NAME

perlsolaris - Perl version 5 on Solaris systems

DESCRIPTION

This document describes various features of Sun's Solaris operating system that will affect how Perl version 5 (hereafter just perl) is compiled and/or runs. Some issues relating to the older SunOS 4.x are also discussed, though they may be out of date.

For the most part, everything should just work.

Starting with Solaris 8, perl5.00503 (or higher) is supplied with the operating system, so you might not even need to build a newer version of perl at all. The Sun-supplied version is installed in /usr/perl5 with */usr/bin/perl* pointing to */usr/perl5/bin/perl*. Do not disturb that installation unless you really know what you are doing. If you remove the perl supplied with the OS, you will render some bits of your system inoperable. If you wish to install a newer version of perl, install it under a different prefix from /usr/perl5. Common prefixes to use are /usr/local and /opt/perl.

You may wish to put your version of perl in the PATH of all users by changing the link */usr/bin/perl*. This is probably OK, as most perl scripts shipped with Solaris use an explicit path. (There are a few exceptions, such as */usr/bin/rpm2cpio* and */etc/rcm/scripts/README*, but these are also sufficiently generic that the actual version of perl probably doesn't matter too much.)

Solaris ships with a range of Solaris-specific modules. If you choose to install your own version of perl you will find the source of many of these modules is available on CPAN under the Sun::Solaris:: namespace.

Solaris may include two versions of perl, e.g. Solaris 9 includes both 5.005_03 and 5.6.1. This is to provide stability across Solaris releases, in cases where a later perl version has incompatibilities with the version included in the preceding Solaris release. The default perl version will always be the most recent, and in general the old version will only be retained for one Solaris release. Note also that the default perl will NOT be configured to search for modules in the older version, again due to compatibility/stability concerns. As a consequence if you upgrade Solaris, you will have to rebuild/reinstall any additional CPAN modules that you installed for the previous Solaris version. See the CPAN manpage under 'autobundle' for a quick way of doing this.

As an interim measure, you may either change the #! line of your scripts to specifically refer to the old perl version, e.g. on Solaris 9 use #!/usr/perl5/5.00503/bin/perl to use the perl version that was the default for Solaris 8, or if you have a large number of scripts it may be more convenient to make the old version of perl the default on your system. You can do this by changing the appropriate symlinks under /usr/perl5 as follows (example for Solaris 9):

```
# cd /usr/perl5
# rm bin man pod
# ln -s ./5.00503/bin
# ln -s ./5.00503/man
# ln -s ./5.00503/lib/pod
# rm /usr/bin/perl
# ln -s ../perl5/5.00503/bin/perl /usr/bin/perl
```

In both cases this should only be considered to be a temporary measure - you should upgrade to the later version of perl as soon as is practicable.

Note also that the perl command-line utilities (e.g. perldoc) and any that are added by modules that you install will be under /usr/perl5/bin, so that directory should be added to your PATH.

Solaris Version Numbers.

For consistency with common usage, perl's Configure script performs some minor manipulations on the operating system name and version number as reported by uname. Here's a partial translation

```
table:          Sun:                        perl's Configure:
 uname    uname -r   Name            osname     osvers
 SunOS    4.1.3      Solaris 1.1     sunos      4.1.3
 SunOS    5.6        Solaris 2.6     solaris    2.6
 SunOS    5.8        Solaris 8       solaris    2.8
 SunOS    5.9        Solaris 9       solaris    2.9
 SunOS    5.10       Solaris 10      solaris    2.10
```

The complete table can be found in the Sun Managers' FAQ *ftp://ftp.cs.toronto.edu/pub/jdd/sunmanagers/faq* under "9.1) Which Sun models run which versions of SunOS?".

RESOURCES

There are many, many sources for Solaris information. A few of the important ones for perl:

Solaris FAQ

The Solaris FAQ is available at *http://www.science.uva.nl/pub/solaris/solaris2.html*.

The Sun Managers' FAQ is available at *ftp://ftp.cs.toronto.edu/pub/jdd/sunmanagers/faq*

Precompiled Binaries

Precompiled binaries, links to many sites, and much, much more are available at *http://www.sunfreeware.com/* and *http://www.blastwave.org/*.

