NAME

dtrace_proc - a DTrace provider for tracing events related to user processes

SYNOPSIS

proc:::create(*struct proc **, *struct proc **, *int*);

proc:::exec(*char **);

proc:::exec-failure(*int*);

proc:::exec-success(*char **);

proc:::exit(*int*);

proc:::signal-clear(*int*, *ksiginfo_t **);

proc:::signal-discard(*struct thread **, *struct proc **, *int*);

proc:::signal-send(*struct thread **, *struct proc **, *int*);

DESCRIPTION

The DTrace **proc** provider provides insight into events related to user processes: process and thread creation and termination events, and process signalling.

The **proc:::create**() probe fires when a user process is created via the fork(2), vfork(2), pdfork(2), or rfork(2) system calls. In particular, kernel processes created with the kproc(9) KPI will not trigger this probe. The **proc:::create**() probe's first two arguments are the new child process and its parent, respectively. The third argument is a mask of rfork(2) flags indicating which process resources are to be shared between the parent and child processes.

The **proc:::exec**() probe fires when a process attempts to execute a file. Its argument is the specified filename for the file. If the attempt fails because of an error, the **proc:::exec-failure**() probe will subsequently fire, providing the corresponding errno(2) value in its first argument. Otherwise, the **proc:::exec-success**() probe will fire.

The **proc:::exit**() probe fires when a process exits or is terminated. Its argument is the corresponding SIGCHLD signal code; valid values are documented in the siginfo(3) manual page and defined in *signal.h*. For example, when a process exits normally, the value of args[0] will be CLD_EXITED.

The **proc:::signal-send**() probe fires when a signal is about to be sent to a process. The

proc:::signal-discard() probe fires when a signal is sent to a process that ignores it. This probe will fire after the **proc:::signal-send**() probe for the signal in question. The arguments to these probes are the thread and process to which the signal will be sent, and the signal number of the signal. Valid signal numbers are defined in the signal(3) manual page. The **proc:::signal-clear**() probe fires when a pending signal has been cleared by one of the sigwait(2), sigtimedwait(2), or sigwaitinfo(2) system calls. Its arguments are the signal number of the cleared signal, and a pointer to the corresponding signal information. The *siginfo_t* for the signal can be obtained from args[1]->ksi_info.

ARGUMENTS

Though the **proc** provider probes use native FreeBSD arguments types, standard D types for processes and threads are available. These are *psinfo_t* and *lwpsinfo_t* respectively, and are defined in */usr/lib/dtrace/psinfo.d*. This file also defines two global variables, *curpsinfo* and *curlwpsinfo*, which provide representations of the current process and thread using these types.

The fields of *psinfo_t* are:

int pr_nlwp	Number of threads in the process.
pid_t pr_pid	Process ID.
pid_t pr_ppid	Process ID of the parent process, or 0 if the process does not have a parent.
pid_t pr_pgid	Process ID of the process group leader.
pid_t pr_sid	Session ID, or 0 if the process does not belong to a session.
pid_t pr_uid	Real user ID.
pid_t pr_euid	Effective user ID.
pid_t pr_gid	Real group ID.
pid_t pr_egid	Effective group ID.
uintptr_t pr_addr	Pointer to the *struct proc* for the process.
string pr_psargs	Process arguments.
u_int pr_arglen	Length of the process argument string.

u_int pr_jailid	Jail ID of the process.

The fields of *lwpsinfo_t* are:

id_t pr_lwpid	Thread ID.
int pr_flag	Thread flags.
int pr_pri	Real scheduling priority of the thread.
char pr_state	Currently always 0.
char pr_sname	Currently always ‘?’.
short pr_syscall	Currently always 0.
uintptr_t pr_addr	Pointer to the *struct thread* for the thread.
uintptr_t pr_wchan	Current wait address on which the thread is sleeping.

FILES

/usr/lib/dtrace/psinfo.d DTrace type and translator definitions for the **proc** provider.

EXAMPLES

The following script logs process execution events as they occur:

```
#pragma D option quiet

proc:::exec-success
{
        printf("%s", curpsinfo->pr_psargs);
}
```

Note that the pr_psargs field is subject to the limit defined by the *kern.ps_arg_cache_limit* sysctl. In particular, processes with an argument list longer than the value defined by this sysctl cannot be logged in this way.

COMPATIBILITY

The **proc** provider in FreeBSD is not compatible with the **proc** provider in Solaris. In particular, FreeBSD uses the native *struct proc* and *struct thread* types for probe arguments rather than translated types. Additionally, a number of **proc** provider probes found in Solaris are not currently available on FreeBSD.

SEE ALSO

dtrace(1), errno(2), fork(2), pdfork(2), rfork(2), vfork(2), siginfo(3), signal(3), dtrace_sched(4), kproc(9)

HISTORY

The **proc** provider first appeared in FreeBSD 7.1.

AUTHORS

This manual page was written by Mark Johnston <*markj@FreeBSD.org*>.