NAME

prex – probe external control

SYNOPSIS

prex [**–o** *trace_file_name*] [**–l** *libraries*] [**–s** *kbytes_size*] *cmd* [*cmd-args* ...]

prex [**–o** *trace_file_name*] [**–l** *libraries*] [**–s** *kbytes_size*] **–p** *pid*

prex **–k** [**–s** *kbytes_size*]

AVAILABILITY

SUNWtnfc

DESCRIPTION

prex is the application used for external control of probes. It is able to find all the probes in a target executable and it provides an interface for the user to manipulate them. **prex** allows a probe to be turned on for tracing, debugging, or both. Tracing generates a TNF trace file that can be converted to ASCII by **tnfdump**(1) and used for performance analysis. Debugging generates a line to standard error whenever the probe is hit at run time.

prex does not work on static executables—it only works on dynamic executables.

Invoking prex

There are three ways to invoke **prex**:

1). Use **prex** to start the target application *cmd*. In this case, the target application need not be built with a dependency on **libtnfprobe** (see **TNF_PROBE**(3X)), because **prex** will **LD_PRELOAD** (see **ld**(1)) the target with **libtnfprobe**. **prex** uses the environment variable **PATH** to find the target application.

2). Attach **prex** to a running application. In this case, the running target application should have **libtnfprobe** already linked in (could have been manually **LD_PRELOAD**'ed in by the user).

3). Use **prex** with the **–k** option, which puts **prex** into **kernel mode** : **prex** is used to control probes in the Solaris kernel. In kernel mode, additional commands are defined, and some commands valid in other modes are invalid. See **Kernel Mode** below.

Control File Format and Command Language

In a future release of **prex**, the command language will be moved to a syntax that is supported by an existing scripting language like **ksh**(1). In the mean time, this is uncommitted.

- Commands should be in ASCII.
- Each command is terminated with the NEWLINE character.
- A command can be continued onto the next line by ending the previous line with the backslash ('\') character.
- Tokens in a command have to be separated by whitespace (one or more spaces or tabs).
- The "#" character implies that the rest of the line is a comment.

Control File Search Path

There are two different methods of communicating with **prex**:

- by specifications in a control file. During start-up, **prex** searches for a file named **.prexrc** in the directories specified below. **prex** does not stop at the first one it finds— this way a user can override any defaults that are set up. The search order is:

  $HOME/
  ./

- by typing commands at the **prex** prompt.

The command language for both methods is the same and is specified in **USAGE**. The commands that return output will not make sense in a control file—the output will go to the standard output.

When using **prex** on a target process, the target will be in one of two states—running or stopped. This can be detected by the presence or absence of the **prex>** prompt. If the prompt is absent, it means that the target process is running. Typing CTRL-C will stop the target process and return the user to the prompt. There is no guarantee that CTRL-C will return to a **prex** prompt immediately. For example, if the target process is stopped on a job control stop (**SIGSTOP**), then CTRL-C in **prex** will wait until the target has been continued (**SIGCONT**). See **Signals to Target Program** below for more information on signals and the target process.

OPTIONS

The following options are supported:

–k **kernel mode**: **prex** is used to control probes in the Solaris kernel. In kernel mode, additional commands are defined, and some commands valid in other modes are invalid. See **Kernel Mode** below.

–l *libraries* The *libraries* mentioned are linked in to the target application using **LD_PRELOAD** (see **ld**(1)). This option cannot be used when attaching to a running process. The argument to the **–l** option should be a space separated string enclosed in double quotes. Each token in the string is a library name. It follows the **LD_PRELOAD** rules on how libraries should be specified and where they will be found.

–o *trace_file_name* File to be used for the trace output. *trace_file_name* is assumed to be relative to the current working directory of **prex** (i.e., the directory that the user was in when **prex** was started).

If **prex** attaches to a process that is already tracing, the new *trace_file_name* (if provided) will not be used. If no *trace_file_name* is specified, the default is **/$TMPDIR/trace-<***pid***>** where **<***pid***>** is the process id of the target program. If **TMPDIR** is not set, **/tmp** is used.

–s *kbytes_size* — Maximum size of the output trace file in Kbytes. The default size of the trace *kbytes_size* is **4096** or **4** Mbytes for normal usage, and **384** or **384** kbytes in kernel mode. The trace file can be thought of as a least recently used circular buffer. Once the file has been filled, newer events will overwrite the older ones.

USAGE

Grammar

Probes are specified by a list of space separated selectors. Selectors are of the form:

<*attribute*>=<*value*>

(see **TNF_PROBE**(3X)). The "<*attribute*>=" is optional. If it is not specified, it defaults to **keys=**.