Solaris Documentation

All Solaris documentation is available on-line at *http://docs.sun.com/*.

SETTING UP

File Extraction Problems on Solaris.

Be sure to use a tar program compiled under Solaris (not SunOS 4.x) to extract the perl-5.x.x.tar.gz file. Do not use GNU tar compiled for SunOS4 on Solaris. (GNU tar compiled for Solaris should be fine.) When you run SunOS4 binaries on Solaris, the run-time system magically alters pathnames matching m#lib/locale# so that when tar tries to create lib/locale.pm, a file named lib/oldlocale.pm gets created instead. If you found this advice too late and used a SunOS4-compiled tar anyway, you must find the incorrectly renamed file and move it back to lib/locale.pm.

Compiler and Related Tools on Solaris.

You must use an ANSI C compiler to build perl. Perl can be compiled with either Sun's add-on C compiler or with gcc. The C compiler that shipped with SunOS4 will not do.

Include /usr/ccs/bin/ in your PATH.

Several tools needed to build perl are located in /usr/ccs/bin/: ar, as, ld, and make. Make sure that /usr/ccs/bin/ is in your PATH.

On all the released versions of Solaris (8, 9 and 10) you need to make sure the following packages are installed (this info is extracted from the Solaris FAQ):

for tools (sccs, lex, yacc, make, nm, truss, ld, as): SUNWbtool, SUNWsprot, SUNWtoo

for libraries & headers: SUNWhea, SUNWarc, SUNWlibm, SUNWlibms, SUNWdfbh, SUNWcg6h, SUNWxwinc

Additionaly, on Solaris 8 and 9 you also need:

for 64 bit development: SUNWarcx, SUNWbtoox, SUNWdplx, SUNWscpux, SUNWsprox, SUNWtoox, SUNWlmsx, SUNWlmx, SUNWlibCx

And only on Solaris 8 you also need:

for libraries & headers: SUNWolinc

If you are in doubt which package contains a file you are missing, try to find an installation that has that file. Then do a

```
$ grep /my/missing/file /var/sadm/install/contents
```

This will display a line like this:

/usr/include/sys/errno.h f none 0644 root bin 7471 37605 956241356 SUNWhea

The last item listed (SUNWhea in this example) is the package you need.

Avoid /usr/ucb/cc.

You don't need to have /usr/ucb/ in your PATH to build perl. If you want /usr/ucb/ in your PATH anyway, make sure that /usr/ucb/ is NOT in your PATH before the directory containing the right C compiler.

Sun's C Compiler

If you use Sun's C compiler, make sure the correct directory (usually /opt/SUNWspro/bin/) is in your PATH (before /usr/ucb/).

GCC

If you use gcc, make sure your installation is recent and complete. perl versions since 5.6.0 build fine with gcc > 2.8.1 on Solaris >= 2.6.

You must Configure perl with

```
$ sh Configure -Dcc=gcc
```

If you don't, you may experience strange build errors.

If you have updated your Solaris version, you may also have to update your gcc. For example, if you are running Solaris 2.6 and your gcc is installed under /usr/local, check in /usr/local/lib/gcc-lib and make sure you have the appropriate directory, sparc-sun-solaris2.6/ or i386-pc-solaris2.6/. If gcc's directory is for a different version of Solaris than you are running, then you will need to rebuild gcc for your new version of Solaris.

You can get a precompiled version of gcc from *http://www.sunfreeware.com/* or *http://www.blastwave.org/*. Make sure you pick up the package for your Solaris release.

If you wish to use gcc to build add-on modules for use with the perl shipped with Solaris, you should use the Solaris::PerlGcc module which is available from CPAN. The perl shipped with Solaris is configured and built with the Sun compilers, and the compiler configuration information stored in Config.pm is therefore only relevant to the Sun compilers. The Solaris:PerlGcc module contains a replacement Config.pm that is correct for gcc - see the module for details.

GNU as and GNU ld

The following information applies to gcc version 2. Volunteers to update it as appropriately for gcc version 3 would be appreciated.

