!Pathname Evolution:Design Meeting 2007-05-02#

Who#

bstecher, mkisel, cburgess, shiv, dbailey

Summary#

what we did:#

We resolved a basic issue of the pathname space: the need for a new name associated with bulk introspection.

Todo:#

- define higher levels of the hierarchy: sub-system (not well defined), and system
- details of registration interface, particularly knowledge of struct hierarchy
- we need a couple of more meetings.

Details#

We discussed:

1. Can we uniformly apply the notion that a URI modifies how something is read?#

The problem is with /proc?tags=... The final consensus was that in Posix, reading a directory always means retrieving dirents. To read data about processes, we would need to somewhere add a non-directory name into the pathname. That's necessary so that whole path would be for a file rather than a directory, since POSIX doesnt have preconcieved notions of what kind of data is read out of an arbitrary file. That means the necessity of a path like /proc/info?tags=...

It seemed ugly to some of us.

This also means we are not introducing new rules as to what illegal characters are for directory names. That would have been a problem. The URI in a filename will be passed to the filesystem to parse or not parse as it sees fit. So, directory name parsing is unchanged.

2. Is writing a URI string to an open FD an alternative to opening path with a contained URI?#

- The advantage is that a user could change filtering options for a bulk transfer by simply writing a new URI string.
- We noted that the customer's HA controller would want to avoid the overhead of opening a new path to the kernel instrospector every time it want to change options.

- We also noted that we have the option of defining whether writing a URI resets the file pointer to the start or not. If not, the user can always use fseek(). However, the customer's HA controller is most likely to read data for all pids and all tids when changing URI options. It seems unlikely that would want different URI options for the first N processes and other options for the rest. This is probably also true for any user. In that case, if we decide that writing a URI sets the file pointer to 0, we will save the users an extra call to fseek(). So having a write of a URI parms reset the read pointer to zero, is desirable.
- We can handle a user writing a URI string to an FD opened on /proc (or longer path) because currrently writing to /proc has no meaning. However, almost all other resmgrs will have had a meaning defined for write. So the conclusion is:
    - we implement both /proc?tags=... and open("/proc");fwrite("tags=...") for /proc
    - we only publish the URI in pathnames to users (and provide a library for parsing) so that the meaning of writing to other resmgrs's FDs is unchanged.
    - we advise Cisco about both interfaces to /proc: URI in the path, or writing the URI to an open FD.

3. How servers register with kernel bulk introspection.#

We'd like to generalize bulk kernel introspection so that we can easily add tagged structures at a later time. We especially want to allow tagged structures to be defined and populated with code in a kernel module. This means we want the bulk transfer code to be generic, i.e. not coded for specific structs, and that there be a "registration" interface . Proc would register the standard debug_process_t and debug_thread_t, etc structs already in debug.h We'd also register the custom structs that might meed needed by indivudal customers.

(Brian indicates he had some ideas on structure tagging that will likely change the following)

For the registration interface:

- We define a global space of tag ids.
    - integers will do
    - we specify a sub-range of tag ids which can be defined by customers
- we provide a registration interface which takes a tag id, tag string name, and fill proc
- in addition the registration interface must know if the parameter of the fillproc is a process, thread, mapinfo or other kernel object. In general, the hierarchy of the tags must be known at the registration interface.

4. Knowledge of tag hierarchy in the registration interface.#

We didn't say much about how the registration would know about the hierarchy of tags. However, Colin suggested the following notation to describe what the bulk transfer implementation must know:

```
process -> ( process_data_structp1,
             proccess_data_structp2,
             thread_iterator,
             mapping_iterator)

thread_iterator -> list of ( thread_data1, thread_data2)

mapping_iterator -> list of (mapping_data1, mapping_data2)
```

However, when the bulk kernel introspector serves a read request, it must know whether it needs to traverse the mapping_iterator or thread_iterater in addition to returning the process stucts for a particular pid is had decided to process. Which led us to discuss:

5. Does the syntax of the URI string specify the hierarchy of tagged structs being returned? Or is simply listing the names of the tagged structs sufficient?