The <*attribute*> or <*value*> (generically called spec) can be any of the following:

IDENT any sequence of letters, digits, _ , \ , ., % not beginning with a digit. **IDENT** implies an exact match.

QUOTED_STR usually used to escape reserved words (any commands in the command language). **QUOTED_STR** implies an exact match and has to be enclosed in single quotes (’ ’).

REGEXP an **ed**(1) regular expression pattern match. **REGEXP** has to be enclosed in slashes (/ /), A / can be included in a **REGEXP** by escaping it with a backslash \ .

The following grammar explains the syntax.

```
selector_list ::=                            /* empty */ |
                   <selector_list> <selector>
selector ::=       <spec>=<spec> |           /* whitespace around ‘=’ opt */
                   <spec>
spec ::=           IDENT |
                   QUOTED_STR |
                   REGEXP
```

The terminals in the above grammar are:

```
IDENT =            [a-zA-Z_\.%]{[a-zA-Z0-9_\.%]}+
QUOTED_STR =       ’[^\n’]*’                 /* any string in single quotes */
REGEXP =           /[^\n/]*/                 /* regexp’s have to be in / / */
```

This is a list of the remaining grammar that is needed to understand the syntax of the command language (defined in next subsection):

```
filename ::=       QUOTED_STR                /* QUOTED_STR defined above */
spec_list ::=      /* empty */ |
                   <spec_list> <spec>        /* <spec> defined above */
fcn_handle ::=     &IDENT                    /* IDENT defined above */
set_name ::=       $IDENT                    /* IDENT defined above */
```

Command Language

1. Set Creation and Set Listing

create $<set_name> <selector_list>
list sets **# list the defined sets**

create can be used to define a set which contains probes that match the <*selector_list*>. The set **$all** is pre-defined as /.*/ — it matches all the probes.

2. Function Listing

list fcns **# list the available <fcn_handle>**

The user can list the different functions that can be connected to probe points. Currently, only the debug function called **&debug** is available.

3. Commands to Connect and Disconnect Probe Functions

connect &<fcn_handle> $<set_name>
connect &<fcn_handle> <selector_list>

The **connect** command is used to connect probe functions (which must be prefixed by &) to probes. The probes are specified either as a single set (with a '$'), or by explicitly listing the probe selectors in the command. The probe function has to be one that is listed by the **list fcns** command. This command does not enable the probes.

4. Disconnects All Connected Probe Functions

clear $<set_name>
clear <selector_list>

The **clear** command is used to disconnect all connected probe functions from the specified probes.

5. Commands to Toggle the Tracing Mode

trace $<set_name>
trace <selector_list>
untrace $<set_name>
untrace <selector_list>

The **trace** and **untrace** commands are used to toggle the tracing action of a probe point (that is, whether a probe will emit a trace record or not if it is hit). This command does not enable the probes specified. Probes have tracing on by default. The most efficient way to turn off tracing is by using the **disable** command. **untrace** is useful if you want debug output but no tracing—if so, set the state of the probe to enabled, untraced, and the debug function connected.

6. Commands to Enable and Disable Probes

enable $<set_name>
enable <selector_list>
disable $<set_name>
disable <selector_list>
list history **# lists probe control command history**

The **enable** and **disable** commands are used to control whether the probes perform the action that they have been setup for. To trace a probe, it has to be both enabled and traced (using the **trace** command). Probes are disabled by default.

list history command is used to list the probe control commands issued: **connect**, **clear, trace**, **untrace**, **enable**, and **disable**. These are the commands that are executed whenever a new shared object is brought in to the target program by **dlopen**(3X). See the subsection, **dlopen'ed Libraries**, below for more information.

7. Commands to List Probes or List Values

list <spec_list> probes $<set_name>	**# e.g. list probes $all**
list <spec_list> probes <selector_list>	**# e.g. list name probes file=test.c**
list values <spec_list>	**# e.g. list values keys**

The first two commands list the selected attributes and values of the specified probes. They can be used to check the state of a probe. The third command lists the various values associated with the selected attributes.

8. Help Command

help

The **help** command lists all the commands available.

9. Source a File

source <filename>

The **source** command can be used to source a file of **prex** commands. **source** can be nested (that is, a file can source another file).

