



Published on Free Software Magazine (<http://www.freesoftwaremagazine.com>)

# Writing a kernel module for FreeBSD

## FreeBSD hacking 101

By Yousef Ourabi

FreeBSD 7.0 has already been released. If you are a real hacker, the best way to jump in and learn it is hacking together an introductory kernel module. In this article I'll implement a very basic module that prints a message when it is loaded, and another when it is unloaded. I'll also cover the mechanics of compiling our module using standard tools and rebuilding the stock FreeBSD kernel. Let's do it!

## Getting Started

There are some prerequisites for this article. I assume you have a little bit of C programming knowledge, though nothing too fancy. If I reference a pointer or a structure, I expect you to understand those concepts without much explanation. I also expect you to be familiar with UNIX-like operating systems and know your way around basic shell usage.

The first thing to do is make sure the development environment you'll be working in includes everything and is configured properly. I'm going to assume you already have FreeBSD installed and running. If you don't, and would like some guidance, you can read my article: [Secure emails servers with FreeBSD](#). I'll be doing a few things differently; so, I would recommend just using it as a resource for the installation, which is identical in both scenarios.

You will also need to make sure the `sudo` utility is installed and configured for your main user account. This is required to run as "root" the utilities `kldload` and `kldunload`. The FreeBSD port for `sudo` is in `/usr/ports/security/sudo`.

```
cd /usr/port/security/sudo
make install && make clean
```

Now `su` to root and run the `visudo` utility. `Visudo` uses the `EDITOR` environment variable to determine which text editor to use. If you don't like the default one, override it with `setenv EDITOR vim` if you are using the default `CSH`, or `export EDITOR=vim` if you are using `bash`. Then simply re-run `visudo`.

```
su
visudo
```

The `visudo` utility does basic sanity checking on the syntax and makes sure no two people edit the `sudoers` file at the same time.

```
# Look for this line, copy it and change the "root" user to your main users name.
root    ALL=(ALL) SETENV: ALL
yourabi ALL=(ALL) SETENV: ALL
```

# Your first kernel

As I mentioned in my recent article [Review of FreeBSD 7.0](#), the ULE scheduler brings a new level of performance and multi-processor scalability to FreeBSD. Though it is now considered stable and ready for prime-time, it will not be enabled by default until the 7.1 release. A good litmus test of your setup will be compiling a custom kernel with the ULE scheduler enabled.

If you are on an x86 based machine, the kernel is located in the `/usr/src/sys/i386` directory. For amd64 machines, simply replace `i386` with `amd64`, which becomes `/usr/src/sys/amd64`. The kernel configuration file is located in a directory called `conf`. Enter that directory and create your own custom kernel configuration file by copying the “generic” default configuration to one of our own naming.

```
cd /usr/src/sys/amd64/conf
cp GENERIC CUSTOM
```

Now it's time to enable the new ULE scheduler. The new ULE scheduler brings improved performance and scalability compared to the legacy scheduler and also serves as a good demonstration of building and installing a custom kernel.

Open the “CUSTOM” file with your text editor of choice and find the line enabling the legacy 44BSD scheduler and replace it with ULE. In the stock kernel configuration file this should be around line 30 (at the time of this writing).

```
options      SCHED_4BSD  # 4BSD scheduler
# BECOMES #
options      SCHED_ULE  # ULE scheduler
```

Now build your new kernel and reboot so it takes effect. FreeBSD has an elegant, simple build system using the standard “make” utility. Simply change directories to the source tree and invoke the make file with the “buildkernel” and “installkernel” targets. If you call them without any parameters, the `GENERIC` (default) kernel will be built and installed. Since you want your custom kernel, pass in the `KERNCONF` flag with the name of the target kernel. In this case, it will be the name that you just copied the generic kernel to: `CUSTOM`.

```
cd /usr/src
make buildkernel KERNCONF="CUSTOM"
make installkernel KERNCONF="CUSTOM"
reboot
```

Congratulations! As the system boots up, it will run the new custom kernel with the ULE scheduler enabled. You have now verified that you are able to compile and install a kernel, so it's time to take on your next task: writing a simple kernel module.

