NAME

polling - device polling support

SYNOPSIS

options DEVICE_POLLING

DESCRIPTION

Device polling (**polling** for brevity) refers to a technique that lets the operating system periodically poll devices, instead of relying on the devices to generate interrupts when they need attention. This might seem inefficient and counterintuitive, but when done properly, **polling** gives more control to the operating system on when and how to handle devices, with a number of advantages in terms of system responsiveness and performance.

In particular, **polling** reduces the overhead for context switches which is incurred when servicing interrupts, and gives more control on the scheduling of the CPU between various tasks (user processes, software interrupts, device handling) which ultimately reduces the chances of livelock in the system.

Principles of Operation

In the normal, interrupt-based mode, devices generate an interrupt whenever they need attention. This in turn causes a context switch and the execution of an interrupt handler which performs whatever processing is needed by the device. The duration of the interrupt handler is potentially unbounded unless the device driver has been programmed with real-time concerns in mind (which is generally not the case for FreeBSD drivers). Furthermore, under heavy traffic load, the system might be persistently processing interrupts without being able to complete other work, either in the kernel or in userland.

Device polling disables interrupts by polling devices at appropriate times, i.e., on clock interrupts and within the idle loop. This way, the context switch overhead is removed. Furthermore, the operating system can control accurately how much work to spend in handling device events, and thus prevent livelock by reserving some amount of CPU to other tasks.

Enabling **polling** also changes the way software network interrupts are scheduled, so there is never the risk of livelock because packets are not processed to completion.

Enabling polling

Currently only network interface drivers support the **polling** feature. It is turned on and off with help of ifconfig(8) command.

The historic *kern.polling.enable*, which enabled polling for all interfaces, can be replaced with the following code:

```
for i in ‘ifconfig -l‘ ;
  do ifconfig $i polling; # use -polling to disable
done
```

MIB Variables

The operation of **polling** is controlled by the following sysctl(8) MIB variables:

kern.polling.user_frac

When **polling** is enabled, and provided that there is some work to do, up to this percent of the CPU cycles is reserved to userland tasks, the remaining fraction being available for **polling** processing. Default is 50.

kern.polling.burst

Maximum number of packets grabbed from each network interface in each timer tick. This number is dynamically adjusted by the kernel, according to the programmed *user_frac*, *burst_max*, CPU speed, and system load.

kern.polling.each_burst

The burst above is split into smaller chunks of this number of packets, going round-robin among all interfaces registered for **polling**. This prevents the case that a large burst from a single interface can saturate the IP interrupt queue (*net.inet.ip.intr_queue_maxlen*). Default is 5.

kern.polling.burst_max

Upper bound for *kern.polling.burst*. Note that when **polling** is enabled, each interface can receive at most (*HZ* * *burst_max*) packets per second unless there are spare CPU cycles available for **polling** in the idle loop. This number should be tuned to match the expected load (which can be quite high with GigE cards). Default is 150 which is adequate for 100Mbit network and HZ=1000.

kern.polling.idle_poll

Controls if **polling** is enabled in the idle loop. There are no reasons (other than power saving or bugs in the scheduler’s handling of idle priority kernel threads) to disable this.

kern.polling.reg_frac

Controls how often (every *reg_frac* / *HZ* seconds) the status registers of the device are checked for error conditions and the like. Increasing this value reduces the load on the bus, but also delays the error detection. Default is 20.

kern.polling.handlers

How many active devices have registered for **polling**.

kern.polling.short_ticks
kern.polling.lost_polls
kern.polling.pending_polls
kern.polling.residual_burst
kern.polling.phase
kern.polling.suspect
kern.polling.stalled
Debugging variables.

SUPPORTED DEVICES

Device polling requires explicit modifications to the device drivers. As of this writing, the bge(4), dc(4), em(4), fwe(4), fwip(4), fxp(4), igb(4), nfe(4), nge(4), re(4), rl(4), sis(4), ste(4), stge(4), vge(4), vr(4), and xl(4) devices are supported, with others in the works. The modifications are rather straightforward, consisting in the extraction of the inner part of the interrupt service routine and writing a callback function, ***_poll**(), which is invoked to probe the device for events and process them. (See the conditionally compiled sections of the devices mentioned above for more details.)

As in the worst case the devices are only polled on clock interrupts, in order to reduce the latency in processing packets, it is not advisable to decrease the frequency of the clock below 1000 Hz.

HISTORY

Device polling first appeared in FreeBSD 4.6 and FreeBSD 5.0.

AUTHORS

Device polling was written by Luigi Rizzo <*luigi@iet.unipi.it*>.