



Linux Kernel Series

By

Devyn Collier Johnson

Linux Kernel – The Series

by

Devyn Collier Johnson (DevynCJohnson)

DevynCJohnson@Gmail.com

Introduction

In 1991, a Finnish student named Linus Benedict Torvalds made the kernel of a now popular operating system. He released Linux version 0.01 on September 1991, and on February 1992, he licensed the kernel under the GPL license. The GNU General Public License (GPL) allows people to use, own, modify, and distribute the source code legally and free of charge. This permits the kernel to become very popular because anyone may download it for free. Now that anyone can make their own kernel, it may be helpful to know how to obtain, edit, configure, compile, and install the Linux kernel.

A kernel is the core of an operating system. The operating system is all of the programs that manages the hardware and allows users to run applications on a computer. The kernel controls the hardware and applications. Applications do not communicate with the hardware directly, instead they go to the kernel. In summary, software runs on the kernel and the kernel operates the hardware. Without a kernel, a computer is a useless object.

There are many reasons for a user to want to make their own kernel. Many users may want to make a kernel that only contains the code needed to run on their system. For instance, my kernel contains drivers for FireWire devices, but my computer lacks these ports. When the system boots up, time and RAM space is wasted on drivers for devices that my system does not have installed. If I wanted to streamline my kernel, I could make my own kernel that does not have FireWire drivers. As for another reason, a user may own a device with a special piece of hardware, but the kernel that came with their latest version of Ubuntu lacks the needed driver. This user could download the latest kernel (which is a few versions ahead of Ubuntu's Linux kernels) and make their own kernel that has the needed driver. However, these are two of the most common reasons for users wanting to make their own Linux kernels.

Before we download a kernel, we should discuss some important definitions and facts. The Linux kernel is a monolithic kernel. This means that the whole operating system is on the RAM reserved as kernel space. To clarify, the kernel is put on the RAM. The space used by the kernel is reserved for the kernel. Only the kernel may use the reserved kernel space. The kernel owns that space on the RAM until the system is shutdown. In contrast to kernel space, there is user space. User space is the space on the RAM that the user's programs own. Applications like web browsers, video games, word processors, media players, the wallpaper, themes, etc. are all on the user space of the RAM. When an application is closed, any program may use the newly freed space. With kernel space, once the RAM space is taken, nothing else can have that space.

The Linux kernel is also a preemptive multitasking kernel. This means that the kernel will pause some tasks to ensure that every application gets a chance to use the CPU. For instance, if an application is running but is waiting for some data, the kernel will put that application on hold and allow another program to use the newly freed CPU resources until the data arrives. Otherwise, the system would be wasting resources for tasks that are waiting for data or another program to execute. The kernel will force programs to wait for the CPU or stop using the CPU. Applications cannot unpause or use the CPU without the kernel allowing them to do so.

The Linux kernel makes devices appear as files in the folder /dev. USB ports, for instance, are located in /dev/bus/usb. The hard-drive partitions are seen in /dev/disk/by-label. It is because of this feature that many people say "On Linux, everything is a file.". If a user wanted to access data on their memory card, for example, they cannot access the data through these device files.

The Linux kernel is portable. Portability is one of the best features that makes Linux popular. Portability is the ability for the kernel to work on a wide variety of processors and systems. Some of the processor types that the kernel supports include Alpha, AMD, ARM, C6X, Intel, x86, Microblaze, MIPS, PowerPC, SPARC, UltraSPARC, etc. This is not a complete list.

In the boot folder (/boot), users will see a "vmlinuz" or a "vmlinuz" file. Both are compiled Linux kernels. The one that ends in a "z" is compressed. The "vm" stands for virtual memory. On systems with SPARC processors, users will see a zImage file instead. A small number of users may find a bzImage file; this is also a compressed Linux kernel. No matter which one a user owns, they are all bootable files that should not be changed unless the user knows what they are doing. Otherwise, their system can be made unbootable - the system will not turn on.

Source code is the coding of the program. With source code, programmers can make changes to the kernel and see how the kernel works.

Downloading the Kernel

Now, that we understand more about the kernel, it is time to download the source code. Go to kernel.org and click the large download button. Once the download is finished, uncompress the downloaded file.

For this series, I am using the source code for Linux kernel 3.9.4. All of the instructions in this article series are the same (or nearly the same) for all versions of the kernel.

Source Code: The Files

After the kernel source code is downloaded and uncompressed, users will see many folders and files. It may be a challenge trying to find a particular file. Thankfully, the source code is sorted in a specific way. This enables developers to find any given file or part of the kernel.

The root of the kernel source code contains the folders listed below.

```
arch  
block  
crypto  
Documentation  
drivers  
firmware  
fs  
include  
init  
ipc  
kernel  
lib  
mm  
net  
samples  
scripts  
security  
sound  
tools  
usr  
virt
```

There are also some files that are located in the root of the source code. They are listed in the table below.

COPYING - Information about licensing and rights. The Linux kernel is licensed under the GPLv2 license. This license grants anyone the right to use, modify, distribute, and share the source code and compiled code for free. However, no one can sell the source code.

CREDITS - List of contributors

Kbuild - This is a script that sets up some settings for making the kernel. For example, this script sets up a ARCH variable where ARCH is the processor type that a developer wants the kernel to support.

Kconfig - This script is used when developer configure the kernel which will be discussed in a later article.

MAINTAINERS - This is a list of the current maintainers, their email addresses, website, and the specific file or part of the kernel that they specialize in developing and fixing. This is useful for when a developer finds a bug in the kernel and they wish to report the bug to the maintainer that can handle the issue.

Makefile - This script is the main file that is used to compile the kernel. This file passes parameters to the compiler as well as the list of files to compile and any other necessary information.

README - This text file provides information to developers that want to know how to compile the kernel.

REPORTING-BUGS - This text document provides information on reporting bugs.

The coding for the kernel will be in files with the extension ".c", or ".h". The ".c" extension indicates that the kernel is written in C, one of many programming languages. The ".h" files are Header files, and they are also written in C. The header files contain code that many of the ".c" files use. This saves programmers' time because they can use the contained code instead of writing new code. Otherwise, a group of code that performs the same action would be in many or all of the ".c" files. That would also consume and waste hard drive space.

All of the files in the above listed folders are well organized. The folder names help developers to at least have a good guess on the contents of the folders. A directory tree and descriptions are provided below.

arch - This folder contains a Kconfig which sets up some settings for compiling the source code that belongs in this folder. Each supported processor architecture is in the corresponding folder. So, the source code for Alpha processors belong in the alpha folder. Keep in mind that as time goes on, some new processors will be supported, or some may be dropped. For Linux Kernel v3.9.4, these are the folders under arch:

```
alpha
arc
arm
arm64
avr32
blackfin
c6x
cris
frv
h8300
hexagon
ia64
m32r
m68k
metag
microblaze
mips
mn10300
openrisc
parisc
powerpc
s390
score
sh
sparc
tile
um
```

unicore32
x86
xtensa

block - This folder holds code for block-device drivers. Block devices are devices that accept and send data in blocks. Data blocks are chunks of data instead of a continual stream.

crypto - This folder contains the source code for many encryption algorithms. For example, "sha1_generic.c" is the file that contains the code for the sha1 encryption algorithm.

Documentation - This folder contains plain-text documents that provide information on the kernel and many of the files. If a developer needs information, they may be able to find the needed information in here.

drivers - This directory contains the code for the drivers. A driver is software that controls a piece of hardware. For example, for a computer to understand the keyboard and make it usable, a keyboard driver is needed. Many folders exist in this folder. Each folder is named after each piece or type of hardware. For example, the bluetooth folder holds the code for bluetooth drivers. Other obvious drivers are scsi, usb, and firewire. Some drivers may be more difficult to find. For instance, joystick drivers are not in a joystick folder. Instead, they are under ./drivers/input/joystick. Keyboard and mouse drivers are also located in the input folder. The Macintosh folder contains code for hardware made by Apple. The xen folder contains code for the Xen hypervisor. A hypervisor is software or hardware that allows users to run multiple operating systems on a single computer. This means that the xen code would allow users to have two or more Linux system running on one computer at the same time. Users could also run Windows, Solaris, FreeBSD, or some other operating system on the Linux system. There are many other folders under drivers, but they are too numerous to mention in this article, but they will in a later article.

firmware - The firmware folder contains code that allows the computer to read and understand signals from devices. For illustration, a webcam manages its own hardware, but the computer must understand the signals that the webcam is sending the computer. The Linux system will then use the vicam firmware to understand the webcam. Otherwise, without firmware, the Linux system does not know how to process the information that the webcam is sending. Also, the firmware helps the Linux system to send messages to the device. The Linux system could then tell the webcam to refocus or turnoff.

fs - This is the FileSystem folder. All of the code needed to understand and use filesystems is here. Inside this folder, each filesystem's code is in its own folder. For instance, the ext4 filesystem's code is in the ext4 folder. Within the fs folder, developers will see some files not in folders. These files handle filesystems overall. For example, mount.h would contain code for mounting filesystems. A filesystem is a structured way to store and manage files and directories on a storage device. Each filesystem has its own advantages and disadvantages. These are due to the programming of the filesystem. For illustration, the NTFS filesystem supports transparent compression (when enabled, files are automatically compressed without the user noticing). Most filesystems lack this feature, but they could only possess this ability if it is programmed into the files in the fs folder.

include - The include folder contains miscellaneous header files that the kernel uses. The name for the folder comes from the C command "include" that is used to import a header into C code upon compilation.

init - The init folder has code that deals with the startup of the kernel (INITiation). The main.c file is the core of the kernel. This is the main source code file that connects all of the other files.

ipc - IPC stands for Inter-Process Communication. This folder has the code that handles the communication layer between the kernel and processes. The kernel controls the hardware and programs can only ask the kernel to perform a task. Assume a user has a program that opens the DVD tray. The program does not open the tray directly. Instead, the program informs the kernel that the tray should be opened. Then, the kernel opens the tray by sending a signal to the hardware. This code also manages the kill signals. For illustration, when a system administrator opens a process manager to close a program that has locked-up, the signal to close the program is called a kill signal. The kernel receives the signal and then the kernel (depending on which type of kill signal) will ask the program to stop or the kernel will simply take the process out of the memory and CPU. Pipes used in the command-line are also used by the IPC. The pipes tell the kernel to place the output data on a physical page on in memory. The program or command receiving the data is given a pointer to the page on memory.

kernel - The code in this folder controls the kernel itself. For instance, if a debugger needed to trace an issue, the kernel would use code that originated from source files in this folder to inform the debugger of all of the actions that the kernel performs. There is also code here for keeping track of time. In the kernel folder is a directory titled "power". Some code in this folder provide the abilities for the computer to restart, power-off, and suspend.

lib - the library folder has the code for the kernel's library which is a set of files that the kernel will need to reference.

mm - The Memory Management folder contains the code for managing the memory. Memory is not randomly placed on the RAM. Instead, the kernel places the data on the RAM carefully. The kernel does not overwrite any memory that is being used or that holds important data.

net - The network folder contains the code for network protocols. This includes code for IPv6 and Appletalk as well as protocols for Ethernet, wifi, bluetooth, etc. Also, the code for handling network bridges and DNS name resolution is in the net directory.

samples - This folder contains programming examples and modules that are being started. Assume a new module with a helpful feature is wanted, but no programmer has announced that they would work on the project. Well, these modules go here. This gives new kernel programmers a chance to help by going through this folder and picking a module they would like to help develop.

scripts - This folder has the scripts needed for compiling the kernel. It is best to not change anything in this folder. Otherwise, you may not be able to configure or make a kernel.

security - This folder has the code for the security of the kernel. It is important to protect the kernel from computer viruses and hackers. Otherwise, the Linux system can be damaged. Kernel security will be discussed in a later article.

sound - This directory has sound driver code for sound/audio cards.

tools - This directory contains tools that interact with the kernel.

usr - Remember the vmlinuz file and similar files mentioned in the previous article? The code in this folder creates those files after the kernel is compiled.

virt - This folder contains code for virtualization which allows users to run multiple operating systems at once. This is different from Xen (mentioned previously). With virtualization, the guest operating system is acting like any other application within the Linux operating system (host system). With a hypervisor like Xen, the two operating systems are managing the hardware together and the same time. In virtualization, the guest OS runs on top of the Linux kernel while in a hypervisor, there is no guest OS and all of the operating systems do not depend on each other.

Tip: Never move a file in the kernel source unless you know what you are doing. Otherwise, the compilation will fail due to a "missing" file.

The Linux kernel folder structure has remained relatively constant. The kernel developers have made some modifications, but overall, this setup is the same throughout all kernel versions. The driver folder's layout also remains about the same.

Drivers

Drivers are small programs that enable the kernel to communicate and handle hardware or protocols (rules and standards). Without a driver, the kernel does not know how to communicate with the hardware or handle protocols (the kernel actually hands the commands to the BIOS and the BIOS passes them on to the hardware). The Linux Kernel source code contains many drivers (in the form of source code) in the drivers folder. Each folder within the drivers folder will be explained. When configuring and compiling the kernel, it helps to understand the drivers. Otherwise, a user may add drivers to the kernel that they do not need or leave out important drivers. The driver source code usually includes a commented line that states the purpose of the driver. For example, the source code for the tc driver has a single commented line that says the driver is for TURBOchannel buses. Because of the documentation, users should be able to look at the first few commented lines of future drivers to learn their purpose.

There are different terms that should be understood so that the information below is understandable. An I/O device is an Input/Output device. A modem and network card are examples of this; they send and receive data. A monitor is an output device - information only comes out. A keyboard, mouse, and joystick are input only - data goes into the system. Storage devices store data. Examples of these include SD cards, Hard-drives, CD-roms, memory cards, etc. The CPU (also called a processor) is the "brain" or "heart" of the computer. Without this single processing chip, the computer cannot function. A motherboard (mainboard) is a small board with printed circuits that connect to the components that are on the board. The board and the components are essential to the functionality of a computer. Many computer users say that the motherboard is the heart of the computer (the motherboard holds the CPU). The motherboard contains ports for peripherals. Peripherals include the input, output, and storage devices. A bus is the circuitry of the motherboard and connection to peripherals. Network devices deal with the connection of two or more computers. Ports are devices that users can plug another device or cable into. For example, users would plug a FireWire memory stick into a FireWire port; an Ethernet cable would go into an Ethernet port. Optical discs are read using lasers that read off of reflective surfaces that will either scatter or reflect the laser light. A common optical disk is the DVD. Many systems are 32-bit or 64-bit systems; this refers to the number of bits of registers, address buses, or data buses. For instance, on a 64-bit motherboard, the data buses (the silver lines going between components) have sixty-four lines running parallel to the same destination. Memory addresses are addresses to the memory in the form of bits (ones and zeros). So, a 32-bit memory address contains thirty-two ones and zeros that give the location of a spot on the memory.

Many of the drivers are generic driver meaning that a generic keyboard driver will allow the kernel to handle nearly every keyboard. However, some drivers are specialized. Apple and Commodore, for instance, have made specialized hardware for their Apple computer and Amiga system, respectively. The Linux kernel contains drivers for many devices like Smartphones, Apples, Amiga systems, Sony's Playstation3, Android tablets, and many others.

Notice that some drivers overlap categories. For instance, radio drivers exist in the net folder and the media directory.

accessibility - These drivers offer support for accessibility devices. In Linux kernel 3.9.4, only one driver is in this folder, and that is the braille device driver.

acpi - The Advanced Configuration and Power Interface (ACPI) drivers manage power usage.

amba - Advanced Microcontroller Bus Architecture (AMBA) is a protocol for the management and interconnection in a System-on-Chip (SoC). A SoC is a single chip that contains many or all essential components of a computer in one chip. The AMBA drivers in this folder allow the kernel to run on these chips.

ata - This directory contains drivers for PATA and SATA devices. Serial ATA (SATA) is a computer bus interface that connects host bus adapters to storage devices like hard-drives. Parallel ATA (PATA) is a standard for connecting storage devices like hard-drives, floppy drives, and optical disc drives. PATA is commonly known as IDE.

atm - Asynchronous Transfer Mode (ATM) is a standard for telecommunications. There are a variety of drivers in here from PCI bridges (they connect to PCI buses) and Ethernet controllers (integrated circuit chip that controls Ethernet communications).

auxdisplay - This folder provides three drivers - LCD framebuffer driver, LCD Controller driver, and a LCD driver. These drivers manage Liquid Crystal Display monitors. LCD monitors are the screens that ripple when pressed. NOTE: The screen can be damaged when pressed, so do not poke or push on LCD screens.

base - This is an important directory that contains essential base drivers for firmware, the system bus, hypervisor abilities, etc.

bcma - These are drivers for buses that use a protocol that are based on the AMBA protocol. These new buses are made by Broadcom.

block - These drivers provide the kernel with support for block devices like floppy-disk readers, SCSI tapes, block devices over TCP, etc.

bluetooth - Bluetooth is a secure wireless standard for Personal Area Networks (PANs). The bluetooth drivers are in this folder and allow the system to use the different bluetooth devices. For example, a bluetooth mouse lacks a cable, and the computer has a dongle (small USB receiver). The Linux system must be able to understand the mouse signals that are coming in through the dongle. Otherwise, the bluetooth device would not work.

bus - This directory contains three drivers. One converts the ocp interface protocol to scp protocol. The other is a driver for interconnection between devices and the third driver is error handling for interconnection.

cdrom - Two drivers exist in this directory. One is for cd-roms - this includes reading and writing DVDs and CDs. The driver second is for gd-roms (Gigabyte Disc Read-Only Memory). A GD is an optical disc with a 1.2GB storage capacity. This is like a large CD or small DVD. GDs are commonly used in Dreamcast game consoles.

char - Character device drivers are stored here. Character devices transmit data one character at a time. Some included drivers in the folder are printers, PS3 flash ROM storage driver, Toshiba SMM drivers, and random number generator driver.

clk - These drivers are for the system clock.

clocksource - These drivers use the clock as a timer.

connector - These drivers supply the kernel with the ability to know when processes fork and execute as well as changing the UID (User ID), GID (Group ID), and SID (session ID) using what is called a proc connector. The kernel needs to know when process fork (run multiple tasks in the CPU) and execute. Otherwise, the kernel may have inefficiencies in managing resources.

cpufreq - These drivers control the CPU by changing power consumption.

cpuidle - These drivers manage the idleness of the CPU/s. If the system uses multiple CPUs, then one of the drivers will try to keep the idleness the same.

crypto - These drivers provide cryptographic features like encryption.

dca - Direct Cache Access drivers allow the kernel to access the CPU cache. The CPU cache is like a RAM storage built into the CPU chip. The CPU cache is faster than the RAM chip. However, the CPU cache has a much lower space capacity than the RAM. The CPU stores the most important and executed code on this cache system.

devfreq - This driver provides a Generic Dynamic Voltage and Frequency Scaling (DVFS) Framework which changes the CPU frequency as needed to conserve energy. This is known as CPU throttling.

dio - The Digital Input/Output bus drivers allow the kernel to use DIO buses.

dma - The Direct memory access (DMA) driver allows devices to access without needing the CPU. This reduces the load on the CPU.

edac - The Error Detection And Correction drives help reduce and correct errors.

eisa - The Extended Industry Standard Architecture drivers provide EISA bus support to the kernel.

extcon - The EXTernal CONnectors driver detects changes in the ports when a device is plugged in. For instance, extcon will detect if a user plugs in a USB drive.

firewire - These drivers control FireWire devices which are USB-like devices made by Apple.

firmware - These drivers communicate with the firmware of the device like the BIOS (the Basic Input Output System firmware of a computer). The BIOS is used to boot up the operating system and control the hardware and firmware of the device. Some BIOS systems allow user's to overclock the CPU. Overclocking is making the CPU operate at a faster speed. The CPU speed is measured in MHz (Mega-Hertz) or GHz. A CPU with a clock speed of 3.7GHz is significantly faster than a 700MHz processor.

gpio - General Purpose Input/Output (GPIO) is a generic pin on a chip whose behavior can be controlled by the user. The drivers here control GPIO.

gpu - The drivers in this folder control the Video Graphics Array (VGA), Graphics Processing Unit (GPU), and Direct Rendering Manager (DRM). VGA is the 640×480 resolution analog computer displays or simply the resolution standard. A GPU is a processor for graphics. DRM is a rendering system for Unix systems.

More Linux Related Stuff on: <http://www.linux.org>

hid - These drivers provide support for USB Human Interface Devices.

hsi - This driver offers the kernel the ability to access the cellular modem on the Nokia N900.

hv - These drivers provide Key Value Pair (KVP) functionality to Linux systems.

hwmon - The HardWare MONitoring drivers allow the kernel to get the readings from sensors in the hardware. For example, the CPU has a thermometer. The kernel can then keep track of the temperature and change the fan speed accordingly.

hwspinlock - The hardware spinlock drivers allow systems to use two or more processors that are different or a single processor with two or more different cores.

i2c - I2C drivers enable the I2C protocol which handle low-speed peripherals attached to the motherboard. The System Management Bus (SMBus) driver manages SMBuses which is a single two-wire bus for lightweight communication.

ide - These drivers are for PATA/IDE devices like cdroms and hard-drives.

idle - This driver manages the idleness of Intel processors.

ii0 - The Industrial I/O core drivers handle analog to digital or digital to analog converters.

infiniband - Infiniband is a high-performance port used by enterprise datacenters and some supercomputers. The drivers in this directory support Infiniband hardware.

input - This directory contains many drivers. All of the drivers deal with input and some include drivers for joysticks, mice, keyboards, gameport (old joystick connectors), remotes, haptic controllers, headphone buttons, and many others. Joysticks today use USB ports, but in the 1980s and 1990s, joysticks plugged into gameports.

iommu - Input/Output Memory Management Unit (IOMMU) drivers manage the IOMMU which is a form of Memory Management Unit (MMU). The IOMMU connects a DMA-capable I/O bus to the RAM. The IOMMU is the bridge between devices and access to the RAM without help from the CPU. This helps to reduce the processors load.

ipack - Ipak stands for IndustryPack. This driver is for a virtual bus that allows operations between carrier and mezzanine boards.

irqchip - These drivers allow interrupt requests (IRQ) which are hardware signals sent to the processor that temporarily stop a running program for a special program(called an interrupt handler) to run instead.

isdn - These drivers support Integrated Services Digital Network (ISDN) which is a set of communication standards for simultaneous digital transmission of voice, video, data, and other network services using traditional circuits of the telephone network.

leds - These drivers support LEDS.

More Linux Related Stuff on: <http://www.linux.org>

Iguest - The Iguest drivers manage interrupts that are used with guest operating system. Interrupts are hardware or software signals that interrupt the CPU for important tasks. The CPU then gives the hardware or software some processing resources.

macintosh - Drivers for Apple devices belong in this directory.

mailbox - The driver in this folder (pl320-pci) manages connections for mail systems.

md - The Multiple Devices driver supports RAID (Redundant Array of Independent Disks) - a system of many hard-drives sharing or replicating data.

media - The media drivers offer support in radios, tuners, video capturers, DVB standards for digital television, etc. The drivers also support various media devices that plug in through USB or FireWire ports.

memory - This important driver supports the RAM.

memstick - This driver supports Sony memorysticks.

message - These drivers are to be used with LSI PCI chip/adapter(s) running LSI Fusion MPT (Message Passing Technology) firmware. LSI stands for Large-Scale Integration which are integrated circuits with tens of thousands of transistors per chip.

mfd - MultiFunction Device (MFD) drivers provide support for multifunction devices which are devices that provide multiple services like email, fax, copy machine, scanner, and printer. The drivers in here also add a generic MCP (Multimedia Communications Port) layer which is a protocol for MFDs.

misc - This directory contains miscellaneous drivers that do not fit in any other category like light sensor drivers.

mmc - MultiMediaCard (MMC) drivers handle the MMC standard that is used in flash memory cards.

mtd - Memory technology devices (MTD) are drivers used in Linux for interacting with flash memory like a Flash Translation Layer. Other block and character drivers do not map the same way flash memory devices operate. Although USB memory cards and SD cards are flash drives, they do not use this driver because they are hidden from the system behind a block device interface. This driver is a generic flash drive driver for new flash devices.

net - The network drivers provide network protocols like Appletalk, TCP, and others. The drivers also support modems, USB 2.0 Ethernet Devices, and radio devices.

nfc - This driver is the interface between Texas Instrument's Shared Transport Layer and NCI core.

ntb - The Non-Transparent Bridging driver provides non-transparent bridging in PCI express (PCIe) systems. PCIe is a high-speed expansion bus standard.

nubus - NuBus is a 32-bit parallel computer bus. The driver supports this Apple device.

More Linux Related Stuff on: <http://www.linux.org>

of - This driver provides OF helpers which are procedures for creating, accessing and interpreting the device tree. The Device Tree is a data structure for describing hardware.

oprofile - This driver profiles the whole system from drivers to user-space processes (applications running under the user's name). This helps developers find performance problems.

parisc - These drivers are for PA-RISC devices which are made by HP. PA-RISC is a specific instruction set for processors.

parport - The Parport drivers provides parallel-port support under Linux.

pci - These drivers offer PCI bus services.

pcmcia - These are laptop motherboard drivers.

pinctrl - These drivers handle pin control devices. Pin controllers can disable and enable I/O devices.

platform - This directory contains drivers for the different computer platforms like Acer, Dell, Toshiba, IBM, Intel, ChromeBooks, etc.

pnp - The Plug-aNd-Play drivers allow users to plug in a device, like a USB device, and use it immediately without the need to manually configure the device.

power - The power drivers allow the kernel to measure the battery power, detect chargers, and power management.

pps - Pulse-Per-Second drivers control an electrical pulse rate that is used for time keeping.

ps3 - These are the drivers for Sony's game console - Playstation3.

ptp - Picture Transfer Protocol (PTP) drivers support a protocol for transferring images from digital cameras.

pwm - Pulse-width modulation (PWM) drivers control the pulse of the electricity to devices, mainly motors like the CPU fan.

rapidio - RapidIO drivers manage the RapidIO architecture which is a high-performance packet-switched, interconnect technology for interconnecting chips on a circuit board, and also circuit boards to each other using a backplane.

regulator - The regulator drivers regulate the electricity, temperature, and any other regulator hardware that may exist on a system.

remoteproc - These drivers manage remote processors.

rpmmsg - These drivers control Remote Processor MeSsaginG buses which can support a number of drivers. These buses supply the messaging infrastructure, facilitating client drivers to write their own wire-protocol messages.

More Linux Related Stuff on: <http://www.linux.org>

rtc - The Real Time Clock (RTC) drivers allow the kernel to read the clock.

s390 - The drivers are for the 31/32-bit mainframe architecture.

sbus - The SPARC-based buses are managed by these drivers.

scsi - SCSI drivers allow the kernel to use the SCSI standard with peripheral devices. For instance, Linux would be using a SCSI driver when it transmits data with a SCSI hard-drive.

sfi - The Simple Firmware Interface (SFI) drivers allow firmware to send tables of information to the operating system. These tables of data are called SFI tables.

sh - These drivers are for SuperHyway buses.

sn - These drivers add support for IOC3 serial ports.

spi - These drivers handle the Serial Peripheral Interface Bus (SPI bus) which is a synchronous serial data link standard that operates in full duplex mode. Full duplex mode is seen when two devices can both send and receive information at the same time. Duplex refers to two-way communication. Devices communicate in master/slave mode (device configuration).

ssb - Sonics Silicon Backplane drivers provide support for a mini-bus used on various Broadcom chips and embedded devices.

staging - This directory contains numerous subdirectories with many drivers. All of the contained drivers are drivers that need more development before being added to the mainstream kernel.

target - These are drivers for SCSI targets.

tc - These are drivers for TURBOchannel. TURBOchannel is a 32-bit open bus developed by the Digital Equipment Corporation. These buses are commonly used in DECstations.

thermal - The thermal drivers make sure that the CPU stays cool.

tty - The tty drivers manage the connection to a physical terminal.

uio - This driver allows the user to make drivers that run in the user space instead of the kernel space. This keeps the user's driver from causing the kernel to crash.

usb - The USB drivers allow the kernel to use USB ports. Flash drivers and memory cards already contain firmware and a controller, so these drivers allow the kernel to use the USB ports and talk to the USB device.

uwb - The Ultra-WideBand driver manages very low energy level radio devices for short-range, high-bandwidth communications.

vfio - The VFIO driver allows devices to access the userspace.

vhost - This driver is for a virtio server in the host kernel. This is used for virtualization.

video - Video drivers are needed to manage the graphics card and monitor.

virt - These drivers are for virtualization.

virtio - This driver allows virtio devices to be used over a virtual PCI device. This is used for virtualization.

vlynq - This driver controls a proprietary interface developed by Texas Instruments. These are broadband products, like WLAN and modems, VOIP processors, and audio and digital media processor chips.

vme - VMEbus is a bus standard originally developed for the Motorola 68000 line of processors.

w1 - These drivers control one-wire buses.

watchdog - This driver manages the watchdog timer which is a timer that is used to detect and recover from malfunctions.

xen - This driver is for the Xen hypervisor system. A hypervisor is software or hardware that allows users to run multiple operating systems on a single computer. This means that the xen code would allow users to have two or more Linux system running on one computer at the same time. Users could also run Windows, Solaris, FreeBSD, or some other operating system on the Linux system.

zorro - This driver offers support for the Zorro Amiga buses.

Security

The Linux kernel is the core of all Linux systems. If any malicious code controls or damages any part of the kernel, then the system can get severely damaged, files can be deleted or corrupted, private information can be stolen, etc. Clearly, it is in the user's best interest to keep the kernel secure.

Thankfully, Linux is a very secure system because of the kernel and its security. There are less Linux viruses than Windows viruses even in proportion to the number of users, and Linux users get less viruses than Windows users. (This is one reason why many companies use Linux to manage their servers.) However, this is no excuse to neglect the kernel's security. Linux has many security features and programs, but only the Linux Security Modules (LSM) and other kernel security will be discussed in this article.

AppArmor (Application Armor) is a security module originally made by Immunix. Since 2009, Canonical maintains the code (Novell handled the code after Immunix and before Canonical). This security module has been in the mainstream Linux kernel since version 2.6.36. AppArmor restricts the abilities of programs. AppArmor uses file paths to keep track of program restrictions. Many Linux administrators claim that AppArmor is the easiest security module to configure. However, many Linux users feel that this module provides the worst security compared to alternatives.

Security-Enhanced Linux (SELinux) is an alternative to AppArmor originally made by the United States National Security Agency (NSA). SELinux has been in the mainstream kernel since version 2.6. SELinux makes modifications to the kernel and user-space tools. SELinux gives executables (mainly daemons and server applications) the minimum privileges required to complete their tasks. SELinux could also be used to control user privileges. SELinux does not use file paths like AppArmor, instead SELinux uses the filesystem to mark executables when keeping track of permissions. Because SELinux uses the filesystem itself for managing executables, SELinux cannot offer protection on all filesystems while AppArmor can provide protection.

NOTE: A daemon (pronounced DAY-mon) is a program that runs in the background.

NOTE: Although AppArmor, SELinux, and others are in the kernel, only one security module can be active.

Smack is another choice for a security module. Smack has been in the mainstream Linux kernel since version 2.6.25. Smack is supposed to offer more security than AppArmor and easier configuration than SELinux.

TOMOYO, another security module, has been in the Linux kernel since version 2.6.30. TOMOYO offers security, but its main use is analyzing the system for security flaws.

AppArmor, SELinux, Smack, and TOMOYO make up the four standard LSM modules. All four work by using mandatory access control (MAC) which is a type of access control that restricts a program or user from executing some task. The LSMs have some form of a list of entities and what they are permitted and not permitted to do.

Yama is a new security module that comes with the Linux kernel. Yama is not yet considered a standard LSM module, but in the future, it may be the fifth standard LSM module. Yama uses the same principals as the other security modules.

"grsecurity" is a collection of security patches for enhancing the Linux kernel's security. The majority of the patches apply to remote network connections and buffer overflows (discussed a little later). One interesting component of grsecurity is PaX. PaX patches allow code on memory to use the least amount of needed privileges. For example, memory containing programs is marked as non-writable. Think about it, why would an executed program need to be written while in memory? Now, malicious code cannot change currently executed applications. A buffer overrun is the event where a program (bug or malicious code) write data on memory and goes past its space boundary into the memory pages for other applications. When PaX is active, it helps to prevent these buffer overruns because the program will not have permission to write on other memory pages.

The Linux Intrusion Detection System (LIDS) is a kernel security patch that adds Mandatory Access Control (MAC) features. This patch acts like a LSM module.

Systrace is a utility that reduces and controls application's access to system files and use of system calls. System calls are service requests to the kernel. For instance, when a text editor writes a file to the hard-drive, the applications makes a system call requesting that the kernel write the file to the hard-drive.

These are very important components in the Linux security system. These security modules and patches keep malicious code from attacking the kernel. Without these features, Linux systems would be unsecure computer operating systems.

Configuring the Kernel Part I

Now that we understand the Linux kernel, we can move on to the main event - configuring and compiling the code. Configuring code for the kernel does take a lot of time. The configuration tool asks many questions and allows developers to configure every aspect of the kernel. If unsure about any question or feature, it is best to pick the default value provided by the configuration tool. This tutorial series will walk readers through the whole process of configuring the kernel.

To configure the code, open a terminal in the main source code folder. Once a terminal is up, there are a few ways to configure the code based on the preferred configuration interface.

make config - Plain text interface (most commonly used choice)

make menuconfig - Text-based with colored menus and radiolists. This option allows developers to save their progress. ncurses must be installed (sudo apt-get install libncurses5-dev).

make nconfig - Text-based colored menus - curses (libcdk5-dev) must be installed

make xconfig - QT/X-windows interface – QT is required

make gconfig - Gtk/X-windows interface – GTK is required

make oldconfig - Plain-text interface that updates a config file (that may have been copied from another system) to be compatible with the newer kernel source code.

make silentoldconfig - This is the same as oldconfig except the questions answered by the config file will not be shown

make olddefconfig - This is like silentoldconfig except some questions are answered by their defaults

make defconfig - This option creates a config file that uses default settings based on the current system's architecture.

make \${PLATFORM}_defconfig - Creates a config file using values from arch/\$ARCH/configs/\${PLATFORM}_defconfig.

make allyesconfig - This option creates a config file that will answer yes to as many questions as possible.

make allmodconfig - This option creates a config file that will make as many parts of the kernel a module as possible

NOTE: Code in the Linux kernel can be put in the kernel itself or made as a module. For instance, users can add Bluetooth drivers as a module (separate from the kernel), add to the kernel itself, or not add at all. When code is added to the kernel itself, the kernel requires more RAM space and boot-up time may take longer. However, the kernel will perform better. If code is added as modules, the code will remain on the hard-drive until the code is needed. Then, the module is loaded to RAM. This will reduce the kernel's RAM usage and decrease boot time. However, the kernel's performance may suffer because the kernel and the modules will be spread throughout the RAM. The other choice is to not add some code.

More Linux Related Stuff on: <http://www.linux.org>

For illustration, a kernel developer may know that a system will never use Bluetooth devices. As a result, the drivers are not added to the kernel. This improves the kernel's performance. However, if users later need Bluetooth devices, they will need to install Bluetooth modules or update the whole kernel.

make allnoconfig - This option creates a config file that will only add essential code to the kernel; this answers no to as many questions as possible. This can sometimes make a kernel that does not work on the hardware it was compiled on.

make randconfig - This option makes random choices for the kernel

make localmodconfig - This option creates a config file based on the current list of loaded modules and system configuration.

make localyesconfig - This will set all module options to yes - most (or all) of the kernel will not be in modules

TIP: It is best to use "make menuconfig" because users can save their progress. "make config" does not offer this luxury. Because the configuration process takes a lot of time,

Configuration:

Most developers choose "make menuconfig" or one of the other graphical menus. After typing the desired command, the first question asks whether the kernel to be built is going to be a 64-bit kernel or not. The choices are "Y", "n", and "?". The question mark explains the question, "n" answers no to the question, and "Y" answers yes to the question. For this tutorial, I will choose yes. To do this I type "Y" (this is case-insensitive) and hit enter.

UPDATED NOTE: At first, I used "make config" as I wrote this series. I later used "make menuconfig" (ncurses). "make config" is linear, that is, each question is asked one after another while the ncurses interface is a branching menu. If you are using any of the configuration interfaces that use menus, start at the top of the first (main) menu and work your way down going into each menu and sub-menu. This will allow you to follow this series easily as you configure your own kernel.

NOTE: If the kernel is compiled on a 32-bit system, then the configuration tool would ask if the kernel should be 32-bit. The first question is different on other processors.

The next line shows "Cross-compiler tool prefix (CROSS_COMPILE) []". If you are not cross-compiling, hit enter. If you are cross-compiling, type something like "arm-unknown-linux-gnu-" for ARM systems or "x86_64-pc-linux-gnu-" for 64-bit PC systems. There are many other possible commands for other processor types, but the list can be quite large. Once a developer knows what processor they want to support, it is easy to research the command needed for that processor.

NOTE: Cross-compiling is compiling code to be used on other processors. For illustration, an Intel system that is cross-compiling code is making applications for processors other than Intel. So, this system may be compiling code for ARM or AMD processors.

NOTE: Each choice will change which questions come up and when they are displayed. I will include my choices so readers can follow the configuration process on their own system.

Next, users will see "Local version - append to kernel release (LOCALVERSION) []". This is where

More Linux Related Stuff on: <http://www.linux.org>

developers can give a special version number or name to their customized kernel. I will type "LinuxDotOrg". The kernel version is now "3.9.4-LinuxDotOrg". Next, the configuration tool asks "Automatically append version information to the version string (LOCALVERSION_AUTO) [N/y/?]". If a git tree is found, the revision number will be appended. This example is not using git, so I will answer no. Other wise the git revision number will be appended to the version. Remember vmlinuz and similar files? Well, the next question asks which compression format should be used. The developer can choose one through five. The choices are

1. Gzip (KERNEL_GZIP)
2. Bzip2 (KERNEL_BZIP2)
3. LZMA (KERNEL_LZMA)
4. XZ (KERNEL_XZ)
5. LZO (KERNEL_LZO)

Gzip is the default, so I will press "1" and hit enter. Each compression format has greater or less compression ratios compared to the other formats. A better compression ratio means a smaller file, but more time is needed to uncompress the file while the opposite applies to lower compression ratios.

