Introduction to Kernel Programming

Jesus Ramos

Panther Linux Users Group

Kernel, Operating System, Distribution

Kernel, Operating System, Distribution

Kernel

The kernel is the basic subsystem of your computer. It provides an interface for the user to the hardware. Essentially it protects the user from having to know about the specifics of the hardware they run and give them a programmable interface that is generic enough to work with easily. In some cases it protects the hardware from the user.

Kernel, Operating System, Distribution

Kernel
: The kernel is the basic subsystem of your computer. It provides an interface for the user to the hardware. Essentially it protects the user from having to know about the specifics of the hardware they run and give them a programmable interface that is generic enough to work with easily. In some cases it protects the hardware from the user.

Operating System
: The operating system is the kernel along with some basic utilities such as a basic window manager, editors, tools, etc...

Kernel, Operating System, Distribution (cont.)

Distribution

The particular distribution you choose in the case of Linux just dictates which specific window manager or particular package manager you get. Although they do have their own addition to the operating system and different configuration options, they are essentially the same even though they can look vastly different.

Kernel Programming. Why?

Kernel Programming. Why?

Open Source

Linux is one of the most successful open source operating systems out there, take advantage of this and possibly learn some things you may never have learned otherwise.

Kernel Programming. Why?

Open Source
: Linux is one of the most successful open source operating systems out there, take advantage of this and possibly learn some things you may never have learned otherwise.

Curiosity
: Maybe you just want to know how things work, because Linux is open source you have the power to just open up some files and look at exactly how it works and why.

Kernel Programming. Why? (cont.)

Why not?

Linux is a very easy operating system to build/install and requires very minimal hardware to even run. Compared to the build/install process for most other open source operating systems Linux makes it a walk in the park. Don't like that your computer is running slow? Well Linux allows you to change that.

So what do I need?

So what do I need?

Computer with Linux

That's about it. Most Linux distributions come with the base-devel packages required to build just about anything and an operating system is no exception to that. Although some distributions make it easier to do than others. Usually the easier it is to build and install the harder it is to install the kernel the distribution itself.

So what do I need?

Computer with Linux
: That's about it. Most Linux distributions come with the base-devel packages required to build just about anything and an operating system is no exception to that. Although some distributions make it easier to do than others. Usually the easier it is to build and install the harder it is to install the kernel the distribution itself.

Basic Programming Skills
: Understanding of C and concepts of memory allocation and pointers. That's it. No seriously that's it. You have to remember that the operating system sits even below the compiler itself so it doesn't have access to a lot of the fancy functions you normally see used. It's meant to be as simple as possible.

Downloading the source

https://github.com/torvalds/linux or http://kernel.org/ You can obtain the source from either Linus Torvalds himself or from the site that hosts the latest releases as well as older versions (for which more documentation is available) from the kernel.org site.

For the first option you need to have git installed and it's as simple as copying the link provided on that page into a git clone command and it will download all the files for you.

For the second option just unpack the code into any directory you want and you should be ready to go.

Building the Source Code

Building an operating system from source isn't as hard as you might think. The Make utility in Linux allows use to build an operating system in a couple of simple steps.

`make menuconfig` and save the default configuration
`make` or `make -j(number of cores + 1)` ex. `make -j9` for 8 cores

And that's it, no crazy command line wizardry needed here, just some simple commands that make calls to widely used Unix development utilities.

Installing the Kernel

Installing the kernel on some systems is as easy as typing
`make install`
The next steps however vary widely depending on your particular distribution and can range from nothing to running one or two extra commands to make sure your bootloader knows where to find the kernel to launch it.

How can it be so simple? Remember that we're just installing the kernel we're not installing any applications or setting up any partitions on a hard drive, we're just making an executable file and telling the bootloader where it is. As mentioned before, the kernel is just an interface to the hardware and not really much else.

Reading Kernel Code

Some of the kernel is written in assembly because there needs to be a layer to interface with the hardware directly but unless you're really interested in learning about hardware specifics I would stay away from it.

The rest of the kernel is coded in C. The good thing about this is that C (despite some of the neat tricks used) is a really easy language to read. Statements do exactly what they say (although can be really hard to read sometimes).

Examples

```
static inline void __list_add(struct list_head *new,
                              struct list_head *prev,
                              struct list_head *next)
{
        next->prev = new;
        new->next = next;
        new->prev = prev;
        prev->next = new;
}
```

Easy right? Most of the code in the kernel is actually pretty straightforward and usually functions aren't very long for readability purposes.

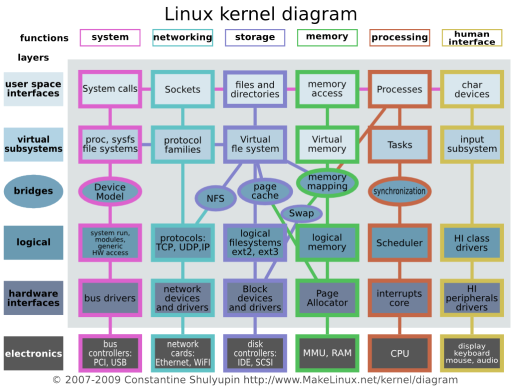
Examples (cont.)

```
#ifndef offset_of
#define offset_of(type, memb) \
        ((unsigned long)(&((type *)0)->memb))
#endif
#ifndef container_of
#define container_of(obj, type, memb) \
        ((type *)(((char *)obj) - offset_of(type, memb)))
#endif
```

Fortunately, very little of the kernel is as cryptic as this. The reason this is so cryptic is because it is essentially a little memory hack to allow for object oriented generalizations in a language that doesn't support it.

Operating System Concepts

The Linux kernel is divided into a number of easy to understand and well divided layers. The advantage of this is you don't have to learn how the whole kernel works just to work on one part.



Operating System Concepts (cont.)

File System
How else will the OS know where your files are? This is the layer where all the handling of reading from disk and organizing your files on the disk is done.

Virtual Memory
Memory is kind of a big deal. This is where the kernel handles managing memory for applications as well as memory for the kernel itself (the kernel isn't magic it needs memory too). When you launch an application this layer kicks in to make sure the application has the memory it needs to run and isn't up to any malicious activity when it comes to memory access.

Operating System Concepts (cont.)

Process Management
: Your system is running about 70+ processes at the same time and you probably only have 4 cores, but how? Process management that's how, this layer handles scheduling of processes so it looks like you're doing hundreds of things at once but in reality you're doing a couple of things at once and switching between them really, really fast.

System Calls
: Your programs need to be able to talk to the operating system somehow. The methods implemented here are your programs way of telling the operating system that you need to open a file, write to a file, check the system time, establish an internet connection and even run another program.

Questions or Comments?

If you have any questions or comments you can ask me now or you can contact me at jramo028@fiu.edu or for a copy of this presentation and any others you can visit `http://plug.cs.fiu.edu`.

PLUG is located in ECS 280C, feel free to drop by any time with general Unix/Linux questions or even ideas for projects you might be working on or want to work on or suggestions for future presentations.

If you have an idea for a future presentation that you may want to do email myself (Jesus Ramos) or Joseph Rivera (jrive034@fiu.edu) or come talk to us and we're more than willing to give you a venue to present whatever you would like.