't detect the removal, lsof may not be able to identify socket files when executed by the affected user ID. Only certain dialects are at risk to this attack -- e.g., SCO OpenServer and Solaris 2.x, 7, 8, and 9.

If you're tracking a network intruder with lsof, that could be important to you. If you suspect that someone has corrupted the device cache file you're using, I recommend you use lsof's -Di option to tell it to ignore it and use the contents of /dev (or /devices) instead; or remove the device cache file (usually .lsof_hostname, where hostname is the first component of the host's name returned by gethostname(2)) from the user ID's home directory and let lsof create a new one for you.

18.2.2 How do I put the full host name in a personal device cache file path?

Lsof constructs the personal device cache file path name from a format specified in the HASPERSDC #define in the dialect's machine.h header file. As distributed HASPERSDC declares the path to be ``.lsof_'' plus the first component of the host name with the format ``.lsof_%L''.

If you want to change the way lsof constructs the personal device cache file path name, you can change the HASPERSDC #define and recompile lsof. If, for example, you #define HASPERSDC to be ``.lsof_%l'' (note the lower case `l'),

```
        Configure and remake lsof, then the personal device cache
        file path will be ``.lsof_'' plus the host name returned
        by gethostname(2).

        See the 00DCACHE file of the lsof distribution for more
        information on the formation of the personal device cache
        file path and the use of the HASPERSDC #define.

18.2.3      How do I put the personal device cache file in /tmp?

        Change the HASPERSDC definition in your dialect's machine.h
        header file.

        When you redefine HASPERSDC, make sure you put at least
        one user identification conversion in it to keep separate
        the device cache files for each user of lsof.  Also give
        some thought to including the ``%0'' conversion to define
        an alternate path for setuid-root and root processes.

        Here's a definition that puts a personal device cache file
        in /tmp with the name ``.lsof_login_hostname_pers''.

            #define HASPERSDC "/tmp/.lsof_%u_%l_pers"

        Thus the /tmp personal device cache file path for login
        "abe" on host "lsof.itap.purdue.edu" would be:

            /tmp/.lsof_abe_lsof.itap.purdue.edu_pers

        You can add the User ID (UID) with the "%U" conversion and
        the first host name component with the ``%L'' conversion.

        CAUTION: be careful using absolute paths like /tmp lest
        lsof processes that are setuid-root or whose real UID is
        root be used to exploit some security weakness via /tmp.
        Elect instead to add an alternate path for those processes
        with the ``%0'' conversion.  Here's an extension of the
        previous HASPERSDC format for /tmp that declares an alternate
        path:

            #define HASPERSDC "/tmp/.lsof_%u_%l_pers%0%h/.lsof_%L"

        When the lsof process is setuid-root or its real UID is
        root, presuming root's home directory is `/' and the host's
        name is ``lsof.itap.purdue.edu'', the extended format yields:

            /.lsof_vic

18.3  Why doesn't lsof know about AFS files on my favorite dialect?

        Lsof currently supports AFS for these dialects:

            AIX 4.1.4 (AFS 3.4a)
            Linux 1.2.13 (AFS 3.3)
            NEXTSTEP 3.2 (AFS 3.3)
            Solaris 2.[56] (AFS 3.4a)

        It may recognize AFS files on other versions of these
        dialects, but I have no way to test that.  Lsof may report
        correct information for AFS files on other dialects, but
        I can't test that either.

        AFS support must be custom crafted for each UNIX dialect
        and then tested.  If lsof supports your favorite dialect,
```

      but doesn't recognize its AFS files, probably I don't have
      access to a test system. If you want AFS support badly
      for your dialect, consider helping me do the development
      and testing.

18.3.1   Why doesn't lsof report node numbers for all AFS volume files,
      or how do I reveal dynamic module addresses to lsof?

      When AFS is implemented via dynamic kernel modules -- e.g.,
      in NEXTSTEP -- lsof can't obtain the addresses of AFS
      variables in the kernel that it uses to identify AFS vnodes.
      It can guess that a vnode is assigned to an AFS file and
      it can obtain other information about AFS files, but it
      has trouble computing AFS volume node numbers.

      To determine node numbers for AFS volumes other than the
      root volume, /afs, lsof needs access to a hashed volume
      structure pointer table. When it can't find the address
      of that table, because AFS support is implemented via
      dynamic kernel modules, lsof will return blanks in the
      INODE column for AFS volume files. Lsof can identify the
      root volume's node number (0), and can compute the node
      numbers for all other AFS files.

      If you have a name list file that contains the addresses
      of the AFS dynamic modules -- e.g., you saved module symbols
      when you created a loadable module kernel with modload(8)
      by specifying -sym -- lsof may be able to find the kernel
      addresses it needs in that file.

      Lsof looks up AFS dynamic kernel addresses for these dialects
      at these default paths:

```
    NEXTSTEP 3.2  /usr/vice/etc/afs_loadable
```

      A different path to a name list file with AFS dynamic kernel
      addresses may be specified with the -A option, when the -A
      option description appears in lsof's -h or -? (help) output.

      If any addresses appear in the -A name list file that also
      appear in the regular kernel name list file -- e.g., /vmunix
      -- they must match, or lsof will silently ignore the -A
      addresses on the presumption that they are out of date.

Making and Installing lsof 4

```
*******************************************************************
| The latest release of lsof is always available via anonymous ftp |
| from lsof.itap.purdue.edu.  Look in pub/tools/unix/lsof.         |
*******************************************************************
```

Contents

=======================

Pre-built Lsof Binaries

=======================

Avoid using pre-built lsof binaries if you can; build your own instead.

When lsof is built its Configure script tunes lsof to the features available on the building system, often embodied in supporting header files and libraries. If the building system doesn't have support for a particular feature, lsof won't be built to support the feature on any system.

The Veritas VxFS file system is a good example of a feature that requires build-time support.

UNIX dialect version differences -- Solaris 8 versus 9, AIX 4.3.3 vesus 5.2, etc. -- can also render a pre-built lsof binary useless on a different version. So can kernel bit size.

There are so many potential pitfalls to using an lsof binary
improperly that I strongly recommend lsof be used only where it is
built.

===========
Making Lsof
===========

```
      $ cd <lsof source directory>
      $ ./Configure <your dialect's abbreviation>
      $ make
```

(Consult the 00FAQ and 00XCONFIG files of the lsof distribution
for information about using make command invocations and environment
variables to override lsof default Makefile strings.)

This lsof distribution can be used with many UNIX dialects. However,
it must be configured specifically for each dialect. Configuration
is done in three ways: 1) by changing definitions in the machine.h
header file of the UNIX dialect of interest; 2) by defining
environment variable values prior to calling Configure (see the
00XCONFIG file, the Environment Variabls and Environment Variables
Affecting the Configure Script sections of this file); and 3) by
running the Configure shell script found in the top level of the
distribution directory.