Now, this line is displayed - "Default hostname (DEFAULT_HOSTNAME) [(none)]". The default hostname can be configured. Usually, developers leave this blank (I left it blank) so that Linux users can set up their own hostname.

Next, developers can enable or disable the use of swap space. Linux uses a separate partition called "swap space" to use as virtual memory. This is equivalent to Windows' paging file. Typically, developers answer yes for the line "Support for paging of anonymous memory (swap) (SWAP) [Y/n/?]".

The next line (System V IPC (SYSVIPC) [Y/n/?]) asks if the kernel should support IPC. Inter Process Communication allows processes to communicate and sync. It is best to enable IPC, otherwise, many applications will not work. Answering yes to this question will cause the configuration tool to ask "POSIX Message Queues (POSIX_MQUEUE) [Y/n/?]". This question will only be seen if IPC is enabled. POSIX message queues is a messaging queue (a form of interprocess communication) where each message is given a priority. The default choice is yes. Hit enter to choose the default choice (indicated by the capitalized choice).

The next question (open by fhandle syscalls (FHANDLE) [Y/n/?]) is asking if programs will be permitted to use file handles instead of filenames when performing filesystem operations if needed. By default, the answer is yes.

Sometimes, when a developer has made certain choices, some questions will automatically be answered. For instance, the next question (Auditing support (AUDIT) [Y/?]) is answered yes without prompting because previous choices require this feature. Auditing-support logs the accesses and modifications of all files. The next question relates to auditing (Enable system-call auditing support (AUDITSYSCALL) [Y/n/?]). If enabled, all system calls are logged. If the developer wants performance, then as much auditing features as possible should be disabled and not added to the kernel. Some developers may enable auditing for security monitoring. I will select "no" for this question. The next audit question (Make audit loginuid immutable (AUDIT_LOGINUID_IMMUTABLE) [N/y/?]) is asking if processes can change their loginuid (LOGIN User ID). If enabled, processes in userspace will not be able to change their own loginuids. For better performance, we will disable this feature.

More Linux Related Stuff on: <http://www.linux.org>

NOTE: When configuring via “make config”, the questions that are answered by the configuration tool are displayed, but the user does not have a way to change the answer. When configuring via “make menuconfig”, the user cannot change the option no matter what button is pressed. Developers should not want to change options like that anyway because a previous choice requires another question to be answered a certain way.

Configuring the Kernel Part II

The next portion of the kernel that is configured is the IRQ subsystem. An Interrupt ReQuest (or IRQ) is a signal from the hardware to the processor to temporarily stop a running program and allow a special program to execute in its place.

The first question belonging to this category of kernel features (Expose hardware/virtual IRQ mapping via debugfs (IRQ_DOMAIN_DEBUG)) asks if the IRQ numbers of the hardware and the respective can be mapped using a virtual debugging filesystem. This is for debugging purposes. Most users will not need this, so I will select “no”.

The next heading displays “Timers subsystem”. The first question pertaining to the timers subsystem is “Tickless System (Dynamic Ticks) (NO_HZ)”. Selecting “yes” (which I did) will enable a tickless system. This means that the timer interrupts will be used as needed. Timer interrupts allow tasks to be executed at particular timed intervals. The next question (High Resolution Timer Support (HIGH_RES_TIMERS)) asks if high resolution timer support can be enabled. Not all hardware supports this. Generally, if the hardware is slow or old, then select “no”, else select “yes” as I did.

The next heading - “CPU/Task time and stats accounting” - pertains to keeping track of processes. The first question looks like this

Cputime accounting

1. Simple tick based cputime accounting (TICK_CPU_ACCOUNTING)
2. Full dynticks CPU time accounting (VIRT_CPU_ACCOUNTING_GEN) (NEW)
3. Fine granularity task level IRQ time accounting (IRQ_TIME_ACCOUNTING)

TICK_CPU_ACCOUNTING checks /proc/stat on every CPU tick. This is the default choice. This accounting method is very simple.

NOTE: A CPU tick is an abstract way of measuring time by the CPU. Every processor, operating system, and installation is different. For example, a more powerful processor will have more CPU ticks than an old processor. If you install a Linux system and then reinstall from the same disk, you may have a faster or slower CPU tick time (at least that is what some computer techs say). Generally, a larger clock speed means more CPU ticks.

If VIRT_CPU_ACCOUNTING_GEN is enabled, task and CPU time accounting will be implemented by watching kernel-user boundaries. This choice comes with a price – extra overhead.

The IRQ_TIME_ACCOUNTING accounting method works by checking a time-stamp between IRQ states. The performance cost is small.

I chose “1” and was then asked about BSD Accounting “BSD Process Accounting (BSD_PROCESS_ACCT)”. This kernel feature logs a variety of information for each process that closes. For a smaller and faster kernel, I will choose “no”.

The next set of questions looks like the following.

Export task/process statistics through netlink (TASKSTATS)
Enable per-task delay accounting (TASK_DELAY_ACCT)

More Linux Related Stuff on: <http://www.linux.org>

Enable extended accounting over taskstats (TASK_XACCT)

TASKSTATS gives the kernel the ability to export process statistics through a netlink socket. Netlink sockets are a form of IPC between the kernel and user space processes. TASK_DELAY_ACCT watches the processes and the delays concerning the access of resources. For example, TASK_DELAY_ACCT would see that process-X is waiting for some CPU time. The process is then given some CPU time if TASK_DELAY_ACCT notices that the process waits too long.

TASK_XACCT collects extra accounting data. I will disable this for less kernel overhead.

Now, the next category is displayed – RCU Subsystem. The Read-Copy-Update subsystem is a low-overhead syncing mechanism that allows programs to view files that are in the process of being modified/updated. The configuration tool answered the first question.

RCU Implementation

> 1. Tree-based hierarchical RCU (TREE_RCU)
choice[1]: 1

This is to select the type of RCU. Besides TREE_RCU, there is classic RCU (the older implementation). The next question (Consider userspace as in RCU extended quiescent state (RCU_USER_QS) [N/y/?]) asks if RCU can be put in a special state when the CPU runs in userspace. This feature is usually disabled because this adds too much overhead. After this question is another RCU question (Tree-based hierarchical RCU fanout value (RCU_FANOUT) [64]) asking for a fanout value. The next question (Tree-based hierarchical RCU leaf-level fanout value (RCU_FANOUT_LEAF) [16]) is another fanout value question except this one deals with the leaf-level. Yet again, another RCU question (Disable tree-based hierarchical RCU auto-balancing (RCU_FANOUT_EXACT) [N/y/?]) asking if the RCU auto-balancing can be disabled.

Next, the configuration script asks this question “Accelerate last non-dyntick-idle CPU's grace periods (RCU_FAST_NO_HZ)”. After that, “Offload RCU callback processing from boot-selected CPUs (RCU_NOCB_CPU)” is displayed.

This next question is very important (Kernel .config support (IKCONFIG)). The developer has the choice of saving the settings made in this configuration tool into a file. This file can be placed in the kernel, in a module, or not saved at all. This file can be used by other developers that want to compile a kernel exactly as someone else. This file can also help developers recompile a kernel using a newer compiler. For illustration, a developer configures and compiles a kernel. The compiler has some bugs, but the developer still needs a kernel with these settings. Thankfully, the developer can upgrade their compiler and use the settings file to save them time from configuring the kernel again. The developer could also save the source code and config file and compile the kernel on another computer. As for another purpose, the developer can load this file and tweak the settings as needed. I chose to save the config file in a module. “Enable access to .config through /proc/config.gz (IKCONFIG_PROC)” asks if this file is accessible. I chose yes.

The next question asks how large to make the kernel log buffers (Kernel log buffer size (16 => 64KB, 17 => 128KB) (LOG_BUF_SHIFT) [17]). Smaller buffers indicate logs are not kept as long as logs set to higher buffers. This choice depends on how long the developer wishes the logs to last. I chose “12”.

Again, another question appears. This one asks about enabling NUMA (Non-Uniform Memory Access) aware memory/task placement (Automatically enable NUMA aware memory/task placement

(NUMA_BALANCING_DEFAULT_ENABLED)). If set and if the computer is a NUMA machine, then NUMA balancing will be enabled. Under NUMA, a processor can access its own local memory faster than non-local memory (memory local to another processor or memory shared between processors). If the above is enabled (I enabled it), then it would be best to answer yes to “Memory placement aware NUMA scheduler (NUMA_BALANCING)”. This is the NUMA scheduler.

Under the new heading “Control Group support”, the choice “Control Group support (CGROUPS)” is automatically answered “yes” due to previous choices.

The following setting (Example debug cgroup subsystem (CGROUP_DEBUG)) is for enabling a simple cgroup subsystem for debugging the cgroups framework. The next choice (Freezer cgroup subsystem (CGROUP_FREEZER)) allows programmers to allow the ability to freeze and unfreeze tasks in a cgroup.

NOTE: A cgroup is a group of processes.

Next, we are asked “Device controller for cgroups (CGROUP_DEVICE)”. Cgroups (Control Groups) is a feature used to control resource usage. Answering “yes” will allow cgroup whitelists for the devices cgroups can open or mknod (system call for creating a file system node).

The next question (Cpuset support (CPUSETS)) asks if it can allow the creation and management of CPUSETS. This would allow administrators to dynamically partition sets of Memory Nodes and CPUs on a system and assign tasks to run on those sets. This is usually used on SMP and NUMA systems. I answered “no” on this question.

NOTE: Remember, if I do not specify what I chose, then I chose the default.

Enabling a cgroup accounting subsystem (Simple CPU accounting cgroup subsystem (CGROUP_CPUACCT)) makes a resource controller for monitoring the CPU usage of individual tasks in a cgroup. I chose “no”.

Resource counters (Resource counters (RESOURCE_COUNTERS)) enables controller independent resource accounting infrastructure that works with cgroups. I chose “no”.

The next question (Enable perf_event per-cpu per-container group (cgroup) monitoring (CGROUP_PERF)) allows developers to extend the per-cpu mode to make it only monitor threads of a particular cgroup on a specific CPU. I chose “no”.

The next section is “Group CPU Scheduler”. The first two pre-answered questions include

Group CPU scheduler (CGROUP_SCHED)

Group scheduling for SCHED_OTHER (FAIR_GROUP_SCHED)

The first answerable question (CPU bandwidth provisioning for FAIR_GROUP_SCHED (CFS_BANDWIDTH)) asks if the kernel should allow users to set CPU bandwidth limits for tasks executing in the fair group scheduler. Groups with no limit set are considered to be unconstrained and will run with no restriction.

NOTE: Not all of the kernel options are in groups. I mention groups only for the ease of reading and to indicate a new, large topic. It is not important to know the groups. This grouping system is helpful when configuring the kernel using the graphical tools. Then, developers can look through the menus that are grouped when searching for a particular setting.

Developers can enable users to allocate CPU bandwidth to task groups by answering yes to “Group scheduling for SCHED_RR/FIFO (RT_GROUP_SCHED)”.

The next question is “Block IO controller (BLK_CGROUP)”. Task groups are recognized and their disk bandwidth is allocated by the CFQ IO scheduler which uses the block IO controller to do so. The BIO throttling logic in the block layer uses a block IO controller to provide upper limit in IO rates on a device.

Here is a debugging question (Enable Block IO controller debugging (DEBUG_BLK_CGROUP) [N/y/?]) that is asking to enable debugging for block IO controllers. To make a streamlined kernel, it is best to disable this feature.

To enable checkpoint and restore features in the kernel answer yes to “Checkpoint/restore support (CHECKPOINT_RESTORE)”. I chose no for less overhead. Enabling this feature would add auxiliary prctl codes to setup process text, data and heap segment sizes, and a few additional proc entries.

Next, we will configure namespace support. A namespace is a container for a group of identifiers. For illustration, /usr/lib/python3/dist-packages/re.py is an identifier, /usr/lib/python3/dist-packages/ is the namespace, and re.py is the localname in the namespace.

The first namespace question (Namespaces support (NAMESPACES)) asks if namespaces can be enabled. This will allow the same PIDs (Process ID) to be used but in different namespaces. Otherwise, PIDs can never be duplicated.

The next question (UTS namespace (UTS_NS)) asks to enable the ability for tasks in the UTS namespace to see different information in the uname() system call. The uname() system call provides information about the machine and operating system.

Enabling the IPC namespace (IPC namespace (IPC_NS)) will allow tasks in this namespace to work with IPC IDs which correspond to different IPC objects in different namespaces.

PID Namespaces (PID Namespaces (PID_NS)) are process ID namespaces. This allows multiple processes, each in different PID namespaces, to use the same PID. This is a building block of containers.

Next, enabling network namespaces (Network namespace (NET_NS)) will allow users to make a network stack appear to have multiple instances.

When enabled, Automatic process group scheduling (SCHED_AUTOGROUP) populates and creates task groups to optimize the scheduler for desktop workloads. This will put applications that take up a lot of resources in their own task group. This helps performance.

Here is a debugging feature that should be disabled unless it is specifically needed. This question (Enable deprecated sysfs features to support old userspace tools (SYSFS_DEPRECATED)) asks if

More Linux Related Stuff on: <http://www.linux.org>

sysfs should be enabled. This is a virtual filesystem for debugging the kernel.

Next, the question “Kernel->user space relay support (formerly relayfs) (RELAY)” is answered “yes” because it is needed with the current configuration. It is best to allow initrd support (Initial RAM filesystem and RAM disk (initramfs/initrd) support (BLK_DEV_INITRD)).

The user is asked where to put the initramfs source files. If none are needed, then leave this blank.

Next, the developer is asked about the supported compression format for the initial ramdisks (Linux image files for the kernel). It is fine to enable support for all of the compression formats.

Support initial ramdisks compressed using gzip (RD_GZIP)

Support initial ramdisks compressed using bzip2 (RD_BZIP2)

Support initial ramdisks compressed using LZMA (RD_LZMA)

Support initial ramdisks compressed using XZ (RD_XZ)

Support initial ramdisks compressed using LZO (RD_LZO)

Here, the compiling options for the kernel are set (Optimize for size (CC_OPTIMIZE_FOR_SIZE)). The developer can have the compiler optimize the code when compiling. I chose “yes”.

For users wanting to configure more kernel features, then they can answer “yes” for this next question (Configure standard kernel features (expert users) (EXPERT)).

To enable legacy 16-bit UID syscall wrappers, answer yes to this question (Enable 16-bit UID system calls (UID16)). These system calls use 16-bit user IDs.

It is recommended to enable sysctl syscall support (Sysctl syscall support (SYSCTL_SYSCALL)). This allows /proc/sys to be the interface for binary paths.

The next two questions about debugging have been pre-answered “yes” - “Load all symbols for debugging/ksymoops (KALLSYMS)” and “Include all symbols in kallsyms (KALLSYMS_ALL)”. These enable debugging symbols.

Next, developers should enable printk support (Enable support for printk (PRINTK)). This prints kernel messages to the kernel log. This is important if something goes wrong with the kernel. Having a mute kernel is not a good idea. However, some developer saw a purpose for it if we have a choice. Otherwise, we would not have a choice.

Unless needed, developers can disable bug support (BUG() support (BUG)). Disabling this will remove support for WARN and BUG messages. This reduces the kernel's size.

The next question (Enable ELF core dumps (ELF_CORE)) asks about enabling the ability for the kernel to generate core dumps. This feature makes the kernel four kilobytes larger. I chose “no”.

NOTE: A core dump (memory or system dump) is the recorded state of an application before it crashed. Core dumps are used for debugging issues. This core dump file is in the Executable and Linkable Format (ELF) format.

Configuring the Kernel Part III

Here, we are still configuring the kernel. There are many more features to configure.

The next question (Enable ELF core dumps (ELF_CORE)) asks about enabling the ability for the kernel to generate core dumps. This feature makes the kernel four kilobytes larger. I chose “no”.

NOTE: A core dump (memory or system dump) is the recorded state of an application before it crashed. Core dumps are used for debugging issues. This core dump file is in the Executable and Linkable Format (ELF) format.

Next, PC-Speakers can be enabled (Enable PC-Speaker support (PCSPKR_PLATFORM)). Most computers users have and use speakers, so this is enabled.

Although this next feature increases the kernel size (Enable full-sized data structures for core (BASE_FULL)), performance is increased. I chose “yes”.

For the kernel to run glibc-based programs, FUTEX must be enabled (Enable futex support (FUTEX)). This feature enables Fast Userspace muTEXes.

NOTE: glibc (GNU C Library) is the GNU's implementation of the standard C library.

NOTE: FUTEX (fast userspace mutex) is used for preventing two threads from accessing a shared resource that should not be used by more than one thread at once.

The epoll system calls can be disabled by answering “no” to this next question (Enable eventpoll support (EPOLL)). However, it helps to have epoll system calls, so I chose “yes”. Epoll is an I/O event notification system.

To receive signals on file descriptors, enable signalfd system calls (Enable signalfd() system call (SIGNALFD)).

This feature allows applications to get file descriptors to use with timer events if enabled (Enable timerfd() system call (TIMERFD)).

The eventfd system call must be enabled with our current configuration (Enable eventfd() system call (EVENTFD)). The ability to use a shmem filesystem is enabled by default (Use full shmem filesystem (SHMEM)). A shmem filesystem is a virtual RAM filesystem.

The next question that can be answered is “Enable AIO support (AIO)”. This feature enables POSIX asynchronous I/O that threaded application use. This features takes up seven kilobytes of space. I disabled this feature.

NOTE: Asynchronous I/O is input/output processing that allows other threads to get processed before transmission is complete.

If embedding a kernel for embedded systems, select “yes” for the question “Embedded system (EMBEDDED)”. Otherwise, choose no as I have done.

More Linux Related Stuff on: <http://www.linux.org>

NOTE: Embedded systems are real-time computers that run in a larger electronic system.

Now, we can configure kernel performance events and counters. The configuration tool enables events and counters without giving the developer a choice (Kernel performance events and counters (PERF_EVENTS)). This is an important feature.

Next, we can disable another debugging feature (Debug: use vmalloc to back perf mmap() buffers (DEBUG_PERF_USE_VMALLOC)).

If VM event counters are enabled, then event counts will be shown in the /proc/vmstat (Enable VM event counters for /proc/vmstat (VM_EVENT_COUNTERS)). If disabled, event counts will not be shown and /proc/vmstat will only display page counts.

For better support for PCI chipsets, answer yes (Enable PCI quirk workarounds (PCI_QUIRKS)). This will enable workarounds for PCI quirks and bugs.

Next is another debugging feature that can be disabled as I did (Enable SLUB debugging support (SLUB_DEBUG)). This feature takes up a lot of space and disables SLB sysfs which is used for debugging the kernel. If this feature is disabled, then /sys/slab will not exist and cache validation support will not exist on the system.

Heap randomization is a feature that makes heap exploits more difficult (Disable heap randomization (COMPAT_BRK)). However, this should not be enabled because any libc5-based software will not work on the system. Only enable this feature if you have a specific reason for doing so or if you will not use libc5-based software. I disabled this feature. When making a general kernel, developers will want to disable this feature.

Next, a SLAB allocator must be chosen. A SLAB allocator is a memory management system for placing kernel objects in memory in an efficient way without fragmentation. The default is choice “2”.

Choose SLAB allocator

1. SLAB (SLAB)
 - > 2. SLUB (Unqueued Allocator) (SLUB)
 3. SLOB (Simple Allocator) (SLOB)
- choice[1-3?]: 2

To enable extended profiling support, answer “yes” (Profiling support (PROFILING)).

The next question gives developers the choice of enabling the OProfile system. It can be disabled, enabled, or added as a module to be loaded when needed. I chose to disable this feature.

Kprobes allows users to trap nearly any kernel address to start a callback function. This is a debugging tool that can be disabled as I did (Kprobes (KPROBES)).

This optimization feature should be enabled (Optimize very unlikely/likely branches (JUMP_LABEL)). This makes branch prediction easier and reduces overhead.

The configuration tool enabled an experimental feature (Transparent user-space probes (EXPERIMENTAL) (UPROBES)). Do not worry, the system will be fine. Not all experimental features are unstable or bad.

Next, we are asked about gcov-based kernel profiling (Enable gcov-based kernel profiling (GCOV_KERNEL)). This can be disabled.

To allow the kernel to load modules, enable loadable module support (Enable loadable module support (MODULES)).

The Linux kernel will only load modules with version numbers. To allow the kernel to load modules with missing version numbers, enable this feature (Forced module loading (MODULE_FORCE_LOAD)). It is generally a bad idea to do this, so disable this feature as I have done, unless you have a specific need to such a feature.

The Linux kernel can also unload modules if that feature is enabled which is best to do (Module unloading (MODULE_UNLOAD)). If the kernel feels that unloading a module is a bad idea, then the user cannot unload the module. Enabling force-unload is possible, but is a bad idea (Forced module unloading (MODULE_FORCE_UNLOAD)).

To use modules that did not come with your kernel or are not meant for your kernel version, enable versioning support (Module versioning support (MODVERSIONS)). It is best not to mix versions, so I will disable this feature.

Modules can have a field in their modinfo (Module Information) section titled “srcversion”. This field allows developers to see what source was used to make the module. Enabling this option will add this field when the modules are compiled. This is not necessary, so I will disable it (Source checksum for all modules (MODULE_SRCVERSION_ALL)). If the previous option was enabled, developers could have the checksums added to the modules (Source checksum for all modules (MODULE_SRCVERSION_ALL)).

To enable module signature verification (Module signature verification (MODULE_SIG)), answer “yes” for this option. Because it is not needed, I will answer “no”. Otherwise, the kernel will check and verify the signature before loading a module.

To enable block layer support (Enable the block layer (BLOCK)), choose “yes” as I have done. Disabling this will make block devices unusable and certain file systems will not be enabled

Next, SG support is enabled by default (Block layer SG support v4 (BLK_DEV_BSG)), and the helper library is also enabled (Block layer SG support v4 helper lib (BLK_DEV_BSGLIB)).

The next answerable question is about data integration support for block devices (Block layer data integrity support (BLK_DEV_INTEGRITY)). This allows better data integrity to help protect data on devices that support such a feature. Many devices do not support this feature, so I will disable it.

IO device rates can be limited if block layer bio throttling is enabled (Block layer bio throttling support (BLK_DEV_THROTTLING)).

To enable support for foreign partitioning schemes, answer “yes” to the next question (Advanced

More Linux Related Stuff on: <http://www.linux.org>

partition selection (PARTITION_ADVANCED)). I will disable this feature.

To enable the CSCAN service and FIFO expiration of requests, enable the deadline IO scheduler (Deadline I/O scheduler (IOSCHED_DEADLINE)).

The CFQ IO scheduler distributes bandwidth evenly between the processes. It is a good idea to enable this feature (CFQ I/O scheduler (IOSCHED_CFQ)).

Next, developers can enable or disable CFQ group support (CFQ Group Scheduling support (CFQ_GROUP_IOSCHED)). Then, developers can choose the default IO scheduler. It is best to pick DEFAULT_DEADLINE.

For devices with less than 32-bit addressing, this next feature allocated the first 16 megabytes of address space (DMA memory allocation support (ZONE_DMA)). If the kernel is not meant for such devices, this can be disabled, so I disabled this feature.

For systems with more than one CPU, it is best to enable SMP (Symmetric multi-processing support (SMP)). For single processor devices, the kernel will execute faster with this feature disabled. I enabled this feature.

For CPUs that offer x2apic, enable x2apic support (Support x2apic (X86_X2APIC)). If your system lacks this feature, then disable it as I have done.

Next, we can enable a MPS table which is for old SMP systems that lack appropriate ACPI support (Enable MPS table (X86_MPPARSE)). Newer systems that have ACPI support, DSDT, and MADT do not need this feature. I disabled the feature.

The following question allows us to enable support for extended x86 platforms (Support for extended (non-PC) x86 platforms (X86_EXTENDED_PLATFORM)). Only enable this if you need a general kernel or a kernel that will run on certain processors that need extended support. I disabled extended support.

To support an Intel Low Power Subsystem, enable this feature (Intel Low Power Subsystem Support (X86_INTEL_LPSS)).

Single-depth WCHAN output (Single-depth WCHAN output (SCHED OMIT FRAME POINTER)) is used to calculate batter /proc/<PID>/wchan values. However, this will cause more overhead.

Next, we can enable virtual guest system support (Paravirtualized guest support (PARAVIRT_GUEST)). This will allow a guest operating system to run with the main OS. I will disable this feature.

Memtest is software that checks the RAM when the system starts. Memtest can be configured to run every time the system starts or sometimes. Memtest is not required, so I will disable it.

Here, we can select the processor family that the kernel should support. I will pick 5 – Generic-x86-64. This is a 64-bit system, a x86 is a 32-bit system,

Next, we can choose to support x86 processors (32-bit) (Supported processor vendors

More Linux Related Stuff on: <http://www.linux.org>

(PROCESSOR_SELECT)). Disable this if you are making a kernel that you plan to use only on 64-bit devices.

To find the machine's quirks, we can enable DMI scanning (Enable DMI scanning (DMI)). This will detect quirks.

To enable DMA access of 32bit memory devices with systems with more than 3GB of RAM, answer “yes” to this next question (GART IOMMU support (GART_IOMMU)).

Configuring the Kernel Part IV

Here, in part four of “The Linux kernel: Configuring the Kernel”, we are continuing with more settings and features to configure.

Here, we are asked about “IBM Calgary IOMMU support (CALGARY_IOMMU)”. This option will enable support for IOMMUs that belong to the xSeries x366 and x460 by IBM. This will also allow 32-bit PCI that do not support DAC (Double Address Cycle) devices of such systems to run properly because this system setup will have issues accessing more than 3GB of RAM. If needed, these IOMMU devices can be turned off at boot time using the “iommu=off” parameter. (These kernel/module parameters will be discussed in a later article.)

An IOMMU (input/output memory management unit) is a memory management unit (MMU) that connects a DMA-capable I/O bus to the main memory. DMA (Direct Memory Access) is a feature of many computers that allows certain devices to access the memory without help from the CPU. Double Address Cycle (DAC) is 64-bit DMA; regular DMA uses 32-bits.

Next, we are asked about enabling Calgary by default (Should Calgary be enabled by default? (CALGARY_IOMMU_ENABLED_BY_DEFAULT)). Calgary is the same concept as the IOMMU support mentioned above. The difference between the two is this is support for such a feature on many devices while the one above is specifically for IOMMU IBM devices. If this is disabled and you need to use it later, use this kernel parameter (iommu=calgary).

Here is a question that should be handled carefully (Enable Maximum number of SMP Processors and NUMA Nodes (MAXSMP)). Only enable this if the system the kernel will run on has many SMP processors and NUMA nodes like the Core i7 and many AMD CPU chips. If such a system lacks SMP processors and NUMA nodes or has a very little amount, the kernel can be inefficient. It is best to select “No”.

Non-Uniform Memory Access (NUMA) is a system of memory where each part of the memory takes longer to access than others. A node is a set of memory. For instance, a NUMA system may have three RAM chips. Each chip is one node. There is a node/chip on the motherboard with the CPU (this is the fastest node). The other two nodes are on a different bus. These two nodes take more time to access than the first node.

NOTE: ccNUMA and NUMA are now the same or at least very similar.

Symmetric Multi-Processing (SMP) is an alternative to NUMA. The memory is on the same bus. Only so many CPUs can access a bus, so this limits the number of processors on a SMP system. However, the memory is equally fast.

REMEMBER: I am compiling a kernel for an AMD64 system, so I will tell you what choice I made to help readers to understand the process and choices. If I do not specify what I choose, then I choose the default choice. When you are compiling for a different system or you have different needs, you will need to make alternate decisions based on your situation.

Next, choose the maximum number of CPUs the kernel will support unless the configuration tool chose for you. This configuration can optimize the kernel for the given amount.

Then, enable or disable “SMT (Hyperthreading) scheduler support (SCHED_SMT)”. The SMT scheduler improves the CPU's decision-making on Pentium 4 processors that use HyperThreading. However, there will be extra overhead. For some systems, it is best to choose “no” as I have done.

HyperThreading is a proprietary SMT parallelization for microprocessors (Intel invented the implementation). This is a special form of multitasking/multithreading (doing many tasks at once). Simultaneous multithreading (SMT) improves multithreading efficiency.

After that, enable or disable “Multi-core scheduler support (SCHED_MC)”. This feature also improves the CPU's decision making on multi-core chips. However, the cost is extra overhead. I chose “No”.

Here, in this next option, the preemption model can be chosen.

Preemption Model

1. No Forced Preemption (Server) (PREEMPT_NONE)
 - > 2. Voluntary Kernel Preemption (Desktop) (PREEMPT_VOLUNTARY)
 3. Preemptible Kernel (Low-Latency Desktop) (PREEMPT)
- choice[1-3]: 2

Preemption is the process of pausing an interrupting task with the intent of allowing it to continue executing later. Preemption forces the task to pause. The task cannot ignore preemption.

Next, we are asked about “Reroute for broken boot IRQs

(X86_REROUTE_FOR_BROKEN_BOOT_IRQS)”. This is simply a fix for spurious interrupts. A spurious interrupt is an unwanted hardware interrupt. These are usually triggered by electrical interference or improperly connected electronics. Remember, an interrupt is a signal to the processor that needs immediate attention.

This option is vital for every machine; I doubt anyone would have a reason for disabling this feature (Machine Check / overheating reporting (X86_MCE)). The kernel must be aware of overheating and data corruption, otherwise, the system will continue to operate only to receive further damage.

Next, users can enable/disable “Intel MCE features (X86_MCE_INTEL)”. This is extra support for Intel MCE features like a thermal monitor. I chose “no” since I am compiling for an AMD64 processor. Machine Check Exception (MCE) is a type of error produced when the processor finds a hardware issue. An MCE will usually cause a kernel panic (equivalent to the “Blue-Screen-of-Death” on Windows).

Here is the same question again except for AMD devices (AMD MCE features (X86_MCE_AMD)).

Next is a debugging feature that I will disable (Machine check injector support (X86_MCE_INJECT)). This allows injecting machine checks. If you do perform machine injections occasionally, it may be better to enable this as a module rather than build it into the kernel. Machine injection makes a device send an error message even though no real error exists. This is used to make sure the kernel and other processes act correctly towards errors. For instance, if the CPU over heats, then it should shutdown, but how would a developer test such code without harming the CPU. Injecting errors is the best way since it is just software that tells hardware to send an error signal.

NOTE: Modules are for features/drivers that may be used or are very rarely executed. Only add

More Linux Related Stuff on: <http://www.linux.org>

features/drivers to the kernel itself if it will be used by many systems that will use the kernel being built.

If the kernel will likely be used on Dell laptops, then enable this feature (Dell laptop support (I8K)). Otherwise, add it as a module if some users of this kernel may use it on a Dell laptop. If this kernel is not planned for Dell laptops, then disable this support as I have done. Specifically, this support is a driver that allows the System Management Mode of the processor to be accessed on the Dell Inspiron 8000. The system management mode's purpose is to get the processor's temperature and fan status which are needed information for controlling the fans on such systems.

Next, users can allow microcode loading support (CPU microcode loading support (MICROCODE)). This will allow users to update the microcode on AMD and Intel CPU chips that support such a feature.

NOTE: To load microcode, you must have a legal copy of the microcode binary-file that is designed for your processor.

For loading microcode patches (to fix bugs or add minor features) to Intel chips (Intel microcode loading support (MICROCODE_INTEL)), this must be enabled. I disabled this feature.

This is the same as above except for AMD chips (AMD microcode loading support (MICROCODE_AMD)).

Enabling this support (/dev/cpu/*/msr - Model-specific register support (X86_MSR)) will allow certain processes to have permission to use x86 Model-Specific Registers (MSRs). These registers are a character devices that include major 202 and minors 0 to 31 (/dev/cpu/0/msr to /dev/cpu/31/msr). This feature is used on multiprocessor systems. Each virtual character device is linked to a specific CPU.

NOTE: MSRs are used for changing CPU settings, debugging, performance monitoring, and execution tracing. MSRs use the x86 instruction set.

After that, we have an option for “CPU information support (X86_CPUID)”. Enabling this feature allows processes to access the x86 CPUID instructions needed to execute on a particular CPU through character devices. These character devices that include major 202 and minors 0 to 31 (/dev/cpu/0/msr to /dev/cpu/31/msr), just like x86_MSR support above.

For processors that support it, enable the kernel linear mapping to use 1GB pages (Enable 1GB pages for kernel pagetables (DIRECT_GBPAGES)). Enabling this feature helps performance by reducing TLB pressure.

A page is the basic unit of memory itself (bits are the basic units of data). Page size is determined by the hardware itself. A page table is the mapping between virtual and physical memory. Physical memory is the memory on devices. The virtual memory is the addresses to the memory. Depending on the architecture of a system, there may be more addresses the hardware has the ability to access than what there is to access. For instance, on a 64-bit system with a 6GB RAM chip, an administrator can add more RAM if needed. This is because there are still more virtual addresses. However, on many 32-bit systems, an administrator can add an 8GB RAM chip, but the system will not use all of it because there are not enough virtual addresses for the system to use to access the large amount of RAM. Translation Lookaside Buffer (TLB) is a cache system that improves virtual address translation speed.

More Linux Related Stuff on: <http://www.linux.org>

Reducing pressure on it keeps it from being so busy.

Next, we have a NUMA option (Numa Memory Allocation and Scheduler Support (NUMA)). This will allow the kernel to allocate memory used by the CPU on the local memory controller of the CPU. This support also makes the kernel more NUMA aware. Very few 32-bit systems need this feature, but some common 64-bit processors use this feature. I chose “no”.

For the system to detect AMD NUMA node topology using an old method, enable this feature (Old style AMD Opteron NUMA detection (AMD_NUMA)). Next is an option for a newer detection method (ACPI NUMA detection (X86_64_ACPI_NUMA)). If both are enabled, the newer one will dominate. Some hardware works better using one of the methods instead of the other.

For NUMA emulation for the purpose of debugging, enable this next feature (NUMA emulation (NUMA_EMU)).

NOTE: If you do not plan to do debugging and you need a fast, light-weight system, disable as many debugging features as you can.

In this next option, choose the maximum number of NUMA nodes your kernel is planned to handle. Then, choose the memory model. There may be only one choice for a memory model. The memory model specifies how the memory is stored.

Maximum NUMA Nodes (as a power of 2) (NODES_SHIFT) [6]

Memory model

> 1. Sparse Memory (SPARSEMEM_MANUAL)

choice[1]: 1

To help performance, there is this option for optimizing pfn_to_page and page_to_pfn operations via a virtually mapped memmap (Sparse Memory virtual memmap (SPARSEMEM_VMEMMAP)). Page Frame Number (pfn) is a number given to each page. These two operations get a page from a number or a number from a page.

Next is an option that allows a node to move memory (Enable to assign a node which has only movable memory (MOVABLE_NODE)). Kernel pages cannot normally be moved. When enabled, users can hotplug memory nodes. Also, movable memory allows memory defragmentation. As data goes in and out of memory, a set of data may get divided across the memory where ever there is space available.

Following the previous memory question, we have more. These may be preconfigured by the configuration tool. The third option (BALLOON_COMPACTION), when enabled, helps to lessen memory fragmentation. Fragmented memory can slow down the system. The fourth option (COMPACTATION) allows memory to be compressed. The fifth option listed below (MIGRATION) allows pages to be moved.

Allow for memory hot-add (MEMORY_HOTPLUG)

Allow for memory hot remove (MEMORY_HOTREMOVE)

Allow for balloon memory compaction/migration (BALLOON_COMPACTION)

Allow for memory compaction ()

Page migration (MIGRATION)

NOTE: Enabling movable memory will enable the five features listed above.

Next, we can “Enable KSM for page merging (KSM)”. Kernel Samepage Merging (KSM) views memory that an application says can be merged. This saves memory because if two pages are identical, one can be deleted or merged and only one will be used.

The configuration tool may automatically choose how much memory to save for user allocation (Low address space to protect from user allocation (DEFAULT_MMAP_MIN_ADDR) [65536]).

This next option is important (Enable recovery from hardware memory errors (MEMORY_FAILURE)). If the memory fails and the system has MCA recovery and ECC memory, the system can continue to run and try to recover. To have such a feature, the hardware itself must be able to support it as well as the kernel.

Machine Check Architecture (MCA) is a feature of some CPUs where they can send hardware error messages to the operating system. Error-correcting code memory (ECC memory) is a form of a memory device with error detection and correction.

Next, the configuration tool automatically enables “HWPoison pages injector (HWPOISON_INJECT)”. This feature allows the kernel to mark a bad page as “poisoned”, and then the kernel kills the application that created the bad page. This helps to stop and correct errors.

To allow the kernel to use large pages (Transparent Hugepage Support (TRANSPARENT_HUGEPAGE)), enable this feature. This speeds up the system, but more memory is needed. Embedded system should not use this feature. Embedded systems generally have very small amounts of memory.

If the above is enabled, then the sysfs support for huge pages must be configured.

Transparent Hugepage Support sysfs defaults

1. always (TRANSPARENT_HUGE_PAGE_ALWAYS)
 - > 2. madvise (TRANSPARENT_HUGE_PAGE_MADVISE)
- choice[1-2?]: 2

This next option is for adding system calls process_vm_ready and process_vm_writev (Cross Memory Support (CROSS_MEMORY_ATTACH)). This allows privileged processes to access the address space of another application.

If tmem is present, enabling cleancache will generally be a good idea. (Enable cleancache driver to cache clean pages if Transcendent Memory (tmem) is present (CLEANCACHE)). When needed pages are removed from the memory, cleancache will place the page on cleancache-enabled filesystems. When the page is needed, it is placed back on memory. Transcendent Memory (tmem) is memory without a set known size. The kernel indirectly addresses such memory.

This next option allows caching swap pages if tmem is active (Enable frontswap to cache swap pages if tmem is present (FRONTSWAP)). Frontswap places data on swap partitions. This is needed for swap support.

