NAME

dtrace_sched - a DTrace provider for tracing CPU scheduling events

SYNOPSIS

sched:::change-pri(*struct thread *, struct proc *, uint8_t*);

sched:::dequeue(*struct thread *, struct proc *, void **);

sched:::enqueue(*struct thread *, struct proc *, void *, int*);

sched:::lend-pri(*struct thread *, struct proc *, uint8_t, struct thread **);

sched:::load-change(*int, int*);

sched:::off-cpu(*struct thread *, struct proc **);

sched:::on-cpu();

sched:::preempt();

sched:::remain-cpu();

sched:::surrender(*struct thread *, struct proc **);

sched:::sleep();

sched:::tick(*struct thread *, struct proc **);

sched:::wakeup(*struct thread *, struct proc **);

DESCRIPTION

The DTrace **sched** provider allows the tracing of events related to CPU scheduling in the 4BSD and ULE schedulers.

The **sched:::change-pri**() probe fires when a thread's active scheduling priority is about to be updated. The first two arguments are the thread whose priority is about to be changed, and the corresponding process. The third argument is the new absolute priority for the thread, while the current value is given by args[0]->td_priority. The **sched:::lend-pri**() probe fires when the currently-running thread elevates the priority of another thread via priority lending. The first two arguments are the thread whose priority is about to be changed, and the corresponding process. The third argument is the new absolute priority

for the thread. The fourth argument is the currently-running thread.

The **sched:::dequeue**() probe fires immediately before a runnable thread is removed from a scheduler run queue. This may occur when the thread is about to begin execution on a CPU, or because the thread is being migrated to a different run queue. The latter event may occur in several circumstances: the scheduler may be attempting to rebalance load between multiple CPUs, the thread's scheduling priority may have changed, or the thread's CPU affinity settings may have changed. The first two arguments to **sched:::dequeue**() are the thread and corresponding process. The third argument is currently always NULL. The **sched:::enqueue**() probe fires when a runnable thread is about to be added to a scheduler run queue. Its first two arguments are the thread and corresponding process. The third argument is currently always NULL. The fourth argument is a boolean value that is non-zero if the thread is enqueued at the beginning of its run queue slot, and zero if the thread is instead enqueued at the end.

The **sched:::load-change**() probe fires after the load of a thread queue is adjusted. The first argument is the cpuid for the CPU associated with the thread queue, and the second argument is the adjusted load of the thread queue, i.e., the number of elements in the queue.

The **sched:::off-cpu**() probe is triggered by the scheduler suspending execution of the currently-running thread, and the **sched:::on-cpu**() probe fires when the current thread has been selected to run on a CPU and is about to begin or resume execution. The arguments to **sched:::off-cpu**() are the thread and corresponding process selected to run following the currently-running thread. If these two threads are the same, the **sched:::remain-cpu**() probe will fire instead.

The **sched:::surrender**() probe fires when the scheduler is called upon to make a scheduling decision by a thread running on a different CPU, via an interprocessor interrupt. The arguments to this probe are the interrupted thread and its corresponding process. This probe currently always fires in the context of the interrupted thread.

The **sched:::preempt**() probe will fire immediately before the currently-running thread is preempted. When this occurs, the scheduler will select a new thread to run, and one of the **sched:::off-cpu**() or **sched:::remain-cpu**() probes will subsequently fire, depending on whether or not the scheduler selects the preempted thread.

The **sched:::sleep**() probe fires immediately before the currently-running thread is about to suspend execution and begin waiting for a condition to be met. The **sched:::wakeup**() probe fires when a thread is set up to resume execution after having gone to sleep. Its arguments are the thread being awoken, and the corresponding process.

The **sched:::tick**() fires before each scheduler clock tick. Its arguments are the currently-running thread and its corresponding process.

ARGUMENTS

The **sched** provider probes use the kernel types *struct proc* and *struct thread* to represent processes and threads, respectively. These structures have many fields and are defined in *sys/proc.h*. In a probe body, the currently-running thread can always be obtained with the *curthread* global variable, which has type *struct thread* *. For example, when a running thread is about to sleep, the **sched:::sleep**() probe fires in the context of that thread, which can be accessed using *curthread*. The *curcpu* global variable contains the cpuid of the CPU on which the currently-running thread is executing.

EXAMPLES

The following script gives a breakdown of CPU utilization by process name:

```
sched:::on-cpu
{
        self->ts = timestamp;
}

sched:::off-cpu
/self->ts != 0/
{
        @[execname] = sum((timestamp - self->ts) / 1000);
        self->ts = 0;
}
```

Here, DTrace stores a timestamp each time a thread is scheduled to run, and computes the time elapsed in microseconds when it is descheduled. The results are summed by process name.

COMPATIBILITY

This provider is not compatible with the **sched** provider found in Solaris. In particular, the probe argument types are native FreeBSD types, and the **sched:::cpucaps-sleep**(), **sched:::cpucaps-wakeup**(), **sched:::schedctl-nopreempt**(), **sched:::schedctl-preempt**(), and **sched:::schedctl-yield**() probes are not available in FreeBSD.

The **sched:::lend-pri**() and **sched:::load-change**() probes are specific to FreeBSD.

SEE ALSO

dtrace(1), sched_4bsd(4), sched_ule(4), SDT(9), sleepqueue(9)

HISTORY

The **sched** provider first appeared in FreeBSD 8.4 and 9.1.

AUTHORS

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