You may not need to change any machine.h definitions, but you might
want to look at them anyway. Pay particular attention to the
definitions that are discussed in the Security section of this
file. Please read that section.

The Configure script calls three other scripts in the lsof
distribution: AFSConfig; Inventory; and Customize. The AFSConfig
script is called for selected dialects (AIX, HP-UX, NEXTSTEP, and
Solaris) to locate AFS header files and determine the AFS version.
See The AFSConfig Script section of this file for more information.

The Inventory script checks the completeness of the lsof distribution.
Configure calls Inventory after it has accepted the dialect
abbreviation, but before it configures the top-level directory for
the dialect. See The Inventory Script section of this file for
more information.

Configure calls the Customize script after it has configured the
top-level lsof directory for the declared dialect. Customize helps
you modify some of the important compile-time definitions of
machine.h. See the The Customize Script section.

You should also think about where you will install lsof and its
man page, and whom you will let execute lsof. Please read the
Installing Lsof section of this file for information on installation
considerations.

Once you have inspected the machine.h file for the dialect for
which you want to build lsof, and made any changes you need, run
the Configure script, supplying it with the abbreviation for the
dialect. (See the following table.) Configure selects the
appropriate options for the dialect and runs the Mksrc shell script
in the dialect sub-directory to construct the appropriate source
files in the top-level distribution directory.

Configure may also run the MkKernOpts script in the dialect sub-directory to propagate kernel build options to the dialect Makefile. This is done for only a few dialects -- e.g., DC/OSx, and Reliant UNIX.

Configure creates a dialect-specific Makefile. You may want to inspect or edit this Makefile to make it conform to local conventions. If you want the Makefile to install lsof and its man page, you will have to create an appropriate install rule.

Lsof may be configured using UNIX dialect abbreviations from the following table. Alternative abbreviations are indicated by a separating `|'. For example, for SCO OpenServer you can use either the ``osr'' or the ``sco'' abbreviation:

```
      $ Configure osr
    or
      $ Configure sco
```

Abbreviations	UNIX Dialect
aix	IBM AIX 4.3.2, 5L, and 5.[12] using IBM's C Compiler
aixgcc	IBM AIX 4.3.2 and 5.[12] using gcc
bsdi	BSDI BSD/OS 4.[13]
darwin	Apple Darwin 6.x and 7.x for Power Macintosh systems
decosf	DEC OSF/1, Digital UNIX, Tru64 UNIX 4.0 and 5.[01]
digital_unix	Digital UNIX, DEC OSF/1, Tru64 UNIX 4.0 and 5.[01]
du	Digital UNIX, DEC OSF/1, Tru64 UNIX 4.0 and 5.[01]
freebsd	FreeBSD 4.[2-9], 4.10 and 5.[012]
hpux	HP-UX 11.00 and 11.11, using HP's C Compiler, both /dev/kmem-based and PSTAT-based
hpuxgcc	HP-UX 11.00 and 11.11, using gcc, both /dev/kmem-based and PSTAT-based
linux	Linux 2.1.72 and above for x86-based systems
netbsd	NetBSD 1.[456] and 2.0
next	NEXTSTEP 3.[13]
nextstep	NEXTSTEP 3.[13]
ns	NEXTSTEP 3.[13]
nxt	NEXTSTEP 3.[13]
openbsd	OpenBSD 2.[89] and 3.[012345]
openstep	OPENSTEP 4.x
openunix	Caldera OpenUNIX 8
os	OPENSTEP 4.x
osr	SCO OpenServer Release 5.0.[46], using the C compiler from the SCO developer's kit
osrgcc	SCO OpenServer Release 5.0.[46], using gcc
ou	Caldera OpenUNIX 8
sco	SCO OpenServer Release 5.0.[46], using the C compiler from the SCO developer's kit
scogcc	SCO OpenServer Release 5.0.[46], using gcc
solaris	Solaris 2.x, 7, 8, 9 and 10 using gcc
solariscc	Solaris 2.x, 7, 8, 9 and 10 using Sun's cc
tru64	Tru64 UNIX, DEC OSF/1, Digital UNIX 4.0 and 5.[01]
unixware	SCO|Caldera UnixWare 7.1.[134]
uw	SCO|Caldera UnixWare 7.1.[134]

If you have an earlier version of a dialect not named in the above list, lsof may still work on your system. I have no way of testing that myself. Try configuring for the named dialect -- e.g., if you're using Solaris 2.1, try configuring for Solaris 2.5.1.

After you have configured lsof for your UNIX dialect and have selected options via the Customize script (See the The Customize Script section.) , use the make command to build lsof -- e.g.,

```
$ make
```

Other Configure Script Options

There are three other useful options to the Configure script besides the dialect abbreviation:

```
      -clean          may be specified to remove all traces of
                  a dialect configuration, including the
                  Makefile, symbolic links, and library files.

      -h              may be specified to obtain a list of
      -help       Configure options, including dialect
                  abbreviations.

      -n              may be specified to stop the Configure
                  script from calling the Customize and
                  Inventory scripts.

                  Caution: -n also suppresses the AFSConfig
                  step.
```

Environment Variables

Lsof configuration, building, and execution may be affected by environment variable settings. See the Definitions That Affect Compilation section in the 00PORTING file, the General Environment Variables section in the 00XCONFIG file, the Dialect-Specific Environment Variables section in the 00XCONFIG file, and the Environment Variables Affecting the Configure Script section of this file for more information.

Note in the General Environment Variables section of the 00XCONFIG file that there are five environment variables that can be used to pre-define values in lsof's -v output: LSOF_BLDCMT, LSOF_HOST, LSOF_LOGNAME, LSOF_SYSINFO, and LSOF_USER.

Security

If the symbol HASSECURITY is defined, a security mode is enabled, and lsof will allow only the root user to list all open files. Non-root users may list only open files whose processes have the same user ID as the real user ID of the lsof process (the one that its user logged on with).

However, if HASNOSOCKSECURITY is also defined, anyone may list anyone else's open socket files, provided their listing is enabled with the "-i" option.

Lsof is distributed with the security mode disabled -- HASSECURITY is not defined. (When HASSECURITY is not defined, the definition of HASNOSOCKSECURITY has no meaning.) You can enable the security mode by defining HASSECURITY in the Makefile or in the machine.h header file for the specific dialect you're using -- e.g. dialects/aix/machine.h.

The Customize script, run by Configure when it has finished its
work, gives you the opportunity to define HASSECURITY and
HASNOSOCKSECURITY. (See the The Customize Script section.)

