Generalised Kernel Hooks Interface

A High Speed Call-back Mechanism for the Linux Kernel

UKUUG Linux 2001

Manchester

Richard J. Moore

richardj_moore@uk.ibm.com

IBM Linux Technology Centre

1st July 2001

0. Overview

1. What are Hooks and Why have them?
2. What or Who would use them?
3. Why not Patch?
4. Hooking Scheme
5. Basic Requirements
6. The Managed Interface
7. Hook Dispatching Mechanism
8. Hook Activation - 1
9. Hook Activation - 2
10. Hook Implementation
11. Kernel Hook Applications
12. DProbes without GKHI
13. DProbes with GKHI
14. First Failure System Technology
15. What's Next?
16. Questions

1. What are Hooks and Why have them?

- Code locations where added function may be inserted
- Supplement or replace standard function - subclassing
- Function may not be known at build or run time
- Function may load later therefore simple call cannot be used
- Kernel has a particular need to implement hooks

- Many applications provide a framework in which additional function can be inserted. There may be a need to add functionality as in security checking or even replace it as in a scheduling algorithm.
- Doing this generically for function that that might dynamically load after the underlying framework is loaded needs more than a simple subroutine calling mechanism.
- The kernel has a particular problem in that it is first to load so there is no way it can have calls to external modules in the usual sense.
- Some form of call-back or call-out mechanism is required.

2. What or Who would use them?

- Security extensions
  - May need multiple policy implementors
- Serviceability extensions
  - Data Capture
  - Assertion Checking
- File System Extensions
  - Data Integrity
  - Compression
  - Security

IBM LTC 15/06/01

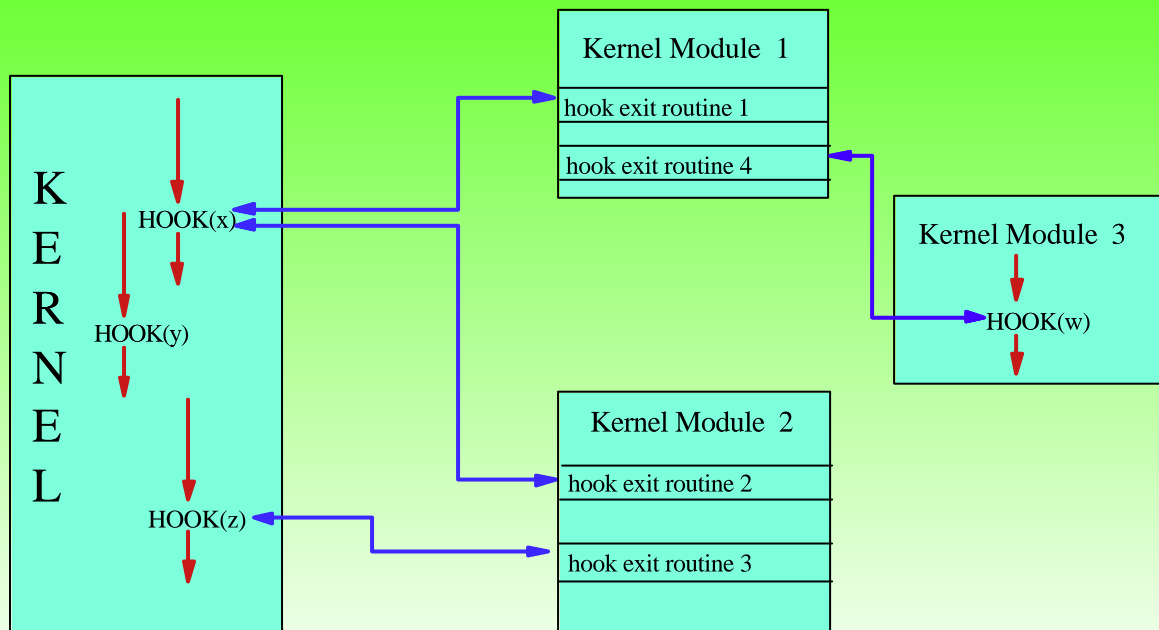
- Security extensions typically have this need. But its not inconceivable that multiple policies my be implemented for a given check. For example, during file open one policy might be to check additional user attributes not represented in the base file system. But a further refinement might be to add a further authorisation filter based on file properties.
- Here we have a need to two functional insertions for the same underlying code - file-open.
-
- Serviceability facilities, typically System Dump, Tracing and Assertion Checking S/W might not be desired at all times. So there's a requirement to have these facilities load when needed rather than bloat the underlying S/W with mostly dormant facilities.
-
- File System have a need when implementing data integrity and compression on top of the normal file system functions.