## **When running FreeBSD in VMWare make sure to lower the kernels timer frequency**

Note: If you are running FreeBSD in VMWare there is one very important performance tweak to make to your system to follow this article. The kernel's timer frequency needs to be lowered from '1000' to '100' ticks per second. Edit `/boot/loader.conf` with your favorite editor and add the following line.

```
echo kern.hz=100 >> /boot/loader.conf
```

# Kernel Hello World

As you may have noticed, FreeBSD makes efficient use of the make utility for building and installing kernels (and the rest of the operating system). What you may not know yet, but will come as no surprise, is that the FreeBSD developers have also developed make files to ease part of the difficulty of kernel module development.

An in depth look at make files and the make utility are beyond the scope of this article. However, two points of immediate relevance are the `bsd.kmod.mk` make file and the ability to include other make files within each other.

The `bsd.kmod.mk` makefile resides in `/usr/src/share/mk/bsd.kmod.mk` and takes all of the pain out of building and linking kernel modules properly. As you are about to see, you simply have to set two variables:

- the name of the kernel module itself via the “KMOD” variable;
- the source files configured via the intuitive “SRCS” variable;

Then, all you have to do is include `<bsd.kmod.mk>` to build the module. This elegant setup lets you build your kernel module with only the following skeletal make file and a simple invocation of the “make” utility.

The Makefile for our introductory kernel module looks like this:

```
# Note: It is important to make sure you include the <bsd.kmod.mk> makefile after declaring the K

# Declare Name of kernel module
KMOD    =  hello_fsm

# Enumerate Source files for kernel module
SRCS    =  hello_fsm.c

# Include kernel module makefile
.include <bsd.kmod.mk>
```

Create a new directory called `kernel`, under your home directory. Copy and paste the text above into a file called `Makefile`. This will be your working base going forward.

## Creating a module

Now that you have a clue about the build environment, it’s time to take a look at the actual code behind a FreeBSD kernel module and the mechanisms for inserting and removing a module from a running kernel.

A kernel module allows dynamic functionality to be added to a running kernel. When a kernel module is inserted, the “load” event is fired. When a kernel module is removed, the “unload” event is fired. The kernel module is responsible for implementing an event handler that handles these cases.

The running kernel will pass in the event in the form of a symbolic constant defined in the `/usr/include/sys/module.h` (`<sys/module.h>`) header file. The two main events you are concerned with are `MOD_LOAD` and `MOD_UNLOAD`.

## Writing a kernel module for FreeBSD

How does the running kernel know which function to call and pass an event type as a parameter to? The module is responsible for configuring that call-back as well by using the `DECLARE_MODULE` macro.

The `DECLARE_MODULE` macro is defined in the `<sys/module.h>` header on line 117. It takes four parameters in the following order:

1. `name`. Defines the name.
2. `data`. Specifies the name of the `moduledata_t` structure, which I've named `hello_conf` in my implementation. The `moduledata_t` type is defined at line 55 of `<sys/module.h>`. I'll talk about this briefly.
3. `sub`. Sets the subsystem interface, which defines the module type.
4. `order`. Defines the modules initialization order within the defined subsystem

The `moduledata` structure contains the name defined as a `char` variable and the event handler routine defined as a `modeventhand_t` structure which is defined at line 50 of `<sys/module.h>`. Finally, the `moduledata` structure has void pointer for any extra data, which you won't be using.

If your head is about to explode from the overview without any code to put in context, fear not. That is the sum of what you need to know to start writing your kernel module, and so with that, "once more into the breach dear friends". Before you get started, make sure you are in the same `kernel` directory where you previously created the `Makefile` file. Fire up your text editor of choice and open a file called `hello_fsm.c`.

First include the header files required for the data types used. You've already seen `<sys/module.h>` and the other includes are supporting header files.

```
#include <sys/param.h>
#include <sys/module.h>
#include <sys/kernel.h>
#include <sys/system.h>
```

Next, you are going to implement the `event_handler` function. This is what the kernel will call and pass either `MOD_LOAD` or `MOD_UNLOAD` to via the `event` parameter. If everything runs normally, it will return a value of 0 upon normal completion. However, you should handle the possibility that something will go wrong and if the `event` parameter is neither `MOD_LOAD` or `MOD_UNLOAD`, you will set `e`, your error tracking variable, to `EOPNOTSUPP`.

```
/* The function called at load/unload. */
static int event_handler(struct module *module, int event, void *arg) {
    int e = 0; /* Error, 0 for normal return status */
    switch (event) {
        case MOD_LOAD:
            uprintf("Hello Free Software Magazine Readers! \n");
            break;
        case MOD_UNLOAD:
            uprintf("Bye Bye FSM reader, be sure to check http://freesoftwaremagazine.com !\n");
            break;
        default:
            e = EOPNOTSUPP; /* Error, Operation Not Supported */
            break;
    }

    return(e);
}
```