The lsof -h output indicates the state HASSECURITY and HASNOSOCKSECURITY
had when lsof was built, reporting:

```
    "Only root can list all files;"
      if HASSECURITY was defined and HASNOSOCKSECURITY wasn't
      defined;

    "Only root can list all files, but anyone can list socket files."
      if HASSECURITY and HASNOSOCKSECURITY were both defined;

    "Anyone can list all files;"
      if HASSECURITY wasn't defined.  (The definition of
      HASNOSOCKSECURITY doesn't matter when HASSECURITY isn't
      defined.)
```

You should carefully consider the implications of using the default
security mode. When lsof is compiled in the absence of the
HASSECURITY definition, anyone who can execute lsof may be able to
see the presence of all open files. This may allow the lsof user
to observe open files -- e.g., log files used to track intrusions
-- whose presence you would rather not disclose.

All pre-compiled binaries on lsof.itap.purdue.edu and mirrored from
it were constructed without the HASSECURITY definition.

As distributed, lsof writes a user-readable and user-writable device
cache file in the home directory of the real user ID executing
lsof. There are other options for constructing the device cache file
path, and they each have security implications.

The 00DCACHE file in the lsof distribution discusses device cache
file path construction in great detail.   It tells how to disable
the various device cache file path options, or how to disable the
entire device cache file feature by removing the HASDCACHE definition
from the dialect's machine.h file. There is also information on
the device cache file feature in the 00FAQ file. (The 00DCACHE
and 00FAQ files are part of the lsof distribution package.)

The Customize script, run by Configure after it has finished its
work, gives you the opportunity to change the compile-time options
related to the device cache file. (See The Customize Script
section.)

Since lsof may need setgid or setuid-root permission (See the Setgid
Lsof Dialects and Setuid-root Lsof Dialects sections.), its security
should always be viewed with skepticism. Lest the setgid and
setuid-root permissions allow lsof to read kernel name list or
memory files, declared with the -k and -m options, that the lsof
user can't normally access, lsof uses access(2) to establish its
real user's authority to read such files when it can't surrender
its power before opening them. This change was added at the
suggestion of Tim Ramsey.

Lsof surrenders setgid permission on most dialects when it has
gained access to the kernel's memory devices. There are exceptions
to this rule, and some lsof implementations need to run setuid-root.
(The Setgid Lsof Dialects and Setuid-root Lsof Dialects sections
contains a list of lsof implementations and the permissions
recommended in the distribution's Makefiles.)

The surrendering of setgid permission is controlled by the WILLDROPGID definition in the dialect machine.h header files.

In the end you must judge for yourself and your installation the risks that lsof presents and restrict access to it according to your circumstances and judgement.

Run-time Warnings

Lsof can issue warning messages when it runs -- e.g., about the state of the device cache file, about an inability to access an NFS file system, etc. Issuance of warnings are enabled by default in the lsof distribution.

Issuance or warnings may be disabled by default by defining WARNINGSTATE in the dialect's machine.h. The Customize script may also be used to change the default warning message issuance state. (See The Customize Script section.)

The ``-w'' option description of the ``-h'' option (help) output will indicate the default warning issuance state. Whatever the state may be, it can be reversed with ``-w''.

Device Access Warnings

When lsof encounters a /dev (or /devices) directory, one of its sub-directories, or one of their files that it cannot access with opendir(3) or stat(2), it issues a warning message and continues. Lsof will be more likely to issue such a warning when it has been installed with setgid(<some group name>) permission; it won't have trouble if it has been installed with setuid(root) permission or is being run under the root login.

The lsof caller can inhibit or enable the warning with the -w option, depending on the issuance state of run-time warnings. (See the Run-time Warnings section.)

The warning messages do not appear when lsof obtains device information from a device cache file that it has built and believes to be current or when warning message issuance is disabled by default. (See the "Caches -- Name and Device" section for more information on the device cache file.)

The lsof builder can inhibit the warning by disabling the definition of WARNDEVACCESS in the dialect's machine.h or disable all warnings by defining WARNINGSTATE. WARNDEVACCESS is defined by default for most dialects. However, some dialects have some device directory elements that are private -- e.g., HP-UX -- and it is more convenient for the lsof user if warning messages about them are inhibited.

Output from lsof's -h option indicates the status of WARNDEVACCESS. If it was defined when lsof was compiled, this message will appear:

```
    /dev warnings = enabled
```

If WARNDEVACCESS was not defined when lsof was compiled, this message will appear instead:

```
    /dev warnings = disabled
```

The Customize script, run by Configure after it has finished its work, gives you the opportunity to change the WARNDEVACCESS definition. (See The Customize Script section.)

NFS Blocks

Lsof is susceptible to NFS blocks when it tries to lstat() mounted file systems and when it does further processing -- lstat() and readlink() -- on its optional file and file system arguments.

Lsof tries to avoid being stopped completely by NFS blocks by doing the lstat() and readlink() functions in a child process, which returns the function response via a pipe. The lsof parent limits the wait for data to arrive in the pipe with a SIGALRM, and, if the alarm trips, terminates the child process with a SIGINT and a SIGKILL.

This is as reliable and portable a method for breaking NFS deadlocks as I have found, although it still fails under some combinations of NFS version, UNIX dialect, and NFS file system mount options. It generally succeeds when the "intr" or "soft" mount options are used; it generally fails when the "hard" mount option is used.

When lsof cannot kill the child process, a second timeout causes it to stop waiting for the killed child to complete. While the second timeout allows lsof to complete, it may leave behind a hung child process. Unless warnings are inhibited by default or with the -w option, lsof reports the possible hung child.

NFS block handling was updated with suggestions made by Andreas Stolcke. Andreas suggested using the alternate device numbers that appear in the mount tables of some dialects when it is not possible to stat(2) the mount points.

The -b option was added to direct lsof to avoid the stat(2) and readlink(2) calls that might block on NFS mount points and always use the alternate device numbers. If warning message issuance is enabled and you don't want warning messages about what lsof is doing, use the -w option, too.

The -O option directs lsof to avoid doing the potentially blocking operations in child processes. Instead, when -O is specified, lsof does them directly. While this consumes far less system overhead, it can cause lsof to hang, so I advise you to use -O sparingly.

Caches -- Name and Device

Robert Ehrlich suggested that lsof obtain path name components for open files from the kernel's name cache. Where possible, lsof dialect implementations do that. The -C option inhibits kernel name cache examination.

Since AFS apparently does not use the kernel's name cache, where lsof supports AFS it is unable to identify AFS files with path name components.

Robert also suggested that lsof cache the information it obtains via stat(2) for nodes in /dev (or /devices) to reduce subsequent running time. Lsof does that, too.

In the default distribution the device cache file is stored in .lsof_hostname, mode 0600, in the home directory of the login of the user ID that executes lsof. The suffix, hostname, is the first component of the host's name returned by gethostname(2). If lsof is executed by a user ID whose home directory is NFS-mounted from several hosts, the user ID's home directory may collect several device cache files, one for each host from which it was executed.