3. Why not Patch?

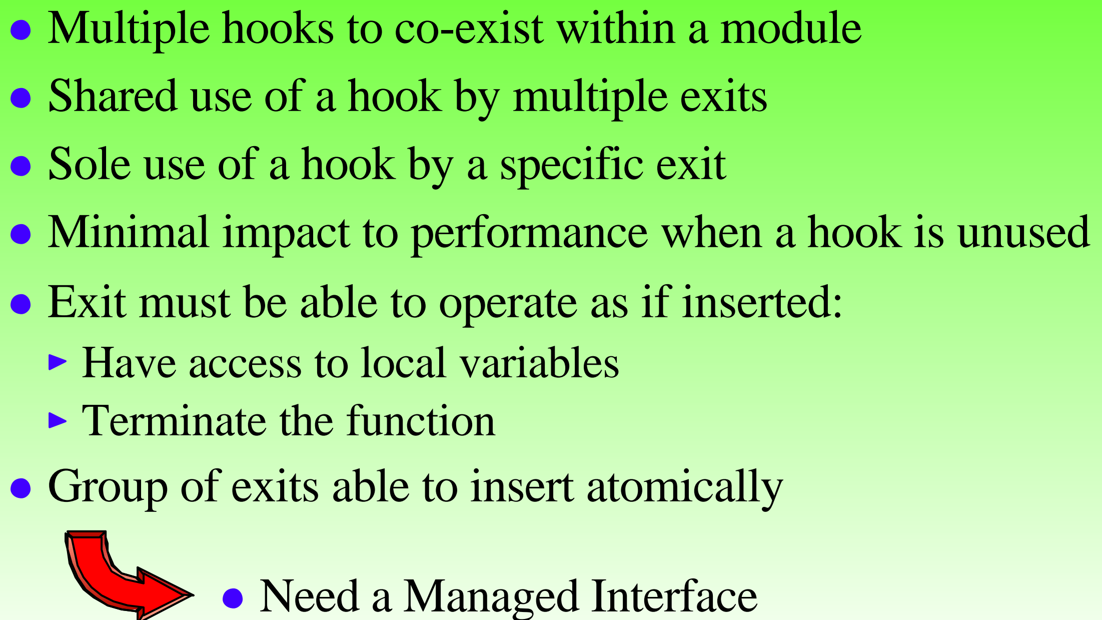
- Inconvenient
  - Multiple patches may require manual rework
- Inflexible
  - Cannot select additional functions at run-time
- Code Bloat
  - Additional functions always present

IBM LTC 15/06/01

- We could write patches to the source of the underlying function but there are issues:
-
- It's very inconvenient to have to make multiple patches work together - a lot of manual effort.
-
- It's very inflexible - once patched in the code is there and cannot be brought in dynamically when needed. If more is needed (e.g. in the Serviceability arena) then an additional patch will have to be worked into the underlying code.
-
- Trying to cater for all possibilities will bloat the underlying S/W enormously.



- The basic scheme:
-
- a HOOK macro is coded where insertions may take place
-
- Inserted function - Hook Exits may exist in loadable modules and be dynamically loaded.
-
- Multiple HOOKs make exit
-
- Multiple Exits may use a given HOOK
-
-
- HOOK - Hook Exit is a many-many relationship



- The complete list of basic requirements is:
-
  - 1) That multiple hooks can be coded
  - 2) Hook can be used by multiple exits
  - 3) Exits have the option of specifying that they are to be the only, first, last or don't care where in exit dispatching priority.
  - 4) Hooks which are not in use by an exit - inactive hooks - should have minimal impact on the system.
  - 5) Exits need to be able to access data local the routine they are inserting into - in other word to be able to have parameters passed to them.
  - 6) They also need to be able to cause the hooked routine to terminate. This is needed where function replacement is provided by the exit.
  - 7) For complex implementations such as security implementations, multiple exits will need to become inserted atomically.
-
- There is therefore a need for a managed interface.