It is best to enable this next feature (Check for low memory corruption (X86_CHECK BIOS CORRUPTION)). This will check the low memory for memory corruption. This feature is disabled at runtime. To use this feature, add “memory_corruption_check=1” to the kernel command-line (this will be discussed in a later article; this is not the same as any command-line). This feature, even when actively being executed, uses very little overhead (nearly none).

Next, we can “Set the default setting of memory_corruption_check (X86_BOOTPARAM_MEMORY_CORRUPTION_CHECK)”. This will set whether memory_corruption_check is on or off. It is best to have the memory checked otherwise data can be lost and the system can crash if an important part of the memory is corrupted.

This option concerns the BIOS (Amount of low memory, in kilobytes, to reserve for the BIOS (X86_RESERVE_LOW) [64]). The configuration tool usually knows the best amount of memory to reserve for the BIOS.

For Intel P6 processors, developers can enable memory type range registers (MTRR (Memory Type Range Register) support (MTRR)). This is used for AGP and PCI cards with a VGA card attached. Enabling this feature creates /proc/mtrr.

If X drivers need to add writeback entries, then enable this following option (MTRR cleanup support (MTRR_SANITIZER)). This will convert MTRR layout from continuous to discrete. Memory type range registers (MTRRs) provide software a way to access CPU cache.

Next, some MTRR options are set by the configuration tool.

MTRR cleanup enable value (0-1) (MTRR_SANITIZER_ENABLE_DEFAULT) [1]

MTRR cleanup spare reg num (0-7) (MTRR_SANITIZER_SPARE_REG_NR_DEFAULT) [1]

To set up page-level cache control, enable PAT attributes (x86 PAT support (X86_PAT)). Page Attribute Table (PATs) are the modern equivalents of MTRRs and are much more flexible than MTRRs. If you experience bootup issues with this enabled, then remake the kernel with this feature disabled. I chose “no”.

Configuring the kernel Part V

The Linux kernel is large with numerous features that can be configured. There are still many more features that can be configured.

The next kernel feature that can be configured is a x86 random number generator (x86 architectural random number generator (ARCH_RANDOM)). Remember, we are configuring the kernel source code for an AMD64 system. This number generator uses the x86 RDRAND instructions for Intel processors. It is not necessary for general use, so I will disable this for a more lightweight kernel.

Next, we can enable or disable "Supervisor Mode Access Prevention (X86_SMAP)". This is a security feature used by some Intel processors. SMAP will only allow the kernel to access user-space in some instances. This helps to protect user-space. There is a performance and size cost if enabled, but the cost is small. Since I am configuring for an AMD system, I will disable this feature.

Developers can enable "EFI runtime service support (EFI)". Only enable this on systems with EFI firmware. With this feature, the kernel can use available EFI services. EFI is a specification of how the operating system interacts with the hardware, so EFI firmware is hardware code that uses this specification. I disabled the support since I do not have a system with EFI firmware.

This is a useful security method that should be enabled (Enable seccomp to safely compute untrusted bytecode (SECCOMP)). This security feature is used with number crunching applications (software that performs extensive calculations) that use untrusted bytecode. Bytecode (p-code/portable code) is code that is made to be read efficiently by an interpreter. Bytecode is not source code, but it is not assembly or binary code either. Untrusted code is code that may cause system/data damage. The untrusted bytecode that may ruin the system or harm data are isolated in a separate address space via seccomp. This is done by using file descriptors as methods of transport. In general, it is best to enable security features even at the cost of performance unless you are making a kernel that absolutely needs incredible performance abilities.

Here is another security feature (Enable -fstack-protector buffer overflow detection (CC_STACKPROTECTOR)). A buffer overflow (buffer overrun) is where data is written past its memory boundary and into adjacent memory. This can be a security threat. Some malware uses buffer overruns to exploit systems. Enabling this will use the "-fstack-protector" GCC parameter. GCC is a Linux compiler; this compiler will compile the kernel when you are done configuring the options. This compiler parameter will add a canary value (special security code) on the stack just before the return address. The value is validated before the return. When a buffer overflow occurs, the canary value will get overwritten. When this happens, a kernel panic is initiated. As many know, a kernel panic means the system will crash, but that is better than the system or data being permanently ruined. With a kernel panic, the system can be rebooted, but if a buffer overrun gains a chance to damage the system, a simple reboot will not fix the destruction. You must compile the kernel with GCC version 4.2 or higher to support the parameter.

NOTE: To figure out what version you have, type “gcc --version” in the command-line.

After that, we can configure the timer frequency. The configuration tool recommends 250Hz, so we will use that value.

Timer frequency

More Linux Related Stuff on: <http://www.linux.org>

1. 100 HZ (HZ_100)
> 2. 250 HZ (HZ_250)
3. 300 HZ (HZ_300)
4. 1000 HZ (HZ_1000)
choice[1-4?]: 2

Using 1000HZ is generally considered too fast for many systems. The timer frequency determines how often the timer interrupt is used. This helps the system operate on a timeline. Applications do not just execute a command randomly. Rather, they wait until a timer interrupt has gone off. This keeps process organized and structured. The time between interrupts on a timer frequency of 100HZ is 10ms, 250HZ is 4ms, and 1000HZ is 1ms. Now, many developers will instantly think that 1000HZ is the best. Well, it depends what effects you will be fine with. A large timer frequency means more power consumption and with more energy being utilized, more heat will be produced. More heat means the hardware may wear down faster.

NOTE: If a particular feature does not matter to you specifically or you are not sure what to choose, use the default value chosen by the configuration tool. For example, for the kernel that I am making, it does not matter to me which timer value to use. In summary, if you do not have a specific reason to select any of the choices, the default is fine.

This interesting system call may be useful to some users (kexec system call (KEXEC)). The kexec call shuts down the current kernel to start another or restart the current. The hardware is not powered-off and this call works without help from the firmware. The bootloader is not executed. (The bootloader is the software that starts the operating system) This restart takes place at the level of the operating system not the hardware. Using this system call is faster than performing a standard power-off or restart. This keeps the hardware on. This system call will not work on all systems. For maximum performance, enable hotplugging.

To use kexec, use the command below replacing “<kernel-image>” with the kernel that will be used after reboot. Also, replace “<command-line-options>” with some of those kernel parameters we had discussed previously. (I will go into greater depth in a later article.)

Code:

```
kexec -l <kernel-image> --append=<command-line-options>"
```

Specifically, I would type “kexec -l /boot/vmlinuz-3.8.0-27-generic –append="root=/dev/sda1””

NOTE: The hardware does need to be reset sometimes, so do not depend on kexec entirely.

Next, we have a debugging feature that works with kexec (kernel crash dumps (CRASH_DUMP)). When kexec is called, a crash dump is generated. Unless you need to debug kexec, this is not needed. I disabled this feature.

Again, we have another kexec feature (kexec jump (KEXEC_JUMP)). kexec jump allows users to switch between the original kernel and the one started by kexec.

It is best to use the default value for the address the kernel starts (Physical address where the kernel is loaded (PHYSICAL_START) [0x1000000]).

This next kernel option (Build a relocatable kernel (RELOCATABLE)) allows the kernel to be placed

More Linux Related Stuff on: <http://www.linux.org>

somewhere else in the memory. The kernel file will be 10% larger, but this excess is removed from memory on execution. Many may wonder why this is important. Before kernel 2.6.20, rescue kernel had to be configured and compiled differently to be able to run on a different memory address. After this feature was invented, developers no longer needed to make two kernels. A rescue kernel will not load where the first kernel is/was loaded because that portion of memory is occupied or corrupted. (If you are using a rescue kernel then obviously the first kernel had errors)

This feature should be enabled on systems where CPUs can be added unless there is a specific reason for not doing so (Support for hot-pluggable CPUs (HOTPLUG_CPU)). The configuration tool may auto-enable this ability. With this feature, you can active/deactivate a CPU on a system that has many processors. This does not mean adding a new CPU to a system. All CPUs must already be in the system.

The next option will allow us to set whether the above ability is enabled by default (Set default setting of cpu0_hotpluggable (BOOTPARAM_HOTPLUG_CPU0)). It is better to have this feature inactive for performance purposes until it is needed.

This debugging feature allows developers to debug the CPU hotplug abilities (Debug CPU0 hotplug (DEBUG_HOTPLUG_CPU0)). I disabled this feature.

To support older versions of glibc (<2.3.3), enable this feature (Compat VDSO support (COMPAT_VDSO)). This will apply the old-style address via map on the 32-bit VDSO. Glibc is Gnu C Library; this is the GNU Project's implementation of the C standard library.

If the system the kernel is intended for lacks a fully functional boot-loader, then enable this feature (Built-in kernel command line (CMDLINE_BOOL)). This will allow users to use a command-line on the kernel itself so administrators can fix kernel issues. If the bootloader has a command-line (like Grub), then this feature is not needed.

Now, we can configure ACPI and power. First, we are given the choice to allow the system to suspend to RAM (Suspend to RAM and standby (SUSPEND)). Advanced Configuration and Power Interface (ACPI) is an open standard for device configuration and power management. Suspending a system places data on RAM and the hardware goes into a low-power state. The system is not shutdown entirely. This is useful if a user needs to put the computer in a low-power state but wants to retain the currently open applications. Shutting-down a system completely powers off a system and clears the memory.

Next, we can enable hibernation (Hibernation (aka 'suspend to disk') (HIBERNATION)). Hibernation is like suspend mode, but all data in the memory is saved to the hard-drive and the device is completely powered-off. This allows the user to continue using their open applications when the system is powered back on.

Here, we can set the default resume partition (Default resume partition (PM_STD_PARTITION)). Very few developers and administrators will need this feature. When a system returns from hibernation, it will load off of the default resume partition.

After that, we can enable "Opportunistic sleep (PM_AUTOSLEEP)". This lets the kernel to initiate suspend or sleep mode when no active wakeup calls are called. This means that an idling system will initiate suspend mode to save energy. I enabled this feature.

More Linux Related Stuff on: <http://www.linux.org>

Next, we are asked about "User space wakeup sources interface (PM_WAKELOCKS)". Enabling this will allow wakeup source objects to be activated, deactivated, and created by the user space via a sysfs-based interface. Wakeup source objects track the source of wakeup events.

Sysfs is a virtual filesystem located /sys/. This virtual filesystem contains information about devices. When going to /sys/, it appears to be part of the hard-drive, but this is really a mount point. The files are actually found in the memory. This is the same concept for /proc/.

NOTE: “/sysfs/” is a folder while “/sysfs” would be a file on the root named “sysfs”. Many Linux users mix up the two naming conventions.

If the above option is enabled, then you can set the "Maximum number of user space wakeup sources (0 = no limit) (PM_WAKELOCKS_LIMIT)". It may be best to select the default. Then, you can enable the garbage collector (Garbage collector for user space wakeup sources (PM_WAKELOCKS_GC)). Garbage collection is a memory management method.

NOTE: On systems that need more memory, it is usually best in most cases to enable as many garbage collectors as possible. Otherwise, the memory will fill up faster and be disorganized.

The next power option concerns IO devices (Run-time PM core functionality (PM_RUNTIME)). This option will permit IO hardware to go into low power states on run time. The hardware must allow this feature; not all hardware will do this.

As with many other components of the kernel, the power-management code also has debugging support, if enabled (Power Management Debug Support). I will disable this option.

NOTE: Notice that the options/questions from the configuration tool that I quote/display are no longer showing the option code (the letters in all caps between the parenthesis). This is because I am no using the ncurses-based configuration tool (make menuconfig) instead of the default tool to get the options/settings/questions. Remember, “make config” lacks the ability to save the current progress.

After that, the configuration tool will enable "ACPI (Advanced Configuration and Power Interface) Support". It is best to allow this power management specification. Usually, the configuration file will enable this feature.

To allow backwards compatibility, enable "Deprecated /proc/acpi files". The new implementation uses the newer functions in /sys/. I disabled this option. A similar question asks about "Deprecated power /proc/acpi directories". Usually, if you disable the files, you will not need the folders, so I disabled them. Some older applications may use these files and folders. If you are compiling a new kernel for an old Linux system, it may be best to enable this option.

Next, we have another file interface that can be enabled/disabled (EC read/write access through). This will create an embedded controller interface in /sys/kernel/debug/ec/. Embedded controllers usually are found in laptops to read the sensors. The Linux kernel accesses the embedded controllers through ACPI code given by the BIOS tables of the system.

Here is another old feature that can be enabled for backwards compatibility (Deprecated /proc/acpi/event support). The acpid daemon may read /proc/acpi/event to manage ACPI-generated

More Linux Related Stuff on: <http://www.linux.org>

events. Instead of this interface, the daemon uses netlink events or input layer to get these events to the user-space. The acpid daemon manages ACPI events.

The next option allows developers to enable a feature that will inform the kernel whether it is using AC or battery (AC adapter). The next option provides battery information from /proc/acpi/battery/ (Battery).

To allow the kernel to behave differently when the power/sleep button is pressed or when the lid is closed, enable this option (Button). These events are controlled in /proc/acpi/event/. For instance, this will make the system suspend when the laptop lid is closed if in the user account power options such a behavior is enabled. On many Linux distros, users can go to the system settings to make the laptop stop suspending when the lid is closed.

The next ACPI extension to be enable/disabled is for video adapters (Video).

ACPI fans can be enabled/disabled (Fan). It is best to enable ACPI fans management. This will help to conserve energy.

We are getting further in configuring the kernel, but there is still more to do in the next articles.

Configuring the kernel Part VI

This particular article will mainly discuss configuring PCI and ACPI.

Here, we can enable support for ACPI-controlled docking stations and removable drive bays (Dock). Remember, ACPI (Advanced Configuration and Power Management Interface) is a power management system. A docking station is a device that allows extra devices to be plugged in through extra ports. A docking station may contain many various ports and connectors. So, an ACPI-controlled docking station is a docking station that has its power managed by ACPI. A drive bay is a set of hardware for adding hard-drives. This too can be managed by ACPI.

Next, we can allow ACPI to be used as the idle manager for the CPU (Processor). This will cause the processor to enter the ACPI C2 and C3 states when idle. This will save power and reduce the temperature of the CPU chip. Processors only idle when they are 100% free. No application must request CPU resources for a set period of time.

There are four CPU power states – C0, C1, C2, and C3. C0 is the active operating state. C1 (Halt) is an active state that is not executing instructions, but can do so instantly. C2 (Stop-Clock) is a powered down state. C3 (Sleep) is more powered down than C2. In C3, the cache is now longer synchronized or managed until the CPU leaves this state. There is a fifth state called C1E (Enhanced Halt State) that has lower power consumption.

If the IPMI driver is enabled, then ACPI can access the BMC controller (IPMI). A Baseboard Management Controller is a microcontroller that manages the connection between the software and hardware. The Intelligent Platform Management Interface (IPMI) is a framework for managing the computer through a direct network at the hardware level rather than going through a login shell or the operating system.

The ACPI v4.0 process aggregator allows the kernel to apply a CPU configuration to all processors on the system (Processor Aggregator). As of ACPI v4.0, only idling can be configured with this method.

After that, the ACPI thermal zone can be enabled (Thermal Zone). Most hardware supports this feature. This allows the fan's power to be managed by ACPI.

A custom DSDT can be linked to the kernel if this option is enabled (Custom DSDT Table file to include). In this setting, the developer must include the full path name to the file. The Differentiated System Description Table is a file that contains the system's information for supported power events. It is not required to enter a path name. These tables exist on the firmware. The kernel will handle this for you. The main purpose for this is if a developer need to use tables that are different from ones that are built-in to a device.

Arbitrary ACPI tables can be overridden using initrd (ACPI tables override via initrd). ACPI tables are basically rules and instructions on how to control and interact with the hardware.

Like all other parts of the kernel, the ACPI system can also generate debugging messages (Debug Statements). Like other debugging features, you may want to disable this and save about fifty kilobytes.

Enabling this next feature will create files (/sys/bus/pci/slots/) for each PCI slot detected on the system

More Linux Related Stuff on: <http://www.linux.org>

(PCI slot detection driver). A PCI slot is a port on a PCI motherboard that allows users to attach other PC devices. PCI is a type of motherboard. PCI refers to the way the components communicate with one another. Some applications may need these files.

The power management timer is another power-managing system (Power Management Timer Support). This is one of many system timers for keeping track of time. This one requires less power. The processor idling, voltage/frequency scaling, and throttling do not effect this timer. Numerous systems require this feature to be enabled.

Next, ACPI module and container device drivers can be enabled (Container and Module Devices). This enables hotplug support for processors, memory, and nodes. This is needed for NUMA systems.

This following driver offers support for ACPI memory hotplugging (Memory Hotplug). Some devices will not support hotpluggable memory even with this driver enabled. If this driver is added as a module, the module will be called acpi_memhotplug.

NOTE: For the kernel to have a particular feature, the hardware, BIOS, and firmware must support the feature in question. Some systems have a BIOS that does not control the hardware much. This type of BIOS will not restrict features often. If the kernel does have a particular feature, the hardware must have the ability to complete such a task.

The Smart Battery System driver offers access to the battery's status and information (Smart Battery System).

Next, we have a driver for a "Hardware Error Device". This device reports hardware errors through SCI. Usually, most of the reports will be on corrected errors.

Here is another ACPI debugging feature (Allow ACPI methods to be inserted/replaced at run time). This permits ACPI AML methods to be managed without rebooting the system. AML stands for ACPI Machine Language. With this debugging feature, the AML code can be changed and tested with requiring a reboot.

APEI is the ACPI error interface (ACPI Platform Error Interface (APEI)). APEI reports errors from the chipset to the operating system. This error interface also offers error injection abilities.

The hardware's firmware can send messages to the operating system when "SFI (Simple Firmware Interface) Support" is enabled. The firmware communicates with the operating system through static tables in memory. SFI-only computers will require this feature for the kernel to work.

To be able to change the processor's clock speed on runtime, enable this feature (CPU Frequency scaling). CPU frequency scaling means changing the processor's clock speed. This driver can be used to lower the clock speed to conserve power.

Next, is another power management subsystem (CPU idle PM support). When the processor is not active, it is best that it idles in an efficient way to reduce power consumption and reduce wear-and-tear on the CPU. Reduced power consumption will also lower the heat production from the internal components.

The Linux kernel offers many CPU idle drivers. On systems with multiple processors, some users may

More Linux Related Stuff on: <http://www.linux.org>

have a reason to use a different driver on each CPU (Support multiple cpuidle drivers). Enabling this driver will allow users to set a different driver to each processor.

For Intel processors, the kernel has a driver specific for managing the idleness of such CPU chips (Cpuidle Driver for Intel Processors).

When the memory chips are idle, those can also use reduced power (Intel chipset idle memory power saving driver). This driver is specific for Intel devices with IO AT support.

Different computers use different types of motherboards (PCI support). One type is PCI. This driver will allow the kernel to run on PCI motherboards.

Next, we can enable/disable "Support mmconfig PCI config space access".

After that, we have an option of enabling/disabling a driver for host bridge windows (Support mmconfig PCI config space access). WARNING: This driver is incomplete (at least in kernel version 3.9.4)

As mentioned above, there are other types of motherboards. This next option offers a driver for "PCI Express (PCIe) support". PCIe is an improved and faster version of PCI.

After that, this following driver should be enabled to allow hotplugging on PCIe motherboards (PCI Express Hotplug driver).

Next, we can enable/disable error reporting for PCIe motherboards (Root Port Advanced Error Reporting). This is the PCI Express AER driver.

This next feature can allow users to override BIOS and firmware settings for PCIe ECRC (PCI Express ECRC settings control). In the next option, there is an error injector for PCIe (PCIe AER error injector support).

The following setting offers the operating system control over PCIe active state and clock power management (PCI Express ASPM control). Normally, the firmware would control the ASPM, but this feature allows the operating system to take control.

Again, like so many components of the kernel, there is debugging support for ASPM (Debug PCI Express ASPM).

Next, in this menu, select the "Default ASPM policy".

After that choice, the next one is about allowing device drivers to enable Message Signaled Interrupts (MSI). It is usually best to allow devices to send the CPU interrupts.

To add numerous debugging messages to the system log, enable "PCI Debugging".

This next setting allows the PCI core to detect if it needs to enable PCI resource re-allocation (Enable PCI resource re-allocation detection).

When hosting a virtual operating system on Linux, it can sometimes help to reserve a PCI device for

More Linux Related Stuff on: <http://www.linux.org>

the virtual OS (PCI Stub driver). With operating system virtualization, one OS is running inside or beside another operating system. Sometimes they can compete for resources. Being able to reserve a device for the guest system (the virtual OS) can reduce competition and increase performance.

The next driver offered allows hypertransport devices to use interrupts (Interrupts on hypertransport devices). Hypertransport is a bus system/protocol for high-speed communication between processors.

This next driver for PCI virtualization allows virtual devices to be made that share their owned physical resources (PCI IOV support).

The PCI Page Request Interface (PRI) gives PCI devices that are behind an IOMMU (input/output memory management unit) to recover from page faults (PCI PRI support). A page fault is not an error; it refers to the event of software trying to access data not on physical memory.

Again, there are still more features to configure in the Linux kernel as you will see in the following articles.

Configuring the kernel Part VII

We will continue configuring the PCI features following the most important characteristic of computers - networking.

Process Address Space Identifiers (PASIDs) allow PCI devices to access multiple IO address spaces simultaneously (PCI PASID support). This feature requires an IOMMU that offers PASIDs support.

Next, we can enable/disable "PCI IO-APIC hotplug support". APIC stands for Advanced Programmable Interrupt Controllers. A programmable interrupt controller (PIC) gathers all interrupts from all of the different sources to a single or multiple CPU lines. An advanced PIC are the same as PICs, but they have better features like Advanced Interrupt Request Management and more priority models. Hotplugging is the ability to add a device while the system is still on and not required to be reset. This driver is for PCI motherboards to have the ability to handle input/output APIC hot-swapping.

After that, the following question asks about enabling "ISA-style DMA support". As mentioned in a previous article, DMA is direct memory access which is the ability for devices to access memory without help from the CPU. ISA stands for Industry Standard Architecture which is a bus standard like PCI. This feature would allow DMA support that is used with ISA motherboards.

Now, we can move on to "PCCard (PCMCIA/CardBus) support". PCMCIA stands for Personal Computer Memory Card International Association. PC-cards, PCMCIA cards, and Cardbus cards are all peripheral laptop devices that are in the shape of cards.

The next PCMCIA options deals with "16-bit PCMCIA support". Some older computers use 16-bit PCMCIA cards.

To load a Card Information Structure (CIS) from userspace to make a PCMCIA card work properly, this feature should be enabled (Load CIS updates from userspace).

CardBus is the newer 32-bit version of 16-bit PCMCIA. This driver offers support for such devices (32-bit CardBus support). To use 32-bit PC-cards, a Cardbus compatible host bridge is required.

This next driver provides support for the CardBus bridges mentioned above (CardBus yenta-compatible bridge support). This bridge is the hardware port that the PCMCIA cards plugin.

The next three options are "Special initialization for O2Micro bridges", "Special initialization for Ricoh bridges", and "Special initialization for TI and EnE bridges". These are all types of bridges for cards.

Next, the driver for "Auto-tune EnE bridges for CB cards" is offered.

"Special initialization for Toshiba ToPIC bridges" can be enabled/disabled in the next option.

The next device driver offered is "Cirrus PD6729 compatible bridge support". This is needed in some older laptops.

The next PCMCIA bridge driver is for Intel "i82092 compatible bridge support". This is also found in

More Linux Related Stuff on: <http://www.linux.org>

some older laptops. This is again another bridge driver.

After that, the following settings asks about enabling "Support for PCI Hotplug".

Next, ACPI PCI hotplugging can be enabled (ACPI PCI Hotplug driver). This driver allows the hotplugging of PCI devices with ACPI (that power management feature that was discussed previously).

For IBM systems, this next driver will need to be enabled for ACPI PCI hotplugging (ACPI PCI Hotplug driver IBM extensions). This is like the above feature but this is specific for IBM devices.

For systems with a CompactPCI system card that has CompactPCI hotswap support, enable the "CompactPCI Hotplug driver".

Next, we have an option for another type of CompactPCU system card (Ziatech ZT5550 CompactPCI Hotplug).

CompactPCI cards that use the #ENUM hotswap signal as a system register bit through the standard IO port will need this driver (Generic port I/O CompactPCI Hotplug).

Motherboards with SHPC PCI hotplug controllers need this next driver (SHPC PCI Hotplug driver). SHPC stands for Standard Hot-Plug Controller. This is a generic hotplug system for PCI motherboards.

RapidIO interconnected devices also need a special driver (RapidIO support). RapidIO chips and boards are faster than PCI and PCI-express.

The "IDT Tsi721 PCI Express SRIO Controller" is a specific type of RapidIO controller.

This next option allows the developer to enter in how long the system's discovery node should wait (in seconds) for a host to finish enumeration before giving up. It is usually best to stick to the defaults.

This feature will allow the RapidIO system to accept other traffic besides maintenance signals (Enable RapidIO Input/Output Ports).

To transfer RapidIO data to and from RIO devices using a DMA Engine framework, enable this driver (DMA Engine support for RapidIO). RIO devices are Reconfigurable Input/Output devices. RapidIO uses NREAD and NWRITE requests to transfer data between local and remote memory, so a driver is needed to allow RapidIO to use DMA to access RIO devices. A DMA controller is needed to complete this feature on a kernel with this feature.

The RapidIO can provide debugging messages if permitted (RapidIO subsystem debug messages). As mentioned before, debugging features can be disabled unless you or the person using the kernel being made needs debugging features.

The next driver offers "IDT Tsi57x SRIO switches support". This is a group of serial RapidIO switches. The next four options are for different serial RapidIO switch drivers - "IDT CPS-xx SRIO switches support", "Tsi568 SRIO switch support", "IDT CPS Gen.2 SRIO switch support", and "Tsi500 Parallel RapidIO switch support".

After managing those drivers, we can move on to other kernel options. The next option offers support

More Linux Related Stuff on: <http://www.linux.org>

for ELF (Kernel support for ELF binaries). Executable and Linkable Format (ELF) support is a format specification for executables. It is highly recommended that this be enabled.

To execute scripts and binaries that require an interpreter, this feature must be enabled (Kernel support for MISC binaries). These types of executables are often called wrapper-driven binary formats. Examples include Python2/3, .NET, Java, DOS executables, etc.

The kernel can generate core dumps when this option is enabled (Enable core dump support). This is a debugging feature. Unless this kernel is intended to be used for debugging (whether the kernel itself or software) this is not needed.

64-bit processors can execute 32-bit programs if "IA32 Emulation" is enabled. It is best that this feature is enabled unless the developer is sure that such a kernel will never need to run 32-bit code.

The old a.out binaries can also be supported (IA32 a.out support). Assembler Output, as it is called, is a file format of a type of compiled code. It is best to enable this because some shared libraries may use this format.

The next setting can allow 32-bit processes to access the whole 64-bit register file and wide data path (x32 ABI for 64-bit mode). However, 32-bit pointers are still used. These 32-bit processes will use less memory than the same process compiled to 64-bits because they use 32-bit pointers.

Now, we will move on to network support.

Our first network setting is to enable networking in general (Networking Support). Very few developers will disable this feature. If they do, the kernel will be small and fast, but it will never be able to use Wifi, Bluetooth, Ethernet, or any type of connection dealing with network devices or protocols. Some programs on stand-alone systems require this feature even though no network devices exist with the hardware. For instance, X11 depends on networking features. Only disable networking features in the kernel if you can provide an alternate way to display graphics on the screen.

"Packet socket" allows processes to communicate with network devices without a mediator. This can enhance performance.

The ss tool needs this feature enabled for packet monitoring (Packet: sockets monitoring interface). Packet monitoring means watching the network traffic concerning the local device.

"Unix domain sockets" are used to establish and access network connections. The X Windowing system requires this feature; this is the perfect example for why networking should be enabled in the kernel even for systems that will not use networks. Unix domain sockets is a network protocol that works between processes on the same running machine.

The above Unix sockets can be monitored by the ss tool, but this next feature must be enabled first (UNIX: socket monitoring interface).

The Transformation (XFRM) user configuration interface is used by many native Linux utilities, so this feature is highly recommended (Transformation user configuration). This enables Ipsec – Internet Protocol SECurity. Ipsec controls the authentication and/or encryption of the IP packets.

The next feature allows developers to give network packets a second policy (called a sub-policy) (Transformation sub policy support).

IPsec security association locators can be updated dynamically when this feature is enabled (Transformation migrate database). Devices using Mobile IPv6 need this feature. When a computer sets up a network connection with a router or any type of network device, the security protocols make sure the two do not accidentally get connected to some other device on the network. The IP packet is set to go to a particular device. However, mobile systems that will use a different network providing a 4g signal, for instance, needs to be able to make the same connection to the new network point. Even though it may be the same 4g provider, a different device would provide a 4g connection to that geographic area. When the device is in the new area, it will still use the same IP address.

Next is a feature that displays statistics on transformation errors during packet processing (Transformation statistics). This useful for developers; this is not needed, so it can be disabled.

The "PF_KEY sockets" are compatible with KAME sockets and are need when using IPsec tools ported from KAME. KAME is a free protocol stack for IPv4 IPsec, IPv6 IPsec, and IPv6.

This is another required Mobile IPv6 feature that adds a PF_KEY MIGRATE message to PF_KEYv2 sockets (PF_KEY MIGRATE).

Next is the most important and number one well known feature in networking that should be enabled - "TCP/IP networking". Most networks (including the Internet) depend on this protocol. Even the X Windowing system uses TCP/IP. This feature even permits users to ping themselves (Command: ping 127.0.0.1). To be able to use the Internet and/or X11, this must be enabled.

To address several computers on a network, "IP: multicasting" must be enabled. Multicasting is the ability to send messages to more than one computer, but not all of them. Broadcasting sends signals to all computers on the network.

If this kernel is for a Linux system that will act as a router, then enable this option (IP: advanced router).

With this following feature enabled, the IP address will be automatically configured at boot time (IP: kernel level autoconfiguration). This is useful when a user needs to connect to a network without needing to configure settings.

With the DHCP protocol support enabled, the Linux system can mount its root file system over a network using NFS and the IP address will be discovered using DHCP (IP: DHCP support). This allows the Linux system to have its root filesystem on a remote computer over the network without the user needing to manually manage the process each time the system is booted.

The next options is just like the above except the BOOTP protocol is used instead of DHCP (IP: BOOTP support). BOOTP is the bootstrap protocol; this protocol uses UDP instead of TCP and will only use IPv4 networks.

RARP is an old protocol that is now obsolete due to BOOTP and DHCP, but it can still be added to the kernel (IP: RARP support).

One network protocol can be used within another - a concept called tunneling. This feature can be used in the Linux kernel (IP: tunneling). The secure shell protocol (SSH) is an example of a tunneling protocol. This feature is needed for SSH.

The next driver can demultiplex GRE (Generic Routing Encapsulation) packets (IP: GRE demultiplexer). Demultiplexing is the process of making a single signal into many separate parts (this is not copying the message, only breaking it down). GRE (Generic Routing Encapsulation) is a tunneling protocol.

The following feature permits GRE tunnels to form in IP connections (IP: GRE tunnels over IP). This will allow GRE tunnels to form on IP networks.

A broadcast using GRE over IP can be done when this feature is enabled (IP: broadcast GRE over IP).

A Linux system intended to be a router for IP packets going to multiple places should have this enabled (IP: multicast routing).

Configuring the kernel Part VIII

In this article, we will continue configuring the network abilities. Remember, networking is the most important characteristic of computers, so this article and the networking ones after this are important to know.

Before we begin the next part of this series, I want to clarify something. This configuration process does not edit the current kernel on your system. This process is configuring the source code before a new kernel is compiled (or cross-compiled). Once I finish going through the configuration process, then I will discuss the topics that were suggested to me by my readers. Also, as a reminder, the text in the quotes or parenthesis of the first or (rarely) the second sentence of each paragraph is the name of the setting in the configuration tool.

First, we can enable two different Sparse Mode Protocol Independent Multicast routing protocols ("IP: PIM-SM version 1 support" and "IP: PIM-SM version 2 support"). A multicast is like a broadcast, but a broad cast sends signals to all computers while a multicast sends messages to a selective group or set of computers. All PIM protocols are multicasting routing protocols that work on IP.

NOTE: When a computer communicates with one other computer or server, this is called a unicast – just in case you were wondering.

The next network feature to configure is "ARP daemon support". This allows the kernel to have a list of IP addresses and their respective hardware addresses in internal cache. ARP stands for Address-Resolution-Protocol.

For extra security, "TCP syncookie support" should be enabled. This protects computers from SVN flood attacks. Hackers or malware could send SVN messages to a server to consume resources that would allow real visitors to use the service provided by the server. The SVN message opens a connection between a computer and a server. The Syncookie blocks SVN messages that are not legitimate. Then, real users can still access the site without hackers consuming bandwidth. Servers should have this feature enabled.

The following feature is for "Virtual (secure) IP: tunneling". Tunneling is the encapsulation of one network protocol into another. Secure tunnels are needed when making virtual private networks (VPN).

Next, enabling "AH transformation" adds support for the IPsec Authentication Header. This is a security measure that manages data authentication.

After that, enabling "ESP transformation" adds support for the IPsec Encapsulation Security Protocol. This is a security measure with encryption and optional data authentication.

The Linux kernel supports the IP Payload Compression Protocol if this feature is enabled (IP: IPComp transformation). This is a lossless compression system. Lossless refers to the fact that the data remains in its complete form. After the uncompression, there is no difference in the data before or after compression. The compression is performed before encryption. This compression protocol speeds up networks because less data is moving.

The next three settings deal with different IPsec features ("IP: IPsec transport mode", "IP: IPsec tunnel

More Linux Related Stuff on: <http://www.linux.org>

mode", and "IP: IPsec BEET mode"). IPsec stands for Internet Protocol SEcurity. Transport mode is the default IPsec mode for connections between two computers and/or servers. Transport mode uses either an AH or an ESP header and only encrypts the IP header. In tunnel mode, the IP header and payload are encrypted. Tunnel mode is usually used when connecting a gateway to a server or gateway or a server to a server. BEET mode (Bound End-to-End Tunnel) does not reconnect when the IP address changes; the BEET mode connection remains connected. BEET mode uses less bytes than the other modes.

Next, the kernel can be given support for large IPv4/TCP offloads that are received (Large Receive Offload (ipv4/tcp)). The network interface cards (NIC) process the TCP/IP stacks. This feature adds code to the kernel to process large stacks.

INET socket monitoring can be enabled (INET: socket monitoring interface). INET sockets go to the Internet. This feature (when enabled) watches the connections and traffic to and from the Internet.

Here is another socket monitoring interface (UDP: socket monitoring interface). This one is for User Datagram Protocol (UDP). Again, this feature watches the sockets for UDP.

The following setting enables various TCP congestion controls (TCP: advanced congestion control). If the network gets too busy or the bandwidth is full, then many computers must wait for some bandwidth or their data stream will be slow. It helps network performance if the traffic is managed properly.

TCP connections can be protected with MD5 (TCP: MD5 Signature Option support). This is used for protecting Border Gateway Protocol (BGP) connections between core routers. Core routers are the main routers of a network; these routers are sometimes referred to as the "backbone of the Internet/network". BGP is a protocol for making routing decisions.

The next setting allows you to enable/disable "The IPv6 protocol". When this is enabled, IPv4 will still work well.

The following feature is a special privacy ability (IPv6: Privacy Extensions (RFC 3041) support). This makes the system generate and use a different random address for the network interface.

NOTE: Nothing is computers is truly random. Random numbers and strings in computers are often called pseudo-random.

On networks with multiple routers, this feature will allow the system to more efficiently figure out which one to use (IPv6: Router Preference (RFC 4191)).

After that, an experimental feature for dealing with route information can be enabled/disabled (IPv6: Route Information (RFC 4191)). Remember, when making a stable kernel, only install experimental features unless you really need the feature in question.

Sometimes, when a system autoconfigures its IPv6 address, it may get an IPv6 address that is already used on the network. This experimental feature allows Duplicate Address Detection (DAD) (IPv6: Enable RFC 4429 Optimistic DAD).

IPv6 can get support for various IPsec features ("IPv6: AH transformation" and "IPv6: ESP

More Linux Related Stuff on: <http://www.linux.org>

transformation").

IPv6 can also use the IP Payload Compression Protocol discussed previously (IPv6: IPComp transformation).

There is even mobility support for IPv6 (IPv6: Mobility). This permits mobile devices using IPv6 to be able to use other networks while retaining the same IP address.

Again, there are some IPsec features for IPv6 ("IPv6: IPsec transport mode", "IPv6: IPsec tunnel mode", and "IPv6: IPsec BEET mode").

When enabled, IPv6 can support MIPv6 route optimization (IPv6: MIPv6 route optimization mode). This ensures the shortest and best network path is made. The download and upload speeds are faster if the messages are sent through less routers and other network devices.

If an administrator needs to connect two IPv6 networks, but it can only be done through an IPv4 connection, this kernel feature will make that possible (IPv6: IPv6-in-IPv4 tunnel (SIT driver)). This tunnels IPv6 through IPv4.

This tunneling feature is for IPv6-in-IPv6 and IPv4 tunneled in IPv6 (IPv6: IP-in-IPv6 tunnel (RFC2473)).

Another tunneling feature is seen (IPv6: GRE tunnel). This one allows GRE tunnels.

Multiple routing tables can be supported (IPv6: Multiple Routing Tables). A routing table is a list of network locations and the paths to take to get data to the destinations.

Routing can be done by source address or prefix with IPv6 if enabled (IPv6: source address based routing).

"IPv6 Multicast routing" is still experimental. IPv4 and IPv6 handle multicasting differently.

Typical multicast routers decide what should be done to multicast packets based on destination and source addresses (IPv6: multicast policy routing). Enabling this feature will allow interfaces and packet marks to be included in the decision making.

Next, the PIM-SMv2 multicast routing protocol can be enabled for IPv6 (IPv6: PIM-SM version 2 support). This is the same PIM mentioned previously with IPv4. Because IP v4 and v6 are different, PIM can be activated for v4 and/or v6.