The versions of as and ld supplied with Solaris work fine for building perl. There is normally no need to install the GNU versions to compile perl.

If you decide to ignore this advice and use the GNU versions anyway, then be sure that they are relatively recent. Versions newer than 2.7 are apparently new enough. Older versions may have trouble with dynamic loading.

If you wish to use GNU ld, then you need to pass it the -Wl,-E flag. The hints/solaris_2.sh file tries to

do this automatically by setting the following Configure variables:

```
ccdlflags="$ccdlflags -Wl,-E"
lddlflags="$lddlflags -Wl,-E -G"
```

However, over the years, changes in gcc, GNU ld, and Solaris ld have made it difficult to automatically detect which ld ultimately gets called. You may have to manually edit config.sh and add the -Wl,-E flags yourself, or else run Configure interactively and add the flags at the appropriate prompts.

If your gcc is configured to use GNU as and ld but you want to use the Solaris ones instead to build perl, then you'll need to add -B/usr/ccs/bin/ to the gcc command line. One convenient way to do that is with

```
$ sh Configure -Dcc='gcc -B/usr/ccs/bin/'
```

Note that the trailing slash is required. This will result in some harmless warnings as Configure is run:

```
gcc: file path prefix `/usr/ccs/bin/' never used
```

These messages may safely be ignored. (Note that for a SunOS4 system, you must use -B/bin/ instead.)

Alternatively, you can use the GCC_EXEC_PREFIX environment variable to ensure that Sun's as and ld are used. Consult your gcc documentation for further information on the -B option and the GCC_EXEC_PREFIX variable.

Sun and GNU make

The make under /usr/ccs/bin works fine for building perl. If you have the Sun C compilers, you will also have a parallel version of make (dmake). This works fine to build perl, but can sometimes cause problems when running 'make test' due to underspecified dependencies between the different test harness files. The same problem can also affect the building of some add-on modules, so in those cases either specify '-m serial' on the dmake command line, or use /usr/ccs/bin/make instead. If you wish to use GNU make, be sure that the set-group-id bit is not set. If it is, then arrange your PATH so that /usr/ccs/bin/make is before GNU make or else have the system administrator disable the set-group-id bit on GNU make.

Avoid libucb.

Solaris provides some BSD-compatibility functions in /usr/ucblib/libucb.a. Perl will not build and run correctly if linked against -lucb since it contains routines that are incompatible with the standard Solaris libc. Normally this is not a problem since the solaris hints file prevents Configure from even looking in /usr/ucblib for libraries, and also explicitly omits -lucb.

Environment for Compiling perl on Solaris

PATH

Make sure your PATH includes the compiler (/opt/SUNWspro/bin/ if you're using Sun's compiler) as well as /usr/ccs/bin/ to pick up the other development tools (such as make, ar, as, and ld). Make sure your path either doesn't include /usr/ucb or that it includes it after the compiler and compiler tools and other standard Solaris directories. You definitely don't want /usr/ucb/cc.

LD_LIBRARY_PATH

If you have the LD_LIBRARY_PATH environment variable set, be sure that it does NOT include /lib or /usr/lib. If you will be building extensions that call third-party shared libraries (e.g. Berkeley DB) then make sure that your LD_LIBRARY_PATH environment variable includes the directory with that library (e.g. /usr/local/lib).

If you get an error message

```
dlopen: stub interception failed
```

it is probably because your LD_LIBRARY_PATH environment variable includes a directory which is a symlink to /usr/lib (such as /lib). The reason this causes a problem is quite subtle. The file libdl.so.1.0 actually *only* contains functions which generate 'stub interception failed' errors! The runtime linker intercepts links to "/usr/lib/libdl.so.1.0" and links in internal implementations of those functions instead. [Thanks to Tim Bunce for this explanation.]

RUN CONFIGURE.

See the INSTALL file for general information regarding Configure. Only Solaris-specific issues are discussed here. Usually, the defaults should be fine.

64-bit perl on Solaris.

