Kernel Introspection:Design Meeting 2007-04-17#

Who#

bstecher, cbugess, adanko, dbailey, mkisel

Summary#

We distilled customer's requirements for their HA controllers and separated the need for detection from the need for reporting into seven major requirements.

Our general conclusions were:

- we can support requirements with possibly more than one api (different api for different requiremments)
- general design approach:
  - eliminate the current heavy-weight polling with custom proc/ devctls
  - replace with notification followed by focused polling. The "Smoke alarm + Flashlight" approach

Smoke Alarm + Flashlight#

We will provide fast, but approximate detection mechanisms, to alert a customer's HA controller that there might be a problem (problem == a condition customer looks for). The notification is delivered with a sigevent (Smoke Alarm). Customer's controller responds by calling a slower APIs to methodically extract the thread/ process details it needs to apply its precise criteria. The Smoke-Alarm tests in the kernel may deliver reasonable number of false positives but may never return a false negative.

Summary of Requirements#

These requirements are distilled from customers own design documents for their high-availablity reporting and control systems. We've attempted, where ever possible, to separate the need for detection from reporting, in the hopes that they might have different performance requirements. We've only include those requirements that might require Neutrino support. Our objective is make sure Neutrino provides a suitably efficient interface to the customer's controller so it can meet these requirements. So for example, when the customer's controller needs to keep a history of a value, we have the choice of keeping the history in Neutrino, or giving the customer an efficient interface to poll the value and letting it keep the history.

C1. Detect Low System Memory#

- watermarks are part of criterion
- 100ms dectection time

C2. Detect Processes Hogging FDs#

- RLIMIT_NOFOILE not enough. Want lower threshold to warn approach of the hard RLIMIT

C3. Detect Process Memory Hogs#

- may suffice to report top N offenders

- currently, cisco uses watermarks as part of detection criteria
- 1min detection time

C4. Record Process Memory History#

- must distinguish at least heap from shared
- need 1min, 5min, 15min, 1hour, ... 48hour samples.
  - We doubt customers really need 1 minute samples. We conjecture they do not and will assume 15minute sampling for the rest of this design
  - We conjecture customers do not really need an instantaneous snapshot of all process times (one kernel call) or would accept a slightly skewed picture make up from many kernel calls.

C5. Detect Process CPU Hogs#

- complex critera based on prio, %cpu %cpu = (total_cpu - time_used_by_procnto) and averaged over 500ms
- 1minute detection time

C6. Record Process CPU history#

- 1min, 15min, 1hr,.... 48hr samples
- like memory history, they may not need 1minute samples.

C7. Detect Deadlocks Threads#

- 1 or 2 minute detection time
- in future, Cisco would like cs_time recorded per thread: Time since that thread last ran.

Brainstorming list of APIs#

A1. the existing devctls, called for each process and thread #

- very slow. Expensive if called at some customer's current polling rates.

A2. bulk devctls #

- read data for all processes and threads in one gulp
- possibly focused to read a subset of thread/process information for specific purposes

A3. bulk devctls based on a user supplied callback which defines selection criterion#

- unworkable: not reasonable to provide a user callback with a read only view of the kernel space. Also, when the callback throws an exception, the kernel crashes.

A4. bulk devctls based on a selection criteria specified by a user supplied data structure #

- assuming we can make the struct's type sufficiently complete to handle all reasonable criteria
- may provide new debug thread and process structs that are focused to a particular task rather than returning all possible data about a thread/process.

A5. a general notification interface#

- API implemented as a resmgr on a path like /dev/proc/notifier/cpuhog/<pid>...
- intenteded to be used my memory partitioning and perhaps any Neutrino application.
- user specifies edge-detection thresholds (level plus direction).
- system delivers sigevents when criteria is detected
- user does not poll.
- some form of throttling may be necessary (ex: requiring user to re-arm after each notification delivery)

A6. Read-only shared-memory window into kernel's thread/process data #

- A customer's suggestion. We're not making this up. Really!
- Nasty
- We'd have to be a copy of kernel data (probably a double-buffered copy). The kernel would have to spend a huge about of updating thread and process states in the buffers in addition to updating its own states. Would be a large change to the kernel

A7. use callouts on trace events #

- application would have to handle every scheduling event and infer state changes and compute run times.
- would require running kernel with tracing on.
- expensive. use lots of kernel time and lots of user time

A8. do nothing #

- convince cisco they dont need it
- (i.e. start writing our resumes)

A9. api to read RLIMIT usage levels #

A10. Generalize RLIMIT #

- allow edge-detect thresolds that deliver sigevents intead of hard errors (in addition to the hard limits)
- build on top of A9 and A5

A11. converge (A5, A9, A10) for Smoke-Alarm, and use A1 and A4 for Flashlight#

A12. faster way to read memory usage #

- without scanning mappings

Mapping requirements to Brainstomed APIs#

C1. Low System Memory#

- Use memory patritioning's notification interface, and suitably configured paritions to detect low system memory
- advise customer to use a conservative threshold (Smokealarm)
- advise customer to apply watermarking critera when memory partitioning delivers a low memory sigevent (flashlight)

C2. FD Hogs#

Alternatives:

1. A8
2. A10
3. A5 plus A9
4. A11

This is case of the general resource limit problem.

C3. Memory Hogs#

Alternatives:

1. A10
2. advise customer to enable A10 only after low system memory detected

C4. Memory History#

- advise customer to poll with individual devctls once every 15 mins
- A12

C5. CPU Hogs#

Alternatives:

1. advise customer to use APS to detect the existinance of some cpu hog, followed by A4
2. A11

C6. CPU History#

- advise customer to poll only once every 15 mins
- A1, A2 or A4

C7. Deadlocks#

Alternatives:

1. Default timers:
    - kernel adds a default timer to every blocking call (that doesn't already have a timer).
    - on timeout the sigenvets are delivered to Wdsysmon
    - this alerts customer's HA controller that some thread has been blocked for more than a minute
    - customer's HA controller responds by using A4 to read focused sync information for all blocked threads and then runs its deadlock detection algorithm.
    - Notes:
        - we need to restart the timer automatically after each firing
        - customer's HA controller may want to throttle these notifications
        - dependency on Brian's optimized timer queue implementation

1. Quick deadlock check in kernel
    - on each blocking operation, do a quick check to assert a deadlock has not been triggered
        - from the newly blocked thread, do a depth-first search to transitively find other threads were blocked on
        - make no attempt to detect loops
        - just count the depth. If the depth is small, no deadlock is possible

- report possible deadlock, with a sigevent, when sync count is greater than say 7. (A loop will looks like an infinite sync count)
- advise customre to respond to the sigevent by asking for all thread data (A4) and then running their own loop detection algorithm.
- customer's controller may want to throttle these "possible deadlock" sigevents