Network packet labeling protocols (like CIPSO and RIPSQ) can be enabled (NetLabel subsystem support). These labels contain security information and permissions.

Network packets can be made more secure by enabling security marking (Security Marking).

This network feature adds some overhead (Time-stamping in PHY devices). PHY devices can time-stamp network packets. PHY stands for “PHYSical layer”. These devices manage received and transmitted data.

A netfilter can be enabled (Network packet filtering framework). Netfilters filter and mangle passing network packets. A packet filter is a type of firewall. If the packet matches certain criteria, the packet is not allowed through.

The Datagram Congestion Control Protocol can be enabled (The DCCP Protocol). DCCP permits bi-directional unicast connections. DCCP is helpful with streaming media, Internet telephony, and on-line games.

Next, the Stream Control Transmission Protocol can be enabled (The SCTP Protocol). SCTP works on top of IP and is a stable and reliable protocol.

The following protocol is the Reliable Datagram Sockets protocol (The RDS Protocol).

RDS can use Infiniband and iWARP as a transport that supports RDMA operations (RDS over Infiniband and iWARP). Infiniband and iWARP are both protocols. RDMA stands for remote direct memory access. RDMA is seen when a remote computer accesses another's RAM directly without the need of the local computer's operating system assistance. It is like direct memory access (DMA) but with a remote computer rather than this taking place locally.

RDS can also use TCP as a transport (RDS over TCP).

Next, "RDS debugging messages" should be disabled.

This next network protocol is for clusters (The TIPC Protocol). A cluster is a group of computers that act as one. They need a way to communicate, so they use the Transparent Inter Process Communication (TIPC) protocol.

This high-speed protocol uses packets with a fixed size (Asynchronous Transfer Mode (ATM)).

IP can be used on ATM to communicate with systems using IP that are connected to a ATM network (Classical IP over ATM).

This next feature disables the "ICMP host unreachable" error message (Do NOT send ICMP if no neighbor). This prevents problems when the ATMARP tables from being removed due to a revalidation. ATMARP tables manage address resolution. ICMP stands for Internet Control Message Protocol and is usually used for sending error messages across a network.

LAN Emulation (LANE) emulates LAN services on an ATM network (LAN Emulation (LANE) support). A LANE computer can be a proxy that bridges Ethernet and ELAN segments.

"Multi-Protocol Over ATM (MPOA) support" allows ATM devices to send connections over subnetwork boundaries.

With this feature, ATM PVCs will behave like Ethernet, at least from the kernel's perspective (RFC1483/2684 Bridged protocols). PVC stands for permanent virtual circuit. A virtual connection is a packet-based connection that uses other higher protocols along with the main/original protocol.

The "Layer Two Tunneling Protocol (L2TP)" makes tunneling transparent to applications. Virtual Private Networks (VPNs) uses L2TP.

More Linux Related Stuff on: <http://www.linux.org>

To make a Linux-based Ethernet bridge, enabled this bridging feature (802.1d Ethernet Bridging). In networking, a bridge connects two or more networks together. An Ethernet bridge is a hardware bridge that uses Ethernet ports.

"IGMP/MLD snooping" is the ability for an Ethernet bridge to be able to selectively forward multicast signals based on the IGMP/MLD load. Disabling this feature can reduce the kernel size significantly. IGMP stands for Internet Group Management Protocol which is a protocol used to set up multicast groups. MLD stands for Multicast Listener Discovery.

This next filtering feature allows Ethernet bridges to selectively manage traffic based on VLAN information seen in each packet. Disabling this feature reduces the size of the kernel being made.

VLAN interfaces can be created on Ethernet interfaces with this feature enabled (802.1Q VLAN Support).

Next, "GVRP (GARP VLAN Registration Protocol)" support allows the GVRP protocol to be used to register certain values from network devices.

After that, "MVRP (Multiple VLAN Registration Protocol) support" can be enabled. MVRP is a newer alternative to GVRP.

"DECnet Support" is a network protocol made by Digital. This is a stable and safe protocol.

"DECnet router support" allows users to make a Linux-based router that supports DECnet.

NOTE: Linux can be used as a server, workstation, router, cluster, firewall, and supports many other uses.

This next feature will enable Logical Link Layer type 2 (ANSI/IEEE 802.2 LLC type 2 Support). This layer allows multiple protocols to be used on the same network equipment. It is highly recommended that this be enabled in environments where networks are important. It may be best if all kernels supported this feature.

Configuring the kernel Part IX

The Novell networking protocol, IPX, is commonly used between Windows systems and NetWare servers (The IPX protocol). IPX stands for Internetwork Packet Exchange. This is a network layer that is commonly used with the SPX protocol that is the transport layer.

To give a NetWare server the same IPX address on all of the networks it serves, enable this next feature (IPX: Full internal IPX network). Otherwise, each network will see the server as having a different IPX address.

NOTE: The IPX protocol uses IPX addressing, not IP addressing. IP addresses are not the only network addresses in computer networking.

For Linux systems on an Apple network, it will help to enable Appletalk (Appletalk protocol support). Apple computers and Apple printers commonly use Appletalk to communicate across a network. Appletalk does not require a central router/server and the network system is plug-and-play.

Linux systems that need to use IP on an Appletalk network will need "Appletalk interfaces support".

The next feature allows users to have IP tunneled in Appletalk (Appletalk-IP driver support).

Next, this feature permits IP packets to be encapsulated in Appletalk frames (IP to Appletalk-IP Encapsulation support). In networking, a frame is a special sequence of bits that marks the beginning and end of packets. This feature would place IP packets inside of Appletalk packets.

This setting allows the decapsulator to be enabled for the previous feature (Appletalk-IP to IP Decapsulation support). The decapsulator takes the IP packet out of the Appletalk packet.

This is another protocol layer called "X.25" (CCITT X.25 Packet Layer). This layer is usually enabled for very large networks like country wide public networks. Many banks use this in their extensive network systems. X25 (spelled "X25" or "X.25") networks have packet-switching exchange (PSE) nodes which are devices that package the incoming data into packets. X25 is being replaced by IP which is more simple than X25. X25 is an old protocol that is not as efficient as TCP/IP, but some companies find it very useful in large, complex networks.

LAPB is a data link layer used by X.25 (LAPB Data Link Driver). If the above is enabled, then this should also be enabled. LAPB stands for "Link Access Procedure Balanced". LAPB can also be used on Ethernet and X.21 (this is not a typo) network cards. X.21 is a protocol used on the physical layer (hardware) and X.25 is used on the network layer. LAPB checks for errors and ensures the packets are placed back in the correct sequence.

Nokia modems use the phone network protocol commonly called PhoNet (Phonet protocols family). Linux computers controlling Nokia phones remotely need this feature.

The next network protocol is for small wireless connections between various automatic devices (IEEE Std 802.15.4 Low-Rate Wireless Personal Area Networks support). 802.15.4 is a simple low data rate protocol that requires little amounts of power. This wireless protocol extends out a maximum of about ten meters. This can be useful in robotics when connecting sensors through a wireless network. Any type of machinery that should not have cables may benefit from this local wireless network in place of

More Linux Related Stuff on: <http://www.linux.org>

cords.

If the above feature was enabled, it may be wise to enable this IPv6 compression feature (6lowpan support over IEEE 802.15.4).

SoftMAC devices that support IEEE 802.15.4 at the PHY level can use this feature (Generic IEEE 802.15.4 Soft Networking Stack (mac802154)).

When many packets need to be transmitted, the kernel must decide which ones to send out first (they cannot all be sent out at once), so they feature will help the kernel prioritize the packets (QoS and/or fair queuing). If this is not enabled, then the kernel will use a "first come, first serve approach". This would mean urgent networking messages will need to wait for their turn for transmission.

On networks with a data center server, this feature is highly recommended (Data Center Bridging support). This feature enhances the Ethernet connections for data center networks.

DNS lookups will be enabled with this next ability (DNS Resolver support). Most systems store DNS cache that allows the computer to perform a DNS lookup itself without the aid of a DNS server.

Next is another routing protocol for multi-hop ad-hoc mesh networks (B.A.T.M.A.N. Advanced Meshing Protocol). "B.A.T.M.A.N." stands for "better approach to mobile ad-hoc networking". This works on wired and wireless networks. Ad-hoc networks have no central infrastructure like routers and such. Each device on the network behaves like a router. Mesh networks are a similar concept. Each node must route data sent to it. In a mesh network, each computer is connected to all or nearly all of the other network devices. When such networks are written on a paper to make a map, the network looks like a mesh.

When many mesh nodes are connected to the same LAN and mesh, some network signals may loop (Bridge Loop Avoidance). This feature will prevent such looping. These loops may never end or harm performance. Preventing such looping is called Bridge Loop Avoidance (BLA).

Distributed ARP Tables (DAT) are used to enhance ARP reliability on sparse wireless mesh networks (Distributed ARP Table).

The BATMAN protocol has some debugging features that may be used by developers (B.A.T.M.A.N. Debugging). As with any debugging feature, it is usually best to disable it to save space and have a better optimized kernel.

Virtualized environments can benefit from "Open vSwitch". This is a multilayer Ethernet switch. Open vSwitch supports numerous protocols.

Network connections between virtual machines, hypervisors, and the host needs the "virtual socket protocol". This is similar to TCP/IP. These sockets are like other network sockets, but they are for virtual machines. This allows a guest OS to have a network connection with the host.

There is a cgroup subsystem that can manage network priorities (Network priority cgroup). This control group gives network traffic priority based on the sending application.

BPF filtering is handled by an interpreter, but the kernel can perform BPF filtering natively with this

More Linux Related Stuff on: <http://www.linux.org>

enabled (enable BPF Just In Time compiler). BPF stands for "Berkeley Packet Filter". This allows the computer system to support raw link-layer packets.

Next, we have two network testing tools. The first one is a "Packet Generator" which injects packets (makes void packets) for testing the network. The second, allows an alert system to be setup to alert the user/system when packets are dropped (Network packet drop alerting service).

The Linux kernel can be made to run radio systems or control them remotely. The "Amateur Radio AX.25 Level 2 protocol" is used for computer communication via radio. This radio protocol can support TCP/IP among many other protocols.

To prevent collisions on an AX.25 network, enable DAMA (AX.25 DAMA Slave support). So far, Linux cannot act as a DAMA server, but it can be a DAMA slave. DAMA stands for Demand Assigned Multiple Access. DAMA assigns network traffic to particular channels.

NET/ROM is a routing layer for AX.25 (Amateur Radio NET/ROM protocol).

An alternative to NET/ROM is Packet Layer Protocol (PLP) which also runs on top of AX.25 (Amateur Radio X.25 PLP (Rose)).

The Controller Area Network (CAN) bus devices need this driver (CAN bus subsystem support). CAN is a serial protocol with various purposes.

The CAN bus can be accessed through the BSD socket API with this feature (Raw CAN Protocol (raw access with CAN-ID filtering)).

A broadcast manager for the CAN protocol is available for the kernel (Broadcast Manager CAN Protocol (with content filtering)). This manager offers many controls including content filtering.

To make a Linux box a CAN Router and/or gateway, this feature is needed (CAN Gateway/Router (with netlink configuration)).

NOTE: A gateway is an interface device between two or more networks that each support different protocols. A simple definition would be a gateway is a protocol converter.

NOTE: A router redirects network traffic and connects networks together that use the same protocols.

There are many CAN devices (mainly controllers) and interfaces that the Linux kernel can support if enabled. All of the CAN drivers are for different brands and models of these types of devices. In the configuration tool, they have titles as seen below.

- Virtual Local CAN Interface (vcan)
- Serial / USB serial CAN Adaptors (slcan)
- Platform CAN drivers with Netlink support
- Enable LED triggers for Netlink based drivers
- Microchip MCP251x SPI CAN controllers
- Janz VMOD-ICAN3 Intelligent CAN controller
- Intel EG20T PCH CAN controller

More Linux Related Stuff on: <http://www.linux.org>

Philips/NXP SJA1000 devices
Bosch C_CAN/D_CAN devices
Bosch CC770 and Intel AN82527 devices
CAN USB interfaces
Softing Gmbh CAN generic support
Softing Gmbh CAN pcmcia cards

Like the many other features in the kernel, CAN devices also have debugging abilities that can be enabled (CAN devices debugging messages). Again, remember the purpose of your kernel - will you need debugging or do you want performance?

The Linux kernel also supports protocols for infrared signals (IrDA (infrared) subsystem support). IrDA stands for "Infrared Data Associations"; this is a standard for infrared signals.

Many people today will want these Bluetooth abilities (Bluetooth subsystem support).

RxRPC session sockets can be enabled (RxRPC session sockets). These sockets carry network connections that are using the RxRPC protocol. RxRPC runs on top of UDP.

The Linux kernel can support "RxRPC dynamic debugging" if enabled.

RxRPC has kerberos 4 and AFS kaserver security abilities that can be enabled (RxRPC Kerberos security). Kerberos is an authentication protocol where each network device is required to prove each other's identity to each other before any data transaction.

The configuration tool for wireless LAN (802.11) devices should be enabled for computers that have a wireless device like Wifi (cfg80211 - wireless configuration API). cfg80211 stands for Configuration 802.11. 802.11 is a wireless specification.

The "nl80211 testmode command" is used for calibration and/or validation utilities that perform such tasks on chips for wireless devices.

The next setting allows users to "enable developer warnings" for cfg80211 devices.

Next, "cfg80211 regulatory debugging" can be enabled.

The following cfg80211 setting is for "cfg80211 certification onus".

Powersaving features should be enabled for cfg80211 compatible devices (enable powersave by default).

cfg80211 supports debugfs entries (cfg80211 DebugFS entries).

Wireless devices have regulatory rules that they follow; these are stored in a database (use statically compiled regulatory rules database).

Some wireless extensions that use cfg80211-based drivers may need to use an old userspace; this feature permits that action (cfg80211 wireless extensions compatibility).

lib80211 can provide debugging features (lib80211 debugging messages).

The hardware independent IEEE 802.11 standard can be enabled (Generic IEEE 802.11 Networking Stack (mac80211)). mac80211 is a framework used to make SoftMAC wireless drivers. SoftMAC allows better control and configuration of the device.

This next feature allows the mac80211 to use a PID controller to manage the TX rate (PID controller based rate control algorithm). TX rate is the BPS (Bits per minute). Specifically, this feature is the algorithm for controlling the rate of data flow.

Another algorithm for the same feature is called "Minstrel". This is a much more precise and efficient TX rate managing algorithm.

Minstrel also supports 802.11n (Minstrel 802.11n support).

Because there are two TX rate controlling algorithms, only one can be used, so a default must be set (Default rate control algorithm (Minstrel)). Generally, it is best that Minstrel be the default.

Draft 802.11s mesh networking can be supported by the kernel (Enable mac80211 mesh networking (pre-802.11s) support). Draft 802.11s is a wireless standard for mesh networking.

For hardware that supports this feature, LED triggers for various packet traffic events can be enabled (Enable LED triggers). On many Ethernet devices, there is an LED that lights up when the port is active. This driver will make those LEDs work during packet traffic events.

mac80211 also supports debugfs features (Export mac80211 internals in DebugFS).

Here is a feature that collects mac80211 debug messages independently of the typical logging system (Trace all mac80211 debug messages).

There is also another set of mac80211 debugging abilities, but these use the typical logging system (Select mac80211 debugging features --->). In this menu, select the debugging features you need (if any).

Configuring the kernel Part X

Wireless broadband devices that use the WiMAX protocol can be enabled (WiMAX Wireless Broadband support). This type of wireless connection usually works only if the connection service is provided by a service provider (this is the same concept as with 3G/4G). WiMAX stands for Worldwide Interoperability for Microwave Access. WiMAX is intended to be a replacement for DSL. Broadband refers to the wide bandwidth and transportation of numerous signals.

RF switches are used in many Wifi and Bluetooth cards (RF switch subsystem support). The “RF” stands for Radio Frequency. RF switches route high-frequency signals.

Input support for RF switches also exists in the kernel (RF switch input support).

The kernel can control and query radio transmitters (Generic rfkill regulator driver). Enabling this will make a device file (/dev/rfkill). This device file acts as an interface to such radio devices.

The Linux kernel supports the 9P2000 protocol (Plan 9 Resource Sharing Support (9P2000)). This network protocol is sometimes called Styx. Plan 9's windowing system (Rio) uses Styx the same way Linux's X11 uses Unix Network Sockets. Linux systems may use Styx on Styx networks. Plan 9 and Linux can use Styx on a network.

The "9P Virtio Transport" system provides transports to and from guest and host partitions on virtual systems.

RDMA transport is also supported by the kernel (9P RDMA Transport (Experimental)). RDMA stands for Remote Direct Memory Access. This is Plan 9's protocol for accessing memory from a remote computer.

The 9P system has debugging support like many of the other kernel components (Debug information).

"CAIF support" support can also be enabled in the kernel. CAIF stands for Communication CPU to Application CPU Interface. This is a MUX protocol that uses packets and is used with ST-Ericsson's modems. ST-Ericsson is the company that developed this protocol. Android and MeeGo phones use this protocol. (Yes, MeeGo and Android are Linux systems, and yes, I am talking about the popular Android by Google.) A MUX protocol is a multiplexing protocol. Multiplexing was mentioned in a previous article.

Next, cephlib can be added to the kernel which is used in the rados block devices (rbd) and the Ceph filesystem (Ceph core library). cephlib is the complete core library for Ceph. Ceph is a storage platform. CephFS (the Ceph filesystem) is a filesystem that runs on top of another. Usually, CephFS runs on top of EXT2, ZFS, XFS, or BTRFS. Rados devices are block storage units that use CephFS.

This debugging feature for ceph harms the kernels performance, so only use it if needed (Include file:line in ceph debug output).

The CONFIG_DNS_RESOLVER facility will perform DNS lookups when this feature is enabled (Use in-kernel support for DNS lookup).

Near Field Communication (NFC) devices are also supported by the Linux kernel (NFC subsystem)

More Linux Related Stuff on: <http://www.linux.org>

support).

The NFC Controller Interface (NCI) should be enabled if the above feature is enabled (NCI protocol support). This allows the host and the NFC controller to communicate.

NFC devices that process HCI frames will need this next feature to be enabled (NFC HCI implementation).

Some HCI drivers need a SHDLC link layer (SHDLC link layer for HCI based NFC drivers). SHDLC is a protocol that checks integrity and manages the order of the HCI frames.

"NFC LLCP support" is usually enabled if NFC features (like the above) are enabled.

Next, there are some drivers for specific NFC devices. The first one is a "NXP PN533 USB driver".

The next NFC driver supports Texas Instrument's BT/FM/GPS/NFC devices (Texas Instruments NFC WiLink driver).

Next is the "NXP PN544 NFC driver".

The driver for Inside Secure microread NFC chips is also provided by the kernel (Inside Secure microread NFC driver).

Now, we will be moving on to various drivers that are not network related. First, we can allow a path to the uevent helper (path to uevent helper). Many computers today should not have this feature because one uevent helper is executed for every one fork a process possess. This can quickly consume resources.

On boot-up, the kernel will make a tmpfs/ramfs filesystem (Maintain a devtmpfs filesystem to mount at /dev). This offers the complete /dev/ directory system. Of the two filesystems (tmpfs and ramfs), ramfs is the most simple of the two. "tmpfs" stands for temporary filesystem and "ramfs" stands for ram filesystem.

The next setting is the code for the devtmpfs filesystem which is also mounted at /dev/ (Automount devtmpfs at /dev, after the kernel mounted the rootfs).

The following feature allows modules to be loaded into user space (Userspace firmware loading support).

To "Include in-kernel firmware blobs in kernel binary" (which will add proprietary firmware to the kernel) enable this feature.

Some binary proprietary drivers need to be used on boot-up. This feature allows such software to do so (External firmware blobs to build into the kernel binary). Some computers have boot devices that require special firmware that may only be proprietary binaries. Without this feature enabled, the system will not boot.

Enabling "Fallback user-helper invocation for firmware loading" allows user-helper (udev) to load firmware drivers as a fallback for the kernel fails to load firmware drivers. udev can load firmware that

More Linux Related Stuff on: <http://www.linux.org>

resides in a non-standard path for firmware.

The part of the kernel that manages drivers can produce debugging messages if permitted (Driver Core verbose debug messages).

Next, the devres.log file will be used if this feature is enabled (Managed device resources verbose debug messages). This is a debugging system for device resources.

This next feature makes a connection between the userspace and kernelspace via a netlink socket (Connector - unified userspace <-> kernelspace linker). This socket uses the netlink protocol. This is another example of a Linux system needing networking abilities even if the computer will never be on a physical network.

The userspace can be informed on process events via a socket (Report process events to userspace). Some reported events include ID changes, forks, and exit status. Some previously enabled kernel features may need this. It is best to follow what the configuration tool recommends.

Systems that use solid state drives will need MTD support (Memory Technology Device (MTD) support). MTD devices are solid state storage devices. Typical storage drivers are different than Solid State Drives (SSD). The standard routines used on magnetic storage units do not work on SSDs (read, write, and erase).

Most desktop computers have parallel ports (a connector with 25 holes), so they need this feature (Parallel port support). Parallel ports are used for printers and ZIP drives among many other less known uses. Parallel ports are the ports with twenty-five pins.

Enable this feature for IBM compatible computers (PC-style hardware). There are different types of computers. Besides IBM computers (which commonly run Windows), there are Apple computers. Linux runs on nearly every type of computer.

Linux also supports multi-IO PCI cards (Multi-IO cards (parallel and serial)). Multi-IO PCI cards have both parallel and serial ports. Serial ports send or receive one bit at a time.

This next feature allows the kernel to "Use FIFO/DMA if available". This is used on certain parallel port cards to speed up printing. FIFO stands for "First In, First Out". DMA is Direct Memory Access as mentioned before.

The next feature is for probing Super-IO cards (SuperIO chipset support). These probes find the IRQ numbers, DMA channels, and other types of addresses/numbers of such devices. Super-IO is a type of integrated IO controller.

PCMCIA support for parallel ports can be enabled (Support for PCMCIA management for PC-style ports).

NOTE: For many of these features, it may be best to do what the configuration tool recommends unless you have a specific reason for not doing so. Usually, if you are cross-compiling or making a kernel for a broad range of computers, then you should be familiar with what you are wanting to support and make the choices accordingly.

Parallel ports on AX88796 network controllers need this support (AX88796 Parallel Port).

"IEEE 1284 transfer modes" allows Enhanced Parallel Port (EPP) and Enhanced Capability Port (ECP) on parallel ports and status readback for printers. Status readback is the retrieval of the printer's status.

"Plug and Play support" (PnP) should be enabled. This allows users to plugin devices while the system is still on and then immediately utilize them. Without this feature, users could not plugin a USB device, printer, or some other device without performing any special tasks. The system will manage the rest automatically.

Next, users can enable block devices (Block devices). This is a feature that should be enabled because block devices are very common.

Floppy disks are block devices that can be enabled (Normal floppy disk support).

IDE devices that connect to parallel ports are also supported (Parallel port IDE device support). Some external CD-ROM devices can connect via parallel ports.

External IDE storage units can also be connected to parallel ports (Parallel port IDE disks).

ATA Packet Interface (ATAPI) CD-ROM drives that connect to parallel ports will need this driver (Parallel port ATAPI CD-ROMs). ATAPI is an extension of the ATA protocol used in Parallel ATA (PATA) devices.

Other ATAPI disk devices can be plugged into the parallel ports (Parallel port ATAPI disks). This driver will support other disk types besides CD-ROMs.

The kernel also supports ATAPI tape devices that connect via the parallel ports (Parallel port ATAPI tapes).

There are many other ATAPI devices that can connect to the parallel ports. As a result, a generic driver was made to manage the other devices not supported by the previously mentioned drivers (Parallel port generic ATAPI devices).

IDE devices attached to the parallel ports need a special protocol for communication purposes. There are many such protocols, one of them being the "ATEN EH-100 protocol".

An alternate protocol for parallel IDE devices is the "MicroSolutions backpack (Series 5) protocol".

There is yet again another parallel IDE device protocol (DataStor Commuter protocol) and another (DataStor EP-2000 protocol) and another (FIT TD-2000 protocol).

Once again, there is another protocol, but this one is highly recommended for the newer CD-ROM and PD/CD devices that plug into parallel ports (FIT TD-3000 protocol).

This next protocol is mainly for parallel port devices made by SyQuest, Avatar, Imation, and HP (Shuttle EPAT/EPEZ protocol).

Imation SuperDisks need support for the Shuttle EP1284 chip (Support c7/c8 chips).

More Linux Related Stuff on: <http://www.linux.org>

Some other parallel IDE protocols that can be enabled next include

Shuttle EPIA protocol

Freecom IQ ASIC-2 protocol - used by Maxell Superdisks

FreeCom power protocol

KingByte KBIC-951A/971A protocols

KT PHd protocol - used in by 2.5 inch external parallel port hard-drives.

OnSpec 90c20 protocol

OnSpec 90c26 protocol

NOTE: These protocols and support for plugging devices into the parallel port is meant to be like hotplugging devices just like USB devices are put in USB ports. USB and Firewire are still the most popular ports to use because of their size and speed. A parallel storage unit is larger than a USB flash drive because parallel ports are larger than USB ports.

Next, we have a driver for Micron PCIe Solid State Drives (Block Device Driver for Micron PCIe SSDs).

Configuring the kernel Part XI

Linux supports two different Compaq Smart Array controllers (Compaq SMART2 support) and (Compaq Smart Array 5xxx support). Array controllers are devices that present physical storage units as logical units. These controllers may also implement hardware-based RAID. The difference between hardware and software RAID is simple. Linux manages and sees software RAID. Linux sees hardware RAID as any other storage unit. This means Linux is not aware that the device is a RAID drive. The hardware (array controller) manages the RAID system independently of the kernel. This is better for system performance because the kernel does not need to configure or manage RAID. Note, not all array controllers have RAID abilities.

The above mentioned array controllers can access SCSI tapes with this driver (SCSI tape drive support for Smart Array 5xxx). SCSI tapes are magnetic-tape drives that use the SCSI protocol.

The PCI RAID controllers Mylex DAC960, AcceleRAID, and eXtremeRAID are supported by this driver (Mylex DAC960/DAC1100 PCI RAID Controller support). A PCI RAID controller is an array controller connected to a PCI card. RAID controllers are array controllers with RAID capabilities.

The MM5415 battery-backed RAM chips are supported with this driver (Micro Memory MM5415 Battery Backed RAM support). Battery-backed RAM chips allow the data to remain on the RAM device when the power is lost. This helps to protect data. Otherwise, when power is lost, the current computer session is lost.

To use a typical file (such as an ISO image) as a block device and mount it when this feature is enabled (Loopback device support). This is useful for retrieving files from image files without the need to burn the file to a disk or pull it apart. Imagine getting an ISO file from the Internet that contains many files. If only one file in the package is desired and the user does not want to burn the ISO to a disc or does not know how to open an ISO file, the user can instead mount the ISO.

The Linux kernel creates some loop devices during init time, so some loopback devices are already prepared and made (Number of loop devices to pre-create at init time). This feature saves time when a file (like an ISO) or virtual device (like a virtual hard-drive [vhd]) is mounted as a loopback device. This setting allows developers to choose how many the kernel should pre-make.

Ciphers from the CryptoAPI can be used when "Cryptoloop Support" is enabled. This can be used for hard-drive encryption. However, not all filesystems are supported.

Next, users can enable "DRBD Distributed Replicated Block Device support". This is like a network RAID1. These devices own the device files /dev/drbdx. These storage devices are usually used in clusters where each computer in the cluster has a storage unit that is mirrored from the master. This means each computer's hard-drive is a mirrored copy of the hard-drive found in the central "master" computer of the group. A cluster is a group of computers acting as one large powerful unit. However, each cluster has one controlling computer called the master node. The rest of the computers are slave nodes.

The DRBD supports fault injection for testing IO error handling (DRBD fault injection). Remember, fault injection is the process of making a device/software think an error occurred so the developer can test how the hardware/software handles errors.

If the kernel is intended to be a client for network block devices, then enable this feature (Network block device support). The first device file is /dev/nd0. A network block device is a remote storage unit that is accessed via a network.

Solid state drives (SSD) that are directly connected to PCI or PCI Express cards will need this driver (NVM Express block device).

Individual SCSI object-based storage (OSD) objects can be used as a block device with this feature (OSD object-as-blkdev support).

Next is a driver for "Promise SATA SX8 support". This is a driver for a SATA controller made by Promise Technology Inc.

Linux allows a part of the RAM to be utilized as a block device (RAM block device support). This is commonly seen on live Linux distros where the system is running entirely on the RAM. A live Linux distro loads off of a disc and then loads into RAM so the installed OS is not harmed when trying a new operating system, or repairing another.

The next setting allows users to enter in the "Default number of RAM disks".

The "Default RAM disk size" can be set in kilobytes.

The kernel can support XIP filesystems on RAM devices acting as block devices (Support XIP filesystems on RAM block device). This feature will enlarge the kernel's size. XIP (eXecute In Place) filesystems are filesystems that allow executables to store data on the same filesystem instead of utilizing RAM like other applications. Such filesystems are needed when running an executable on a live Linux system which resides on the RAM.

Next, the kernel can be given support for "Packet writing on CD/DVD media".

The kernel developer can set the max amount of active concurrent packets (Free buffers for data gathering). A large number will speed up write performance at the cost of memory. One packet will consume about sixty-four kilobytes.

The Linux kernel can use a rewritable compact-disc as cache space (Enable write caching). This feature is still experimental.

The next feature allows the ATA specification to be used over Ethernet cables (ATA over Ethernet support).

This following driver allows virtual block devices to be created for virtio (Virtio block driver). Virtio is a platform for I/O virtualization.

Old hard-drives will need a special driver (Very old hard disk (MFM/RLL/IDE) driver).

Here is a driver for the previously mentioned Rados device (Rados block device (RBD)).

Next, a device specific driver is available (IBM FlashSystem 70/80 PCIe SSD Device Driver).

Now, we will move on to miscellaneous devices. The first setting is for enabling/disabling potentiometers (Analog Devices Digital Potentiometers).

If the potentiometer(s) is on a I2C bus, then this should be enabled (support I2C bus connection).

If the potentiometer is attached to a SPI bus, then this driver will be needed (support SPI bus connection).

NOTE: The Linux kernel supports many sensors because the Linux kernel is often used in weather devices and robots.

This driver is for the IBM RSA (Condor) service processor (Device driver for IBM RSA service processor).

The kernel also has a driver for the PCI Sensable PHANToM devices (Sensable PHANToM (PCI)).

This driver is for directing trace data from various devices via Parallel Trace Interface (PTI) out to Intel Penwell PTI ports (Parallel Trace Interface for MIPI P1149.7 cJTAG standard). This directed data is for debugging purposes.

Some SGI IO controllers have an IOC4 chip that needs this driver (SGI IOC4 Base IO support). SGI IO controllers are devices made by SGI that manage input/output. The IOC4 chip controls a lot of the tasks performed by such devices. This is a basic driver. Other drivers for such devices will rely on this driver.

There are few Texas Instruments Flash Media adapter drivers for the Linux kernel (TI Flash Media interface support) and (TI Flash Media PCI74xx/PCI76xx host adapter support).

This "Integrated Circuits ICS932S401" driver for ICS932S401 clock control chips.

The Atmel Synchronized Serial Communication peripheral (SSC) has a driver in the kernel (Device driver for Atmel SSC peripheral).

Such a device provides point-to-point serial connections between devices.

The "Enclosure Services" feature supports hard-drive bays.

This is a timer driver for CS5535/CS5536 chips (CS5535/CS5536 Geode Multi-Function General Purpose Timer (MFGPT) support).

This driver gives applications the ability to communicate with iLO management processors in HP ProLiant servers (Channel interface driver for the HP iLO processor). “iLO” stands for Integrity Integrated Lights-Out. iLO allows remote server management.

The Linux kernel supports the ALS APDS9802 light sensor (Medfield Avago APDS9802 ALS Sensor module). Some other supported sensor include

Intersil ISL29003 ambient light sensor

Intersil ISL29020 ambient light sensor

Taos TSL2550 ambient light sensor

ROHM BH1780GLI ambient light sensor

More Linux Related Stuff on: <http://www.linux.org>

BH1770GLC / SFH7770 combined ALS - Proximity sensor
APDS990X combined als and proximity sensors

NOTE: Most of these drivers should be included as modules if the kernel being made is intended for a broad range of computers.

Linux can even use a "Honeywell HMC6352 compass".

The kernel also supports the "Dallas DS1682 Total Elapsed Time Recorder with Alarm".

The 16-bit digital-to-analog converter is supported with this driver (Texas Instruments DAC7512).

The "VMware Balloon Driver" manages physical memory by taking physical pages from guest operating systems that do not need it and giving pages to those that do.

There are also two different pressure sensors (BMP085 digital pressure sensor on I2C) and (BMP085 digital pressure sensor on SPI).

An Intel Input/Output Hub (IOH) is also supported by the kernel (Intel EG20T PCH/LAPIS Semicon IOH(ML7213/ML7223/ML7831) PHUB). Specifically, this is the PCH PHUB (Platform Controller Hub Packet Hub) Intel Topcliff.

The "FSA9480 USB Switch" is a detector that detects when devices are plugged in.

The next option permits bitstream configuration (Lattice ECP3 FPGA bitstream configuration via SPI).

The Silicon microcontroller uses the Silicon Labs C2 Port which needs a special driver (Silicon Labs C2 port support).

Configuring the kernel Part XII

The kernel supports a large variety of hardware, protocols, and features as you have already guessed.

The next set of features we will discuss is "EEPROM support". Electrically Erasable Programmable Read-Only Memory is a form of memory that is not erased when the power is lost or purposely shut off.

The kernel supports EEPROM chips on I2C cards including FRAMs, ROMs, and SRAMs (I2C EEPROMs / RAMs / ROMs from most vendors). FRAM (also called FeRAM) is a Random Access Memory chip that uses ferroelectric principles instead of dielectric materials to store data. A ROM chip is a Read Only Memory chip. SRAM is Static instead of Dynamic like DRAM. DRAM must be refreshed to retain data while SRAM does not require refreshing. However, both lose the stored data when the power is turned off or lost.

The kernel also supports EEPROMs for SPI buses (SPI EEPROMs from most vendors). The Serial Peripheral Interface Bus (SPI) is a full duplex bus system that lacks an error-checking protocol.

The older I2C EEPROM chips require a driver other than the I2C driver above (Old I2C EEPROM reader). The I2C bus is intended for embedded systems and cellphones due to the low-speed bus protocol that is used.

This feature is needed to prevent the Maxim programmable EEPROM from going into read-only mode (Maxim MAX6874/5 power supply supervisor). Specifically, this driver provides better power management to the chip.

There is also a driver for "EEPROM 93CX6 support", "Microwire EEPROM 93XX46 support", and then "ENE CB710/720 Flash memory card reader support".

Like with many other kernel features, there are debugging features for EEPROM devices (Enable driver debugging). Again, for better performance, disable debugging features.

Next, we have a Texas Instruments feature (Shared transport core driver). This driver provides transport protocols for BT/FM and GPS chips.

The following driver supports the I2C LIS3LV02Dx accelerometer (STMicroelectronics LIS3LV02Dx three-axis digital accelerometer (I2C)). The data provided by the device is stored in /sys/devices/platform/lis3lv02d.

Next, the Linux kernel supports a module for downloading firmware to Altera's FPGA (Altera FPGA firmware download module). An FGPA is a field-programmable gate array. These are programmable integrated circuits.

The "Intel Management Engine Interface" offers security and other services to Intel chips.

"ME Enabled Intel Chipsets" can have MEI support. MEI is Management Engine Interface. This driver gives supported chipsets MEI services.

The "VMware VMCI Driver" is a high-speed virtual device for relaying communications between the guest and host. VMCI stands for Virtual Machine Communication Interface.

Next, "ATA/ATAPI/MFM/RLL support" can be enabled/disabled. MFM (Modified Frequency Modulation) is a specific method for encoding floppy disk bits. However, this does not work on all floppy disks. MFM uses the RLL (Run-Length Limited) coding scheme. RLL transfers data going through communication systems that have bandwidth limits. ATAPI is the previously mentioned ATA Packet Interface and ATA is the same ATA mentioned when discussing interface standards.

Now, we will discuss SCSI support. Small Computer System Interface is another interface standard as is SATA. USB and FireWire devices use the SCSI protocol.

The first SCSI setting concerns the "RAID Transport Class". This allows RAID to use the SCSI standard.

To use SCSI targets, enable this feature (SCSI target support).

If the system will run old Linux applications, the system may need "legacy /proc/scsi/ support". This will create the SCSI files once placed in /proc/scsi/.

To support SCSI disks, enable the next feature (SCSI disk support). This is a generic driver.

To support SCSI tapes, enable this feature (SCSI tape support). This is a generic driver. A SCSI tape drive stores data on tape-like magnetic strips.

The OnStream SCSI tapes need this driver instead of the previously mentioned generic SCSI driver (SCSI OnStream SC-x0 tape support).

For "SCSI CDROM support" enable this feature. Some CD-ROMs use the SCSI protocol.

Next, the user can enable "Enable vendor-specific extensions (for SCSI CDROM)".

This is a SCSI driver that is a generic driver for a larger variety of SCSI devices (SCSI generic support). This is mainly used for SCSI scanners and other SCSI devices not supported by the previously mentioned SCSI drivers or those discussed afterwards.

Some SCSI jukeboxes need this SCSI driver (SCSI media changer support).

The Linux kernel offers "SCSI Enclosure Support". A SCSI enclosure is a device that manages the power and cooling of the SCSI device as well as offer other services not related to the data.

The Linux kernel should be set to search for all Logical Unit Numbers (LUN) for each SCSI device (Probe all LUNs on each SCSI device). A LUN is a SCSI address.

Extra error reporting is available for SCSI (Verbose SCSI error reporting (kernel size +=12K)). This will increase the size of the kernel significantly.

There is also a logger for SCSI (SCSI logging facility).

To enhance your system, enable this feature that allows SCSI to be probed while the system boots up rather than booting and then probing (Asynchronous SCSI scanning). Most systems can perform both tasks at once, so why not allow it? For hardware with many SCSI devices attached, this will increase boot time significantly.