See the INSTALL file for general information regarding 64-bit compiles. In general, the defaults should be fine for most people.

By default, perl-5.6.0 (or later) is compiled as a 32-bit application with largefile and long-long support.

General 32-bit vs. 64-bit issues.

Solaris 7 and above will run in either 32 bit or 64 bit mode on SPARC CPUs, via a reboot. You can build 64 bit apps whilst running 32 bit mode and vice-versa. 32 bit apps will run under Solaris running in either 32 or 64 bit mode. 64 bit apps require Solaris to be running 64 bit mode.

Existing 32 bit apps are properly known as LP32, i.e. Longs and Pointers are 32 bit. 64-bit apps are more properly known as LP64. The discriminating feature of a LP64 bit app is its ability to utilise a 64-bit address space. It is perfectly possible to have a LP32 bit app that supports both 64-bit integers (long long) and largefiles (> 2GB), and this is the default for perl-5.6.0.

For a more complete explanation of 64-bit issues, see the "Solaris 64-bit Developer's Guide" at *http://docs.sun.com/*

You can detect the OS mode using "isainfo -v", e.g.

```
$ isainfo -v   # Ultra 30 in 64 bit mode
64-bit sparcv9 applications
32-bit sparc applications
```

By default, perl will be compiled as a 32-bit application. Unless you want to allocate more than ~ 4GB of memory inside perl, or unless you need more than 255 open file descriptors, you probably don't need perl to be a 64-bit app.

Large File Support

For Solaris 2.6 and onwards, there are two different ways for 32-bit applications to manipulate large files (files whose size is > 2GByte). (A 64-bit application automatically has largefile support built in by default.)

First is the "transitional compilation environment", described in lfcompile64(5). According to the man page,

```
The transitional compilation environment exports all the
explicit 64-bit functions (xxx64()) and types in addition to
all the regular functions (xxx()) and types. Both xxx() and
xxx64() functions are available to the program source. A
32-bit application must use the xxx64() functions in order
to access large files. See the lf64(5) manual page for a
complete listing of the 64-bit transitional interfaces.
```

The transitional compilation environment is obtained with the following compiler and linker flags:

```
getconf LFS64_CFLAGS        -D_LARGEFILE64_SOURCE
getconf LFS64_LDFLAG        # nothing special needed
getconf LFS64_LIBS          # nothing special needed
```

Second is the "large file compilation environment", described in lfcompile(5). According to the man page,

```
Each interface named xxx() that needs to access 64-bit entities
to  access  large  files maps to a xxx64() call in the
resulting binary. All relevant data types are defined to  be
of correct size (for example, off_t has a typedef definition
for a 64-bit entity).

An application compiled in this environment is able  to  use
the  xxx()  source interfaces to access both large and small
files, rather than having to explicitly utilize the  transitional
xxx64()  interface  calls to access large files.
```

Two exceptions are fseek() and ftell(). 32-bit applications should use fseeko(3C) and ftello(3C). These will get automatically mapped to fseeko64() and ftello64().

The large file compilation environment is obtained with

```
getconf LFS_CFLAGS      -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64
getconf LFS_LDFLAGS     # nothing special needed
getconf LFS_LIBS        # nothing special needed
```

By default, perl uses the large file compilation environment and relies on Solaris to do the underlying mapping of interfaces.

Building an LP64 perl

To compile a 64-bit application on an UltraSparc with a recent Sun Compiler, you need to use the flag "-xarch=v9". getconf(1) will tell you this, e.g.

```
$ getconf -a | grep v9
XBS5_LP64_OFF64_CFLAGS:         -xarch=v9
XBS5_LP64_OFF64_LDFLAGS:        -xarch=v9
XBS5_LP64_OFF64_LINTFLAGS:      -xarch=v9
XBS5_LPBIG_OFFBIG_CFLAGS:       -xarch=v9
XBS5_LPBIG_OFFBIG_LDFLAGS:      -xarch=v9
XBS5_LPBIG_OFFBIG_LINTFLAGS:    -xarch=v9
_XBS5_LP64_OFF64_CFLAGS:        -xarch=v9
_XBS5_LP64_OFF64_LDFLAGS:       -xarch=v9
_XBS5_LP64_OFF64_LINTFLAGS:     -xarch=v9
_XBS5_LPBIG_OFFBIG_CFLAGS:      -xarch=v9
_XBS5_LPBIG_OFFBIG_LDFLAGS:     -xarch=v9
_XBS5_LPBIG_OFFBIG_LINTFLAGS:   -xarch=v9
```

This flag is supported in Sun WorkShop Compilers 5.0 and onwards (now marketed under the name Forte) when used on Solaris 7 or later on UltraSparc systems.

