



Driver Update Model

Robert Hentosh, Senior Engineer

Dell, Inc.

May 2006



- Distributions package the Linux kernel, test and then support it.
- Enterprise level distributions will support it through a multi-year life cycle.
 - Updates will be released to support new hardware devices, resolve security issues or fix bugs. Enterprise updates will attempt to keep the core system stable while resolving the critical or nagging issues.
- Much of the function of the kernel has been divided into kernel modules which can be loaded at runtime.
 - About $\frac{3}{4}$ of the 1500 some odd kernel modules are for hardware device support.
- Most of the issues can be fixed with the addition of modules or replacement existing ones.

- Sometimes the need of a new device or fix does not match with the release schedule of a distribution or falls outside the distributions scope of functionality or support.
 - This presents a problem to people that support these configuration and leaves them little choice but to wait till the next release or possibly run an unsupported configuration.
- The various maintainers and suppliers of the modules have the problem of getting a fix to the user and distributing the fix to the user.
- We are looking for a common solution to manage, maintain, monitor, distribute and build these modules.

- An initial pass at solving this problem was developed at Dell about 2 years ago.
- DKMS
 - allows one to package modules for distribution
 - assist with creating driver diskettes
 - recompile modules for update kernels
 - obtain status of the version of modules currently modified in the system.
 - allow one to roll back to the previous driver

- Red Hat project under development, led by Jon Masters.
 - Collaborative development among Red Hat, Dell, IBM, HP, and Novell.
 - Project is in the process of being integrated with Fedora.
- Similar goals as DKMS with the addition of:
 - better integration with the RPM method of managing the system image. (does not modify file under RPM)
 - better support for exposing risks to kernel changes effecting the module / kernel interface. (beyond a simple compile)
 - reduce the necessity of recompilation of modules due to kernel updates. (not all customers want a build environment on each deployed machine)

- kernel ABI
 - kABI consists of function calls with parameters and data structures that are passed between the module and kernel.
 - each exported symbol is has a checksum that depends on its dependencies and parameters
 - build a method of tracking the changes that occur to the kernel module interfaces between updates.
 - verify that the module is compatible with the kABI
 - build a database of which modules use which interfaces
 - when ABI changes are required, it will be easier to track what they will effect and be able to weigh the ramification.
 - modules can now verify that the interface they are interested in will be consistent in to the previous version at installation time.
 - database will mark which interfaces are considered stable for 3rd party modules

- To avoid breaking each module into an RPM and increasing the size of RPM dependency tree, modutils has been modified
 - allows overriding module packages to be installed in a separate directory – removes the DKMS need to swap files with the kernel RPM.
 - 2 new directories *extras* and *weak-updates* permit overriding built in module contained in kernel RPM.
 - *extras* directory only overrides the single kernels version
 - *weak-updates* directory allows overriding of a range of kernel versions. It has lower priority than the *extras* directory.
 - verifies kABI via *symvers / symsets* file (similar to *System.map*).

- may require eventual RPM modifications
- currently changes are contained with in RPM spec file.
 - changes INSTALL_MOD_DIR
 - adds macros to RPM spec file, new macros contained in *kmodtool* file with source RPM.
 - modification to rpm package to also handle KMP (kernel module packages)
 - changes in the kernel RPM.
 - changes to anaconda (modifications needed for initrd, driver update disks, etc.)
- Using RPM to package drivers opens one to many ways to distribute a module. (Using RHN, yum, etc...)

- The project is under heavy development by Jon Masters at Red Hat.
 - Jon is working closely with the IHV's and Novell to find an immediate solution
 - Developing project website / wiki.
- Goal of similarity between distributions for module authors and users.
 - Project is working closely with Novell and its KMP (kernel module packages)
 - Work with others in the industry to hopefully come to a standard packaging scheme (simplifying work for the module author and distributors)
- More thought put into optimal method of calculating the kABI checksums.
- More support tool work to develop tools to assist with module symbol verification (certification?)
- Integration with Fedora.

Where does that leave DKMS?

- DKMS is in use now by Dell and will continue as a distribution and managing method until Driver Update Model is accepted into the enterprise distributions (or something functionally similar)
- DKMS has been modified to produce KMP's, it may live on to provide utility to developers in producing module packages, even if all the distros pickup this model.

Where to go for more information?

- Driver Update Model:
 - If you have interest in this project, see me after the presentation or email robert_hentosh@dell.com or jcm@redhat.com
- Novel KMP:
 - http://en.opensuse.org/Kernel_Module_Packages
 - <http://en.opensuse.org/Factory>
- DKMS:
 - <http://linux.dell.com/projects.shtml#dkms>