10. Process Control

continue	**# resumes the target process**
quit kill	**# quit prex, kill target**
quit resume	**# quit prex, continue target**
quit suspend	**# quit prex, leave target suspended**
quit	**# quit prex (continue or kill target)**

The default **quit** will continue the target process if **prex** attached to it. Instead, if **prex** had started the target program, **quit** will kill the target process.

dlopen'ed Libraries

Probes in shared objects that are brought in by **dlopen**(3X) are automatically set up according to the command history of **prex**. When a shared object is removed by a **dlclose**(3X), **prex** again needs to refresh its understanding of the probes in the target program. This implies that there is more work to do for **dlopen**(3X) and **dlclose**(3X) —so they will take slightly longer. If a user is not interested in this feature and doesn't want to interfere with **dlopen**(3X) and **dlclose**(3X), detach **prex** from the target to inhibit this feature.

Signals to Target Program

prex does not interfere with signals that are delivered directly to the target program. However, **prex** receives all terminal generated signals (for example, CTRL-C (**SIGINT**), CTRL-Z (**SIGSTOP**), etc.) and does not forward them to the target program. To signal the target program, use the **kill**(1) command from a shell.

Interactions with Other Applications

Process managing applications like **dbx**, **truss**(1), and **prex** can not operate on the same target program simultaneously. **prex** will not be able to attach to a target which is being controlled by another application. A user can trace and debug a program serially by the following method: first attach **prex** to target (or start target through **prex**), set up the probes using the command language, and then type **quit suspend**. The user can then attach **dbx** to the suspended process and debug it. A user can also suspend the target by sending it a **SIGSTOP** signal, and then by typing **quit resume** to **prex—** in this case, the user should also send a **SIGCONT** signal after invoking **dbx** on the stopped process (else **dbx** will be hung).

Failure of Event Writing Operations

There are a few failure points that are possible when writing out events to a trace file, for example, system call failures. These failures result in a failure code being set in the target process. The target process continues normally (but no trace records are written). Whenever a user types CTRL-C to **prex** to get to a **prex** prompt, **prex** will check the failure code in the target and inform the user if there was a tracing failure.

Target Executing a Fork or exec

If the target program does a **fork**(2), Any probes that the child encounters will be logged to the same trace file. Events are annotated with a process id, so it will be possible to determine which process a particular event came from. In multi-threaded programs, there is a race condition with a thread doing a fork while the other threads are still running. For the trace file not to get corrupted, the user should either use **fork1**(2), or make sure that all other threads are quiescent when doing a **fork**(2),

If the target program itself (not any children it may **fork**(2)) does an **exec**(2), **prex** detaches from the target and exits. The user can reconnect **prex** with **prex** −**p** *pid*.

Kernel Mode

Invoking **prex** with the −**k** flag causes **prex** to run in **kernel mode**. In kernel mode, **prex** controls probes in the Solaris kernel. See **tnf_probes**(4) for a list of available probes in the Solaris kernel. A few **prex** commands are unavailable in kernel mode; other commands are valid in kernel mode only.

The −**l**, −**o**, and −**p** command line options are not valid in kernel mode (that is, they may not be combined with the −**k** flag).

The rest of this section describes the differences in the **prex** command language when running **prex** in kernel mode.

1. Prex will not stop the kernel

When **prex** attaches to a running user program, it stops the user program. Obviously, it cannot do this when attaching to the kernel. Instead, **prex** provides a ''tracing master switch'': no probes will have any effect unless the tracing master switch is on. This allows the user to iteratively select probes to enable, then enable them all at once by turning on the master switch.

The command

ktrace [on | off]

is used to inspect and set the value of the master switch. Without an argument, **prex** reports the current state of the master switch.

Since **prex** will not stop or kill the kernel, the

quit resume

and

quit kill

commands are not valid in kernel mode.

2. No functions may be attached to probes in the kernel

In particular, the debug function is unavailable in kernel mode. Unless a probe is both enabled and traced, the probe has no effect. Thus, the only semantically meaningful values are to have the probe both enabled and traced, or neither enabled nor traced.

3. Trace output is written to an in-core buffer

In kernel mode, a trace output file is not generated directly, in order to allow probes to be placed in time-critical code. Instead, trace output is written to an in-core buffer, and copied out by a separate program, **tnfxtract**(1).

The in-core buffer is not automatically created. The following **prex** command controls buffer allocation and deallocation:

buffer [**alloc** [*size*] | **dealloc**]

Without an argument, the **buffer** command reports the size of the currently allocated buffer, if any. With an argument of **alloc** [*size*], **prex** allocates a buffer of the given size. *size* is in bytes, with an optional suffix of **'k'** or **'m'** specifying a multiplier of **1024** or **1048576**, respectively. If no *size* is specified, the *size* specified on the command line with the **–s** option is used as a default. If the **–s** command line option was not used, the ''default default'' is 384 kilobytes.