If you are using gcc, you would need to use -mcpu=v9 -m64 instead. This option is not yet supported as of gcc 2.95.2; from install/SPECIFIC in that release:

```
GCC version 2.95 is not able to compile code correctly for sparc64
```

```
targets. Users of the Linux kernel, at least, can use the sparc32
program to start up a new shell invocation with an environment that
causes configure to recognize (via uname -a) the system as sparc-*-*
instead.
```

All this should be handled automatically by the hints file, if requested.

Long Doubles.

As of 5.8.1, long doubles are working if you use the Sun compilers (needed for additional math routines not included in libm).

Threads in perl on Solaris.

It is possible to build a threaded version of perl on Solaris. The entire perl thread implementation is still experimental, however, so beware.

Malloc Issues with perl on Solaris.

Starting from perl 5.7.1 perl uses the Solaris malloc, since the perl malloc breaks when dealing with more than 2GB of memory, and the Solaris malloc also seems to be faster.

If you for some reason (such as binary backward compatibility) really need to use perl's malloc, you can rebuild perl from the sources and Configure the build with

```
$ sh Configure -Dusemymalloc
```

You should not use perl's malloc if you are building with gcc. There are reports of core dumps, especially in the PDL module. The problem appears to go away under -DDEBUGGING, so it has been difficult to track down. Sun's compiler appears to be okay with or without perl's malloc. [XXX further investigation is needed here.]

MAKE PROBLEMS.

Dynamic Loading Problems With GNU as and GNU ld

If you have problems with dynamic loading using gcc on SunOS or Solaris, and you are using GNU as and GNU ld, see the section *GNU as and GNU ld* above.

ld.so.1: ./perl: fatal: relocation error:

If you get this message on SunOS or Solaris, and you're using gcc, it's probably the GNU as or GNU ld problem in the previous item *GNU as and GNU ld*.

dlopen: stub interception failed

The primary cause of the 'dlopen: stub interception failed' message is that the LD_LIBRARY_PATH environment variable includes a directory which is a symlink to /usr/lib (such as /lib). See *LD_LIBRARY_PATH* above.

#error "No DATAMODEL_NATIVE specified"

This is a common error when trying to build perl on Solaris 2.6 with a gcc installation from Solaris 2.5 or 2.5.1. The Solaris header files changed, so you need to update your gcc installation. You can either rerun the fixincludes script from gcc or take the opportunity to update your gcc installation.

sh: ar: not found

This is a message from your shell telling you that the command 'ar' was not found. You need to check your PATH environment variable to make sure that it includes the directory with the 'ar' command. This is a common problem on Solaris, where 'ar' is in the /usr/ccs/bin/ directory.

MAKE TEST

op/stat.t test 4 in Solaris

op/stat.t test 4 may fail if you are on a tmpfs of some sort. Building in /tmp sometimes shows this behavior. The test suite detects if you are building in /tmp, but it may not be able to catch all tmpfs situations.

nss_delete core dump from op/pwent or op/grent

See *"nss_delete core dump from op/pwent or op/grent" in perlhpux*.

CROSS-COMPILATION

Nothing too unusual here. You can easily do this if you have a cross-compiler available; A usual Configure invocation when targetting a Solaris x86 looks something like this:

```
sh ./Configure -des -Dusecrosscompile \
    -Dcc=i386-pc-solaris2.11-gcc      \
    -Dsysroot=$SYSROOT                \
    -Alddlflags=" -Wl,-z,notext"      \
    -Dtargethost=... # The usual cross-compilation options
```

The lddlflags addition is the only abnormal bit.