Lsof senses accidental or malicious damage to the device cache file with extensive integrity checks, including the use of a 16 bit CRC. It also tries to sense changes in /dev (or /devices) that indicate the device cache file is out of date.

There are other options for forming the device cache file path. Methods the lsof builder can use to control and employ them are documented in the separate 00DCACHE file of the lsof distribution.

Raw Sockets

On many UNIX systems raw sockets use a separate network control block structure. Display of files for applications using raw sockets -- ping, using ICMP, for example -- need special support for displaying their information. This support is so dialect-specific and information to provide it so difficult to find that not all dialect revisions of lsof handle raw sockets completely.

Other Compile-time Definitions

The machine.h and dlsof.h header files for each dialect contains definitions that affect the compilation of lsof. Check the Definitions That Affect Compilation section of the 00PORTING file of the lsof distribution for their descriptions. (Also see The Customize Script section.)

The AFSConfig Script

Lsof supports AFS on some combinations of UNIX dialect and AFS version. See the AFS section of this document for a list of supported combinations.

When configuring for dialects where AFS is supported, the Configure script calls the AFSConfig script to determine the location of AFS header files and the AFS version. Configure will not call AFSConfig, even for the selected dialects, unless the file /usr/vice/etc/ThisCell exists.

The AFS header file location is recorded in the AFSHeaders file; version, AFSVersion. Once these values have been recorded, Configure can be told to skip the calling of AFSConfig by specifying its (Configure's) -n option.

The Inventory Script

The lsof distribution contains a script, called Inventory, that checks the distribution for completeness. It uses the file 00MANIFEST in the distribution as a reference point.

After the Configure script has accepted the dialect abbreviation, it normally calls the Inventory script to make sure the distribution is complete.

After Inventory has run, it creates the file ".ck00MAN" in the top-level directory to record for itself the fact that the inventory has been check. Should Inventory be called again, it senses this file and asks the caller if another check is in order, or if the check should be skipped.

The -n option may be supplied to Configure to make it bypass the calling of the Inventory script. (The option also causes Configure to avoid calling the Customize script.)

The lsof power user may want to define (touch) the file ".neverInv". Configure avoids calling the Inventory script when ".neverInv" exists.

The Customize Script

Normally when the Configure script has finished its work, it calls another shell script in the lsof distribution called Customize. (You can tell Configure to bypass Customize with its -n option.)

Customize leads you through the specification of these important compile-time definitions for the dialect's machine.h header file:

HASDCACHE	device cache file control
HASENVDC	device cache file environment variable name
HASPERSDC	personal device cache file path format
HASPERSDCPATH	name of environment variable that provides an additional component of the personal device cache file path
HASSYSDC	system-wide device cache file path
HASKERNIDCK	the build-time to run-time kernel identity check
HASSECURITY	the security option
HASNOSOCKSECURITY	the open socket listing option whe HASSECURITY is defined
WARNDEVACCESS	/dev (or /devices) warning message control
WARNINGSTATE	warning message issuance state

The Customize script accompanies its prompting for entry of new values for these definitions with brief descriptions of each of them. More information on these definitions may be found in this file or in the 00DCACHE and 00FAQ files of the lsof distribution.

You don't need to run Customize after Configure. You can run it later or you can edit machine.h directly.

The -n option may be supplied to Configure to make it bypass the calling of the Customize script. (The option also causes Configure to avoid calling the Inventory script.)

The lsof power user may want to define (touch) the file ".neverCust". Configure avoids calling the Customize script when ".neverCust" exists.

Customize CAUTION: the Customize script works best when it is
applied to a newly configured lsof source base -- i.e., the machine.h
header file has not been previously modified by the Customize
script. If you have previously configured lsof, and want to rerun
the Customize script, I recommend you clean out the previous
configuration and create a new one:

```
      $ Configure -clean
      $ Configure <dialect_abbreviation>
      ...
      Customize in response to the Customize script prompts.
```

Cautions

Lsof is a tool that is closely tied to the UNIX operating system
version. It uses header files that describe kernel structures and
reads kernel structures that typically change from OS version to
OS version, and even within a version as vendor patches are applied.

DON'T TRY TO USE AN LSOF BINARY, COMPILED FOR ONE UNIX OS VERSION,
ON ANOTHER. VENDOR PATCHES INFLUENCE THE VERSION IDENTITY.

On some UNIX dialects lsof versions may be even more restricted by
architecture type.

The bottom line is use lsof where you built it. If you intend to
use a common lsof binary on multiple systems, make sure all systems
run exactly the same OS version and have exactly the same patches.

Warranty

Lsof is provided as-is without any warranty of any kind, either
expressed or implied, including, but not limited to, the implied
warranties of merchantability and fitness for a particular purpose.
The entire risk as to the quality and performance of lsof is with
you. Should lsof prove defective, you assume the cost of all
necessary servicing, repair, or correction.

License

Lsof has no license. Its use and distribution are subject to these
terms and conditions, found in each lsof source file. (The copyright
year in or format of the notice may vary slightly.)

```
    /*
     * Copyright 2002 Purdue Research Foundation, West Lafayette,
     * Indiana 47907.  All rights reserved.
     *
     * Written by Victor A. Abell
     *
     * This software is not subject to any license of the American
     * Telephone and Telegraph Company or the Regents of the
     * University of California.
     *
     * Permission is granted to anyone to use this software for
     * any purpose on any computer system, and to alter it and
     * redistribute it freely, subject to the following
     * restrictions:
     *
     * 1. Neither the authors nor Purdue University are responsible
     *    for any consequences of the use of this software.
     *
     * 2. The origin of this software must not be misrepresented,
     *    either by explicit claim or by omission.  Credit to the
     *    authors and Purdue University must appear in documentation
     *    and sources.
     *
     * 3. Altered versions must be plainly marked as such, and must
     *    not be misrepresented as being the original software.
     *
     * 4. This notice may not be removed or altered.
     */
```

Bug Reports

Now that the obligatory disclaimer is out of the way, let me hasten to add that I accept lsof bug reports and try hard to respond to them. I will also consider and discuss requests for new features, ports to new dialects, or ports to new OS versions.

PLEASE DON'T SEND BUG REPORTS ABOUT LSOF TO THE UNIX DIALECT VENDOR.

At worst such bug reports will confuse the vendor; at best, the vendor will forward the bug report to me.

PLEASE DON'T SEND BUG REPORTS ABOUT LSOF BINARIES BUILT OR DISTRIBUTED BY SOMEONE ELSE, BECAUSE I CAN'T SUPPORT THEM.

I do support binaries I built, obtained ONLY from lsof.itap.purdue.edu.

Before you send me a bug report, please do these things:

* Make sure you try the latest lsof revision.