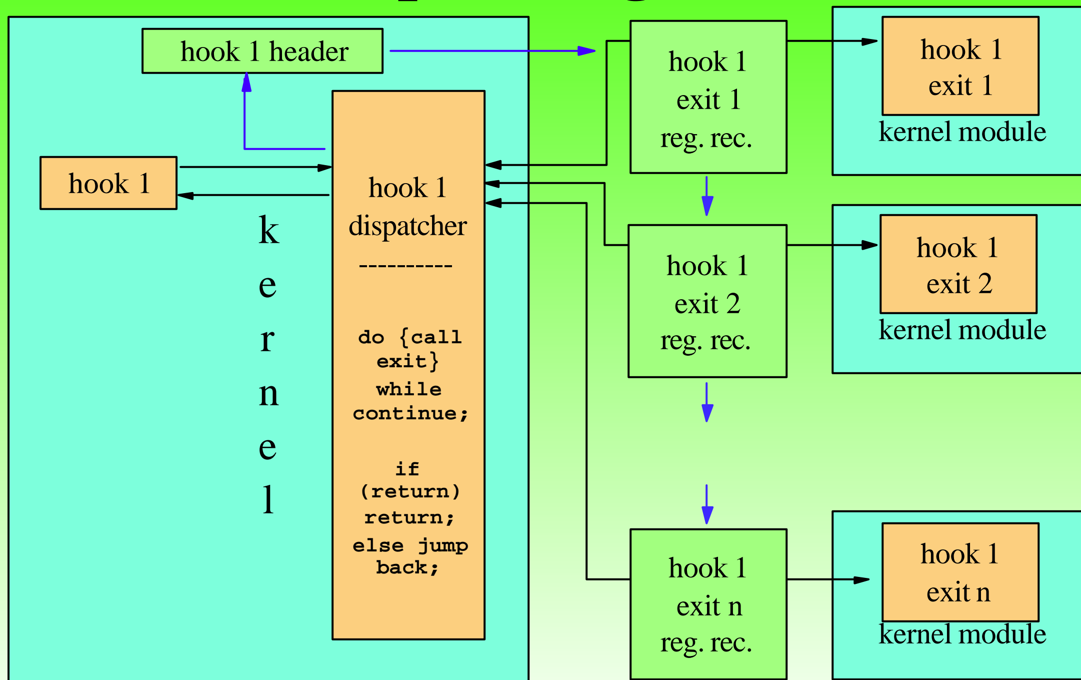
6. The Managed Interface

- For Hooked Code:
  - A **HOOK** macro - indicate the hook location
  - **hook_initialise** - allows use of the hook
  - **hook_terminate** - disallows use of the hook
- For Hook Exits:
  - **hook_register** - identifies exit routine and priority
  - **hook_arm** - activates group of exits
  - **hook_disarm** - deactivate group of exits
  - **hook_deregister** - removes exit from interface

IBM LTC 15/06/01

- For the Hooked Code
-
  - We provide a **HOOK** macro which placed in line where the hook location is. It's a single line update to the logic flow.
  - There are two functions:
    - **hook_initialise** which identifies to the hook management interface that a hook is available for use.
    - **hook_terminate** which removes a hook from use.
    -
- For the exit code for functions are supplied:
  - **hook_register** to notify the management interface of existence of the exit and its dispatching priority.
  - **hook_arm** is used to make a group of exits dispatchable.
  - **hook_disarm** and **hook_deregister** are the reverse functions to arm and register.

7. Hook Dispatching Mechanism



IBM LTC 15/06/01

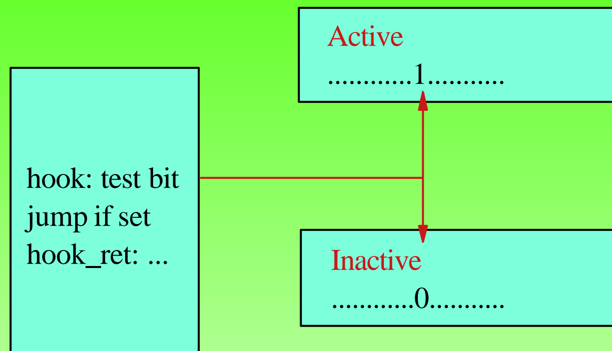
- The hook dispatcher is a small piece of code provided by the hook macro, but physically located in another code section so as not to impact I-cache lines.
- When a hook becomes active - one or more exits has armed it - then the dispatcher is invoked when the HOOK macro is executed.
- The dispatcher locates a header structure and from that a series of hook registration records which are ordered in exit priority. Each registration record contains the address of the exit routine.
- The dispatcher successively calls each exit, passing parameters as necessary (there are sever versions of the hook macro according to the number of parameters passed).
- An exit may return one of three values:
  - CONTINUE: - dispatch the next exit
  - TERMINATE: - return to the hooked code, do not dispatch further exits this time.
  - RETURN: - exit the hooked code, do not dispatch further exits this time.