## Writing a kernel module for FreeBSD

Next, you're going to define the second parameter to the `DECLARE_MODULE` macro, which is of type `moduledata_t`. This is where you set the name of the module and expose the `event_handler` routine to be called when loaded and unloaded from the kernel.

```
/* The second argument of DECLARE_MODULE. */
static moduledata_t hello_conf = {
    "hello_fsm",    /* module name */
    event_handler, /* event handler */
    NULL           /* extra data */
};
```

And finally, you're going to make a call to the much talked about `DECLARE_MODULE` with the name of the module and the `hello_conf` structure.

```
DECLARE_MODULE(hello_fsm, hello_conf, SI_SUB_DRIVERS, SI_ORDER_MIDDLE);
```

All that is left to do is build the module. Double check that you are in the same directory as the module's makefile you saw earlier and run:

```
make
```

## Loading and unloading the module

To load the module, you have two options: the `kldload` utility or the `load` make target via the `<bsd.kmod.mk>` makefile. You must use both options via the "sudo" utility as loading and unloading modules requires root privileges.

```
sudo kldload ./hello_fsm.ko
# or #
sudo make load
```

You should see the message "Hello Free Software Magazine Readers!" on your console. To view all loaded modules, use the `kldstat` utility with no arguments. `kldstat` does not require root privileges and you can verify that the module is indeed loaded.

```
kldstat
Id Refs Address      Size      Name
1     8 0xc0400000 926ed4   kernel
2     1 0xc0d27000 6a1c4    acpi.ko
3     1 0xc317e000 22000    linux.ko
4     1 0xc4146000 2000     hello_fsm.ko
```

To unload the module, use `kldunload` or the `unload` target in the `<bsd.kmod.mk>` make file. You should see the message printed on the `MOD_UNLOAD` case, which is "Bye Bye FSM reader, be sure to check <http://freesoftwaremagazine.com!>"

```
sudo kldunload hello_fsm
or
sudo make unload
```

## Conclusion

## Writing a kernel module for FreeBSD

There you have it, a basic, skeletal kernel module. It prints a message when loaded and a separate message when being unloaded from the kernel. This article covered the mechanics of building, inserting, and removing the module. You now have the basic building blocks to take on more advanced projects: I would recommend looking at writing a character device writer as it is probably the next simplest device driver.

I hope this has been as much fun for you as it has been for me!

## Resources

- [Writing device drivers in Linux](#)
- [FreeBSD's handbook](#)
- [FreeBSD's developers handbook](#)
- [FreeBSD Architecture Handbook](#)
- [FreeBSD under VMWare](#)

## Books:

- The Design and Implementation of the FreeBSD Operating System, by Marshall Kirk McKusick and George V. Neville-Neil
- Designing BSD Rootkits, an Introduction to Kernel Hacking, by Joseph Kong

## Biography

[Yousef Ourabi](#) (/user/43" title="View user profile.): [Yousef Ourabi](http://yousefourabi.com) (<http://yousefourabi.com>) is a developer in the San Francisco bay area. He is currently working at the startup he recently founded, [Zero-Analog](#) ([http://www.zero-analog.com " title="Zero-Analog](http://www.zero-analog.com "Zero-Analog")). Zero-Analog is currently developing an enterprise application, however, one of its stated goals is "to increase the rate of open source adoption in companies of all sizes, across all industries". Zero-Analog also offers consulting services, all based around open source tools, frameworks and applications.

## Copyright information

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU General Public License, Version 2 or any later version published by the Free Software Foundation. A copy of the license is available at <http://www.gnu.org/copyleft/gpl.html>.

---

### Source URL:

[http://www.freesoftwaremagazine.com/articles/writing\\_a\\_kernel\\_module\\_for\\_freebsd](http://www.freesoftwaremagazine.com/articles/writing_a_kernel_module_for_freebsd)

---