  + Download the latest revision from:

    ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof

  + While connected to lsof.itap.purdue.edu, check for patches:

    ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/patches

  + If patches exist, install them in the latest revision you just downloaded. Then build the latest revision and see if it fixes your bug.

* Check the lsof frequently asked questions file, 00FAQ, to see if there's a question and answer relevant to your problem.

When you send a bug report, make sure you include output from your running of lsof's Configure script. If you were able to compile a running lsof, please include output from its -h and -v options.

If you weren't able to compile a running lsof, please send me: the compiler error output; identification of the lsof revision you're using (contents of the lsof version.c file); identification of your system (full uname output or output from whatever other tool identifies the system); and compiler identification (e.g., gcc -v output).

Either set of output will help me understand how lsof was configured
and what UNIX dialect and lsof revision is involved.

Please send all bug reports, requests, etc. to me via e-mail at
<abe@purdue.edu>. Make sure "lsof" appears in the "Subject:" line so
my e-mail filter won't classify your letter as Spam.

The 00FAQ File

The lsof distribution contains an extensive frequently asked
questions file on lsof features and problems. I recommend you
consult it before sending me e-mail. Use your favorite editor or
pager to search 00FAQ -- e.g., supplying as a search argument some
fixed text from an lsof error message.

The lsof-l Mailing List

Information about lsof, including notices about the availability
of new revisions, may be found in mailings of the lsof-l listserv.
For more information about it, including instructions on how to
subscribe, read the 00LSOF-L file of the lsof distribution.

Field Output Example Scripts

Example AWK and Perl 4 or 5 scripts for post-processing lsof field
output are locate in the scripts sub-directory of the lsof distribution.
The scripts sub-directory contains a 00README file with information
about the scripts.

Field Output C Library

The lsof test suite (See "Testing Lsof."), checks basic lsof
operations using field output. The test suite has its own library
of C functions for common test program operations, including
processing of field output. The library or selections of its
functions could be adapted for use by C programs that want to
process lsof field output. See the library in the file LTlib.c
in the tests/ sub-directory

Testing Lsof

Lsof has an automated test suite in the tests/ sub-directory that
can be used to test some basic lsof features -- once lsof has been
configured and made. Tests are arranged in three groups: basic
tests that should run on all dialects; standard tests that should
run on all dialects; and optional tests that may not run on all
dialects or may need special resources to run. See 00TEST for more
information.)

CAUTION!!! Before you attempt to use the test suite make sure that
the lsof you want to test can access the necessary kernel resources
-- e.g., /dev/mem, /dev/kmem, /proc, etc. Usually you want to test
the lsof you just built, so this is an important check. (See

00TEST.)

To run the basic and standard tests, using the lsof in the parent
directory of tests/, do this:

```
      $ cd tests
      $ make test
    or      $ make std
    or      $ make standard
```

The basic and standard tests may be run as silently as possible,
using the lsof in the parent directory of tests/, with:

```
      $ cd tests
      $ make auto
```

This is the "automatic" test mode, designed for use by scripts that
build lsof. The caller is expected to test the make exit code to
determine if the tests succeeded. The caller should divert standard
output and standard error to /dev/null to suppress make's error
exit message.

The optional tests may be run, using the lsof in the parent directory
of tests/, with:

```
      $ cd tests
      $ make opt
    or      $ make optional
```

It's possible to excute individual tests, too. See the 00TEST file
of this distribution for more informaiton on the tests, what they
do, and how to run and possibly customize each test.

It's possible to run the tests, using an lsof other than the one
in the parent directory of /tests, too. See 00TEST for information
about using the LT_LSOF_PATH environment variable to do that.

Dialect Notes

AFS

Lsof recognizes AFS files on the following combinations of UNIX
dialect and AFS versions:

```
      AIX 4.1.4 (AFS 3.4a)
      Linux 1.2.13 (AFS 3.3)
      NEXTSTEP 3.2 (AFS 3.3) (untested on recent lsof revisions)
      Solaris 2.6 (AFS 3.4a)
      Ultrix 4.2 RISC (AFS 3.2b) (no longer available)
```

Lsof has not been tested under other combinations -- e.g. HP-UX
10.10 and AFS 3.4a -- and probably won't even compile there. Often
when a UNIX dialect version or AFS version changes, the new header
files come into conflict, causing compiler objections.

AIX

Specify the aix Configure abbreviation for AIX 4.1.[45], 4.2[.1], 4.3[.123], 5L, and 5.[12].

Specify aixgcc on AIX above 4.1 to use the gcc compiler. (Gcc can't be used to compile lsof on AIX 4.1 and below because of kernel structure alignment differences between it and xlc.) Gcc results sometimes depend on the version of the gcc compiler that is used.

Compilation of lsof with gcc on AIX 4.3[.123], 5L, and 5.[12] has been sparsely tested with varying degrees of success: it has been reported to succeed on AIX 4.3.3 and 32 bit Power AIX 5.1; to fail on ia64 AIX 5.1 and 64 bit Power AIX 5.1; and to succeed on 32 and 64 bit Power AIX 5.2.

At revision 4.61 and above lsof is configured and built to match the bit size of the kernel of Power architecture AIX 5.1 systems. Lsof binaries built for 32 and 64 bit kernels are not interchangeable. See 00FAQ for more information.

The Configure script uses /usr/bin/oslevel to determine the AIX version. If /usr/bin/oslevel isn't executable, the Configure script issues a warning message and uses ``uname -rv'' to determine the AIX version.

When Configure must use ``uname -rv'' to determine the AIX version, the result will lack a correct third component -- e.g., the `4' of ``4.1.4''. If your AIX system lacks lacks an executable oslevel, I suggest you edit the Configure-produced Makefile and complete the _AIXV definition in the CFGF string.

By default lsof avoids using the kernel's readx() function, causing it to be unable to report information on some text and library file references. The ``-X'' option allows the lsof user to ask for the information readx() supplies.

Lsof avoids readx() to avoid the possibility of triggering a kernel problem, known as the Stale Segment ID kernel bug. Kevin Ruderman reported this bug to me. The bug shows up when the kernel's dir_search() function hangs, hanging the application process that called it so completely that the application process can neither be killed nor stopped. The hang is the consequence of another process (perhaps lsof) making legitimate use of the kernel's readx() function to access the kernel memory that dir_search() is examining. IBM has indicated they have no plans to fix the bug.

A fuller discussion of this bug may be found in the 00FAQ file of the lsof distribution. There you will find a description of the Stale Segment ID bug, the APAR on it, and a discussion of the sequence of events that exposes it.

I added the ``-X'' function so you can tell lsof to use readx(), but if you use ``-X'', you should be alert to its possibly serious side effects. Although readx() is normally disabled, its state is controlled with the HASXOPT, HASXOPT_ROOT, and HASXOPT_VALUE definitions in dialects/aix/machine.h, and you can change its default state by changing those definitions. You can also change HASXOPT_ROOT via the Customize script.