8. Hook Activation - 1

Option 1: Use a Hook Switch

Logic:

hook: jump to dispatcher if bit set
return:



Pros:

Simple to implement
Architecturally independent

Cons:

Numbered hooks - uniqueness/registration issues
D-cache hit
I-cache hit
Jump and test it generally more than one instruction
State/register saving may be required
Test made when hook inactive

IBM LTC 15/06/01

- Two basic ways of implementing hooks:
  - Using a data switch
  - Code modification
- Each has pros and cons.
- The Switch Mechanism
  - Is simple to implement but requires more set-up than code modification. There's always an impact whether or not the hook is active. There are always D- and I-chache hits.
  - If a bit-string is used for hook identification then we have registration/uniqueness issues. One could implement a labelled switch-byte that would avoid this but there would be a proliferation of exported public symbols.
  - Use of a bit-string has issues associated with addressibility and extension - though these are minor.

9. Hook Activation - 2

Option 2: Use a Dynamic Code Patch

Logic:

```
hook: jump to return: or nop
jump to dispatcher
return:
```

```
/* Inactive Hook */
security_hook_1: jump security_hook_1_ret
jump hook dispatcher
security_hook_1_ret: ...
```

```
/* Active Hook */
security_hook_1: nop
jump hook dispatcher
security_hook_1_ret: ...
```

Pros:

- Simple to implement
- No D-cache hit
- Numbered hooks not required
- Minimal overhead for inactive hooks
- Minimal I-cache hit

Cons:

- Needs specific implementation per architecture
- Self-modifying code considerations apply
- -O2 Optimisation may need to be suppressed

IBM LTC 15/06/01

- Using a code patch to change - effectively - a no-op to a jump and *vice versa* has minimal impact to inactive hooks.
- There's no D-cache hit and minimal I-cache hit.
- The technique will be architecturally specific but there will be equivalent methods across most architectures.
- We can avoid using numbered hooks by using the label of the hook location as an identifier. (Note this has to be done for the data switch case as well).
- There will be architectural implications because hook activation is a process of code self-modification.
- -O2 gcc optimisation can be a problem - we are investigating this. It may have to be suppressed in module that have hooks. Better would be a #pragma option that would tell gcc treat certain sections of code as atomic from an optimisation perspective.

10. Hook implementation

IA32:

Use a 3-byte nop - jump with zero offset

IA64:

Under investigation

Use a predicate register against with a constant

S/390:

Use a relative branch - similar to IA32 - target address

MIPS:

Under investigation

Delayed branch technique needs thinking about

PPC:

Under investigation

IBM LTC 15/06/01

- So far the GKHI hook implementation is using the code-modification technique.
-
- We have it working on IA32. To avoid MP issues we code a hoop as a pair of jumps. The first will either jump to the next instruction (effectively a no-op) or it will jump passed the next instruction. The second jump is to the dispatcher loop.
-
- When a hook is inactive the first jump offset is non-zero - the value will depend on whether the second jump is 32-bit or 8-bit.
- When a hook is active the first jump offset is zero.
-
- We are investigating other architectures.
  - IA64 should be easy to implement by using an instruction bundle with a jump to dispatcher together with a predicate register.
  -
  - S/390 would work similarly to IA32
  -
  - MIPS needs some thought because of the delayed branch mechanism.
  -
  - PowerPC should be similar to IA32
  -

11. Kernel Hook Applications

Serviceability tools:

Reduce kernel patch to hooks and installable modules
Linux RAS Package

DProbes:

User exit

Linux Trace Toolkit:

Static Trace Hooks.