Next, "Parallel SCSI (SPI) Transport Attributes" allows each SCSI device to send transport information to sysfs. Some systems require this feature.

This next feature is the same as above, but sends information about transport from FiberChannel devices (FiberChannel Transport Attributes). FiberChannel devices use SCSI.

Next, the user can enable/disable "SCSI target support for FiberChannel Transport Attributes".

Transport information from iSCSI devices and SAS devices can also be exported to sysfs (iSCSI Transport Attributes) and (SAS Transport Attributes). SAS stands for Serial Attached SCSI.

Next, ATA support is added to libufs (ATA support for libufs (requires libata)). Note that the configuration tools says libata is required. To satisfy this need, enable ATA support. Most likely, the configuration tool has or will do this for you, but double check anyway. libufs and libata are the libraries that provide support for SAS and ATA, respectively.

The next feature permits SAS interfaces to accept SMP frames (Support for SMP interpretation for SAS hosts). This adds a SMP interpreter to libufs. However, this does increase the kernel's size. SMP frames allow all processors on multi-CPU systems to access SAS devices.

SRP can send transport data to sysfs (SRP Transport Attributes). SRP stands for SCSI RDMA Protocol. RDMA stands for Remote Direct Memory Access. This means SRP is a protocol used to access data from remote SCSI devices attached to another computer.

Next, the user can enable "SCSI target support for SRP Transport".

Low-level SCSI drivers can be enabled (SCSI low-level drivers). This provides many basic drivers.

After that, users can enable/disable "PCMCIA SCSI adapter support". This adapter allows SCSI devices to attach to PC cards.

There are some drivers for specific adapters - (Future Domain PCMCIA support), (Qlogic PCMCIA support), and (Symbios 53c500 PCMCIA support).

Devices with multi-path setups will need this feature (SCSI Device Handlers). This is used in clusters where each node needs a direct path to a SCSI storage unit.

Next, the "OSD-Initiator library" can be enabled. This is a SCSI driver that provides patches, the OSD protocol, and the T10 protocol to SCSI devices. OSD stands for Object-based Storage Device; this is discussed more in the next paragraph.

This feature makes a SCSI upper layer for testing and managing /dev/osdx devices (OSD Upper Level driver). exofs uses this driver for mounting OSD-based filesystems. OSD devices are storage devices that do not use blocks like other storage units. Instead, OSD devices store data in containers called

More Linux Related Stuff on: <http://www.linux.org>

objects. exofs was once called OSDFS.

OSD features provide debugging tools if enabled (Compile All OSD modules with lots of DEBUG prints).

Now, we can discuss Serial ATA and Parallel ATA features and drivers. The first feature to enable/disable is for debugging (Verbose ATA error reporting).

Next, users should enable the Advanced Configuration and Power Interface feature for ATA devices (ATA ACPI Support). This will allow the kernel to manage power usage more efficiently in SATA devices.

The kernel contains a driver for "SATA Zero Power Optical Disc Drive (ZPODD) support". This powers off SATA optical disc drives (ODD) when not in use. This saves energy and reduces wear and tear.

TIP: Even when making a high-performance kernel, try to enable all power management features. This reduces power consumption, operating costs, heat production (heat reduces performance), and wear and tear.

SATA port multipliers need this driver (SATA Port Multiplier support). A port multiplier is a device that has numerous ports but only needs to plug into one port itself. For instance, if a piece of hardware has one SATA port, but more are needed, plug the port multiplier into the one port. Now the device has many SATA ports.

Next is a driver for AHCI SATA (AHCI SATA support). Advanced Host Controller Interface (AHCI) is an operation standard for SATA bus adapters.

For AHCI SATA devices to be supported on SoC hardware, this driver must be enabled (Platform AHCI SATA support). SoC stands for System-on-a-Chip.

Next are some drivers for some specific devices.

Initio 162x SATA support

ACard AHCI variant (ATP 8620)

Silicon Image 3124/3132 SATA support

Configuring the kernel Part XIII

Aloha! In this next article of the Linux kernel series, we are still configuring ATA devices and we then move on to logical volumes/storage.

"ATA SFF support (for legacy IDE and PATA)" should be enabled because this extends the abilities of ATA.

To support Pacific Digital ADMA controllers, enable "Pacific Digital ADMA support".

"Pacific Digital Serial ATA QStor support" is offered by this next driver.

Support for the Promise SATA SX4 device is supported by the Linux kernel (Promise SATA SX4 support (Experimental)).

SFF ATA controllers with BMDMA abilities will need this driver (ATA BMDMA support). BMDMA stands for Bus-Master Direct Memory Access.

Next, this driver provides support for various Intel SATA and PATA controllers (Intel ESB, ICH, PIIX3, PIIX4 PATA/SATA support).

Here is another device-specific driver (Calxeda Highbank SATA support) and another (Marvell SATA support) and another (NVIDIA SATA support) and another (Promise SATA TX2/TX4 support) and another (Silicon Image SATA support) and several more (SiS 964/965/966/180 SATA support), (ServerWorks Frodo / Apple K2 SATA support), (ULi Electronics SATA support), (VIA SATA support), Because many of the SATA/PATA controllers are designed differently, a generic driver cannot be used on many of these devices.

Next, ATA devices on PC Cards are supported by this driver unless a device-specific driver manages the hardware (PCMCIA PATA support).

After that, here is a generic PATA driver that manages all other PATA devices not supported by previously enabled drivers (Generic platform device PATA support).

The power consumption of PATA devices are managed by this ACPI driver (ACPI firmware driver for PATA). It is highly recommended that ACPI be enabled for all hardware on the system. Although this increases the kernel's size, ACPI is better for performance.

"Generic ATA support" is offered by this driver.

Legacy support for ISA, VLB, and PCI bus PATA devices can be added via this driver (Legacy ISA PATA support (Experimental)). This legacy support uses the new ATA layer.

This set of features contains many abilities for RAID and LVM as seen in the following feature options (Multiple devices driver support (RAID and LVM)).

INTERESTING FACT: The kernel is written in C and Assembly.

This driver allows RAID and LVM to be combined (RAID support). This is used to make several

More Linux Related Stuff on: <http://www.linux.org>

LVMs use RAID. Partitions are combined into Logical Block Devices which are then used to form RAID devices.

Many users will want RAID devices to be detected during booting (Autodetect RAID arrays during kernel boot). If you do not have RAID, then do not enable this feature. Otherwise, the boot process will take a few seconds longer than what it needs to be.

NOTE: When configuring the Linux kernel, it is best to follow the "use it or lose it" principal. That is, if you will not use it, then disable the feature.

Hard-drive partitions can be added together using this driver (Linear (append) mode).

The following driver adds RAID-0 support to logical block devices (RAID-0 (striping) mode). Then, there is (RAID-1 (mirroring) mode), (RAID-10 (mirrored striping) mode), and (RAID-4/RAID-5/RAID-6 mode).

The MD framework needs multipath support (Multipath I/O support). The MD framework is a Multi Device framework which manages multiple devices as one unit. For example, combining the partitions of many storage units will make multiple devices act as one. The multipath support is for handling a virtual "single device" with many addresses. Because the single storage unit is physically more than one piece of hardware, it will have more than one hardware address.

These larger multi-disk storage units can be tested for bugs using this debugging driver (Faulty test module for MD).

The "Device mapper support" is a volume manager that is used to map logical sectors. LVM uses sector mapping.

The device mapper can have debugging features if enabled (Device mapper debugging support).

These logical devices can be set to encrypt data if desired (Crypt target support). This feature can allow users to later encrypt such storage devices.

Snapshots of these logical storage units can only be taken if this feature is enabled (Snapshot target).

Thin provisioning allows logical volumes to be setup to have more storage capacity than the physical storage units that make up the logical volume (Thin provisioning target). This feature also provides snapshot abilities of such storage. This excess virtual data space cannot immediately be used. The point of it is to allow users to add a physical storage unit in the future and save time configuring the logical block device.

Thin provisioning can be debugged with this feature (Keep stack trace of thin provisioning block lock holders).

The performance of block devices can be improved by moving the more commonly used data to the faster storage units (Cache target (EXPERIMENTAL)).

Volume managers can be made to mirror logical volumes (Mirror target).

Device-mapper (dm) targets can be made to support the mappings of RAID1, RAID10, RAID4, RAID5 and RAID6 (RAID 1/4/5/6/10 target).

The device-mapper logs can be mirrored in userspace (Mirror userspace logging).

A "Zero target" is a device that disregards writes and returns reads as zero.

Next, volume managers should have multipath support for the hardware (Multipath target).

This driver finds the most efficient path to storage devices for reads and writes (I/O Path Selector based on the number of in-flight I/Os).

The next driver is the same as above, but finds the fastest path (I/O Path Selector based on the service time).

If a physical storage unit on a logical volume is busy, this feature will allow reads/writes to go to another physical volume if possible (I/O delaying target).

udev can generate events for device-manager actions (DM uevents). udev is a device manager for /dev/.

To test how software/hardware reacts to logical devices that occasionally fail input/output tasks, enable this debugging feature (Flakey target).

A logical volume can be created that is a read-only storage unit that validates data for another logical partition (Verity target support).

NOTE: If you enjoy my articles, click "Like" on my posts if you have a Linux.org account. Also, reshare this article on Google, Twitter, and/or Facebook.

ConfigFS and the TCM storage engine can be enabled with this setting (Generic Target Core Mod (TCM) and ConfigFS Infrastructure). ConfigFS is a RAM-based filesystem.

INTERESTING FACT: The Linux kernel does not have a “main()” function. In programs, main() is called by libc which depends on the kernel. The kernel cannot have a main() function because libc would not be able to start the kernel. If the kernel did have a main() function, then we would have a “chicken-or-the-egg” problem – which would come first? Plus, the entry point of the kernel is written in Assembly which does not use the main() function.

Next, the "TCM/IBLOCK Subsystem Plugin for Linux/BLOCK" can be enabled/disabled.

Then, the "TCM/FILEIO Subsystem Plugin for Linux/VFS" can be enabled/disabled.

And again, there are two other TCM features - (TCM/pSCSI Subsystem Plugin for Linux/SCSI) and (TCM Virtual SAS target and Linux/SCSI LDD fabric loopback module).

The Linux-iSCSI.org iSCSI Target Mode Stack for ConfigFS is provided by this driver (Linux-iSCSI.org iSCSI Target Mode Stack).

Next, enable/disable the "FireWire SBP-2 fabric module". This allows a computer to connect to another computer and appear to be a hard-drive.

After that, we can configure "Fusion Message Passing Technology (MPT) device support".

The first option under that heading is a driver for SCSI support for parallel adapters (Fusion MPT ScsiHost drivers for SPI).

SCSI can also get support for fiber channel host adapters (Fusion MPT ScsiHost drivers for FC) and/or SAS adapters (Fusion MPT ScsiHost drivers for SAS).

Next, users can set the "Maximum number of scatter gather entries". A lower number leads to reduced memory consumption per controller instance.

The next driver offers system ioctl calls to manage MPT adapters (Fusion MPT misc device (ioctl) driver).

Fiber channel ports can support IP LAN traffic with this driver (Fusion MPT LAN driver).

Configuring the kernel Part XIV

Aloha! Ready for another cool article on the Linux kernel?

Next, in our long quest, we can enable/disable the "Fusion MPT logging facility". MPT stands for Message Passing Technology. The Fusion driver is made by LSI Logic Corporation. MPT is a specific messaging tactic used between process. This technique is synchronous meaning that process will wait for messages as needed.

After that, the "FireWire driver stack" should be enabled if the computer possesses FireWire ports. If not, then there is no point in enabling FireWire ports for a device that could not use such drivers. FireWire is a lot like USB. The differences lie in the protocols, speed, and the physical shape and layout of the ports. Generally, Apple devices use FireWire and USB. Some PCs have FireWire ports, but this is less common than USB ports.

Some FireWire controllers use the OHCI-1394 specification (OHCI-1394 controllers). If so, enable this driver.

To use FireWire storage devices, enable this next driver (Storage devices (SBP-2 protocol)). This driver provides a protocol that the FireWire storage units use to communicate with the FireWire bus (the card with the attached FireWire ports). Some FireWire scanners also need this driver.

IPv4 can be used on FireWire ports (IP networking over 1394). IEEE 1394 or simply "1394" is FireWire. Multicasting over FireWire using IPv4 has limitations.

Nosy is the traffic monitor for FireWire PCILynx cards (Nosy - a FireWire traffic sniffer for PCILynx cards).

Next, the I2O devices can be supported (I2O device support). Intelligent Input/Output (I2O) bus uses drivers for the hardware and operating system levels. The hardware drivers (HDM) are not specific to any operating system while the OS drivers (OSM) must be used on the intended operating system. The OSM communicates with any of the HDMs. I2O cards/buses have an IOP - Input/Output Processor. This speeds up the system since the main CPU is processing less data.

Only enable "Enable LCT notification" on systems that lack a SUN I2O controller. I2C SUN firmware does not support LCT notifications.

The Adaptec I2O controllers need this next driver if RAID is intended (Enable Adaptec extensions).

Direct memory access of 64-bits can be allowed on Adaptec I2O controllers (Enable 64-bit DMA).

I2O devices can be configured if permitted (I2O Configuration support). This feature is mainly needed for RAID setup.

The old Input/Output controls can be enabled for I2O devices (Enable ioctls (OBSOLETE)).

The OSM software for I2O bus adapters can be enabled (I2O Bus Adapter OSM). This set of OSMs are used to find new I2O devices on the other end of the adapter.

Next, the OSMs for I2O block devices can be enabled (I2O Block OSM). RAID controllers on I2O hardware will need this OSM.

The following OSM is for the SCSI or FiberChannel devices on I2O controllers (I2O SCSI OSM).

Information on I2O devices can be read from /proc/ if enabled (I2O /proc support).

After the I2O features have been enabled/disabled, we can move on to other kernel abilities. Next, we have "Macintosh device drivers". These are only useful to Apple devices. Linux kernels for PCs should not have any of these features enabled. However, as with many statements there is always an exception. Some PC users may use an Apple mouse, keyboard, and/or some other Apple device. Again, it is best to thoroughly understand the need and use of the kernel being developed.

Next, we have a driver for networking (Network device support). X11 and other Linux software do not depend on this driver, so if the kernel will not need to connect to another computer, Internet, Intranet, or network, then this can safely be disabled.

The following driver is like above, but is specific to core drivers (Network core driver support).

Etherchannels are supported by this driver (Bonding driver support). In networking, bonding is the fusion of two or more Ethernet channels. This is also known as trunking.

A dummy network can be set up in Linux with this driver (Dummy net driver support). A dummy network is a virtual network that is like the /dev/null of networking. Any data sent to this dummy network is gone forever as it would be going to /dev/null. The IP address is not set. The user can define their networking equivalent of /dev/null.

Next, EQL can be supported (EQL (serial line load balancing) support). This is needed to allow two computers to communicate via two serial connections that use the SLIP or PPP protocol.

Fiber Channel is a fast serial protocol for connecting storage devices to computers (Fibre Channel driver support).

The MII transceiver needs this driver (Generic Media Independent Interface device support). MII is an interface for Ethernet with speeds up to 100Mbit/s. These Ethernet cables are intended for connecting to a PHYceiver which is an Ethernet transceiver.

To group many Ethernet devices via virtual interfaces, the "Ethernet team driver support" is needed.

"MAC-VLAN support" permits users to have packet maps between a particular MAC address and a certain interface.

A tap character device can be made from the MAC-VLAN network interface (MAC-VLAN based tap driver). Tap devices get packets from the kernel so they can be sent elsewhere.

The next feature allows virtual vxlan interfaces that create Layer 2 networks to be placed above Layer 3 (Virtual eXtensible Local Area Network (VXLAN)). This is commonly used for tunneling to virtual systems.

Messages that the kernel sends on a network can be logged with this feature (Network console logging support). Only enable this if it is important for you to log the network messages. Disabling this feature will enhance performance.

This feature allows various parameters to be changed (Dynamic reconfiguration of logging targets). Some of these parameters include port numbers, MAC addresses, IP addresses, and some other settings.

If user space programs are expected to use TAP devices, then enable this driver that will allow such activity (Universal TUN/TAP device driver support).

This driver is for local Ethernet tunnels (Virtual ethernet pair device).

The "Virtio network driver" is used with QEMU, Xen, KVM, and other virtual machines.

Next, "ARCnet support" can be enabled. ARCnet is a Token-Ring-like Local-Area-Network (LAN) protocol. ARCnet stands for "Attached Resource Computer Network".

Now, we can move on to "ATM drivers". ATM stands for "Asynchronous Transfer Mode". ATM is used for telecommunications.

The Marvell Ethernet switch needs this driver (Marvell 88E6060 ethernet switch chip support). Also, the chip for such a switch is also needed depending on the model (Marvell 88E6085/6095/6095F/6131 ethernet switch chip support) and (Marvell 88E6123/6161/6165 ethernet switch chip support).

Now, we can learn about "Ethernet driver support".

First, we can enable/disable "3Com devices". This then allows kernel developers to choose which 3Com devices to support.

The next set of options is for various "Adaptec devices" and then later "Alteon devices". These are just device/vendor specific drivers. Generally, these drivers are added as modules.

After those two sets of options, there is a set for "AMD devices" and again for "Atheros devices".

NOTE: Keep in mind what type of hardware the kernel will run on. For a large variety of devices, it may be best to add these devices as modules.

There are various other vendor-specific device drivers - "Cadence devices", "Broadcom devices", "Brocade devices", "Chelsio devices", "Cisco devices", and "Digital Equipment devices". Some other device/vendor specific drivers follow.

The next driver that is not vendor/device specific is "SLIP (serial line) support". This driver supports SLIP and CSLIP. SLIP (Serial Line Internet Protocol) is an Internet protocol for modems and serial ports. PPP is now used instead of SLIP. CSLIP is Compressed SLIP.

Next, "CSLIP compressed headers" can be enabled which compresses the TCP/IP headers. CSLIP is faster than SLIP, but to be able to use CSLIP, both the transmitting computer and the receiving computer must be able to understand CSLIP.

When using SLIP on bad analogue lines, then it may be a good idea to enable "Keepalive and linefill" which will help to keep the connection alive.

For running IP on poor-quality networks or are 7-bit, it may help to enable and use "Six bit SLIP encapsulation".

Now, we can move on to the popular USB system, but these are the USB drivers for networking.

The first USB network device to enable/disable is "USB CATC NetMate-based Ethernet device support". This is for 10Mbps USB Ethernet EL1210A chip devices. The USB device will act and appear to be an Ethernet device even though the hardware is USB.

Next, this driver is the same as above except this is for devices with the KLSI KL5KUSB101B chipset (USB KLSI KL5USB101-based ethernet device support).

The Pegasus USB devices are USB-to-Ethernet adapters/converters (USB Pegasus/Pegasus-II based ethernet device support).

Next, there is another USB-to-Ethernet driver (USB RTL8150 based ethernet device support).

Configuring the kernel Part XV

Aloha! In this article of the Linux kernel series, we are still configuring drivers for USB networking. Then, we will move on to input devices.

First, we can enable/disable the "Multi-purpose USB Networking Framework" which allows connecting laptops to desktop systems.

Next, the ASIX USB-to-Ethernet adapter driver can be enabled/disabled (ASIX AX88xxx Based USB 2.0 Ethernet Adapters).

Then, there is another ASIX adaptor driver (ASIX AX88179/178A USB 3.0/2.0 to Gigabit Ethernet).



NOTE: Generally, it is best to add adapter drivers as modules.

The Communication Device Class specification is offered in the driver (CDC Ethernet support (smart devices such as cable modems)). This specification is for USB modems. The Linux system recognizes this USB networking interface as an Ethernet interface and will be designated as "ethX" where "X" is the Ethernet device number.

Next is a specification similar to the above (CDC EEM support). CDC EEM stands for Communication Device Class Ethernet Emulation Model.

The CDC Network Control Model (NCM) also has a driver that offers the specification (CDC NCM support).

The driver providing the CDC MBIM (Mobile Broadband Interface Model) specification is also available for the Linux kernel (CDC MBIM support).

Next, there are several vendor/device specific drivers for various USB networking devices and chipsets.

After those, there is a generic driver for USB network devices that do not require any special drivers (Simple USB Network Links (CDC Ethernet subset)).

Again, there are some more drivers for device/vendor specific devices.

FUN FACT: Linux was used to make the special effects for the movie "Titanic" by James Cameron. "CDC Phonet support" is for USB Nokia modems that use Phonet.



Now, we can move on to Wireless LAN drivers which use the 802.11 specification. Mainly, there is a list of vendor/device specific drivers.

"SoftLED Support" controls the LEDs that are associated with the Wifi cards/devices. Many people forget to enable this and then wonder why the LED will not turn on. So, remember this.

Some chipsets support SDIO as seen by this driver (Atheros ath6kl SDIO support). SDIO is an extension of the Secure Digital specification for wireless SD cards. SDIO stands for Secure Digital Input/Output.



Kernel developers may also notice that some wireless devices can support QoS. QoS stands for Quality

More Linux Related Stuff on: <http://www.linux.org>

of Service. This feature gives network transmissions priority. Assume two sets of data need to be sent over a network. Only one can go first. QoS will send the most important data first.

FUN FACT: Technically, Linux is not an operating system. Linux is the kernel while GNU/Linux is the operating system.

The "Generic HDLC layer" is needed for WAN cards. HDLC stands for High-Level Data Link Control. This is a data link layer protocol.

Raw HDLC can be used with the "Raw HDLC support" driver enabled. The "Raw HDLC Ethernet device support" driver allows the HDLC layer to emulate Ethernet. The cHDLC driver offers a HDLC extension also called Cisco HDLC (Cisco HDLC support).

The Linux kernel also has a driver for "Frame Relay support" for HDLC. Frame Relay is a Layer 2 protocol.

HDLC can also support PPP (Synchronous Point-to-Point Protocol (PPP) support) and X.25 (X.25 protocol support).

Next, this driver offers Frame Relay the ability to use DLCI (Frame Relay DLCI support).

The "LAPB over Ethernet driver" creates a device file that permits the user to make a LAPB point-to-point connection to another computer via Ethernet. The device file is usually /dev/lapb0 for the first of such device.

X.25 frames can be sent over telephone lines with this driver (X.25 async driver). Specifically, this driver allows X.25 to use asynchronous serial lines.

A special driver is needed for ISA SBNI12-xx cards (Granch SBNI12 Leased Line adapter support). These cards are inexpensive substitutes for leased line modems.

The next driver allows parallel connections to carry scheduled traffic (Multiple line feature support). This allows the Linux system to more efficiently manage parallel connection on SBNI12 adapters. Some Linux users claim this driver doubles their speed. However, I have never tested this to know for myself.

Next, "IEEE 802.15.4 drivers" can be configured. This is for slow WAN devices. This is a standard that controls the media and physical layers of the wireless network. This specification uses different frequencies in different continents. For example, in Europe, such wireless devices will use the 868.0-868.6MHz frequency.

The first setting in this category is for a fake LR-WPAN driver (Fake LR-WPAN driver with several interconnected devices). LR-WPAN stand for Low-Rate Wireless Personal Area Network.

FUN FACT: Only about 2% of the current kernel was written by Linus Torvalds.

The vmxnet3 virtual Ethernet used by VMware requires this driver (VMware VMXNET3 ethernet driver). When making a kernel for a large number of users, it is best to enable this as a module because

More Linux Related Stuff on: <http://www.linux.org>

someone may want to use the Ethernet on VMWare.

The Hyper-V virtual network needs this driver (Microsoft Hyper-V virtual network driver). You may be wondering if this is the same Hyper-V virtual network by Microsoft. It is and yes, Linux supports Hyper-V.

The digital telephone service, ISDN, is supported by this driver (ISDN support). ISDN stands for Integrated Services Digital Network. In France, ISDN is known as RNIS which stands for Réseau numérique à intégration de services. With an ISDN adapter, a computer can start and accept voice calls. This allows computers to be used as answering machines or some other telephone service device. ISDN can also carry video information.

Now, we can move on to input devices (Input device support). These are devices that give the computer information. Mice and keyboards are the most commonly used and known input devices. Scanners are another example of input devices.

First, there is a driver that supports various haptic/tactile feed-back devices (Support for memoryless force-feedback devices). For instance, many game controllers vibrate which is haptic/tactile feed-back.

Some input devices check on the status of the hardware (Polled input device skeleton). Such behavior requires this driver.

Input devices that use sparse keymaps need this driver (Sparse keymap support library). A keymap is the layout information for keyboards.

Next, this is another keymap (Matrix keymap support library).

NOTE: When making a kernel for a broad user group, include most or all input devices as modules because the developer usually does not know what type of devices the users may plugin to the computer.

FUN FACT: A Vanilla Kernel is a Linux kernel in its original, unchanged state. For example, if you download a kernel from Kernel.org, then you have just downloaded a Vanilla Kernel.

The "Mouse interface" makes two different device files for the mouse. The two device files are /dev/input/mouseX and /dev/input/mice.

This next driver makes a psaux device file that is an alias to /dev/input/mice (Provide legacy /dev/psaux device). The psaux device file is /dev/psaux.

If the system has a digitizer, then the horizontal resolution needs to be set (Horizontal screen resolution) and then the vertical resolution (Vertical screen resolution). A digitizer is the type of touch-screen that supports touch-pens that allow users to draw. Other touch-screens cannot support such complex input.

The next driver supports joysticks and gamepads (Joystick interface). This driver creates the /dev/input/jsX files.

The "Event interface" driver allows input devices to be accessible via /dev/input/eventX.

More Linux Related Stuff on: <http://www.linux.org>

The "Event debugging" driver outputs all input events to the system's log. Do not enable this for any reason other than debugging systems. Obviously, for performance reasons, but the main reason I recommend this be disabled is for security purposes. All key-presses are plainly logged including passwords.

Next, a list of various keyboard drivers are listed for configuration (Keyboards) followed by mouse drivers (Mice) and then joystick/gamepad drivers (Joysticks/Gamepads).



After that, various drivers for specific tablet hardware/vendors are listed (Tablets). After that is the driver list for "Touchscreens".

The last set of input device drivers is a list of miscellaneous drivers for specific hardware and vendors (Miscellaneous devices).

The next article of this series will discuss input ports. Do not forget to read the other articles of this series and this website. Mahalo!

NOTE: The images in this article are externally linked to their source. Most or all of these images are copyrighted. These images are on the server of their copyright-holder or person with legal permission.

Configuring the kernel Part XVI

In this article, we will discuss the input/output ports.

First, the "i8042 PC Keyboard controller" driver is needed for PS/2 mice and AT keyboards. Before USB, mice and keyboards used PS/2 ports which are circular ports. The AT keyboard is an 84-key IBM keyboard that uses the AT port. The AT port has five pins while the PS/2 port has six pins.

Input devices that use the COM port (sometime called RS232 serial port) will need this diver (Serial port line discipline). The COM port is a serial port meaning that one bit at a time is transferred.

The TravelMate notebooks need this special driver to use a mouse attached to the QuickPort (ct82c710 Aux port controller).

Parallel port adapters for PS/2 mice, AT keyboards, and XT keyboards use this driver (Parallel port keyboard adapter).

The "PS/2 driver library" is for PS/2 mice and AT keyboards.

"Raw access to serio ports" can be enabled to allow device files to be used as character devices.

Next, there is a driver for the "Altera UP PS/2 controller".

The PS/2 multiplexer also needs a driver (TQC PS/2 multiplexer).

The ARC FPGA platform needs special driver for PS/2 controllers (ARC PS/2 support).

NOTE: I want to make it clear that the PS/2 controllers that are discussed in this article are not Sony's game controllers for their PlayStation. This article is discussing the 6-pin mouse/keyboard ports. The controller is the card that holds the PS/2 ports.

The "Gameport support" driver offers support for the 15-pin gameport. Gameport was the 15-pin port used by many input gaming devices until the invention of the USB port.

The next driver is for gameports on ISA and PnP bus cards (Classic ISA and PnP gameport support). ISA stands for Industry Standard Architecture and was a parallel bus standard before PCI. PnP stands for Plug-and-Play and was a common standard before ISA.

"PDPI Lightning 4 gamecard support" provides a driver for a proprietary gamecard with gameports.

The SoundBlaster Audigy card is a proprietary gameport card (SB Live and Audigy gameport support).

The ForteMedia FM801 PCI audio controller has a gameport on the card (ForteMedia FM801 gameport support). This driver only supports the gameport.

Next, we can move on to "Character devices". Character devices transfer data character by character.

First, TTY can be enabled or disabled (Enable TTY). Removing TTY will save a lot of space, but TTY

More Linux Related Stuff on: <http://www.linux.org>

is needed for terminals and such. Unless you know what you are doing, do not disable TTY.

NOTE TO MY FANS: If you know of a reason for disabling TTY, could you post the answer below and share with us. Mahalo!

Next, support for "Virtual terminals" can be enabled/disabled. Again, a lot of space can be saved, but virtual terminals are very important.

This next driver supports font mapping and Unicode translation (Enable character translations in console). This can be used to convert ASCII to Unicode.

Virtual terminals can be used as system consoles with this driver (Support for console on virtual terminal). A system console manages the logins and kernel messages/warnings.

Virtual terminals must channel through a console driver to interact with the physical terminal (Support for binding and unbinding console drivers). Before the virtual terminal can do so, the console driver must be loaded. When the virtual terminal is closed, the console terminal must be unloaded.

The next driver provides support for Unix98 PTY (Unix98 PTY support). This is Unix98 pseudo terminal.

FUN FACT: The Linux kernel allows a filesystem to be mounted many times in many places at once.

Next, "Support multiple instances of devpts" can be supported. The devpts filesystem is for pseudo-terminal slaves.

Legacy support for PTY can also be enabled (Legacy (BSD) PTY support).

The max amount of legacy PTYs in use can be set (Maximum number of legacy PTY in use).

The next driver can be used to offer support to serial boards that the other drivers fail to support (Non-standard serial port support).

Next, there are some drivers for specific boards and cards.

The GSM MUX protocol is supported with this driver (GSM MUX line discipline support (EXPERIMENTAL)).

The next driver enables the kmem device file (/dev/kmem virtual device support). kmem is usually used for kernel debugging. kmem can be used to read certain kernel variables and states.

The Stallion cards have many serial ports on them (Stallion multiport serial support). This driver specifically supports this card.

Next, we can move on to drivers for serial devices. As stated before, serial devices transfer one bit at a time.

The first driver is for standard serial port support (8250/16550 and compatible serial support).

Plug-and-Play also exists for serial ports with this driver (8250/16550 PNP device support).

The following driver allows the serial ports to be used for connecting a terminal to be used as a console (Console on 8250/16550 and compatible serial port).

Some UART controllers support Direct Memory Access (DMA support for 16550 compatible UART controllers). UART stands for Universal Asynchronous Receiver/Transmitter. UART controllers convert serial to parallel and vice versa.

Next, this driver offers support for standard PCI serial devices (8250/16550 PCI device support).

16-bit PCMCIA serial devices are supported by this driver (8250/16550 PCMCIA device support). Remember, PCMCIA is a PC-card that is usually used in laptops.

The maximum number of supported serial ports can be set (Maximum number of 8250/16550 serial ports) and then the maximum that are registered during boot-up (Number of 8250/16550 serial ports to register at runtime).

For extended serial abilities like HUB6 support, enable this driver (Extended 8250/16550 serial driver options).

A special driver is needed to support more than four legacy serial ports (Support more than 4 legacy serial ports).

Serial interrupts can be shared when this driver is used (Support for sharing serial interrupts).

Serial port IRQs can be autodetected using this driver (Autodetect IRQ on standard ports).

RSA serial ports are also supported by the Linux kernel (Support RSA serial ports). RSA stands for Remote Supervisor Adapter. RSA is an IBM-specific hardware.

Next, there are various vendor/device specific drivers.

This is a TTY driver that uses printk to output user messages (TTY driver to output user messages via printk). Printk (print kernel) is a special piece of software that usually prints the boot-up messages. Any string that is displayed by printk is usually put in the /var/log/messages file. The shell command "dmesg" displays all strings that were used by printk.

Next, we can enable/disable support for parallel printers (Parallel printer support).

The next driver allows a printer to be used as a console (Support for console on line printer). This means kernel messages will be literally printed at the printer. Normally when the word "print" was used in this article series, it meant putting data on the screen. This time, this literally means putting the data on paper. The following driver makes the device files at /dev/parport/ (Support for user-space parallel port device drivers). This allows some processes to access.

Again, the Linux kernel has many features and drivers, so we will discuss more drivers in the next article. Mahalo!

Configuring the kernel Part XVII

Aloha! This article will discuss various drivers.

First, the "virtio console" is a virtual console that is used with hypervisors.

The "IPMI top-level message handler" is a message manager for the IPMI system. IPMI stands for Intelligent Platform Management Interface. IPMI is an interface for managing the system via network without using a shell.

"/dev/nvram support" permits the system to read and write memory in the real time clock's memory. Generally, this feature is used for saving data during a power loss.

The next driver supports the Siemens R3964 packet protocol (Siemens R3964 line discipline). This is a device-to-device protocol.

Now, we can move on to PCMCIA character devices. However, most of the drivers here are vendor/device specific.

The RAW driver allows block devices to be bound to the device files /dev/raw/rawN (RAW driver (/dev/raw/rawN)). The advantage to this is efficient zero-copy. However, most software will still prefer to access the storage through /dev/sd** or /dev/hd**.

Next, the maximum number of RAW devices can be supported is set.

The following driver makes the device file /dev/hpet (HPET - High Precision Event Timer).

NOTE: Many of you may be wondering why enabling these device file matter. Well, these device files serve as an interface between the software and hardware.

The HPET timers can be mapped with this driver (Allow mmap of HPET). Mapping is the process of making a list of address in memory of devices and files. The files can then be found faster by getting the address from the memory and then commanding the hard-drive to get the data from the address.

The "Hangcheck timer" is used to detect whether of not the system has locked-up. This timer watches for locked-up processes. As soon as a process freezes, a timer starts. After the timer goes off, if the process has not restarted or closed, then the timer will force the process to close.

Linus Torvalds Quote: Portability is for people who cannot write new programs.

The TPM security chip that uses Trusted Computing Group's specification will need this driver (TPM Hardware Support).

Now, we can move on to I2C devices. I2C stands for Inter-Integrated Circuit and is spoken as "eye two see". However, some people say "eye squared see". I2C is a serial bus standard.

Some old software used I2C adapters as class devices, but software now does not do that (Enable compatibility bits for old user-space). So, this driver will offer backwards compatibility for older software.

More Linux Related Stuff on: <http://www.linux.org>

Next, the I2C device files can be made (I2C device interface).

I2C can support multiplexing with this driver (I2C bus multiplexing support).

I2C can support GPIO-controlled multiplexing with this driver (GPIO-based I2C multiplexer).

Various tests can be performed on I2C and SMBus with this driver for developers (I2C/SMBus Test Stub).

The I2C system will produce debugging messages with this feature enabled (I2C Core debugging messages).

The next driver produces additional I2C debugging messages (I2C Algorithm debugging messages).

Linus Torvalds Quote: The main reason there are no raw devices [in Linux] is that I personally think that raw devices are a stupid idea.

The following driver will cause the I2C drivers to produce debugging messages (I2C Bus debugging messages).

Next, we have Serial Peripheral Interface support (SPI support). SPI is a synchronous serial protocol used on SPI buses.

After that, there is a driver for High speed synchronous Serial Interface support (HSI support). HSI is a synchronous serial protocol.

PPS can also be supported by the Linux kernel (PPS support).

The "IP-over-InfiniBand" driver allows IP packets to be transported over InfiniBand.

After that, there is a debugging driver for IP-over-InfiniBand (IP-over-InfiniBand debugging).

SCSI's RDMA protocol can also travel over InfiniBand (InfiniBand SCSI RDMA Protocol).

There is also an extension for the iSCSI protocol to transmit over InfiniBand (iSCSI Extensions for RDMA (iSER)).

Sometimes, errors occur in the core system that the whole system must know (EDAC (Error Detection And Correction) reporting). This driver sends the core errors to the system. Generally, such low-level errors are reported in the processor and then seen by this driver to let other system processes know about or handle the error.

This driver provides legacy support for EDAC to use older versions of sysfs (EDAC legacy sysfs).

EDAC can be set to send debugging information to the logging system of Linux (Debugging).

Linus Torvalds Quote: Nobody actually creates perfect code the first time around, except me.

The Machine Check Exceptions (MCEs) are converted to a readable form via this driver (Decode MCEs in human-readable form (only on AMD for now)).

MCEs are hardware errors detected by the CPU. MCEs usually trigger kernel panics.