You can also compile lsof with readx() use permanently enabled or disabled -- see the comments about the definitions in the dialects/aix/machine.h header file. You may want to permanently disable lsof's use of readx() if you plan to make lsof publicly executable. You can also restrict -X to processes whose real UID is root by defining HASXOPT_ROOT.

I have never seen lsof cause the Stale Segment ID bug to occur and haven't had a report that it has, but I believe there is a possibility it could.

AFS support for AIX was added with help help from Bob Cook and Jan Tax who provided test systems.

Henry Grebler and David J. Wilson helped with lsof for AIX 4.2.

Bill Pemberton provided an AIX 4.3 test system. Andrew Kephart and Tom Weaver provided AIX 4.3 technical assistance.   Niklas Edmundsson did 4.3.1 testing. Doug Crabill provided an AIX 4.3.2 test system. Jeff W. Stewart provided an AIX 4.3.3 test system.

The SMT file type for AIX 4.1.[45], 4.2[.1], and 4.3[.12] is my fabrication. See the 00FAQ file more information on it.

Loc Le and Nasser Momtaheni of IBM provided test systems for AIX 5L and 5.1. Lsof for AIX 5L and 5.1 needs setuid-root permission to process the -X option on systems whose architecture type is ia64.

Dale Talcott of Purdue has provided AIX 5.1 and 5.2 test systems.

Apple Darwin

The Apple Darwin port was provided by Allan Nathanson for version 1.2. Allan also arranged for access to a test system for maintenance and regression testing. Dale Talcott provided a test system, too. Allan provided 1.4 changes.

Allan supplied patches for updates to 1.4, 5.x, 6.x and 7.0.

BSDI BSD/OS

Terry Kennedy provided a 2.1 test system so that support for BSDI BSD/OS could be revived. (BSDI BSD/OS support was dropped at from version 3 at revision 3.21 when a test system was no longer available.) Terry has also provided 3.0, 3.1, 4.1 and 4.3 test systems.

Jim Reid helped with the 3.0 port and Terry Kennedy provided a test system.

Jeffrey C. Honig packaged lsof for inclusion on the BSDI user-contributed software CD.

DEC OSF/1, Digital UNIX, Tru64 UNIX

Robert Benites, Dean Brock, Angel Li, Dwight McKay, Berkley Shands, Ron Young and Steve Wilson have kindly provided test systems. Jeffrey Mogul has provided technical assistance. Dave Morrison and Lawrence MacIntyre did Digital UNIX V3.2 testing.

Lsof supports the ADVFS/MSFS layered file system product. Lsof can locate all the open files of an ADVFS/MSFS file system when its path is specified, provided the file system is listed in

/etc/fstab with an ``advfs'' type. (This /etc/fstab caveat applies
only to Digital UNIX 2.0.) At Digital UNIX 4.0 and Tru64 UNIX,
using code provided by David Brock, lsof 4.20 and above can locate
ADVFS file paths.

FreeBSD

Bill Bormann of Purdue University provided access to several FreeBSD
test systems. Ade Barkah, John Clear, Ralph Forsythe, Michael
Haro, Kurt Jaeger, and William McVey have also provided FreeBSD
test systems.

The FreeBSD distribution header files are augmented by header files
in the dialects/freebsd/include directory.

David O'Brien maintains the lsof FreeBSD port package.

HP-UX

Lsof has two HP-UX bases: /dev/kmem for HP-UX 11.0 and earlier;
and PSTAT for HP-UX 11.11 and later. The lsof Configure script
will pick the appropriate base.

To use the CCITT x.25 socket support for HP-UX, you must have the
x.25 header files in /etc/conf/x25

Pasi Kaara helped with the HP-UX port, especially with its CCITT
x.25 socket support.

Richard Allen provided HP-UX 10.x and 11.x test systems, as did
Mark Bixby, and Elias Halldor Agustsson.   Marc Winkler helped test
the 10.20 port. Richard J. Rauenzahn provided a 64 bit HP-UX 11
test system and an HP-UX 11.11 development system.

AFS support for HP-UX was added thanks to help from Chaskiel Moses
Grundman, who provided a test system.

The /dev/kmem-based HP-UX 11.00 support is extremely fragile. It
depends on privately developed kernel structure definitions. (See
.../dialects/hpux/hpux11 for the header files making the definitions.)
Those header files and their definitions will not be updated by
HP-UX 11.00 patches, making it likely that any patch changing a
kernel structure critical to lsof will break lsof in some way.

It's possible to build a 64 bit lsof for 64 bit HP-UX 11.00 with
gcc, but you must have a gcc compiler capable of producing 64 bit
executables. See the 00FAQ file for more information.

The PSTAT-based lsof for HP-UX 11.11 and later is much more solid.
I am indebted to the vision of HP for providing an lsof kernel API
through the PSTAT implementation. Specifically I appreciate the
help of HP staff members Carl Davidson, Louis Huemiller, Rich
Rauenzahn, and Sailu Yallapragada that made PSTAT-based HP-UX lsof
possible.

IPv6

Lsof has IPv6 support that has been tested for these UNIX dialects:

AIX 4.3.x; Apple Darwin 5.[12] and 6.0; the INRIA and KAME FreeBSD IPv6 implementations; /proc-based Linux; the INRIA and KAME NetBSD implementations; and Solaris 8 and 9. Lsof has IPv6 support that hasn't been tested for: BSDI BSD/OS4.x; OpenBSD (KAME); OpenUNIX 8; Tru64 Unix 5.[01]; and UnixWare 7.1.[34].

Please let me know if your UNIX dialect has IPv6 support and I'll see if it can be supported by lsof.

Linux

Tim Korb, Steve Logue, Joseph J. Nuspl Jr., and Jonathan Sergent have provided Linux test systems.

Michael Shields helped add and test automatic handling of ELF/COFF form names in /System.map, Marty Leisner and Keith Parks have helped test many lsof revisions. Marty has provided valuable suggestions, Linux hints, and code, too.

The 00FAQ file gives some Linux tips, including information on coping with system map file problems.

To determine the state of the Linux 2.1.x C library lseek() function, the lsof Configure script runs a test program that must have permission to read /dev/kmem. The test determines if the lseek() function properly handles kernel offsets, which appear to be negative because their high order bit is set. If the lseek() test reveals a faulty lseek(), Configure activates the use of a private lseek() function for kernel offset positioning. See the Linux problems section of the 00FAQ file of the lsof distribution for more information.

NetBSD

Greg Earle and Paul Kranenburg have assisted with the NetBSD ports. Paul has provided test systems. Ray Phillips provided a NetBSA Alpha test system. Andrew Brown also provided a test system.