With an argument of **dealloc**, **prex** deallocates the trace buffer in the kernel.

prex will reject attempts to turn the tracing master switch on when no buffer is allocated, and to deallocate the buffer when the tracing master switch is on. **prex** will refuse to allocate a buffer when one is already allocated; use **buffer dealloc** first.

prex will not allocate a buffer larger than one-half of a machine's physical memory.

4. Prex supports per-process probe enabling in the kernel

In kernel mode, it is possible to select a set of processes for which probes are enabled. No trace output will be written when other processes traverse these probe points. This is called ''process filter mode.'' By default, process filter mode is off, and all processes cause the generation of trace records when they hit an enabled probe.

Some kernel events (such as interrupts) cannot be associated with a particular user process. By convention, these events are considered to be generated by process id 0.

prex provides commands to turn process filter mode on and off; to get the current status of the process filter mode switch; to add and delete processes (by process id) from the process filter set; and to list the current process filter set.

The process filter set is maintained even when process filter mode is off, but has no effect unless process filter mode is on.

When a process in the process filter set exits, its process id is automatically deleted from the process filter set. **prex** will report this the next time the user issues a command to **prex.**

The command:

pfilter [**on** | **off** | **add** <*pidlist*> | **delete** <*pidlist*>]

controls the process filter switch, and process filter set membership. With no arguments, **pfilter** prints the current process filter set and the state of the process filter mode switch.

on or **off** set the state of the process filter mode switch.

add <*pidlist*>
delete <*pidlist*> add or delete processes from the process filter set. <*pidlist*> is a comma-separated list of one or more process ids.

EXAMPLES

Example command language:

```
# at start up, all probes are cleared by default
# set creation and set listing

create $foo name='foo'                    # match only on name attr being foo
create $special /thr/=locks name=vm       # matches probes having either
                                          # attribute (reg-exp) /thr/=locks
                                          # or attribute name=vm
list sets                                 # list the defined sets
list fcns                                 # list the defined probe fcns

# Commands to trace and connect probe functions

trace foobar='on'                         # exact match on foobar attribute
trace $all                                # trace all probes
connect &debug $special                   # connect debug to the probes in "special" set
connect &debug /resource/ name=allocate   # connect debug to probes that have
                                          # attribute keys=/resource or
                                          # attribute name=allocate

# Commands to enable and disable probes

enable  $all                              # enable all the probes
enable /vm/ name=allocate destroy         # enable the specified probes
disable $special                          # disable "special" probes
disable /resource/ name='malloc'          # disable the specified probes
```

```
list history                      # list probe control commands issued

# Process control

continue                          # resumes the target process
^C                                # stop target, give control to prex
quit resume                       # exit prex, leave target process running

# Kernel mode

buffer alloc 2m                   # allocate a 2 Megabyte buffer
enable /.*/                       # enable all probes
trace /.*/                        # trace all probes
ktrace on                         # turn tracing on
ktrace off                        # turn tracing back off
pfilter on                        # turn process filter mode on
pfilter add 1379                  # add pid 1379 to process filter
ktrace on                         # turn tracing on
                                  # (only pid 1379 will be traced)
```

FILES

.prexrc	local **prex** initialization file
~/.prexrc	user's **prex** initialization file
/proc/*nnnnn*	process files

SEE ALSO

ed(1), **kill**(1), **ksh**(1), **ld**(1), **tnfdump**(1), **tnfxtract**(1), **truss**(1), **exec**(2), **fork**(2), **fork1**(2), **TNF_DECLARE_RECORD**(3X), **TNF_PROBE**(3X), **dlclose**(3X), **dlopen**(3X), **tnf_process_disable**(3X), **tnf_probes**(4)

NOTES

Currently, the only probe function that is available is the **&debug** function. When this function is executed, it prints out the arguments sent in to the probe as well as the value associated with the **sunw%debug** attribute in the detail field (if any) to **stderr**.

For example, for the following probe point:

```
TNF_PROBE_2(input_values, "testapp main",
        "sunw%debug have read input values successfully",
        tnf_long, int_input, x,
        tnf_string, string_input, input);
```

If *x* was 100 and *input* was the string "success", then the output of the debug probe function would be:

```
probe input_values; sunw%debug "have read input values successfully";
int_input=100; string_input="success";
```

dbx is available with the SPARCworks compiler set.