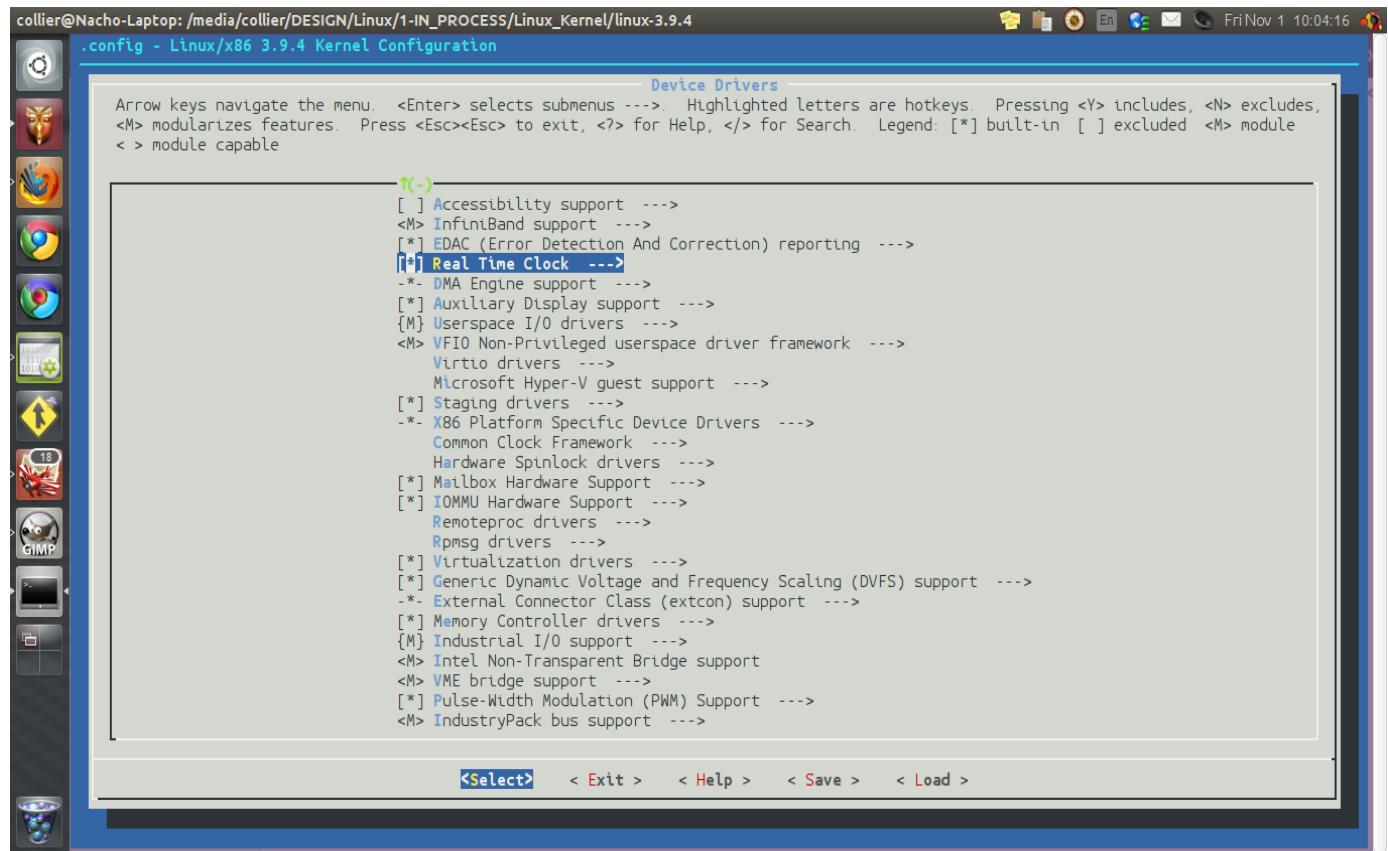
The decoding process for MCE to a readable form can be injected to test error handling (Simple MCE injection interface over /sysfs).

The next driver allows errors to be detected in memory and then corrected (Main Memory EDAC (Error Detection And Correction) reporting).

Next, there are many drivers that detect and correct errors on specific hardware sets.

Linus Torvalds Quote: Theory and practice sometimes clash. And when that happens, theory loses. Every single time.

Now, we can move on to the "Real Time Clock". This is commonly abbreviated "RTC". The RTC keeps track of time.



The next setting allows us to make the Linux system use the time from the RTC as the time on the "wall clock" (Set system time from RTC on startup and resume). The wall clock is the clock on the desktop or the time seen using the "date" command.

Alternately, the wall clock can get the time from an NTP server and then sync with the RTC (Set the RTC time based on NTP synchronization).

Some systems have more than one RTC, so the user must set which is the default (RTC used to set the system time).

It is best to make the first one (/dev/rtc0) the primary clock.

Debugging abilities can be set for the RTC system (RTC debug support).

The RTC can use various interfaces for giving the operating system the current time. Using sysfs will require this driver (/sys/class/rtc/rtcN (sysfs)) while using proc will require this driver (/proc/driver/rtc (procfs for rtcN)). Special RTC character devices can be made and used (/dev/rtcN (character devices)). The shell command "hwclock" uses /dev/rtc, so the RTC character devices are needed.

The next driver allows interrupts of the RTC to be emulated on the /dev/ interface (RTC UIE emulation on dev interface). This driver reads the clock time and allows the new time to be retrieved from /dev/.

The RTC system can be tested with the test driver (Test driver/device).

Next, we will discuss the Direct Memory Access system.

DMA is the process of hardware accessing the memory independently of the processor. DMA increases system performance because the processor will have less to do if the hardware is performing more tasks for itself. Otherwise, the hardware would be waiting for the processor to complete the task.

The debugging engine is for debugging the DMA system (DMA Engine debugging).

Next, there are many vendor/device specific drivers for DMA support.

Some DMA controllers support big endian reading and writing with this driver (Use big endian I/O register access).

Big endian refers to the arrangement of the binary code. The number system used in English speaking countries places the largest end of the number on the left. For example, in the number 17, the most left numbers place is the tens place which is larger than the ones place. In big endian, each byte is arranged with the largest portion on the left. A byte is eight bits. Example: 10110100. Each place has a value of 128, 64, 32, 16, 8, 4, 2, and 1 respectively. So the byte mentioned converts to the decimal number 180.

The DMA system can use the network to reduce CPU usage (Network: TCP receive copy offload).

The "DMA Test Client" is used for testing the DMA system.

Configuring the kernel XVIII

Aloha! Ready for the next article? In this article, we will discuss the auxiliary-screen. The auxiliary displays are small LCD screens; most are equal to or less than 128x64. Then, we will discuss Userspace IO drivers, some virtual drivers, Hyper-V, staging drivers, IOMMU, and other kernel features.

The first driver to configure for the auxiliary display is the "KS0108 LCD Controller" driver. The KS0108 LCD Controller is a graphics controller made by Samsung.

Next, the parallel port address for the LCD can be set (Parallel port where the LCD is connected). The first port address is 0x378, the next is 0x278 and the third is 0x3BC. These are not the only choices of addresses. The majority of people will not need to change this. The shell command "cat /proc/ioports" will list the available parallel ports and the addresses.

The kernel developer can set the writing delay of the KS0108 LCD Controller to the parallel port (Delay between each control writing (microseconds)). The default value is almost always correct, so this typically does not need to be changed.

The "CFAG12864B LCD" screen is a 128x64, two-color LCD screen. This screen relies on the KS0108 LCD Controller.

The refresh rate of these LCD screens can be changed (Refresh rate (hertz)). Generally, a higher refresh rate causes more CPU activity. This means slower systems will need a smaller refresh rate.

After the auxiliary displays are configured, the "Userspace I/O drivers" are then set. The userspace system allows the user's applications and processes to access kernel interrupts and memory addresses. With this enabled, some drivers will be placed in the userspace.

The "generic Hilscher CIF Card driver" is a userspace driver for Profibus cards and Hilscher CIF Cards.

The "Userspace I/O platform driver" creates a general system for drivers to be in the userspace.

The next driver is the same as above, but adds IRQ handling (Userspace I/O platform driver with generic IRQ handling).

The following driver is again like the one before, but with dynamic memory abilities added (Userspace platform driver with generic irq and dynamic memory).

Next, some vendor/device specific drivers are available.

Then, there is a generic PCI/PCIe card driver (Generic driver for PCI 2.3 and PCI Express cards).

The following driver is for "VFIO support for PCI devices". VFIO stands for Virtual Function Input/Output. VFIO allows devices to directly access userspace in a secure fashion.

The "VFIO PCI support for VGA devices" allows VGA to be supported by PCI through VFIO.

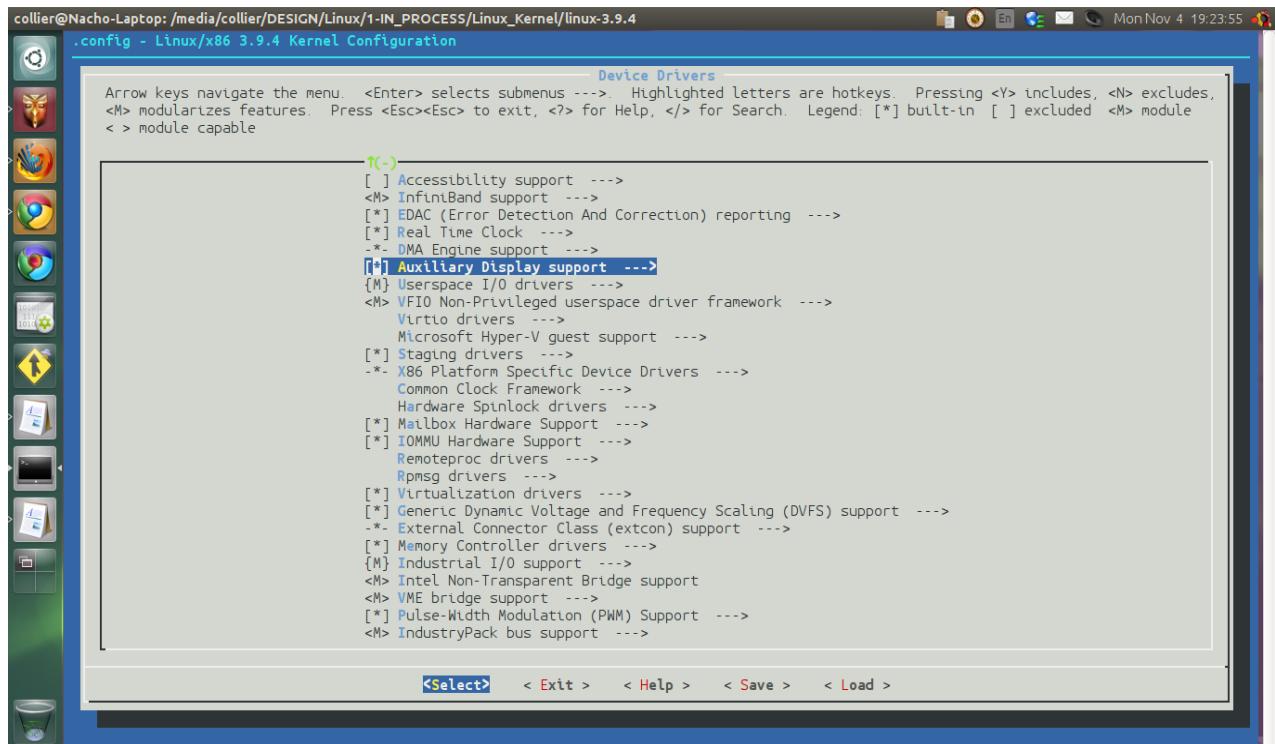
Next, are virtio drivers. Virtio is a IO virtualization platform. This virtual software is for operating system virtualization. This is required for running an operating system in a virtual machine on the Linux system.

The first virtio driver we can configure is the "PCI driver for virtio devices". This allows virtual access to PCI.

The "Virtio balloon driver" allows the memory owned by a virtual system to be expanded or decreased as needed. Generally, no one wants a virtual system to reserve memory it may never use when the host operating system needs the memory.

The following driver supports memory mapped virtio devices (Platform bus driver for memory mapped virtio devices).

If the Linux kernel being configured is intended to run on a Microsoft Hyper-V system, then enable this driver (Microsoft Hyper-V client drivers). This would allow Linux to be the guest/client system on Hyper-V.



Next, we have the staging drivers. These are drivers that are under development, may change soon, or are not up to the standard quality for the Linux kernel. The only group of drivers in this category (in this kernel version 3.9.4) are the Android drivers. Yes, Android uses the Linux kernel which would make Android a Linux system. However, this is still debated. If the kernel is intended for Android, then it may be wise to enable all of the drivers.

The "Android Binder IPC Driver" provides support for Binder which is a system that allows processes to communicate with each other on Android systems.

The ashmem driver can be enabled next (Enable the Anonymous Shared Memory Subsystem). Ashmem
More Linux Related Stuff on: <http://www.linux.org>

stands for "Anonymous SHared MEMory" or "Android SHared MEMory". This supports a file-based memory system for userspace.

The "Android log driver" offers the complete Android logging system.

The "Timed output class driver" and "Android timed gpio driver" allow the Android system to manipulate GPIO pins and undo the manipulations after the timeout.

The "Android Low Memory Killer" closes processes when more memory is needed. This feature kills the tasks that are not used or inactive.

The "Android alarm driver" makes the kernel wakeup at set intervals.

After the staging drivers are configured, the next set of drivers are for the X86 platform. These drivers are vendor/device specific for X86 (32-bit) hardware.

The next driver is for "Mailbox Hardware Support". This framework controls mailbox queues and interrupt signals for hardware mailbox systems.

"IOMMU Hardware Support" links the memory to devices that are able to use DMA. IOMMU enhances DMA. The IOMMU maps addresses and blocks faulty devices from accessing the memory. IOMMU also allows hardware to access more memory than it could without IOMMU.

The "AMD IOMMU support" driver offers better IOMMU support for AMD devices.

Debugging abilities exist for the AMD IOMMU support (Export AMD IOMMU statistics to debugfs).

A newer version of the IOMMU driver exists for AMD hardware (AMD IOMMU Version 2 driver).

The Linux kernel also provides an IOMMU driver specifically for Intel devices (Support for Intel IOMMU using DMA Remapping Devices).

Some devices may be able to accept a variety of voltages and clock frequencies. This driver allows the operating system to control the device's voltage output and clock rate (Generic Dynamic Voltage and Frequency Scaling (DVFS) support). With this driver enabled, other kernel features can be enabled for power/performance management as seen below.

"Simple Ondemand" is like above, but specifically changes the clock rate based on the device's activity. Generally, more activity means the device needs a faster clock speed to accommodate for the larger resource demand.

"Performance" allows the system to set the clock speed to the maximum supported amount for best performance. This increases power consumption.

"Powersave" sets the clock rate to the lowest value to save power.

"Userspace" allows the userspace to set the clock speed.

"External Connector Class (extcon) support" provides the userspace with a way to watch external

More Linux Related Stuff on: <http://www.linux.org>

connectors like USB and AC ports. This allows applications to know if a cable was plugged into a port. Users will almost always want this enabled. If anyone has purposely disabled this for a legitimate reason, please share with us why that would be needed.

The "GPIO extcon support" driver is just like the above driver, but is made specifically for GPIO pins.

Next, there is a list of various vendor/device specific controllers for memory (Memory Controller drivers). Memory chip controllers may be separate devices or built inside the memory chips. These controllers manage the incoming and outgoing data flow.

The "Industrial I/O support" driver provides a standard interface for sensors despite the bus type they are on (that is, PCIe, spi, GPIO, etc.). IIO is a common abbreviation for Industrial Input/Output.

The Linux kernel offers support for a large variety of accelerometers, amplifiers, analog to digital converters, inertial measurement units, light sensors, magnetometer sensors, and many other sensors and converters.

The "Intel Non-Transparent Bridge support" driver supports PCIe hardware bridges which connect to systems. All writes to mapped memory will be mirrored on both systems.

"VME bridge support" is the same as above except the bridge uses VME which is a different bus standard.

"Pulse-Width Modulation (PWM) Support" controls the back-light and fan speed by regulating the average power received by such devices.

"IndustryPack bus support" offers drivers for the IndustryPack bus standards.

Configuring the kernel Part XIX

Aloha and welcome to the next article of the Linux kernel series! We are getting closer to the end of the configuration process. In this article, we will discuss firmware drivers and then the filesystem drivers.

The first driver in this category is for finding the boot-disk (BIOS Enhanced Disk Drive calls determine boot disk). Sometimes, Linux does not know which drive is the bootable drive. This driver allows the kernel to ask the BIOS. Linux then stores the information on sysfs. Linux needs to know this for setting up bootloaders.

Even if BIOS EDD services are compiled in the kernel, this option can set such services to be inactive by default (Sets default behavior for EDD detection to off). EDD stands for Enhanced Disk Drive.

When using kexec to load a different kernel, performance can be increased by having the firmware provide a memory map (Add firmware-provided memory map to sysfs).

The "Dell Systems Management Base Driver" gives the Linux kernel better control of the Dell hardware via the sysfs interface.

The hardware's information can be accessed by the software via /sys/class/dmi/id/ with this driver enabled (Export DMI identification via sysfs to userspace). DMI stands for Desktop Management Interface. The DMI manages the components of the hardware and can access the hardware's data. The structure of the data in the BIOS and hardware is regulated by the System Management BIOS (SMBIOS) specification.

The raw data tables from the DMI can be accessed with this driver (DMI table support in sysfs).

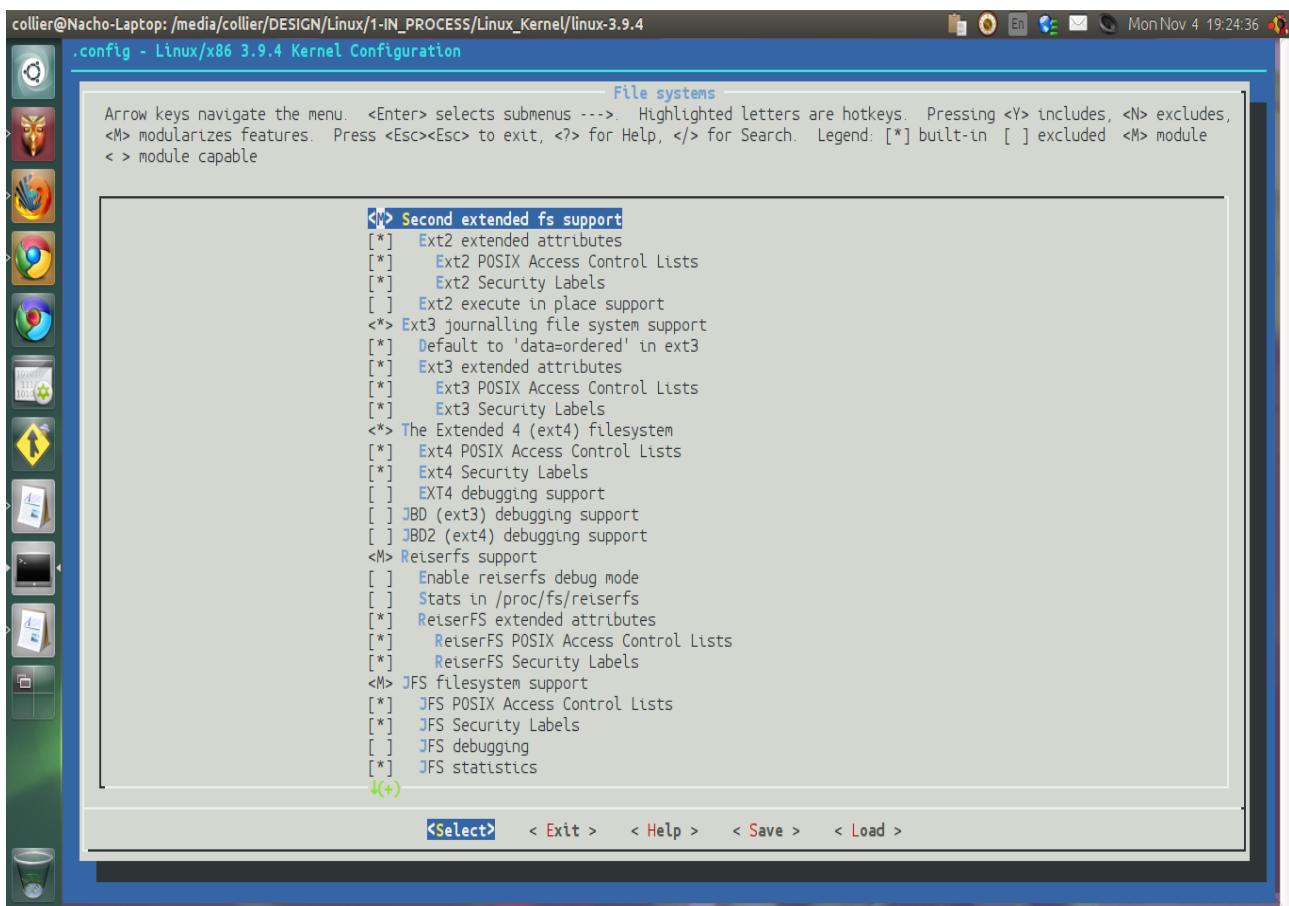
To boot from an iSCSI driver, enable this driver (iSCSI Boot Firmware Table Attributes).

The last firmware driver is a set of "Google Firmware Drivers". These are drivers for Google-specific hardware. Do not enable this driver unless you work for Google and need to use Linux on such hardware or if you are making a Linux kernel for a computer you stole from Google.

Next, we can configure the file system support of the kernel.

The "Second extended fs support" driver provides the EXT2 filesystem.

<http://www.linux.org/threads/ext-file-system.4365/>



The "Ext2 extended attributes" offers the ability to use extra metadata not natively supported by the filesystem.

The "Ext2 POSIX Access Control Lists" driver adds additional permission schemes not native to EXT2.

The "Ext2 Security Labels" enhances the security provided by SELinux.

Enabling "Ext2 execute in place support" allows executables to be executed in the current location without being executed using the paged cache.

The EXT3 filesystem is offered by this driver (Ext3 journaling file system support).

<http://www.linux.org/threads/ext-file-system.4365/>

The "Default to 'data=ordered' in ext3" driver sets the data ordering mode to "Ordered". This deals with the way the journaling and writing work. Data ordering is explained in this article -

<http://www.linux.org/threads/journal-file-system.4136/>

The "Ext3 extended attributes" offers the ability to use extra metadata not natively supported by the filesystem. Again, the following EXT3 drivers/features are the same as for EXT2 - "Ext3 POSIX Access Control Lists" and "Ext3 Security Labels". Also, the same is true for the following EXT4 drivers/features - "Ext4 POSIX Access Control Lists", "Ext4 Security Labels", and "EXT4 debugging support".

More Linux Related Stuff on: <http://www.linux.org>

Journal Block Device debugging is supported by EXT3 (JBD debugging support) and EXT4 (JBD2 debugging support).

The next driver offers the Reiser filesystem (Reiserfs support). <http://www.linux.org/threads/reiser-file-system-reiser3-and-reiser4.4403/>

Debugging exists for the Reiser filesystem (Enable reiserfs debug mode).

The kernel can store ReiserFS statistics in /proc/fs/reiserfs (Stats in /proc/fs/reiserfs).

The following Reiser drivers/features are the same as the ones for EXT2/3/4 - "ReiserFS extended attributes", "ReiserFS POSIX Access Control Lists", and "ReiserFS Security Labels".

JFS is also supported by the Linux kernel and includes various features - "JFS filesystem support", "JFS POSIX Access Control Lists", "JFS Security Labels", "JFS debugging", and "JFS statistics".
<http://www.linux.org/threads/journaled-file-system-jfs.4404/>

Again, XFS is supported with drivers/features that can be enabled - "XFS filesystem support", "XFS Quota support", "XFS POSIX ACL support", "XFS Realtime subvolume support", and "XFS Debugging support". <http://www.linux.org/threads/xfs-file-system.4364/>

The Global FileSystem 2 is supported by the Linux kernel (GFS2 file system support). This filesystem is used to share storage in a cluster.

The "GFS2 DLM locking" driver offers a distributed lock manager (DLM) for GFS2.

The Oracle Cluster FileSystem 2 is supported by the Linux kernel (OCFS2 file system support). This filesystem is used to share storage in a cluster.

The "O2CB Kernelspace Clustering" driver offers various services for the OCFS2 filesystem.

The "OCFS2 Userspace Clustering" driver allows the cluster stack to execute in userspace.

The "OCFS2 statistics" driver allows the user to get statistics concerning the filesystem.

Like with most of the Linux kernel, the OCFS2 offers logging (OCFS2 logging support). This may be used to watch for errors or for debugging purposes.

The "OCFS2 expensive checks" driver offers storage consistency checks at the cost of performance. Some Linux users recommend only enabling this feature for debugging purposes.

The kernel also contains the new B-Tree FileSystem; this driver offers the disk formatter (Btrfs filesystem Unstable disk format). BTRFS is still in development and is planned to one day become as popular or more popular than EXT4. <http://www.linux.org/threads/b-tree-file-system-btrfs.4430/>

The "Btrfs POSIX Access Control Lists" driver adds additional permission schemes not native to BTRFS.

Next, there is a BTRFS check tool (Btrfs with integrity check tool compiled in (DANGEROUS)). Since, BTRFS is a newly developing filesystem, most of the software associated with it are unstable.

The NIL-FileSystem is also supported by Linux (NILFS2 file system support).

<http://www.linux.org/threads/new-implementation-of-a-log-structured-file-system-nilfs.4547/>

To support the flock() system call used by some filesystems, enable this driver (Enable POSIX file locking API). Disabling this driver will reduce the kernel size by about eleven kilobytes. The driver provides file-locking. File-locking is the process of allowing one process to read a file at a time. This is commonly used with network filesystems like NFS.

The "Dnotify support" driver is a legacy filesystem notification system that informs the userspace of events on the file system. One use of this and the successor notifications software is to monitor the filesystem for applications. Certain applications tell this daemon what events to watch. Otherwise, each userspace application would need to complete this task themselves.

Remember, Dnotify is a legacy system, so what is the new notification system? It is Inotify which is provided by this driver (Inotify support for userspace).

An alternative notification system is fanotify (Filesystem wide access notification). Fanotify is the same as Inotify, but fanotify relays more information to the userspace than Inotify.

Fanotify can check permissions with this driver enabled (fanotify permissions checking).

For systems that need to divide the storage space by user will want "Quota support".

<http://www.linux.org/threads/atomic-disk-quotas.4277/>

The following driver allows disk quota warnings and messages to be reported through netlink (Report quota messages through netlink interface). Netlink is a socket interface on the userspace that communicates with the kernel.

Quota messages can also be sent to a console (Print quota warnings to console (OBSOLETE)).

This driver allows the quota system to perform extra sanity checks (Additional quota sanity checks). In computer technology, a sanity check is the process of checking for errors that may be due to poor programming. The files and output are inspected to ensure the data is what it should be and not structured in some odd fashion.

Some old system use the old quota system but want to retain the old quota system when upgrading to a newer kernel. This is easily solved by enabling this driver (Old quota format support). Many readers may be wondering why someone would want to keep the old quota system instead of upgrading. Well, imagine being the manager of the IT department of a very large corporation that has many servers running very important tasks. Would you want to create and configure a new (and possibly large) quota system when you can continue using the one that works well? Generally, with computers, follow the principle - If it is not broken or will not cause security issues, do not fix it.

The newer quota system supports 32-bit UIDs and GIDs with this driver (Quota format vfs0 and vfs1 support).

More Linux Related Stuff on: <http://www.linux.org>

To automatically mount remote storage units, enable this driver (Kernel automounter version 4 support).

FUSE filesystems are supported by this driver (FUSE (Filesystem in Userspace) support). Filesystem in Userspace (FUSE) allows any user to create their own filesystem and utilize it in userspace.

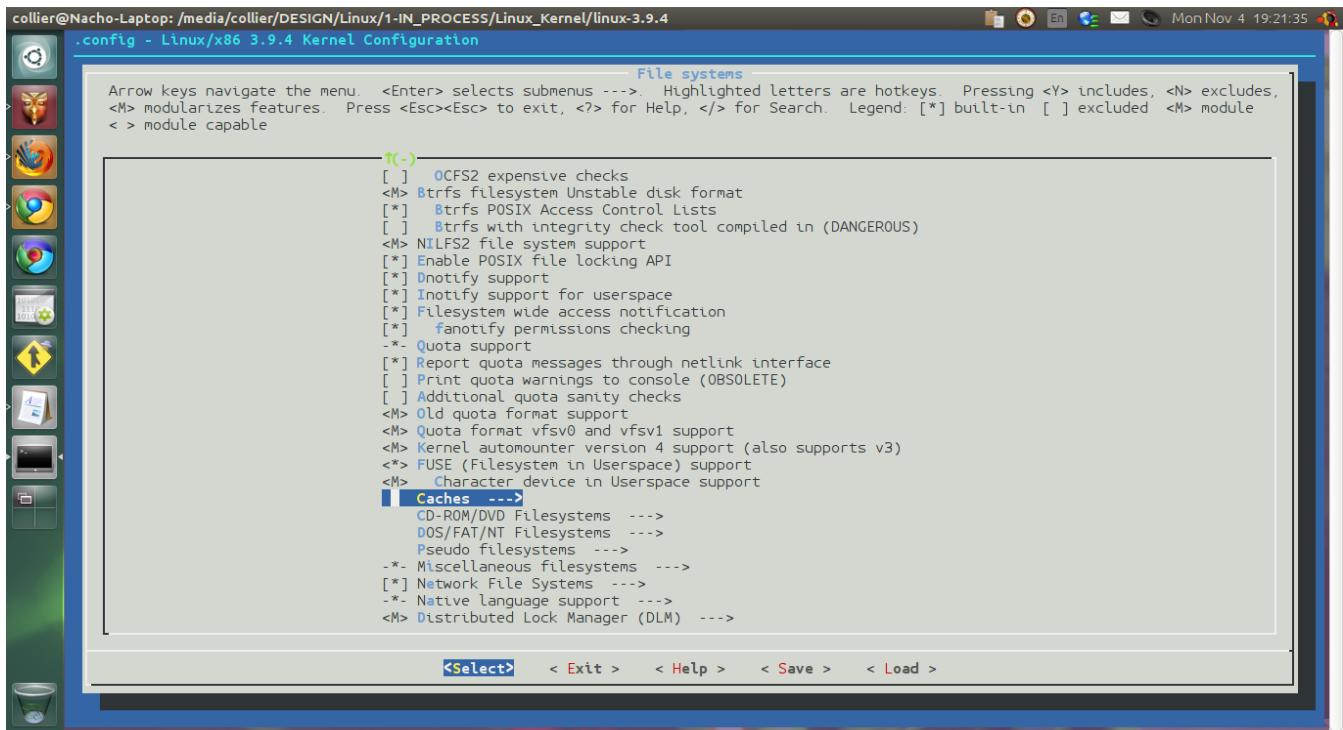
A special extension for FUSE can be used to utilize character devices in userspace (Character device in Userspace support).

Configuring the kernel Part XX

Aloha and welcome to the next Linux kernel article! In this article, we will continue configuring filesystem support.

First, we can enable "General filesystem local caching manager" which allows the kernel to store filesystem cache. This can enhance performance at the cost of storage space.

The caching system can be monitored with statistical information used for debugging purposes (Gather statistical information on local caching). Generally, this feature should only be enabled if you plan to debug the caching system.



This next feature is a lot like the above, but this feature stores latency information (Gather latency information on local caching). Again, this is a debugging feature.

The "Debug FS-Cache" driver offers many other debugging abilities for the cache system.

The next cache debugging tool keeps a global list (any process can access the list) of filesystem cache objects (Maintain global object list for debugging purposes).

To enhance the speed of network filesystems, enable this next driver (Filesystem caching on files). This feature allows a whole local filesystem to be used as cache for remote filesystem and storage units. The Linux kernel will manage this partition.

Two different debugging drivers exist for this local cache system for remote filesystems (Debug CacheFiles) and (Gather latency information on CacheFiles).

The most common optical disc filesystem is ISO-9660 which is ISO standard 9660, hence the name (ISO 9660 CDROM file system support). This driver is needed to read/write the major of optical discs.

When reading an optical disc with files using long Unicode filenames or writing such files, this driver is required (Microsoft Joliet CDROM extensions). This is an extension to the ISO-9660 filesystem.

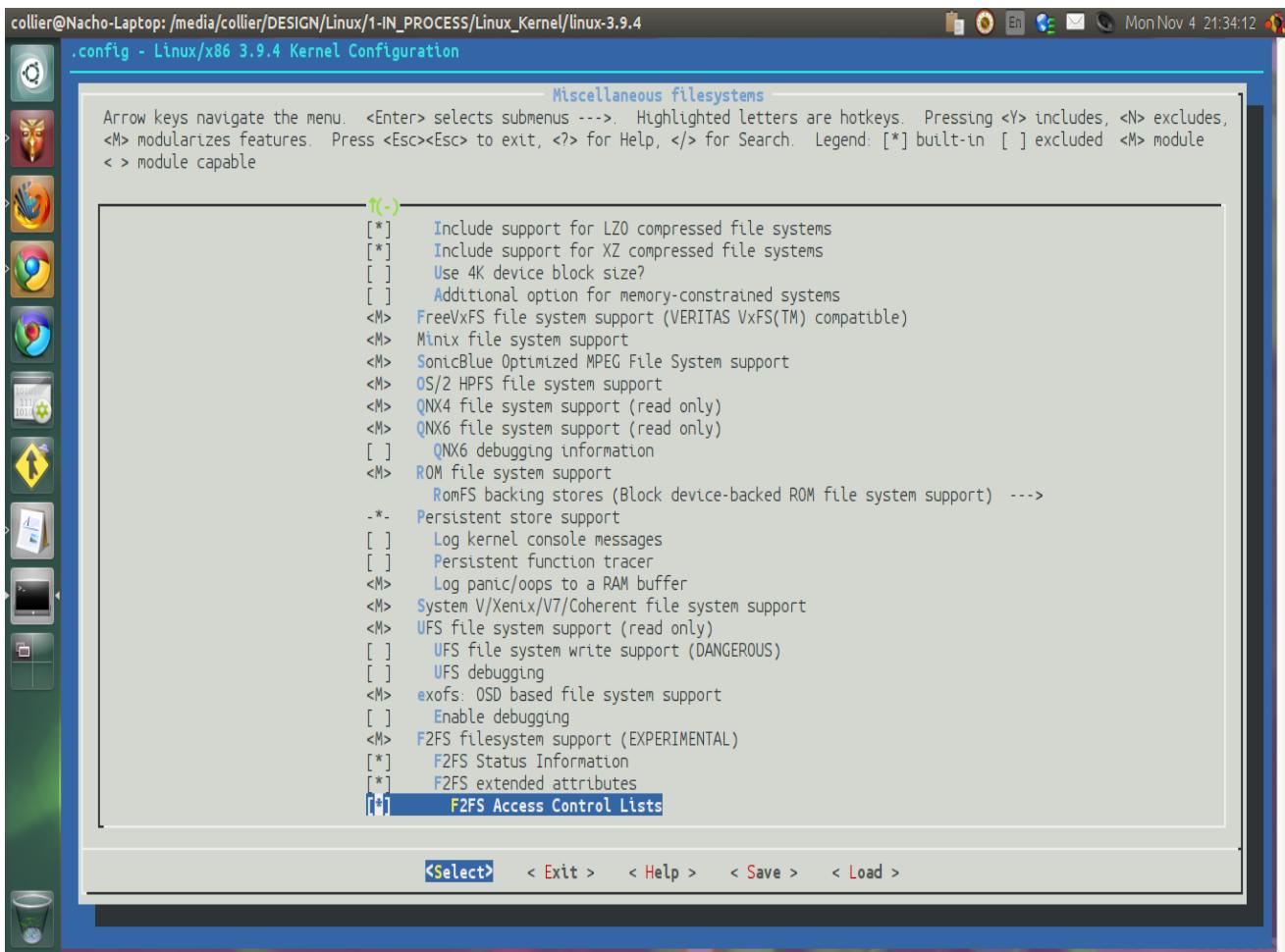
The "Transparent decompression extension" allows data to be written to a disc in a compressed form and read off the disc and decompressed transparently. This will allow more data to be placed on the disc.

"UDF file system support" allows the kernel to read/write rewritable-optical-discs that are using the UDF filesystem. UDF is designed to manage incremental writes. UDF allows the rewritable optical disc to be used more like flash drives. The system can write and update the optical disc's data more quickly than regular writing on ISO-9660 filesystems. However, this is not faster than using flash drives.

As many of you know, Windows is a very popular system, so many storage units are using the FAT filesystem of NTFS. Thankfully, Linux supports such filesystems. The "MSDOS fs support" driver is a general driver for MS-DOS filesystems. This will increase the kernel size significantly, but since the FAT filesystems are very common, this size increase is usually worth the cost.

<http://www.linux.org/threads/file-allocation-table-fat.4472/>

To support the FAT filesystems, enable this driver (VFAT (Windows-95) fs support). At the time this article was written, this driver does not support FAT64 (commonly called exFAT).



The size of the codepage can be set here (Default codepage for FAT).

After that, the default character set is configured for the FAT filesystems (Default iocharset for FAT).

The NTFS file system is offered with this driver (NTFS file system support). The driver provides read-only abilities. To write to NTFS, enable this driver (NTFS write support).

The Linux kernel offers debugging tools for the NTFS filesystem (NTFS debugging support).

To have a proc folder in the root, this feature must be enabled (/proc file system support). Some other similar drivers that rely on this one include (/proc/kcore support), (/proc/vmcore support), and (Sysctl support (/proc/sys)). The proc system (short for “process”) uses the proc-filesystem sometimes called procfs. This filesystem is in the hardware’s memory and is created when Linux boots up. So, when viewing files in proc, the user is browsing the memory as if it were like other storage units. Proc acts as an interface between userspace and the kernelspace. Proc is in the kernelspace.

The "Enable /proc page monitoring" driver offers some proc files that monitor the memory utilization of processes.

The "sysfs file system support" driver creates the /sys/ folder. The sysfs filesystem is in memory and provides an interface to the kernel objects.

More Linux Related Stuff on: <http://www.linux.org>

The tmp directory is needed by many applications and Linux itself, so it is strongly recommended that this driver be enabled (Tmpfs virtual memory file system support (former shm fs)). The tmp filesystem maybe stored on the hard-drive or in memory and is used only to store temporary files.

The "Tmpfs POSIX Access Control Lists" driver offers extra permission features for the files in the tmpfs virtual filesystem.

The "Tmpfs extended attributes" driver provides more attributes to tmpfs files than what they would normally have without he driver.

The "HugeTLB file system support" driver provides the hugetlbfs filesystem, which is ramfs based. This virtual filesystem contains HugeTLB pages.

The configfs filesystem is a kernel object manager in the form of a filesystem (Userspace-driven configuration filesystem). It is highly recommended that this driver be enabled. ConfigFS is a lot like sysfs. However, ConfigFS is used to create and delete kernel object while sysfs is used to view and modify kernel objects.

Next, we can move back to "real" filesystems. That is, the filesystems users themselves use to store their personal files. Next, the kernel can be given the ability to read ADFS filesystems (ADFS file system support).

The ability to write to ADFS filesystems is provided by a separate and unstable driver (ADFS write support (DANGEROUS)). ADFS stands for Advanced Disc Filing System.

Linux also supports the Amiga Fast FileSystem (Amiga FFS file system support).