The NetBSD dialect version of lsof is compiled using the dialect sources it shares with OpenBSD in the n+obsd dialect sub-directory.

NEXTSTEP and OPENSTEP

Virtual memory header files that allow lsof to display text references were derived from the contents of /usr/include/vm of NEXTSTEP 2.0. NeXT did not ship the virtual memory header files with other NEXTSTEP or OPENSTEP versions.

You may use the RC_FLAGS environment variable to declare compiler options outside the Makefile. A common use of this variable is to define the architecture types to be included in a "fat" executable. See the comments in dialects/next/Makefile for an example.

OpenBSD

David Mazieres has provided OpenBSD test systems. The OpenBSD

dialect version of lsof is compiled using the dialect sources it
shares with NetBSD in the n+obsd dialect sub-directory.

Kenneth Stailey has provided OpenBSD testing and advice.

John Dzubera (Zube) reports, "lsof 4.33 compiles and runs on OpenBSD
2.3 for the pmax architecture (decstation 3100)."

Pyramid DC/OSx and Reliant UNIX

As of lsof revision 4.52 support for all Pyramid dialects has been
discontinued. Lsof revision 4.51 with Pyramid support may be
obtained upon request. Send the request to abe@purdue.edu.

These two UNIX dialects are very similar and share dialect-specific
source files from the pyramid sub-directory.

The Reliant Unix Pyramid C compiler issues warning messages that
I haven't found a convenient way to suppress. You can ignore
warning messages about casts and conversions that lose bits. The
message "warning: undefining __STDC__" is intentionally caused by
the lsof MkKernOpts configuration script to suppress warning messages
about cast and conversion problems in standard system header files,
such as <stdio.h> and <string.h>.

Bruce Beare and Kevin Smith provided test systems.

Caldera OpenUNIX

Larry Rosenman provided an OpenUNIX 8 test system. Matthew Thurmaier
provided technical assistance, along with these people from Caldera:
Jack Craig, Robert Lipe, and Bela Lubkin.

Robert Lipe supplied changes to lsof for OpenUNIX 8.0.1. Those
changes were also incorporated in UnixWare 7.1.3 when it became
the release name for OpenUNIX 8.0.1.

SCO OpenServer

Dion Johnson, Bela Lubkin, and Nathan Peterson of SCO gave me copies
of SCO OpenServer and the SCO OpenServer Development System 3.0
and provided technical advice for the lsof port.

Hugh Dickins, Bela Lubkin, Craig B. Olofson, and Nathan Peterson
provided version 5.0 and gave technical advice for porting lsof to
it. Bela provided the 5.0.4 changes. D. Chris Daniels provided
a 5.0.4 test system, Lee Penn provided one for 5.0.5, and John
Dubois for 5.0.6.

The <netdb.h> header file was accidentally omitted from some SCO
OpenServer Development System releases. The Configure script will
sense its absence and substitute an equivalent from the BSD
distribution. The BSD <netdb.h> and the <sys/cdefs.h> header file
it includes are located in the dialects/os/include sub-directory
tree.

To compile lsof from its distribution sources you must have the
TCP/IP and NSF headers in /usr/include. While those are optional

OpenServer packages, I have access to no system that doesn't have them, so I'm unable to build lsof for such a configuration. However, it should be possible to modify the lsof Configure script and sources so lsof would compile and work without those optional packages.

If you have an OpenServer system configured without the TCP/IP and NFS packages, and want to tackle the job of building lsof for it, contact me via e-mail at <abe@purdue.edu>. I'll identify the Configure script, header file, and source file changes you will need to make. (Caution: this is not a simple task, or I would have already done it.)

The optional osrgcc and scogcc Configure abbreviations construct Makefiles for compiling lsof with gcc.

SCO|Caldera UnixWare

D. Chris Daniels, John Hughes, Ken Laing, Andrew Merril, Lee Penn, and Matthew Thurmaier provided test systems. Bela Lubkin provided technical assistance. Larry Rosenman provided 7.1.[34] test systems.

Solaris 2.x, 7, 8, 9 and 10

SEE THE CAUTIONS SECTION OF THIS DOCUMENT.

The latest Solaris revision of lsof 4 might work under Solaris 2.[1-4] and 2.5[.1] and 7 but hasn't been tested there. I have no test systems for those Solaris versions.

Lsof will compile with gcc and the Sun C compiler under Solaris. If you want to use the Sun compiler, use the solariscc Configure abbreviation. If you use a gcc version less than 2.8 on Solaris, make sure the gcc-specific includes have been updated for your version of Solaris -- i.e., run the gcc fixincludes script.

Solaris 7, 8, 9 and 10 support for 64 bit kernels depends on a Sun WorkShop or Forte C compiler version that supports the "-xarch=v9" flag -- usually 5.0 or greater. Gcc versions 2.95 and above *may* be configured and built for 64 bit support, but it takes some extra work, the resulting compiler may be fragile, and the gcc developers discourage it. I've built 64 bit capable gcc compilers for Solaris 7, 8 and 9 from gcc versions 2.95 through 3.0.1 and produced working lsof executables with them. More information on 64 bit gcc for Solaris may be found in the 00FAQ file.

Dave Curry and Steve Kirsch provided resources for Solaris 2.x ports. Casper Dik and Gerry Singleton consulted and provided valuable assistance.

Henry Katz, Joseph Kowalski, Charles Stephens, Mike Sullivan, and Mike Tracy provided technical assistance.

AFS support was added to Solaris lsof with help from Curt Freeland, Heidi Hornstein, Michael L. Lewis, Terry McCoy, Phillip Moore, and Sushila R. Subramanian.

Casper Dik provided valuable assistance for the Solaris 8 support.

Sun has graciously provided me access to BETA versions of Solaris

2.5, 2.6, 7, 8, and 9.

John Dzubera provided Solaris 7 and 8 test systems.

Mike Miscevic provided Solaris 10 test systems.

Ultrix

As of lsof revision 4.52 support for Ultrix is no longer available, because I no longer have an Ultrix test system.

Terry Friedrichsen, Dwight McKay, and Jeffrey Mogul helped me with this port.

DECnet support was added to Ultrix lsof with the help of John Beacom, who kindly provided a test system. The Configure script decides that DECnet support is available if /usr/lib/libdnet.a and /usr/include/netdnet/dn.h exist and are readable.

Veritas VxFS and VxVM

Lsof supports some versions of Veritas VxFS and VxVM on some UNIX dialects. Consult the lsof Configure script for the specific dialect, and consult the lsof dialect-specific source files for the UNIX dialect of interest. Veritas support will usually be found in a source file named dnode[1-9].c.

Since Veritas rarely has a version number that can be extracted with shell commands, lsof doesn't use it. Instead, when lsof supports Veritas, the Configure script will form compile-time definitions starting with HASVXFS.   Check the lsof 00PORTING documentation file for more information.