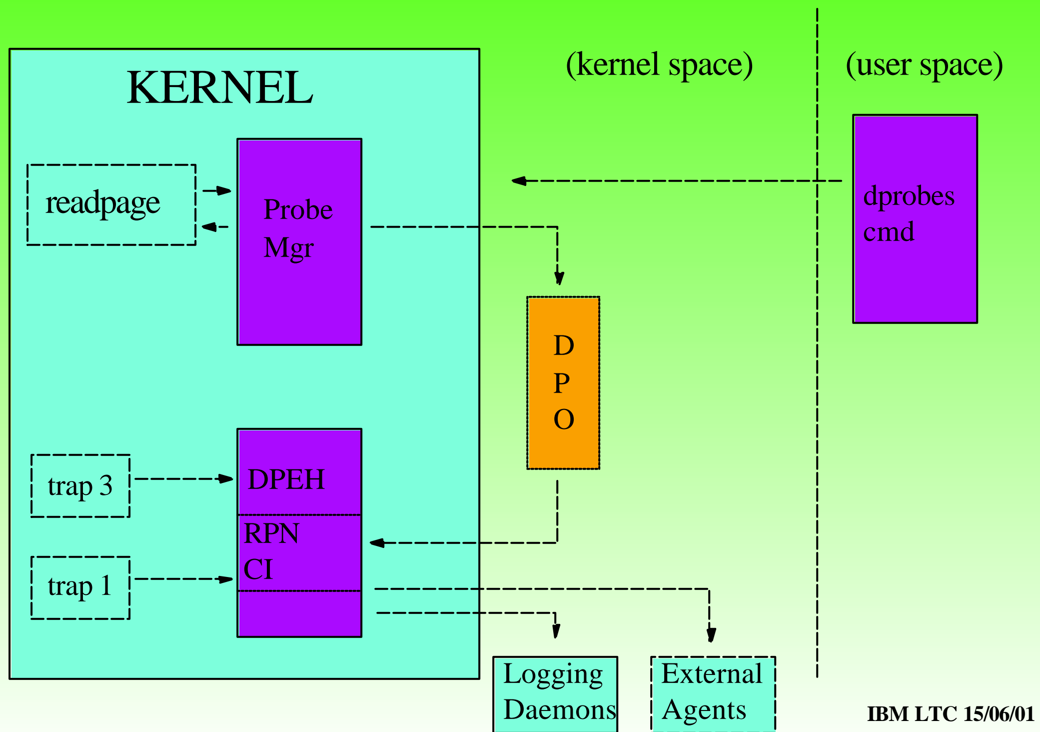
FFST:

Dynamic assertion checking and First Failure Data Capture

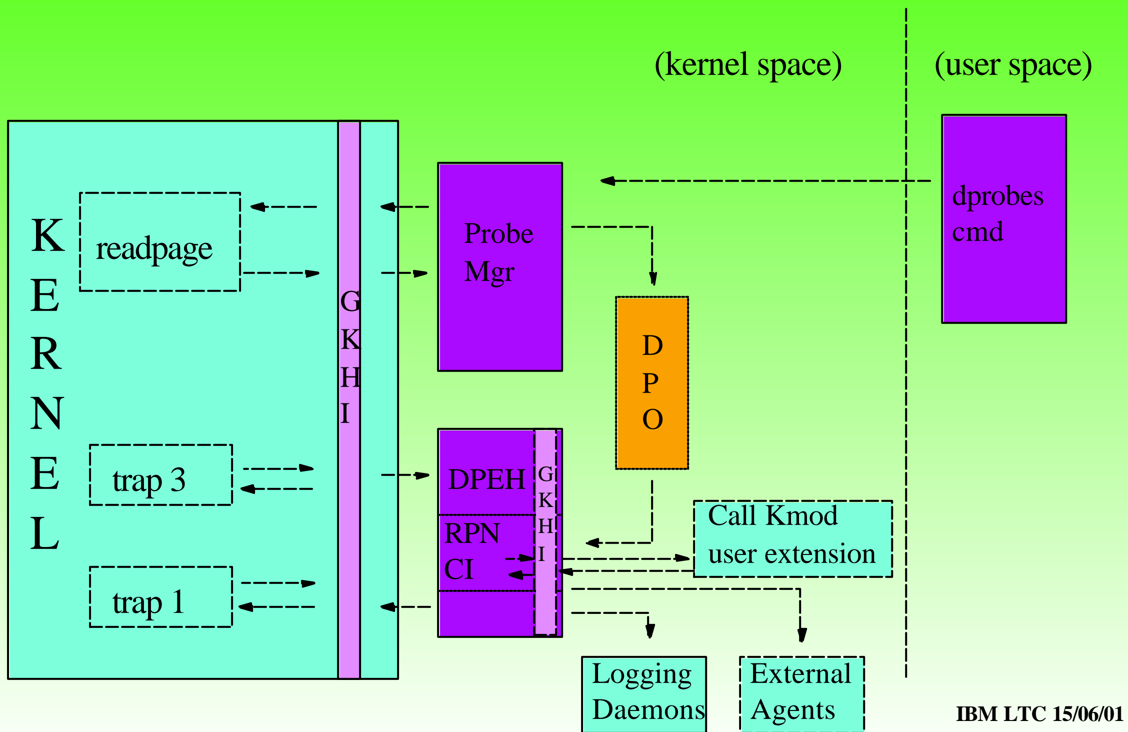
Security Hooks:

