CIS 657 (POS) fall 2013 Lab 6 –

Dynamic Kernel Linker

# What’s KLD (Dynamic Kernel Linker)?

KLD is an object file that contains code to extend the running kernel, or so-called base kernel, of an operating system. KLDs are typically used to add support for new hardware and/or file systems, or for adding system calls. When the functionality provided by a KLD is no longer required, it can be unloaded in order to free memory and other resources.

# What’s the advantage of KLD?

Without dynamic kernel linker, an operating system would have to include all possible anticipated functionality already compiled directly into the base kernel. Much of that functionality would reside in memory without being used, wasting memory, and would require that users rebuild and reboot the base kernel every time they require new functionality. Most operating systems supporting dynamic kernel linker will include modules to support most desired functionality.

# What about the KLD in FreeBSD?

Kernel modules for FreeBSD are stored within /boot/kernel/ for modules distributed with the OS, or usually /boot/modules/ for modules installed from FreeBSD ports or FreeBSD packages, or for proprietary or otherwise binary-only modules. Once the machine has booted, they may be loaded with the kldload command, unloaded with kldunload, and listed with kldstat. Modules can also be loaded from the loader before the kernel starts, either automatically (through/boot/loader.conf) or by hand.

The purpose of this project is to introduce the basics of programming and developing KLDs under the FreeBSD operating system.

# KLD structure

There are two main functions/macros that must be included in all KLDs; they are:

* Load handler function.
* DECLARE\_MODULE macro.

It can be easily compiled via Makefile

#### Load handler function

Basically, the load handler function is, as it states, a function that handles the loading and unloading of a KLD. Hence, when a KLD is kldloaded or kldunloaded, this handler is what, at a very simplistic level, gets called.

#### DECLARE\_MODULE macro

The DECLARE\_MODULE macro is also something that is basic to all KLDs. However, it is not always seen as DECLARE\_MODULE. There are a couple of macros which can be used instead to more easily declare the module as a certain type. DECLARE\_MODULE is itself a macro:

#define DECLARE\_MODULE(name, data, sub, order) which is defined in /usr/include/sys/module.h. The parameters are explained below

|  |  |
| --- | --- |
| name : | The generic module name, this will be used further down in the SYSINIT call. |
| data : | A pointer to the moduledata structure is filled then passed as the data field. This structure contains two main items:   char \*name: The official module name, which will be used in the module structure.   modeventhand\_t evhand: This is our load handler function pointer, therefore, this field gets filled with the name of our load handler function. |
| sub : | This is an argument more directed at the SYSINIT macro. The valued entries for this can be found in /usr/include/sys/kernel.h in the system\_sub\_id enumeration list. These are known types for system startup interfaces. |
| order : | This is another argument that is intended for the later calling of SYSINIT. It represents the KLDs order of initialization within the subsystem. Valid values for this field can be found in /usr/include/sys/kernel.h in the sysinit\_elem\_order enumeration. |

#### Easy compile via makefile

A very neat piece of the Makefile functionality is the ".include" command. A few of the key variables that you may set are

|  |  |
| --- | --- |
| SRCS : | Listing of sources. |
| KMOD : | Name of module to build. |

References:

<http://www.freebsd.org/doc/en/books/arch-handbook/driverbasics-kld.html>

<http://www.freesoftwaremagazine.com/articles/writing_a_kernel_module_for_freebsd>

# Tasks (90)

In this lab, you have to print the following system parameters using Kernel loadable module:

|  |  |
| --- | --- |
| Requirements | Points |
| Version info | 5 |
| Copyrights info | 5 |
| System Uptime in minutes | 10 |
| Physical memory in Megabytes | 10 |
| CPU time for User, System, Interrupts, Idle in seconds | 10 |
| Number of context switches, hardware interrupts, software interrupts | 15 |
| Number of context switches, hardware interrupts, software interrupts from kldload to kldunload | 15 |
| Program that compiles | 10 |
| Program that runs without crash | 10 |

# Submission (10)

Create and attach a README (txt/word/pdf) file at the end of the lab. It doesn't need to be comprehensive, but it should at least cover the following content:

* Which tasks are done, and which are not?
* What’s your basic idea to achieve this task?
* Where is your main function?
* Which files you have modified and under which function?

If you can only finish some of the tasks in this project, please make sure that your code can at least be compiled and installed and also clearly state in the README file about the missing parts of your project.

**Checklist:** To submit your project, you need to:

• Attach the **source code of the kernel module**

• **Makefile**

• Create and attach a **README** file.

• Send this email to the TAs keeping Dr. Chapin < chapin@syr.edu> in the CC with subject line “CIS657: Lab 6”

Saurabh Sabnis < spsabnis@syr.edu >

Prasoon Pandya < [pdpandya@syr.edu](mailto:pdpandya@syr.edu) >