<http://www.linux.org/threads/amiga-fast-file-system-affs.4429/>

The "eCrypt filesystem layer support" driver offers a POSIX-compliant cryptographic filesystem layer. This eCrypt can be placed on every and any filesystem no matter what partition table the filesystem resides on. <http://www.linux.org/threads/intro-to-encryption.4376/>

The eCrypt layer can have a device file if this driver is enabled (Enable notifications for userspace key wrap/unwrap). The device path is /dev/ecryptfs.

Linux also supports HFS and HFS+ (Apple Macintosh file system support) and (Apple Extended HFS file system support). <http://www.linux.org/threads/hierarchical-file-system-hfs.4480/> and <http://www.linux.org/threads/hierarchical-file-system-plus-hfs.4493/>

The BeFS filesystem can be used by Linux as a read-only filesystem (BeOS file system (BeFS) support (read only)). Generally, it is easier to program the reading abilities for a filesystem than the writing features.

Special debugging features exist for BeFS (Debug BeFS).

EFS is another filesystem that Linux can only read, not write (EFS file system support (read only)).

<http://www.linux.org/threads/extent-file-system-efs.4447/>

More Linux Related Stuff on: <http://www.linux.org>

Some flash drives may use the JFFS2 filesystem (Journalling Flash File System v2 (JFFS2) support). Next, the debugging level can be set (JFFS2 debugging verbosity).
<http://www.linux.org/threads/journaling-flash-file-system-version-2-jffs2.4495/>

To use JFFS2 on NAND and NOR flash drives, this driver is needed (JFFS2 write-buffering support).

This next driver offers better error protection (Verify JFFS2 write-buffer reads).

JFFS filesystems can be mounted faster with "JFFS2 summary support" enabled. This driver stores information about the filesystem.

Like the other extended/extra attributes drivers for some filesystems, JFFS2 has such a driver (JFFS2 XATTR support).

The JFFS2 filesystem supports various transparent compression systems. This allows files to be smaller on JFFS2 filesystems and be read without the user needing to perform any special actions. (Advanced compression options for JFFS2), (JFFS2 ZLIB compression support), (JFFS2 LZO compression support), (JFFS2 RTIME compression support), and (JFFS2 RUBIN compression support). The default compression format can be defined in the following option (JFFS2 default compression mode).

A successor for JFFS2 exists and is supported by the kernel (UBIFS file system support). The Unsorted Block Image File System (UBIFS) also competes with LogFS.

The Linux kernel also supports LogFS (LogFS file system).

ROM-based embedded systems need support for CramFS (Compressed ROM file system support (cramfs)).

Alternately, embedded systems could use SquashFS which is a read-only compression filesystem (SquashFS 4.0 - Squashed file system support). The Linux kernel also offers extended attributes for SquashFS (Squashfs XATTR support).

There are three different compression formats supported by SquashFS - (Include support for ZLIB compressed file systems), (Include support for LZO compressed file systems), and (Include support for XZ compressed file systems). The block size for SquashFS can be set to four kilobytes (Use 4K device block size?). Also, the cache size can be set (Additional option for memory-constrained systems).

The Linux kernel supports FreeVxFS (FreeVxFS file system support (VERITAS VxFS(TM) compatible)), Minix (Minix file system support), MPEG filesystem (SonicBlue Optimized MPEG File System support), HPFS (OS/2 HPFS file system support), QNX4 (QNX4 file system support (read only)), QNX6 (QNX6 file system support (read only)), and the ROM filesystem (ROM file system support). <http://www.linux.org/threads/qnx-file-systems.4577/> and <http://www.linux.org/threads/minix-mini-unix-file-system.4545/>

"RomFS backing stores (Block device-backed ROM file system support)" offers a list of various ROMfs extra features and abilities.

The "Persistent store support" driver provides support for the pstore filesystem which allows access to platform level persistent storage.

More Linux Related Stuff on: <http://www.linux.org>

The pstore filesystem can store kernel logs/messages (Log kernel console messages).

When a kernel panic takes place (equivalent to the "Blue-Screen-of-Death" on Windows), the "Log panic/oops to a RAM buffer" driver will store a log in the RAM.

This next single driver offers support for the Xenix, Coherent, Version 7, and System V filesystems (System V/Xenix/V7/Coherent file system support).

The Linux kernel also supports UFS (UFS file system support (read only)), (UFS file system write support (DANGEROUS)), and (UFS debugging).

exofs is also supported by the kernel (exofs: OSD based file system support).

The Flash-Friendly FileSystem is a special filesystem for flash drives (F2FS filesystem support (EXPERIMENTAL)), (F2FS Status Information), (F2FS extended attributes), and (F2FS Access Control Lists). <http://www.linux.org/threads/flash-friendly-file-system-f2fs.4477/>

In the next article, we will configure network filesystems. Mahalo!

Configuring the kernel Part XXI

Aloha! In this article, we will configure network filesystem support for the Linux kernel. A network filesystem is a remote filesystem that computers access via the network.

First, the "NFS client support" driver allows the Linux system to use the NFS network filesystem. There are also three other drivers for different versions of NFS - (NFS client support for NFS version 2), (NFS client support for NFS version 3), (NFS client support for NFS version 4), and (NFS client support for NFSv4.1). If you have a network that possess NFS, either figure out what version of NFS you are using, or enable all of the NFS drivers.

Swap space is not required to be on a local storage unit. This driver allows Linux to use NFS support to use remote swap spaces (Provide swap over NFS support).

The NFS system can be sped up by using a cache system (Provide NFS client caching support). This is local cache.

Enable this driver to allow DNS to use host-names for NFS servers (Use the legacy NFS DNS resolver).

"NFS server support" gives the server providing NFS the features it needs to fulfill such a task. Some other NFS drivers include (NFS server support for NFS version 3) and (NFS server support for NFS version 4).

The "NFS server manual fault injection" driver is a debugging tool that allows developers to make the NFS server think an error occurred with NFS. Specifically, this is used to test how the server handles NFS errors.

The "Secure RPC: Kerberos V mechanism" is needed to make the RPC calls secure. NFS cannot be added to the kernel without this feature for security reasons.

There is a special debugging tool for RPC (RPC: Enable dprintk debugging).

The Linux kernel supports the Ceph filesystem (Ceph distributed file system).

CIFS is a virtual filesystem used by Samba and Windows servers (CIFS support (advanced network filesystem, SMBFS successor)). CIFS stands for Common Internet FileSystem.

There are two features that can be used to debug or monitor the CIFS driver (CIFS statistics) and (Extended statistics).

A special driver is needed to support servers with LANMAN security (Support legacy servers which use weaker LANMAN security). LANMAN or LM hash is a special password hashing function that has some weaknesses.

CIFS requires that Kerberos tickets be used before mounting to secure servers (Kerberos/SPNEGO advanced session setup). This driver offers the ability for CIFS to use userspace tools which is needed to provide the tickets.

More Linux Related Stuff on: <http://www.linux.org>

Like other filesystems, CIFS can have extended abilities (CIFS extended attributes) and (CIFS POSIX Extensions).

This driver gets the Access Control List (ACL) from the CIFS server (Provide CIFS ACL support).

CIFS has two other debugging tools (Enable CIFS debugging routines) and (Enable additional CIFS debugging routines).

CIFS can have "DFS feature support" which allows shares to be accessed even when they are moved. DFS stands for Distributed FileSystem.

SMB2 is an improved alternative to CIFS (SMB2 network file system support). SMB2 stands for Server Message Block version 2.

Clients can store CIFS cache with this driver enabled (Provide CIFS client caching support).

Novell NetWare clients need this driver to access NetWare volumes (NCP file system support (to mount NetWare volumes)). NCP stands for NetWare Core Protocol. NCP is a protocol that allows clients to communicate with the servers hosting NetWare volumes.

NetWare servers can use NFS namespaces if this driver is enabled (Use NFS namespace if available).

NetWare servers can use the OS/2 long namespaces if this driver is enabled (Use LONG (OS/2) namespace if available).

If this driver is enabled, then filenames made by DOS or on storage units owned by a DOS system will be converted to lowercase (Lowercase DOS filenames).

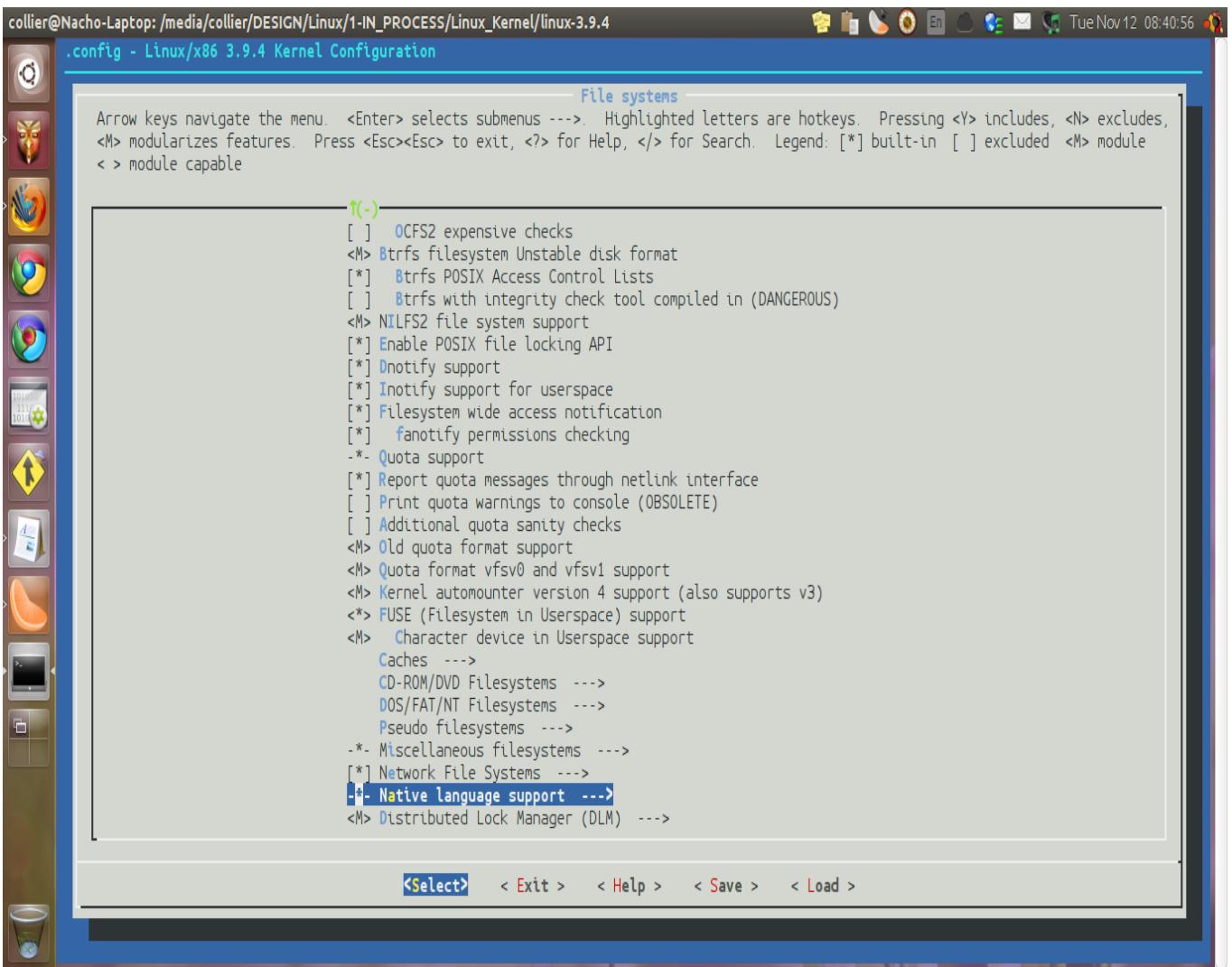
Many filesystems depend on native language support (Use Native Language Support). Specifically, Native Language Support (NLS) allows the different character-sets to be used in filenames.

NCP filesystems can support the execute flag and symbolic links with this driver enabled (Enable symbolic links and execute flags).

The Linux kernel offers support for the Coda filesystem (Coda file system support (advanced network fs)). Coda is one of many network filesystems.

The Linux kernel can support the Andrew Filesystem (Andrew File System support (AFS)). However, the Linux kernel can only read such filesystems in an insecure manner. This driver is intended to allow Linux systems to access AFS. If your network only contains Linux systems, then select a different network filesystem that the kernel can fully support.

The Linux kernel has an experimental driver for accessing Plan 9 resources via the 9P2000 protocol (Plan 9 Resource Sharing Support (9P2000)). The kernel also has cache support (Enable 9P client caching support) and control lists (9P POSIX Access Control Lists) for the previously mentioned Plan 9 feature.



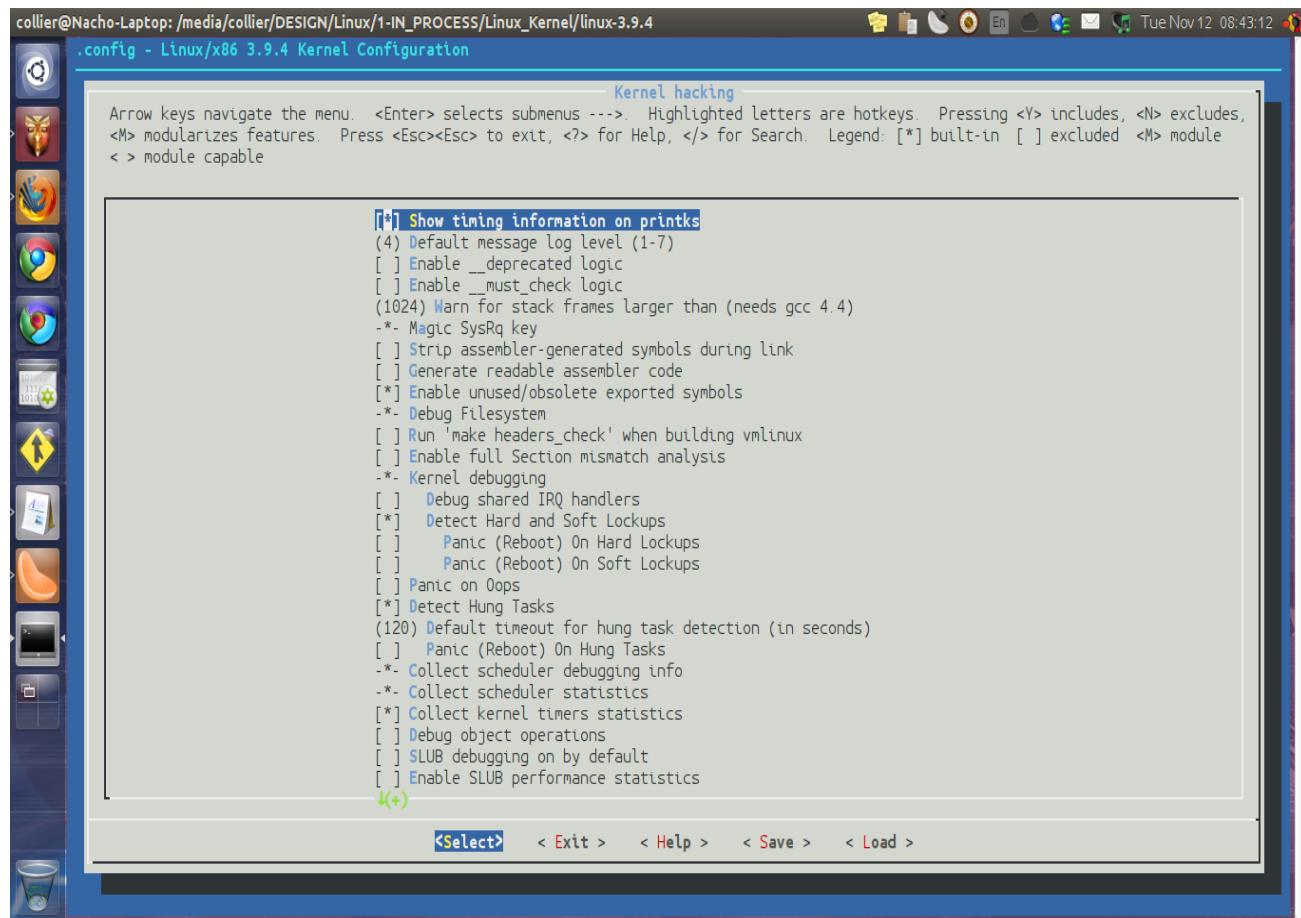
After the network filesystems have been configured, the next part of the kernel to setup is the "Native Language Support". This whole menu contains the drivers for most or all of the character-sets and encodings. Enabling these encodings allows these character sets to be used by the system and applications. UTF-8 is the most commonly used encoding, but it is not the only one. Most applications and driver need UTF-8, so this encoding is already set to be added to the kernel.

After that menu, the "Distributed Lock Manager (DLM)" can be configured. A DLM is used to keep shared resources in sync and performing well. This driver manages the userspace and kernelspace applications that access or manipulate shared resources (like network filesystems). Clusters strongly depend on this driver.

Now that we have finally finished configuring the filesystems and related features, we can now move on to the "Kernel hacking" menu seen on the main (first) screen on the kernel configuration tool (when using the command "make menuconfig"). I am using the ncurses interface (seen in the screenshot) which is initiated with the "make menuconfig" command, so other interfaces may be a little different. Many of the features and settings in the kernel hacking menu contain various settings concerning the kernel itself. Some of these features are debugging tools and some control the kernel's behavior.

This first setting adds the printrk time stamps to the syslog system call output (Show timing information on printks).

The next three features control various debugging features (Default message log level (1-7)), (Enable __deprecated logic), and (Enable __must_check logic).



The next feature is a debugging feature that is active at compiling time ((1024) Warn for stack frames larger than (needs gcc 4.4)). If stack frames are larger than the specified amount, then the compiler will warn the user.

The "Magic SysRq key" driver will enable support for the Magic SysRq key. This allows users to send the kernel special commands when Alt+PrintScreen is pressed. This works in most cases regardless of the kernel's state. However, exceptions exist. It is highly recommended that the Magic SysRq Key be enabled.

During compilation, the assembler's symbolic links will be removed during a link to reduce the output of get_wchan() (Strip assembler-generated symbols during link).

This next feature is for debugging purposes (Generate readable assembler code). If enabled, some kernel optimizations will be disabled so some of the assembly code will be human-readable. This will harm the kernel's speed. Only enable this if you have a specific reason for doing so.

This setting enables/disables commonly unneeded or obsolete symbols (Enable unused/obsolete

More Linux Related Stuff on: <http://www.linux.org>

exported symbols). However, some modules may need such symbols. Enabling this will increase the kernel's size. It is very unlikely that a Linux user will need such symbols. In general, disable this unless you know for a fact the user needs a symbol for an important module.

Sanity checks will be performed on user kernel headers if this setting is enabled (Run 'make headers_check' when building vmlinux).

During compilation, this feature will check for invalid references (Enable full Section mismatch analysis).

The kernel can be configured to detect soft and hard lockups (Detect Hard and Soft Lockups). When the system is frozen for more than twenty second and other tasks cannot execute, this is called a soft-lockup. If the CPU is in a loop that lasts for more than ten seconds and interrupts fail to get execution time, then this is called a hard-lockup.

The next two features set the kernel to reboot on hard and soft lockups respectively, (Panic (Reboot) On Hard Lockups) and (Panic (Reboot) On Soft Lockups).

When the kernel experiences major problems, it can be set to start a kernel panic (Panic on Oops). It is highly recommended that this setting be enabled. This will help to prevent the kernel from causing system damage and data loss.

The kernel can be set to detect hung tasks (Detect Hung Tasks). This is when a process or application locks-up or is frozen. Specifically, the application becomes uninterruptable. The following setting allows the user to define how much time must pass before a process is deemed "hung" (Default timeout for hung task detection (in seconds)).

The kernel can be set to restart when a process hangs (Panic (Reboot) On Hung Tasks). Generally, users will not want to enable this. Would you like your computer to restart every time an application becomes frozen?

The "Kernel memory leak detector" finds and logs memory leaks.

The kernel uses frame pointers to help report errors more efficiently and include more information (Compile the kernel with frame pointers). I will skip a lot of the debugging tools because they are self-explanatory.

As many Linux users know, when the system boots up, the boot messages appear too quickly to be read. This feature sets the delay time which will give users more time to read the messages (Delay each boot printk message by N milliseconds).

This is a special developmental feature for testing backtrace code (Self test for the backtrace code). Backtrace code is a self-test.

Block device number can be extended (Force extended block device numbers and spread them). However, this may cause booting issues, so use with caution.

We will continue to configure the kernel itself in the next article. I hope you have enjoyed this article. Mahalo!

More Linux Related Stuff on: <http://www.linux.org>

Configuring the kernel Part XXII

Aloha! In this article, we will continue to configure the kernel hacks and then we will configure the whole security system.

The next feature to configure is needed by Alpha and s390 processor (Force weak per-cpu definitions). This feature offers a fix for an addressing issue commonly seen in such processors. Other processors do not need this feature enabled.

Kernel dumps can be tested with this special debugging tool (Linux Kernel Dump Test Tool Module). This software will allow a kernel developer to trigger a fake error that will cause a kernel dump. The kernel developers can then ensure that the dumps complete successfully.

The kernel offers some different error injection modules that allow kernel developers to test the notifiers (CPU notifier error injection module), (PM notifier error injection module), and (Memory hotplug notifier error injection module). A notifier informs the system that the hardware is present, which is important for hotplugging. These error injection modules trigger an error in this notification system so developers can test the notification system's error handling abilities.

The "Fault-injection framework" driver offers various tools for testing fault-handling.

The "Latency measuring infrastructure" driver provides the LatencyTOP tool used to find the userspace object that is blocking/interfering with a kernel execution/task.

Next, we have a sub-menu titled "Tracers" that contains a list of various tracers. A tracer is a piece of code that watches various kernel functions. Every time a particular function starts, a tracer will be called to watch the function.

This next module tests the performance of the Red-Black tree library (Red-Black tree test). The Red-Black tree library is a sorting and searching algorithm.

The next feature is the same except that it tests the interval tree library (Interval tree test).

The kernel can also debug FireWire on other systems while that particular remote system is booting (Remote debugging over FireWire early on boot) and (Remote debugging over FireWire with firewire-ohci).

The printk() function can be made to print various debugging messages if this feature is enabled (Enable dynamic printk() support). "printk()" is a commonly discussed system call, so remember that it prints debugging messages about the kernel.

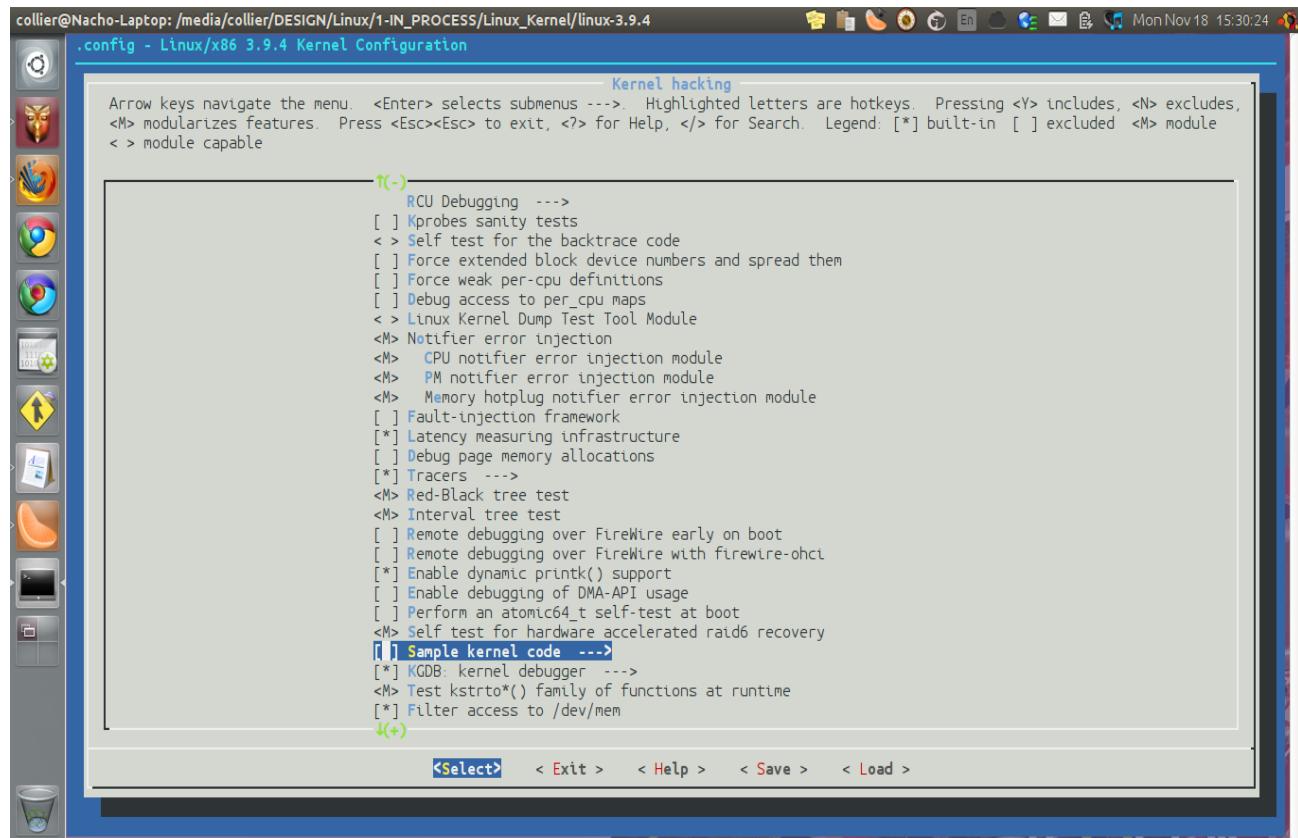
Here is a Direct Memory Access (DMA) debugging driver (Enable debugging of DMA-API usage).

The Atomic64 self-tests checks if the system supports atomic operations (Perform an atomic64_t self-test at boot). This is where a 32-bit system performs a 64-bit operation.

This driver provides a self-test for all of the possible RAID6 recovery systems (Self test for hardware accelerated raid6 recovery).

NOTE: Self-tests are low-level tests and detection software that executes before most of the system's hardware and software turns on and executes. Self-tests search for hardware, failing devices, etc. A self-test may also be code an application uses to test itself.

In the Kernel Hacking menu (if you are using a menu-like interface such as ncurses), there is a sub-menu titled "Sample kernel code". If you make your own personal modules, this is where you can enable them. In a later article, we will discuss how to implement custom/home-made kernel modules. Just remember this is where you enable your module.



The Kernel GNU DeBugger (KGDB) has many features that can be enabled or disabled (KGDB: kernel debugger). This debugger only works with two Linux systems plugged into each other via serial connection.

This feature provides extra boot-up messages for the decompression of the bzimage (Enable verbose x86 bootup info messages). You set the kernel encryption near the beginning of the configuration process.

Printk() prints various information to the boot-screen of dmesg, but after the serial and console drivers load. Enable this driver to make printk start printing messages sooner (Early printk).

This next driver is the same as above, but uses the EHCI port (Early printk via EHCI debug port).

The kernel can be set to watch for stack overflows so the kernel can manage the error better (Check for stack overflows). The kernel will execute more slowly, but overflows will not cause as much damage.

The page-table for the kernel can be seen on debugfs with this enabled (Export kernel pagetable layout to userspace via debugfs). However, this will slow down the kernel. This is needed for debugging purposes.

The kernel's written mistakes can be caught with this feature (Write protect kernel read-only data structures). This option turns the kernel's read-only data to write-protected mode. This debugging tool harms the kernel's speed. That debugging tool has a tool to debug itself (Testcase for the DEBUG_RODATA feature).

To prevent the execution of modules with modified code (due to an error), then enable this protective feature (Set loadable kernel module data as NX and text as RO). The debugging tool for that feature is provided by this driver (Testcase for the NX non-executable stack feature).

The kernel can be set to flush one of the TLB entries at a time or the whole table using this option (Set upper limit of TLB entries to flush one-by-one).

The next feature is an IOMMU debugging feature (Enable IOMMU debugging). There is another debugging test that disables some IOMMU features to test for extra stability (Enable IOMMU stress-test mode). The IOMMU stands for input/output memory management unit.

Enabling this option will make the kernel perform self-tests on the change_page_attr() system call on thirty second intervals (CPA self-test code). This system call changes page attributes.

Any kernel code marked as "inline" can not be manipulated as much as it would by GCC than if it were not marked (Allow gcc to uninlne functions marked). The GCC compiler adds code that it feels will make the code better (GCC is good at doing so). However, some code is not meant to be manipulated by GCC.

This next driver offers sanity checks for the "copy_from_user()" system call (Strict copy size checks). copy_from_user() copies a block of userspace data to kernelspace.

Here is another self-test; this one is for NMI (NMI Selftest).

Now, we can move on to the "Security Options" which is seen as a sub-menu in the main menu if you are using a menu-based interface, like ncurses. The first option allows access keys and authentication tokens to be stored in the kernel (Enable access key retention support). This is used for many reasons like accessing encrypted filesystems.

The following option is for creating and sealing/unsealing keys (TRUSTED KEYS). Encrypted keys are encrypted/decrypted using this driver (ENCRYPTED KEYS).

Keys can be viewed in proc with this feature enabled (Enable the /proc/keys file by which keys may be viewed).

Extra restrictions can be applied to syslog with this security feature (Restrict unprivileged access to the kernel syslog).

If this option is enabled, then the user can select different security models (Enable different security models). Otherwise, the defaults will be used. Disable this if you do not fully understand security or if

More Linux Related Stuff on: <http://www.linux.org>

you are fine with your kernel using the defaults.

The securityfs filesystem is offered by this driver (Enable the securityfs filesystem).

Hooks are added to networking and socket security when this feature is enabled (Socket and Networking Security Hooks). These hooks are access controls.

IPSec networking hooks (also called XFRM networking hooks) are implemented when this option is enabled (XFRM (IPSec) Networking Security Hooks). Security hooks are also available for files (Security hooks for pathname based access control).

The next driver provides support for Intel's Trusted Execution Technology (Enable Intel(R) Trusted Execution Technology (Intel(R) TXT)).

The user can set the range of memory addresses that cannot be reserved for userspace (Low address space for LSM to protect from user allocation). The starting point is 0. The user types the end point for this option. For most platforms, 65536 is a recommended choice.

SELinux (mentioned in the Kernel Security article) is one of the popular Linux-Security-Modules (NSA SELinux Support). Many options and features exist for SELinux. The boot parameter determines whether SELinux is started {1} or not started {0} when the kernel executes (NSA SELinux boot parameter). SELinux can be configured with the ability to be temporarily disabled at times when the Root user needs to do so (NSA SELinux runtime disable). Users can develop and test new policies with this feature enabled (NSA SELinux Development Support). AVC statistics are collected and stored by this feature (NSA SELinux AVC Statistics). A default can be set for the checkreqprot flag; a "1" means SELinux will check the application's requested protection and zero will default to the kernel's protection for mmap and mprotect system calls (NSA SELinux checkreqprot default value). Many SELinux policies exist; the user can set the latest version that they wish SELinux not to exceed (NSA SELinux maximum supported policy format version).

One of the other Linux-Security-Modules (LSM), SMACK, is supported by the kernel (Simplified Mandatory Access Control Kernel Support).

TOMOYO is another supported LSM (TOMOYO Linux Support). The maximum number of entries permitted to be added during learning-mode is set in the following feature (Default maximal count for learning mode). The amount of log entires can also be set (Default maximal count for audit log). Next, this option allows/disallows TOMOYO to be activated without a policy loader (Activate without calling userspace policy loader). The location of the policy loader is configured here ((/sbin/tomoyo-init) Location of userspace policy loader) and the executable that triggers the execution is set here ((/sbin/init) Trigger for calling userspace policy loader).

Again, the kernel supports another LSM - AppArmor (AppArmor support). Like with SELinux, the default boot parameter can be set for AppArmor (AppArmor boot parameter default value).

Yama is another LSM (Yama support). Yama can be used with another LSM if this feature is enabled (Yama stacked with other LSMs).

This driver gives the kernel the ability to use multiple keyrings for verification processes (Digital signature verification using multiple keyrings).

More Linux Related Stuff on: <http://www.linux.org>

Asymmetric keys are supported with this feature (Enable asymmetric keys support).

The kernel can keep and maintain a list of hashes and important system files (Integrity Measurement Architecture(IMA)). Then, if malware changes an important file, the kernel will know because the hashes are checked before the file or executable are used. It is highly recommended that this feature be enabled.

Extra security attributes are added to files if this feature is enabled (EVM support). The version can be set using this next option (EVM HMAC version). The two choices are version 1 and 2.

Remember all of the different Linux Security Modules (LSMs)? Well, the default can be set here (Default security module (AppArmor)).

In the next article, we can configure the "Cryptographic API". There may be two additional configuration articles left than what I estimated. After that, there are other kernel topics to discuss.

SNEAK PEAK: After this article, there is one more about the configuration process. Then, we can finally compile and install the kernel.

Configuring the kernel Part XXIII

Aloha and welcome to the next kernel configuration article! In this article, we will configure the Cryptographic API, Virtualization, and Library Routines. Cryptography refers to encryption and secure communication between desired computers. Users may encrypt data to ensure only the recipient reads the data instead of hackers that may obtain the data.

The Linux kernel requires the "Cryptographic algorithm manager" to be enabled in the kernel. This feature provides the software needed to operate the cryptographic abilities of the kernel.

The userspace can configure the cryptography features when this driver is enabled (Userspace cryptographic algorithm configuration). NOTE: This configuration is referring to the cryptographic setup during kernel runtime, not the tool for making the kernel.

To enhance performance, enable this feature which stops self-tests on cryptographic algorithms (Disable run-time self tests).

The "GF(2^{128}) multiplication functions" is a specific algorithm used by some ciphers. GF stands for Galois field and is a set of finite numbers. These sets are called fields and they come in a variety of sizes.

"Null algorithms" are algorithms used in IPsec. Null encryption means no encryption, so this driver allows IPsec to use no encryption.

Arbitrary algorithms can be converted to parallel algorithms (Parallel crypto engine). This feature provides that converter.

Arbitrary algorithms can also be converted to asynchronous algorithms (Software async crypto daemon).

"Authenc support" is needed by IPsec. Authenc support stands for Authenticated Encryption and offers multiple encryptions to IPsec.

CCM stands for "Counter with CBC MAC" and is needed by IPsec (CCM support).

This driver offers "GCM/GMAC support". GCM is "Galois/Counter Mode" and GMAC is "Galois Message Authentication Code".

NOTE: I will not be able to explain specifically the use and details of some of these features. Cryptography is a detail field of computers and explaining cryptography is beyond the scope of this article.

The "Sequence Number IV Generator" is a special number generator used by some cryptography software.

The Linux kernel supports various cipher algorithms (CBC support), (CTR support), (CTS support), (ECB support), (LRW support), (PCBC support), (XTS support), (HMAC support), (XCBC support), and (VMAC support).

More Linux Related Stuff on: <http://www.linux.org>

The "CRC32c CRC algorithm" is used specifically with SPARC64 processors.

"CRC32c INTEL hardware acceleration" is another processor specific algorithm. It works on Intel processors with SSE4.2.

The kernel also offers many digests, ciphers, and other cryptographic software. Generally, allow the defaults unless you have a specific reason for enabling or disabling features.

NOTE: A digest (like MD5) generates a hash (sequence of characters) based on a file. Hashes are then used to check files. For example, if you download the Ubuntu installation ISO file from Canonical's website, you may want to know if the file on your hard-drive is an exact replica of the server's file. Users do this because the ISO may become corrupted during the long download. The hash is used to prove that the file is unchanged.

NOTE: A cipher is an encryption/decryption algorithm. Encryption is the process of making a file unreadable to anyone other than the intended recipients/owners. Decryption is the process used to view an encrypted file.

The Linux kernel also supports various compression algorithms that users are strongly recommended to enable (Deflate compression algorithm), (Zlib compression algorithm), and (LZO compression algorithm).

The kernel can generate random numbers which are needed for cryptographic software (Pseudo Random Number Generation for Cryptographic modules), (User-space interface for hash algorithms), and (User-space interface for symmetric key cipher algorithms).

"Hardware crypto devices" is a sub-menu that contains a list of drivers for hardware-based cryptography tools. This is hardware that has the algorithms in the firmware.

Various drivers for asymmetric public-keys exist in the "Asymmetric (public-key cryptographic) key type" menu.

Now, we can move on to the next entry on the main menu of the configuration tool (on menu-based interfaces). Virtualization is the ability to host an operating system. This means Linux (and other systems) can run another operating system as if the guest system is an application.

"Kernel-based Virtual Machine (KVM) support" allows the kernel itself to manage the guest system. Computers with Intel processors need this driver (KVM for Intel processors support) and AMD processors need (KVM for AMD processors support).

The memory management unit (MMU) for the Kernel Virtual Machine (KVM) can have an auditing system.

The guest's network can become faster with this feature enabled (Host kernel accelerator for virtio net).

After configuring the virtual machine abilities for the kernel, the last menu on the main screen of the configuration tool (when using a menu interface) is the last portion of the kernel to configure. This menu is for the "Library Routines", also called library functions. Parts of the kernel can be used as a linkable library. For example, XZ filters (compression algorithms) can be accessed by external

More Linux Related Stuff on: <http://www.linux.org>

programs. The different libraries are listed below.

NOTE: The CRC functions are mostly the same but with different features or performance. Generally, it is best to allow the defaults.

CRC-CCITT functions - The Cyclic Redundancy Check function tests for changes in raw data

CRC16 functions - The Cyclic Redundancy Check function tests for changes in raw data

CRC calculation for the T10 Data Integrity Field - The Cyclic Redundancy Check function tests for changes in raw data

CRC ITU-T V.41 functions - The Cyclic Redundancy Check function tests for changes in raw data

CRC32/CRC32c functions - The Cyclic Redundancy Check function tests for changes in raw data

CRC32 perform self test on init - The Cyclic Redundancy Check function tests for changes in raw data

CRC32 implementation (Slice by 8 bytes) - The Cyclic Redundancy Check function tests for changes in raw data

CRC7 functions - The Cyclic Redundancy Check function tests for changes in raw data

CRC32c (Castagnoli, et al) Cyclic Redundancy-Check - The Cyclic Redundancy Check function tests for changes in raw data

CRC8 function - The Cyclic Redundancy Check function tests for changes in raw data

* BCJ filter decoder - XZ decoder design for a specific processor where "*" is the processor. The kernel lists a few different architectures.