- We could define a URI syntax that specifies which tagged structs to return, and for each tagged struct, which of it's iterators should be followed. The syntax would be recursive.
- We concluded that we could create a very powerful syntax that would allow the URI string to specify whether thread information or mapinfo information was wanted for pids.
- We concluded that the syntax could be even powerful enough to specify a different selection filter on tids and tids.
    - for example: a URI meaning "return all process structs for all pids using more than 10meg ram, and all thread structs for those pids, but only if the thread has used more than 10% cpu time.
- However, the complexity did not seem worthhwile. It is also possible to define the hierachy of tagged structs to be fixed properties of the tags, probably at registration time.
- So we concluded that path like /proc/info?tags=P1+P2+T1+T2 would suffice. That particular example would mean:
    - return struct types P1 and P2 for all processes
    - return struct types T1 and T2 for all threads
    - the knowledge that we also return information about threads of a pid we've selected to process comes from attributes of P1 and P2 tag.
- Though we concluded that /proc/info?tags=P1+P2+T1+T2 was sufficent and simple, we did not describe in detail the implementation that would know the hierarchy.
- we also observed that any mechanism we choose should allow for two hierarchies above process:
    - sub-system (not clearly defined yet, but something like APS or memory partitions)
    - the whole system.

Whiteboard Snapshots#

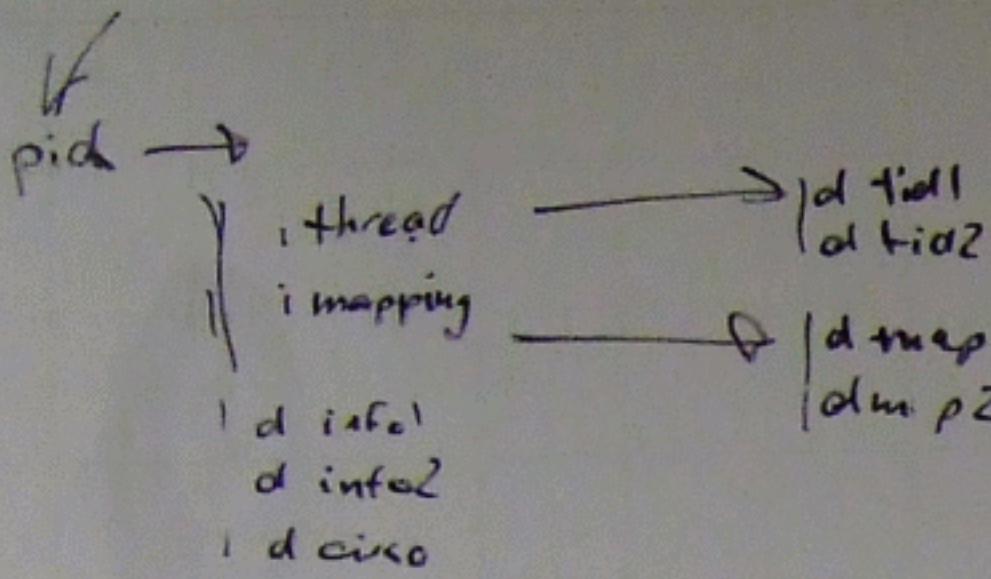
For those who want the original scribble notes.

PATHNAMES FOR

- BULK KERNEL INTROSPECTI
- GENERIC NOTIFIER
- MEM_PART. NOTIFICATI
- VAPS NOTIFICATION

FOR BULK

- READ ALL DATA ABOUT AL
- FILTER BY SYMLT
- DATA FOR 1 PID / TID



pid (1, 3, 77000, ... , / 1)

tid (1, 3) : tid2

— FILE FUNCTION REGISTRATION IN HIFANCY

/PROC/4721/#1

/PROC/4721/1/72

/PROC/4721/INFO? TAG=P

PROBES

(PATH, URI)

- MAY BE SPECIFIED
- OR READ PAT
  - EVENT D

ALTERNATIVE : REOPEN

/ PROC / 4721 / NF

/ PROC / INFO ? WHAT

* MUST HAVE CONCEPT

/PROC/INFO →