Suggested approach for hooking

- We are using hooks to simplify a consolidated package of serviceability tools - the Linux RAS Package.
-
- We have used it in Dynamic Probes to provide a user extension to the RPN command interpreter - the so-called "call kmod" interface.
-
- We are investigating migrating the LTT trace hooks to use GKHI hooks.
-
- First Failure System Technology is an organised way to perform dynamic assertion checking. This is something we are working on - details a little later.
-
- We have suggested that the GKHI might be adopted for implementing kernel security hooks. So far no comment from the security guys - they are busy deciding where they want hooks rather than the implementation.

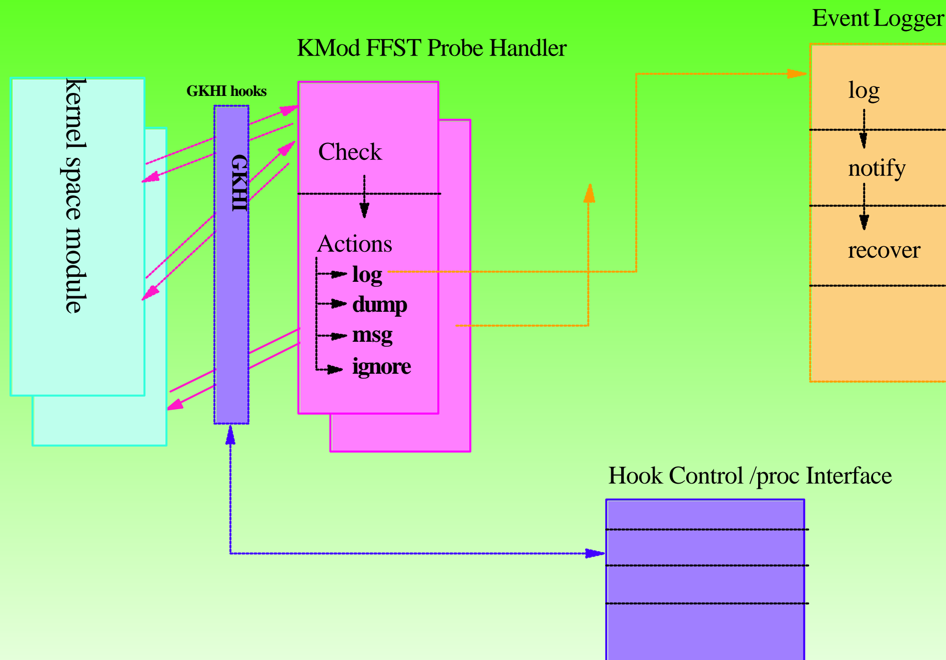


- This shows the structure of DProbes integrated as a kernel patch.



- When hooks are used the addition to the kernel is minuscule.
-
- Also note the "call kmod" hook in the RPN interpreter which was not possible to implement without a hooking mechanism.

14. First Failure System Technology



IBM LTC 15/06/01

- In the FFST scheme we are proposing an implementation where hooks are coded where assertion check are required.
- Hook exits would arm these hooks when assertion checking was required.
- We are also considering extending the management interface of GKHI to allow hook status to be reporter using the **/proc** file system. And in addition to allow the activation state of some hooks to be changed by the user through this interface.

15. What's Next?

- Repackage GKHI as a kernel patch
- Repackage DProbes to use GKHI
- Base Linux RAS Package on GKHI
- Introduce /PROC user interface to GKHI for FFST

IBM LTC 15/06/01

- We are currently re-packaging GKHI as a kernel patch. We are changing the names and coding style to conform with the kernel source style. For example **GKH_register** is becoming **hook_register**. The package currently available is the pre-kernel patch version.
- We are using GKHI to help implement and maintain a consolidate RAS (system serviceability) Package comprising:
  - SGI Kernel Debugger
  - SGI Kernel Crash Dump
  - IBM Dynamic Probes
  - Opersys Linux Trace Toolkit.
  - To be added IBM Linux Event Log
- The next functional enhancement to GKHI is the **/proc** user interface.

16. Questions?

End of Presentation

Mailing List: *dprobes@oss.software.ibm.com*
Web Page: *http://oss.software.ibm.com/developerworks/opensource/linux/projects/gkhi*

Core Team:

Richard Moore (RAS Architect)
S. Vamsikrishna
Subodh Soni
Bharata B. Rao
Suparna Bhattacharya