PREBUILT BINARIES OF PERL FOR SOLARIS.

You can pick up prebuilt binaries for Solaris from *http://www.sunfreeware.com/*, *http://www.blastwave.org*, ActiveState *http://www.activestate.com/*, and *http://www.perl.com/* under the Binaries list at the top of the page. There are probably other sources as well. Please note that these sites are under the control of their respective owners, not the perl developers.

RUNTIME ISSUES FOR PERL ON SOLARIS.

Limits on Numbers of Open Files on Solaris.

The stdio(3C) manpage notes that for LP32 applications, only 255 files may be opened using fopen(), and only file descriptors 0 through 255 can be used in a stream. Since perl calls open() and then fdopen(3C) with the resulting file descriptor, perl is limited to 255 simultaneous open files, even if sysopen() is used. If this proves to be an insurmountable problem, you can compile perl as a LP64 application, see *Building an LP64 perl* for details. Note also that the default resource limit for open file descriptors on Solaris is 255, so you will have to modify your ulimit or rctl (Solaris 9 onwards) appropriately.

SOLARIS-SPECIFIC MODULES.

See the modules under the Solaris:: and Sun::Solaris namespaces on CPAN, see *http://www.cpan.org/modules/by-module/Solaris/* and *http://www.cpan.org/modules/by-module/Sun/*.

SOLARIS-SPECIFIC PROBLEMS WITH MODULES.

Proc::ProcessTable on Solaris

Proc::ProcessTable does not compile on Solaris with perl5.6.0 and higher if you have LARGEFILES defined. Since largefile support is the default in 5.6.0 and later, you have to take special steps to use this module.

The problem is that various structures visible via procfs use off_t, and if you compile with largefile support these change from 32 bits to 64 bits. Thus what you get back from procfs doesn't match up with the structures in perl, resulting in garbage. See proc(4) for further discussion.

A fix for Proc::ProcessTable is to edit Makefile to explicitly remove the largefile flags from the ones MakeMaker picks up from Config.pm. This will result in Proc::ProcessTable being built under the correct environment. Everything should then be OK as long as Proc::ProcessTable doesn't try to share off_t's with the rest of perl, or if it does they should be explicitly specified as off64_t.

BSD::Resource on Solaris

BSD::Resource versions earlier than 1.09 do not compile on Solaris with perl 5.6.0 and higher, for the same reasons as Proc::ProcessTable. BSD::Resource versions starting from 1.09 have a workaround for the problem.

Net::SSLeay on Solaris

Net::SSLeay requires a /dev/urandom to be present. This device is available from Solaris 9 onwards. For earlier Solaris versions you can either get the package SUNWski (packaged with several Sun software products, for example the Sun WebServer, which is part of the Solaris Server Intranet Extension, or the Sun Directory Services, part of Solaris for ISPs) or download the ANDIrand package from *http://www.cosy.sbg.ac.at/~andi/*. If you use SUNWski, make a symbolic link /dev/urandom pointing to /dev/random. For more details, see Document ID27606 entitled "Differing /dev/random support requirements within Solaris[TM] Operating Environments", available at *http://sunsolve.sun.com* .

It may be possible to use the Entropy Gathering Daemon (written in Perl!), available from *http://www.lothar.com/tech/crypto/*.

SunOS 4.x

In SunOS 4.x you most probably want to use the SunOS ld, /usr/bin/ld, since the more recent versions of GNU ld (like 2.13) do not seem to work for building Perl anymore. When linking the extensions, the GNU ld gets very unhappy and spews a lot of errors like this

```
  ... relocation truncated to fit: BASE13 ...
```

and dies. Therefore the SunOS 4.1 hints file explicitly sets the ld to be */usr/bin/ld*.

As of Perl 5.8.1 the dynamic loading of libraries (DynaLoader, XSLoader) also seems to have become broken in in SunOS 4.x. Therefore the default is to build Perl statically.