Lsof Veritas support requires that the supporting Veritas header files be installed -- e.g., in /usr/include/sys/fs. (The location will depend in the dialect's header file conventions.)

Some information on lsof support for Veritas extensions may be found in the lsof 00DIST file.

Chris Kordish and Andy Thomas have provided Solaris VxFS test systems.

User-contributed Dialect Support

There are some user-contributed dialect versions of lsof; more information on them can be found at:

      ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/contrib

Check the 00INDEX file there for details.

Dialects No Longer Supported

Because I don't have access to test systems, these UNIX dialects
are no longer supported by lsof:

```
CDC EP/IX
/dev/kmem-based Linux
MIPS RISC/os
Motorola V/88
Pyramid DC/OSx
Pyramid Reliant UNIX
Sequent DYNIX
SGI IRIX
SunOS 4.x
Ultrix
UnixWare below 7.0
```

Remnants of the support lsof once provided for these dialects may
be found in:

```
ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/OLD/binaries
```

and

```
ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/OLD/dialects
```

===============
Installing Lsof
===============

The distributed Makefiles do not have actions that will install
lsof. I've come to the conclusion there is no standard for installing
lsof or its man page, so I no longer distribute make rules for
installing them. You should adjust the Makefile for your local
preferences.

The Makefile does have an install rule that will cause lsof to
compile by virtue of its dependency clause. Some Makefiles also
have a dependency that causes the production of a man page that is
ready to install. However, the actions of the install rule will
not cause the lsof executable or its man page to be installed in
any UNIX system-wide directory.

Instead, after the compilation and optional man page production
are completed, the install rule will produce a brief description
of what actions you might add to the install rule. The description
will suggest the possible modes, ownerships, permissions, and
destinations your install rule might employ to install the lsof
executable and man page.

As you form your install rule, keep in mind that lsof usually needs
some type of special permission to do its job. That may be permission
to read memory devices such as /dev/kmem, /dev/mem, or /dev/swap,
or it may be authorization to read entries in the /proc file system.

Memory device access can usually be provided by setting the modes
of the lsof executable so that it's effective group identifier when
it runs is the same as the group that has permission to read the
memory devices -- i.e., it is setgid-group. The privileged group
is usually kmem, sys, or system.

Don't overlook using ACLs -- e.g., on AIX or Solaris 8 -- to give
lsof permission to access memory devices. ACLs, coupled to a
separate group like kmem, can be safer than giving lsof setgid
authorization to a commonly used system group.

When lsof needs to read /proc file system entries, it must be installed with modes that make its effective user identifier root when it runs -- i.e., it must be setuid-root. If lsof must be installed setuid-root (only the AIX 5L, PSTAT-based HPUX, and /proc-based Linux, ports need such power.), then access to memory devices is automatic (or not needed in the case of /proc-based Linux).

Your choice of permissions for lsof may also be affected by your desire to allow anyone to use it or your need to restrict its usage to specific individuals. You will have to be guided by local policy and convention in this case.

The next two sections, Setgid Lsof Dialect Versions and Setuid-root Lsof Dialect Versions, list recommended install permissions.

The system directory where you install the lsof executable is also open to choice. A traditional place for a tool like lsof is /usr/local/etc, but recent changes in directory structure organization suggest that somewhere in /opt may be more suitable.

Bear one other factor in mind when choosing a location for the lsof executable -- it usually is a shared executable, requiring access to shared libraries. Thus, locations like /sbin or /usr/sbin are probably unsuitable.

Once you've chosen a location for the executable you may find that the location for the man page follows -- e.g., if the executable goes in /usr/local/etc, then the man page goes in /usr/local/man. If the executable location doesn't imply a location for the man page, you'll have to let local custom guide you.

Setuid-root Lsof Dialect Versions

These dialect versions should be installed with setuid-root permission -- i.e., the lsof binary should be owned by root and its setuid execution bit (04000) should be set.

```
      AIX 5L and above for full use of the -X option
      PSTAT-based HP-UX 11.11
      /proc-based Linux (generally 2.1.72 and above)
```

Setgid Lsof Dialect Versions

These dialect versions should be installed with setgid permission, owned by the group that can read kernel memory devices such as /dev/drum, /dev/kmem, /dev/ksyms, /dev/mem, /dev/swap. ACLs may be another mechanism (e.g., under AIX or Solaris 8) you can use to grant read permission to the kernel memory devices.

```
      AIX 4.1.[45], 4.2[.1], and 4.3[.123]
      BSDI BSD/OS 2.1, 3.[01], and 4.[013]
      DEC OSF/1, Digital UNIX, Tru64 UNIX 2.0, 3.2, 4.0, and 5.[01]
      FreeBSD 2.1.6, 2.2[.x], 3.[012345], 4.[234567], and 5.0
      /dev/kmem-based 11.00
      NetBSD 1.[456] and 2.0
      NEXTSTEP 3.[13]
      OpenBSD 2.[89] and 3.[012345]
      OPENSTEP 4.x
      Caldera OpenUNIX 8
      SCO OpenServer 5.0.[46]
      SCO UnixWare 7.0 and 7.1.[0134]
      Solaris 2.6, 8, 9 and 10
      Ultrix 4.2 (no longer available)

====================================
Porting lsof 4 to a New UNIX Dialect
====================================

If you're brave enough to consider this, look at the 00PORTING
file.  Please contact me before you start.  I might be able to help
you or even do the port myself.

Don't overlook the contrib/ directory in pub/tools/unix/lsof on my
ftp server, lsof.itap.purdue.edu.  It contains user-contributed ports
of lsof to dialects I don't distribute, because I can't test new
revisions of lsof on them.


=========================
Quick Start to Using lsof
=========================

For information on how to get started quickly using lsof, consult
the 00QUICKSTART file of the lsof distribution.  It cuts past the
formal density of the lsof man page to provide quick examples of
using lsof to solve common open file display problems.


======================
Cross-configuring Lsof
======================

Using environment variables it is possible to Configure (and possibly
build) lsof for one UNIX dialect on a different one -- e.g., you
are running Configure on a Linux 2.3 system and you want to Configure
and build lsof for Linux 2.4.

See the 00XCONFIG file of the lsof distribution for a discussion
of how to do this.


=====================================================
Environment Variables Affecting the Configure Script
=====================================================

Configure script actions can be modified by introducing values to
the script via environment variables.  In many cases the environment
variable values take the place of test operations the Configure
script makes.

For more information on environment variables that can affect
Configure, consult the 00XCONFIG file of the lsof distribution.
See the General Environment Variables sections for descriptions of
ones that affect all dialects.  Consult the Dialect-Specific
Environment Variables section for ones that might affect the dialect
you are trying to configure.


Vic Abell <abe@purdue.edu>
July 6, 2004
```