Running the test suite in SunOS 4.1 is a bit tricky since the *dist/Tie-File/t/09_gen_rs.t* test hangs (subtest #51, FWIW) for some unknown reason. Just stop the test and kill that particular Perl process.

There are various other failures, that as of SunOS 4.1.4 and gcc 3.2.2 look a lot like gcc bugs. Many of the failures happen in the Encode tests, where for example when the test expects "0" you get "0" which should after a little squinting look very odd indeed. Another example is earlier in *t/run/fresh_perl* where chr(0xff) is expected but the test fails because the result is chr(0xff). Exactly.

This is the "make test" result from the said combination:

```
  Failed 27 test scripts out of 745, 96.38% okay.
```

Running the `harness` is painful because of the many failing Unicode-related tests will output megabytes of failure messages, but if one patiently waits, one gets these results:

```
 Failed Test                    Stat Wstat Total Fail  Failed  List of
Failed

-------------------------------------------------------------------------
--
 ...
 ../ext/Encode/t/at-cn.t          4  1024    29    4  13.79%  14-17
 ../ext/Encode/t/at-tw.t         10  2560    17   10  58.82%  2 4 6 8 10
12
                                                              14-17
 ../ext/Encode/t/enc_data.t      29  7424    ??   ??       %  ??
 ../ext/Encode/t/enc_eucjp.t     29  7424    ??   ??       %  ??
 ../ext/Encode/t/enc_module.t    29  7424    ??   ??       %  ??
```

```
 ../ext/Encode/t/encoding.t         29  7424    ??   ??       %  ??
 ../ext/Encode/t/grow.t             12  3072    24   12  50.00%  2 4 6 8 10
12 14
                                                                16 18 20 22
 24
  Failed Test                    Stat Wstat Total Fail  Failed  List of
Failed

-------------------------------------------------------------------------------
 ../ext/Encode/t/guess.t          255 65280    29   40 137.93%  10-29
 ../ext/Encode/t/jperl.t           29  7424    15   30 200.00%  1-15
 ../ext/Encode/t/mime-header.t      2   512    10    2  20.00%  2-3
 ../ext/Encode/t/perlio.t          22  5632    38   22  57.89%  1-4 9-16
19-20
                                                                23-24 27-32
 ../ext/List/Util/t/shuffle.t       0   139    ??   ??       %  ??
 ../ext/PerlIO/t/encoding.t                   14    1   7.14%  11
 ../ext/PerlIO/t/fallback.t                    9    2  22.22%  3 5
 ../ext/Socket/t/socketpair.t       0     2    45   70 155.56%  11-45
 ../lib/CPAN/t/vcmp.t                        30    1   3.33%  25
 ../lib/Tie/File/t/09_gen_rs.t      0    15    ??   ??       %  ??
 ../lib/Unicode/Collate/t/test.t             199   30  15.08%  7 26-27
71-75
                                                                81-88 95
101
                                                                103-104 106
 108-
                                                                109 122 124
 161
                                                                169-172
 ../lib/sort.t                      0   139   119   26  21.85%  107-119
 op/alarm.t                                  4    1  25.00%  4
 op/utfhash.t                               97    1   1.03%  31
 run/fresh_perl.t                           91    1   1.10%  32
 uni/tr_7jis.t                              ??   ??       %  ??
 uni/tr_eucjp.t                    29  7424     6   12 200.00%  1-6
 uni/tr_sjis.t                     29  7424     6   12 200.00%  1-6
 56 tests and 467 subtests skipped.
 Failed 27/811 test scripts, 96.67% okay. 1383/75399 subtests failed,
   98.17% okay.
```

The alarm() test failure is caused by system() apparently blocking alarm(). That is probably a libc bug, and given that SunOS 4.x has been end-of-lifed years ago, don't hold your breath for a fix. In addition to that, don't try anything too Unicode-y, especially with Encode, and you should be fine in SunOS 4.x.

AUTHOR

The original was written by Andy Dougherty *doughera@lafayette.edu* drawing heavily on advice from Alan Burlison, Nick Ing-Simmons, Tim Bunce, and many other Solaris users over the years.

Please report any errors, updates, or suggestions to *perlbug@perl.org*.