XZ decompressor tester - Debug functions for testing the XZ decoder

Averaging functions - Load average as seen in the output of "uptime"

CORDIC algorithm - hyperbolic and trigonometry functions

JEDEC DDR data - JEDEC Double Data Rate SD-RAM specification

Compiling and Installing kernel

Aloha! After you have spent a lot of time configuring your kernel to make the kernel you need, you can now compile it. The source code is C code in the form of plain text. This is readable to humans but not computers. Compiling the code converts the code to a form computers understand called binary (ones [on] and zeros [off]). Compilation will also make all of those kernel files one file called the kernel.

To compile the kernel, type "make" in a terminal that is in the same directory as the kernel's source code folders. This will take some time. Once done, the modules must be compiled by typing "make modules". To make the compiling process easier from the beginning, type "make; make modules". This will make the kernel and then the modules instead of the user coming back later to type "make modules".

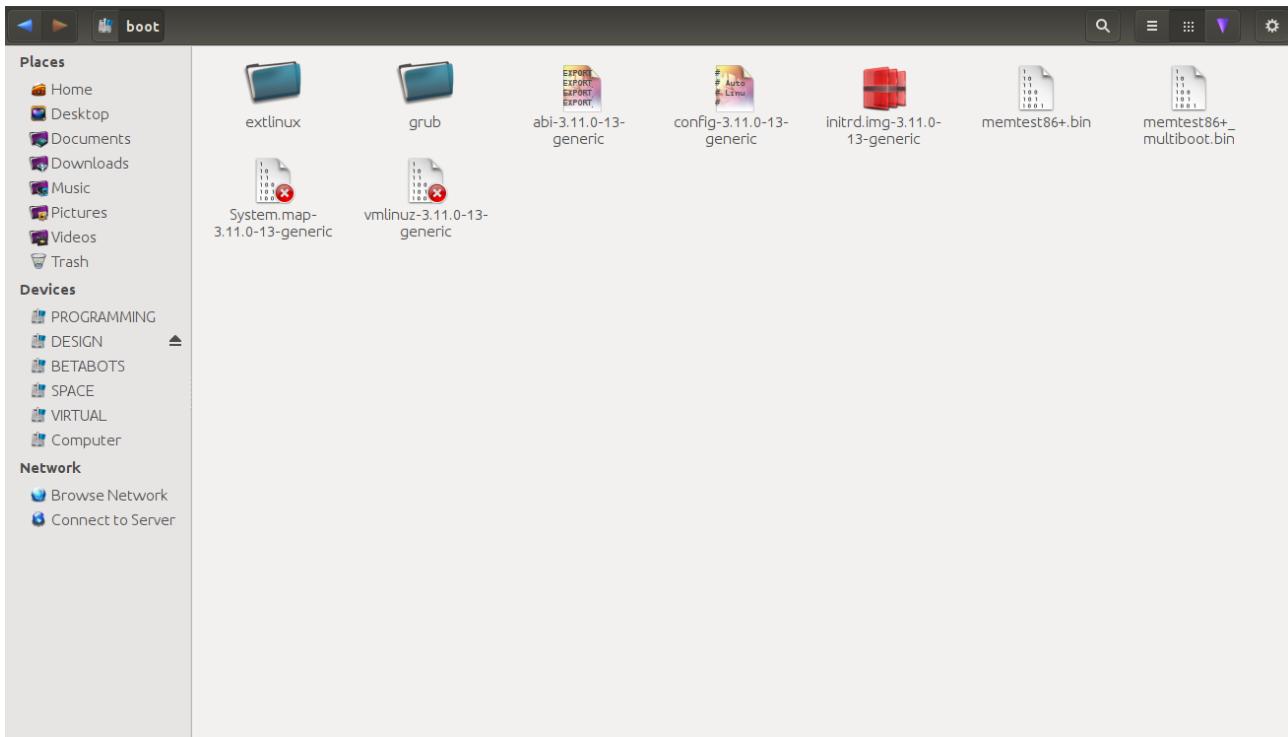
The terminal window shows the command 'make' being run in the directory '/media/collier/DESIGN/Linux/1-IN_PROCESS/Linux_Kernel/linux-3.9.4'. The output of the command is displayed, showing various compilation steps for different kernel components like syscalls, headers, and modules. The file browser window shows the directory structure of the kernel source code, including subdirectories for Documentation, drivers, firmware, fs, ipc, kernel, lib, mm, net, security, sound, tools, usr, virt, and modules.order. It also shows files like Kconfig, Makefile, and CREDITS.

```
collier@Nacho-Laptop:/media/collier/DESIGN/Linux/1-IN_PROCESS/Linux_Kernel/linux-3.9.4$ make
HOSTCC scripts/kconfig/conf.o
HOSTLD scripts/kconfig/conf
scripts/kconfig/conf --silentoldconfig Kconfig
SYSHDR arch/x86/syscalls/. /include/generated/uapi/asm/unistd_32.h
SYSHDR arch/x86/syscalls/. /include/generated/uapi/asm/unistd_64.h
SYSHDR arch/x86/syscalls/. /include/generated/uapi/asm/unistd_x32.h
SYSTBL arch/x86/syscalls/. /include/generated/asm/syscalls_32.h
SYSHDR arch/x86/syscalls/. /include/generated/asm/unistd_32_ta32.h
SYSHDR arch/x86/syscalls/. /include/generated/asm/unistd_64_x32.h
SYSTBL arch/x86/syscalls/. /include/generated/asm/syscalls_64.h
HOSTCC arch/x86/tools/relocs
WRAP arch/x86/include/generated/asm/clkdev.h
CHK include/generated/uapi/linux/version.h
UPD Trash include/generated/uapi/linux/version.h
CHK include/generated/utsrelease.h
UPD include/generated/utsrelease.h
CC kernel/bounds.s
GEN DESIGN include/generated/bounds.h
CC arch/x86/kernel/asm-offsets.s
GEN BETA include/generated/asm-offsets.h
CALL BACK scripts/checksyscalls.sh
HOSTCC scripts/genksyms/genksyms.o
SHIPPED scripts/genksyms/lex.lex.c
SHIPPED scripts/genksyms/keywords.hash.c
SHIPPED scripts/genksyms/parse.tab.h
HOSTCC scripts/genksyms/lex.lex.o
SHIPPED scripts/genksyms/parse.tab.c
HOSTCC scripts/genksyms/parse.tab.o
HOSTLD scripts/genksyms/genksyms
CC scripts/mod/empty.o
HOSTCC scripts/mod/nk_elfconfig
MKELF scripts/mod/elfconfig.h
CC scripts/mod/devicetable-offsets.s
GEN scripts/mod/devicetable-offsets.h
HOSTCC scripts/mod/file2alias.o
HOSTCC scripts/mod/modpost.o
```

WARNING: Before you install a kernel, backup all of the important data. Be sure to make a copy of /boot/ onto a FAT32 memory card. This helps to repair the system if the installation fails. FAT32 does not store permissions, so it will be easier to use a live disk to copy the files back. Remember to set the original file permissions and executable bits.

Once the compilation has finished successfully, we can then install the kernel to the local system (I will explain how to install the kernel on other systems in a moment [cross-compiling]). In the same terminal, after compilation, type "make install". This will place some files in the /boot/ directory. The "vmlinuz" (or some similar name) is the kernel itself. "initrd" is the temporary RAM-based filesystem that is placed in memory and used during boot-up. The "System-map" contains a list of kernel symbols. These are global variables and functions used in the kernel's code. "config" is the configuration file for the kernel. grub.cfg will automatically be updated. However, some other bootloaders may need to be manually configured. The kernel installer can configure the Grub, LILO, and SysLinux bootloader automatically. Bootloaders like BURG may need to be manually configured. The modules must also be installed by typing "make modules_install".

More Linux Related Stuff on: <http://www.linux.org>



NOTE: Both the kernel and modules can be installed using one line - “make install && make modules_install”

```
IHEX2FW firmware/emi26/loader.fw IHEX2FW firmware/emi26/firmware/fw IHEX2FW firmware/emi26/bitstream/fw
IHEX2FW firmware/emi62/loader.fw IHEX2FW firmware/emi62/bitstream/fw IHEX2FW firmware/emi62/spdif/fw
IHEX2FW firmware/emi62/midt/fw IHEX2FW firmware/kaweth/new_code.bin
IHEX2FW firmware/kaweth/trigger_code.bin IHEX2FW firmware/kaweth/new_code_fix.bin
IHEX2FW firmware/kaweth/trigger_code_fix.bin IHEX2FW firmware/tl_3410/fw
IHEX2FW firmware/tl_5052/fw IHEX2FW firmware/mts_cdma/fw
IHEX2FW firmware/mts_gsm/fw IHEX2FW firmware/mts_edge/fw
IHEX2FW firmware/edgeport/boot/fw IHEX2FW firmware/edgeport/bot2/fw
IHEX2FW firmware/edgeport/down/fw IHEX2FW firmware/edgeport/down2/fw
IHEX2FW firmware/edgeport/down3/btn IHEX2FW firmware/whiteheat_loader/fw
IHEX2FW firmware/whiteheat/fw IHEX2FW firmware/keysan_pda/keysan_pda/fw
IHEX2FW firmware/keysan_pda/xircom_pgs/fw IHEX2FW firmware/cptia2/stv0672_vp4.bln
IHEX2FW firmware/yam/1200/bin IHEX2FW firmware/yam/9600/bin
collier@Nacho-Laptop:/media/collier/DESIGN/Linux/1-IN_PROCESS/Linux_Kernel/linux-3.9.4$ make modules
make[1]: Nothing to be done for 'all'.
make[1]: Nothing to be done for 'relocs'.
      CHK    include/generated/uapi/linux/version.h
      CHK    include/generated/utsrelease.h
      CALL   scripts/checksyscalls.sh
      CC    scripts/mod/devicetable-offsets.s
      GEN   scripts/mod/devicetable-offsets.h
      HOSTCC scripts/mod/file2alias.o
      HOSTLD scripts/mod/nodpost
Building modules, stage 2.
MODPOST 3739 modules
collier@Nacho-Laptop:/media/collier/DESIGN/Linux/1-IN_PROCESS/Linux_Kernel/linux-3.9.4$
```

Once that process is complete, the user can ensure the kernel was installed by restarting the system and typing "uname -r" in a terminal when the system is back on. If the system fails to boot or uname reports a different version number than expected, the issue may be due to one of many issues. Either the

More Linux Related Stuff on: <http://www.linux.org>

bootloader was improperly setup, feature/configuration conflict, compilation error, improperly installed, or some other reason. The best way to start finding the source of the issue is to look at the systems logs (if the system boots up enough to produce logs). "dmesg" is a command that prints the kernels logs to the screen. Look for any errors, warnings, or unexpected results. If the system does not boot or does not boot-up enough to produce logs, use a live Linux disc to perform diagnostics and repairs. If all else fails, compile the kernel again and make sure you installed the kernel as Root or used "sudo".

NOTE: The best way to repair such a system is to use a live Linux distro to remove the new/broken kernel and then manually fix (or paste a backup) Grub's files.

Some Linux users like to have the documentation installed as well, but this is not required. For those that like to have the documentation installed, type this line where version is the kernel version - "install -d /usr/share/doc/linux-VERSION && cp -r Documentation/* /usr/share/doc/linux-VERSION" (VERSION is the kernel's version number). Obviously, Root privileges are required.

To compile a newer kernel with the same features as your current kernel, then type this command "zcat /proc/config.gz > .config". This file may not exist, if so, you may be able to ask the developers of your distro/kernel for the file. The "zcat" command uncompresses the data and places it in the file ".config". Remember to type where you want ".config". This file is to be placed in the Linux kernel directory and allow it to replace the current file. Then, compile and install the kernel as you normally would.

Cross-compiling is slightly different. Configure the kernel for the intended system. Make sure that when the kernel was configured, that it was configured with cross-piling in mind. When cross-compiling, there are two terms to be familiar with. "Host" is the system performing the compilation. The "Target" is the system that will receive the new kernel. Make sure that the host system has the proper compilers. For example, to cross-compile for ARM systems, users will need gcc-arm-linux-gnueabi on the host system. Generally, the developer can do a search in their package manager or Google for the proper/best cross-compiler for their needs. The specific command used to cross-compile for ARM systems is "make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-". The "ARCH=arm" refers to the target's processor type and "CROSS_COMPILE" declares the cross-compiler. Notice that the cross-compiler is missing the "gcc-" at the beginning and ends in a dash. That is the format users must use when using the cross-compiler as a parameter. The modules can be cross-compiled by typing "make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- modules". To install the kernel on the target system, copy the Linux kernel folder to the target system. Once the files are on the target system and a terminal is open in that directory, type "make install && make modules_install". Of course, you must be Root or use "sudo".

INFO: Kernel.org hosts a list of supported cross-compilers
(<https://www.kernel.org/pub/tools/crosstool/>).

FUN FACT: Some Linux distros store the kernel's config file in the /boot/ directory. Developers wanting to compile a kernel with the same settings as their current kernel can copy the file to their build directory. cp /boot/config-\$(uname -r) /PATH/TO/.config

Compilation and Installation Summary:

Standard:

Code:

```
make && make modules && make install && make modules_install
```

Make a newer version or remix of your kernel:

Code:

```
zcat /proc/config.gz > .config && make && make modules && make install && make modules_install
```

Cross-compile:

Code:

```
make ARCH={TARGET-ARCHITECTURE} CROSS_COMPILE={COMPILER}; make A
```

Modules

Now that we have our new kernel installed, we may have a module we wish to manipulate. Modules allow users to get extra hardware support without making a new kernel. For example, if a kernel is made without HFS support, a user can still use the HFS filesystem without making a new kernel.

Just in case you do not fully understand modules, Window's equivalent to Linux's modules are called drivers. As an analogy - Linux is to Windows as module is to driver. However, many people still call modules "drivers". That is fine, people and search engines know what you mean. Sometimes, modules are referred to as Loadable Kernel Modules (LKMs) because they can be loaded without changing the kernel.

Modules are stored under `/lib/modules/` this directory has a directory for each installed kernel. The module files themselves end in ".ko" which stands for "Kernel Object".

NOTE: Not all ".ko" files are modules. Linux also uses ".ko" for Linux's analogy of Window's ".dll" files.

To access the module directory (via command-line) of the currently active kernel, type this command - `/lib/modules/$(uname -r)/`. The "`$(uname -r)`" will be replaced by the output of "`uname -r`", so the user does not need to know or memorize the name/version of the active kernel. This folder is well organized. In `./kernel/`, these are the following directories.

arch - Architecture specific modules

crypto - Cryptography modules

drivers - Many modules are stored here and are sorted by hardware type. For example, ATA modules are under the "ata" folder.

fs - Filesystem modules are kept here. For instance, the module file for the Minix filesystem is `/lib/modules/$(uname -r)/kernel/fs/minix/minix.ko`

lib - The library routines are stored under this directory

mm - Modules for managing memory and debugging the kernel are stored here

net - Network modules are stored here

sound - This is an obvious one

Some other folders may exist. For instance, I have a "ubuntu" directory which contains modules specific to Ubuntu or were added by the Ubuntu developers.

When managing modules, Root privileges must be used. So, use "sudo" or login as Root.

To load a module, use this command (when logged in as Root. Otherwise, remember sudo.)

More Linux Related Stuff on: <http://www.linux.org>

Code:

```
modprobe <MODULE>
```

For illustration, to use the HFS+ filesystem, load its module.

Code:

```
modprobe hfsplus
```

Include the "-v" parameter for verbose messaging and "-s" sends error messages to the syslog. To see what modules the desired module requires, use "--show-depends" as a parameter. To remove a module, use "-r".

To get information about a module, use the "modinfo <MODULE>" command. You may notice information about aliases. Some modules have an alias which can be used to reference the module. The real name and alias each work well. Use which ever you remember better.

The "lsmod" command lists the currently loaded modules. This is helpful when users need to ensure a module is loaded.

Some modules can be given special parameters before they are loaded. To view these parameters (and aliases) for a specific module, use this command

Code:

```
modprobe -c | grep <MODULE>
```

To set a parameter for a module at runtime (the module is already running/loaded), use this command

Code:

```
insmod MODULE PARAM_NAME=VALUE
```

If you wish to load a module and also set the parameter at the same time, use this command

Code:

```
modprobe MODULE PARAM_NAME=VALUE
```

Alternately, parameters can be set at runtime by changing files in /sys/module/. For illustration, to change a parameter for a bluetooth module, use something like this

Code:

```
sudo echo -n "VALUE" > /sys/module/bluetooth/parameters/Filename
```

If a module has issues loading, like an "unknown symbol error", the module may still be able to be loaded by using the "-f" parameter. Use this parameter with caution.

Code:

```
modprobe -f <MODULE>
```

To load modules from different kernel versions, use the same command above after you copy the

More Linux Related Stuff on: <http://www.linux.org>

module to the active kernel's module path. Or, try

Code:

```
insmod /PATH/TO/MODULE.ko
```

Now, you can enjoy your kernel even more with module manipulation.

Patches

Sometimes, the kernel developers may release a patch for a particular Linux kernel. It helps to know how to apply these patches which will usually fix bugs or enhance performance. Patches can be obtained from "kernel.org". Once a patch is downloaded to your local system, place it in the directory containing the folder of your kernel's source code. Ensure that the kernel and patch are compatible, meaning, they must be the same version. Patches are applied to the uncompressed source code before the kernel is configured.

To apply a patch, type the following where "PATCH" is the patches file name. Also, use this command while the active directory is the kernel's source code.

Code:

```
patch -p1 < ./PATCH
```

If a patch is accidentally applied or the user decides not use a patch, then the patch can be removed/undone in the following manner,

Code:

```
patch -R -p1 < ./PATCH
```

The "-R" reverses the patches changes.

Compressed patches can be applied using one of these commands depending on the compression type. Again, you must be in the directory with the kernel's code.

Code:

```
zcat ./PATCH.gz | patch -p1  
xzcat ./PATCH.xz | patch -p1  
bzcat ./PATCH.bz2 | patch -p1
```

However, patches can be applied other ways. Ksplice is an administrator tool that allows users to apply minor security patches to the kernel that is currently running. To apply the security patches, type "uptrack-upgrade -y" in a terminal. "uptrack-show --available" displays available patches and "uptrack-remove --all" removes the patches. Ksplice functions like a package manager. Ksplice searches its repositories for patches and then installs the patches. However, unlike a package manager, Ksplice applies these patches to the kernel in memory. So, if the computer reboots, the updates will not exist.

Kernel Names

When using patches, it is important to understand the name of the kernel. For example, kernel v3.12.5 is not the same as v3.12.5-rc2 or v3.12.5-mm. "-rc#" means release candidate. These are kernels that will soon be released after some final testing. "-mm" indicates the kernel is experimental, so such kernels may be unstable and are not suitable for mainstream use. "-git" is found on the daily snapshots of the kernel from GitHub.

Updating the Kernel

For those of you that dislike compiling your own kernel or are happy with the configuration of other developers, there are some options for you in getting a newer kernel.

The number one method of updating the system's kernel is via the package manager. Updating the kernel is just like updating any other software. Open your package manager and refresh the repository list to see the latest updates. If updates exist for your kernel, then select the updates and apply them. If you are using a command-line, then follow the normal procedures for applying updates.

To Update or not to Update

If you are unsure about whether an update is worth-while, ask yourself these questions -

Why do I want to upgrade?

Does my system have a bug that I want fixed?

Is there some hardware I want supported by my Linux system?

Do I have hardware that is partially supported?

What if the update goes wrong?

What if the update introduces problems or new bugs?

Would updating improve system security?

Will the system continue to function well without the update?

If you need help making a decision, read the changelog to see the changes in the newer version of the kernel. Go to Kernel.org to read the changelogs. Changelogs include detailed information on changes and improvements as well as bug fixes.

Overall, only perform updates/upgrades on important computers when necessary. When working for a large company (or any company), try to refrain from updating the kernel unless the server is at risk to malware and bugs.

Types of Kernels

Many Linux users are familiar with Linux being called "GNU/Linux". This means something special. "GNU" refers to the userland and "Linux" refers to the kernel. Now, you may be wondering if these two components can be swapped. Yes, they can. For example, in the beginning of Linux's history, the GNU community had no kernel at the time and Linux was just a kernel. So, they put the two projects together. Today, the GNU community has a kernel called "Hurd". They then swapped the Linux kernel with the Hurd kernel to make "GNU/Hurd" systems. For instance, Arch exists as "GNU/Linux" and "GNU/Hurd". Not only do various userland and kernel combinations exist, there are different Linux kernels. Also, Linux is not the only operating system with a module (obvious, right?).

Many kernels exist for various operating systems. Two main types of kernels exist - monolithic kernels and microkernels. Linux is a monolithic kernel and Hurd is a microkernel. **Microkernels** offer the bare essentials to get a system operating. Microkernel systems have small kernelspaces and large userspaces. **Monolithic** kernels, however, contain much more. Monolithic systems have large kernelspaces. For instance, one difference is the placement of device drivers. monolithic kernels contain drivers (modules) and place them in kernelspace while microkernels lack drivers. In such systems, the device drivers are offered in another way and placed in the userspace. There is a lot more to the definition and more differences, but these are the main defining characteristics.

One other type of kernel is called a **hybrid kernel** which lies on the boundary between monolithic kernels and microkernels.

Userland refers to the user-space applications. For instance, Mastodon Linux (FreeBSD/Linux) uses the Linux kernel, but has FreeBSD applications (userland).

Debian is a system that has many variants. All Debian systems use GNU, but may have different kernels. Most people have at least heard of the GNU/Linux form that has many derivatives (like Ubuntu). Some interesting forms of Debian include GNU/Hurd, GNU/NetBSD, and GNU/kFreeBSD (FreeBSD Kernel). Obviously, people need to be more clear on which Debian system they have when they need help fixing an issue.

NOTE: The Hurd kernel contains the GNU-Mach kernel.

Nextenta OS is an OpenSolaris system (GNU/kOpenSolaris). Once installed, it appears to be GNU/Linux with the GNOME user-interface and typical applications seen on Linux systems. However, Nextenta OS is not Linux.

MkLinux is technically not Linux. This system uses the Mach kernel. Even though the userland is specifically RedHat Linux, this does not make MkLinux a Linux distro. The Mach kernel is an example of a microkernel.

StormOS is another example of an operating system that uses the GNU userland, but is not Linux. StormOS is a GNU/Illumos system.

This information may now answer the popular question - "Is Android a Linux system?". Yes, **Android** is Linux, but not GNU/Linux. Android uses the Linux kernel and the Dalvik userland, thus making Android - "Dalvik/Linux". Android also uses a modified Linux kernel. Yes, there are different Linux

kernels, but they are all still Linux. By the way, MeeGo is also Linux.

Food for thought: GNU = "GNU is Not Unix". What does the "G" stand for in "GNU"?

Now, remember, I said there are different Linux kernels. All Linux kernels have come from the Vanilla kernel directly or indirectly. The Vanilla kernel is the Linux kernel that can be downloaded from Kernel.org. The Vanilla kernel is the mainstream, official kernel that is made and managed by Linus Torvalds.

NOTE: The Vanilla kernel is developed on GitHub.com.

RTLinux is a real-time microkernel from of Vanilla. Yes, not all Linux kernels are monolithic, but most are monolithic. (<http://www.rtlinuxfree.com/>)

μClinux not only refers to the distro, but also the specialized kernel for very small, weak microcontrollers.

Firefox-OS is a mobile operating system that uses the Gonk kernel which contains various needed drivers (modules) for mobile devices like phones. Gonk also has enhancements for the system's software.

DS-Linux is the kernel used by Nintendo for the Nintendo-DS consoles. Sony also has a special Linux kernel for their Playstation. The kernel is called "Runix" or "PSXLinux".

Linux-libre is a Linux kernel that completely lacks proprietary code and modules. Basically, it is the Vanilla Kernel with code removed. This is perfect for people that want to avoid all proprietary software. Beware, some device will not work with this kernel.

CoLinux is a specialized kernel that was modified to allow Linux to run with Windows at the same time. (<http://www.colinux.org/>)

Compute-Node-Linux (CNL) is a kernel for the Cray computers. The INK kernel is used in IBM's Blue Gene Supercomputer.

The *L4Linux* kernel is designed to run on the L4 microkernel.

L4Android is the kernel that Dalvik/Android used for version 2.2-2.3. This is a combination of Google's changes and code with L4Linux. (<http://l4android.org/>)

NOTE: These kernels are different not because they were configured or compiled differently. Rather, the source code itself was greatly changed.

Some individual versions of the Linux kernel are given nicknames/codenames, but these are not used in mainstream use. (http://en.wikipedia.org/wiki/List_of_Linux_kernel_names)

For some people, it is important to have a deep understanding of some of these different systems. In the next article, we will discuss Dalvik/Android's kernel.

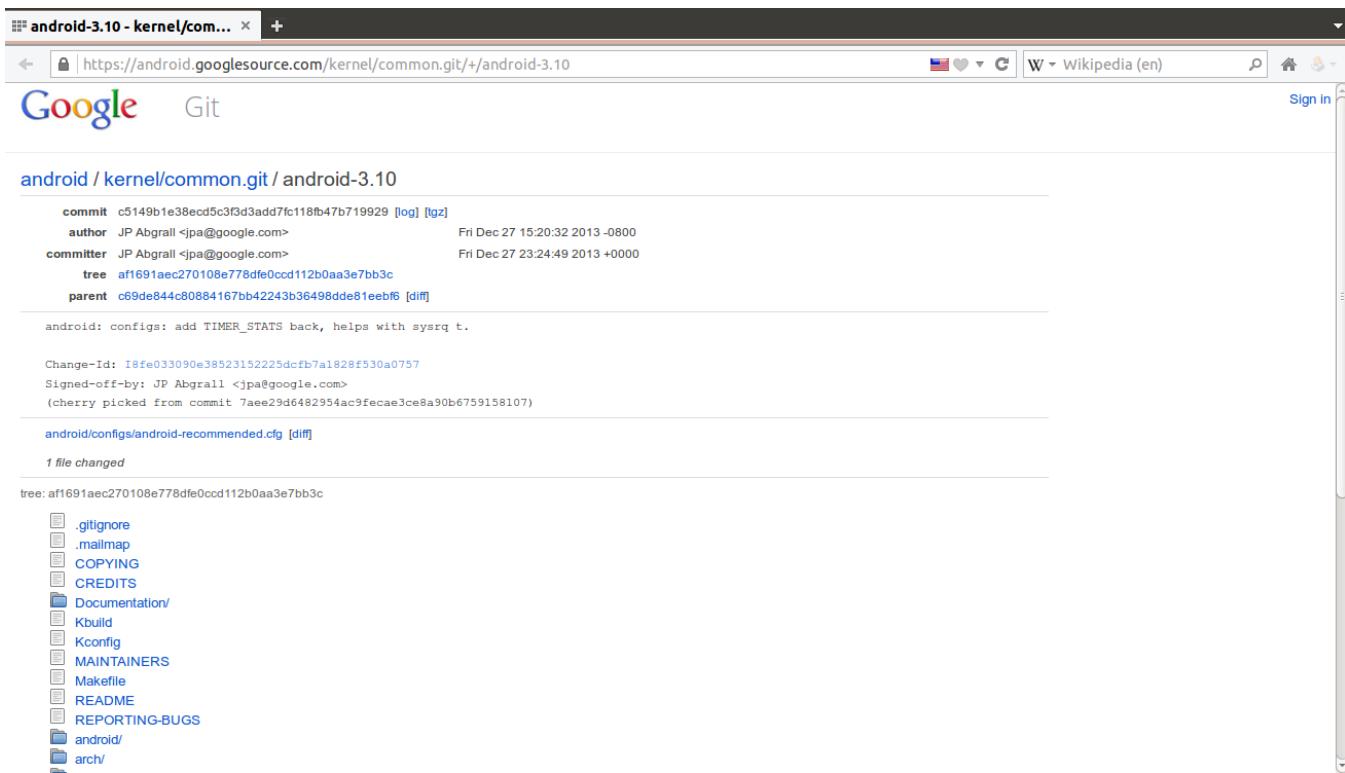
Linux Kernel Android

Aloha! Now that we have studied the Linux kernel very well and learned how to make our own, we will move on to a slightly different direction in this series. Many of you may be unaware of this, but Android is Linux. True, they are not quite the same, but *Android is Linux*. For example, Ubuntu is "GNU/Linux" while Android is "Dalvik/Linux". If an operating system uses the Linux kernel, then it is a Linux system. The userland (GNU and Dalvik) does not determine whether an OS is Linux or not. Android uses a modified Linux kernel. As we know, Android runs on phones. As you may remember from configuring the kernel, there were no drivers for phone devices (like small keypads, 3G/4G cards, SIM cards, etc.). The Linux kernel used in Android lacks drivers that would not be in phones and instead has drivers for phone devices. In other words, no Android system uses a Vanilla Kernel.

NOTE: Some people say "Android/Linux" instead of "Dalvik/Linux". However, both are valid.

Linux kernels for Android can be downloaded or viewed here -

(<https://android.googlesource.com/?format=HTML>). Just like the Vanilla kernel, the Android kernel is open-source software (due to the GPL license on the Vanilla kernel). The generic Android kernel can be seen here - (<https://android.googlesource.com/kernel/common.git/> /android-3.10) [for kernel v3.10] which looks very similar to the Vanilla kernel.



The screenshot shows a web browser window with the URL <https://android.googlesource.com/kernel/common.git/+/android-3.10>. The page displays the commit history for the android-3.10 branch. The first commit shown is:

```
commit c5149b1e38ecd5c3f3d3add7fc118fb47b719929 [log] [tgz]
author JP Abgrall <jpa@google.com>
committer JP Abgrall <jpa@google.com>
tree af1691aec270108e778dfe0ccd112b0aa3e7bb3c
parent c69de844c80884167bb42243b36498dde81eebf6 [diff]

android: configs: add TIMER_STATS back, helps with sysrq t.

Change-Id: I8fe033090e38523152225dcfb7a1828f530a0757
Signed-off-by: JP Abgrall <jpa@google.com>
(cherry picked from commit 7aee29d6482954ac9fecae3ce8a90b6759158107)

android/configs/android-recommended.cfg [diff]

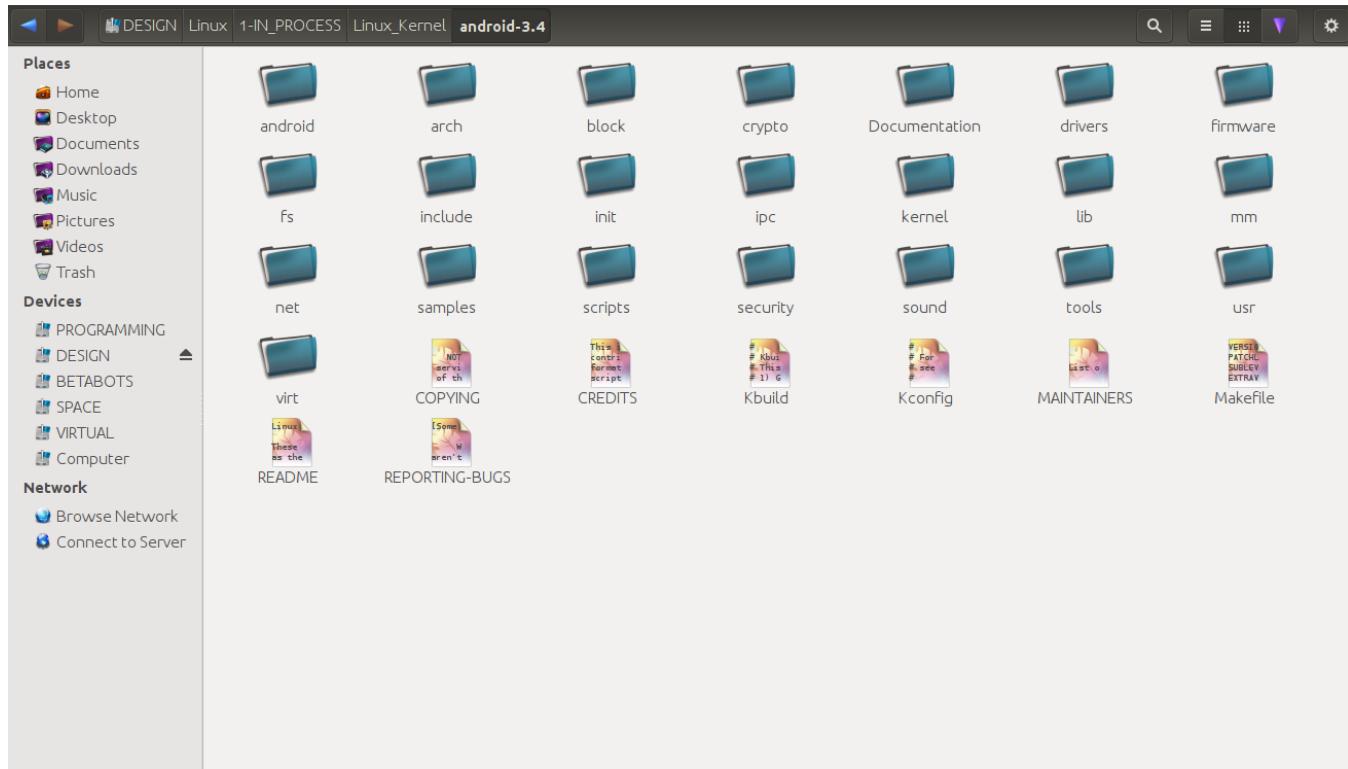
1 file changed

tree: af1691aec270108e778dfe0ccd112b0aa3e7bb3c
```

Below the commit details, there is a sidebar showing the directory structure of the kernel source code, including .gitignore, .mailmap, COPYING, CREDITS, Documentation/, Kbuild, Kconfig, MAINTAINERS, Makefile, README, REPORTING-BUGS, android/, and arch/.

NOTE: In this article and series, "Android Kernel" will be used interchangeably with "Linux kernel for Android" and other similar phrases.

The ./android/ directory contains a configuration file for making the Android kernel. These files tell the configuration tool which features/modules/drivers to enable or disable (<https://android.googlesource.com/kernel/common.git/> /android-3.10/android/configs/android-base.cfg).

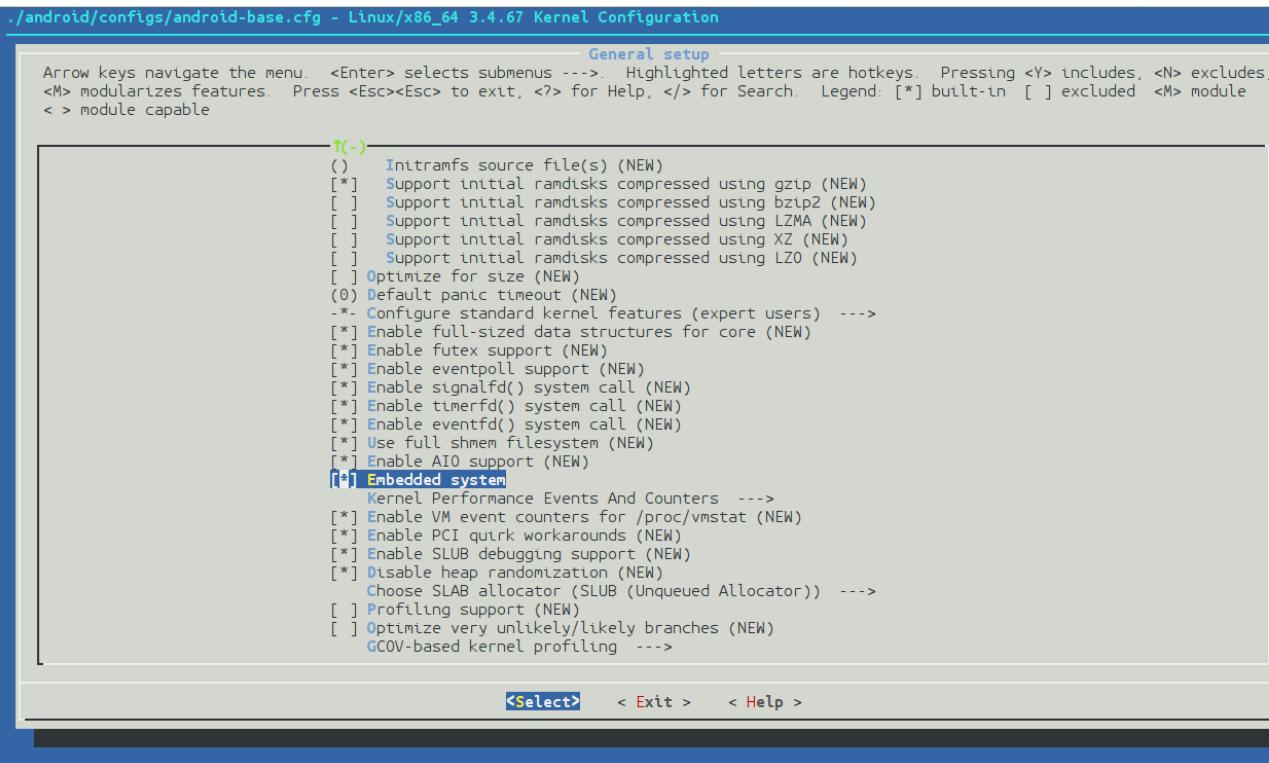


Some proprietary or special hardware may have a kernel designed specifically for them. For instance, kernels for Samsung phones can be downloaded here (<https://android.googlesource.com/kernel/samsung/>).

DOWNLOAD: To download an Android kernel, look to the "tgz" hyperlink near "Commit" around the top of the screen. Here is a download link - <https://android.googlesource.com/kernel/common.git/archive/android-3.10.tar.gz>

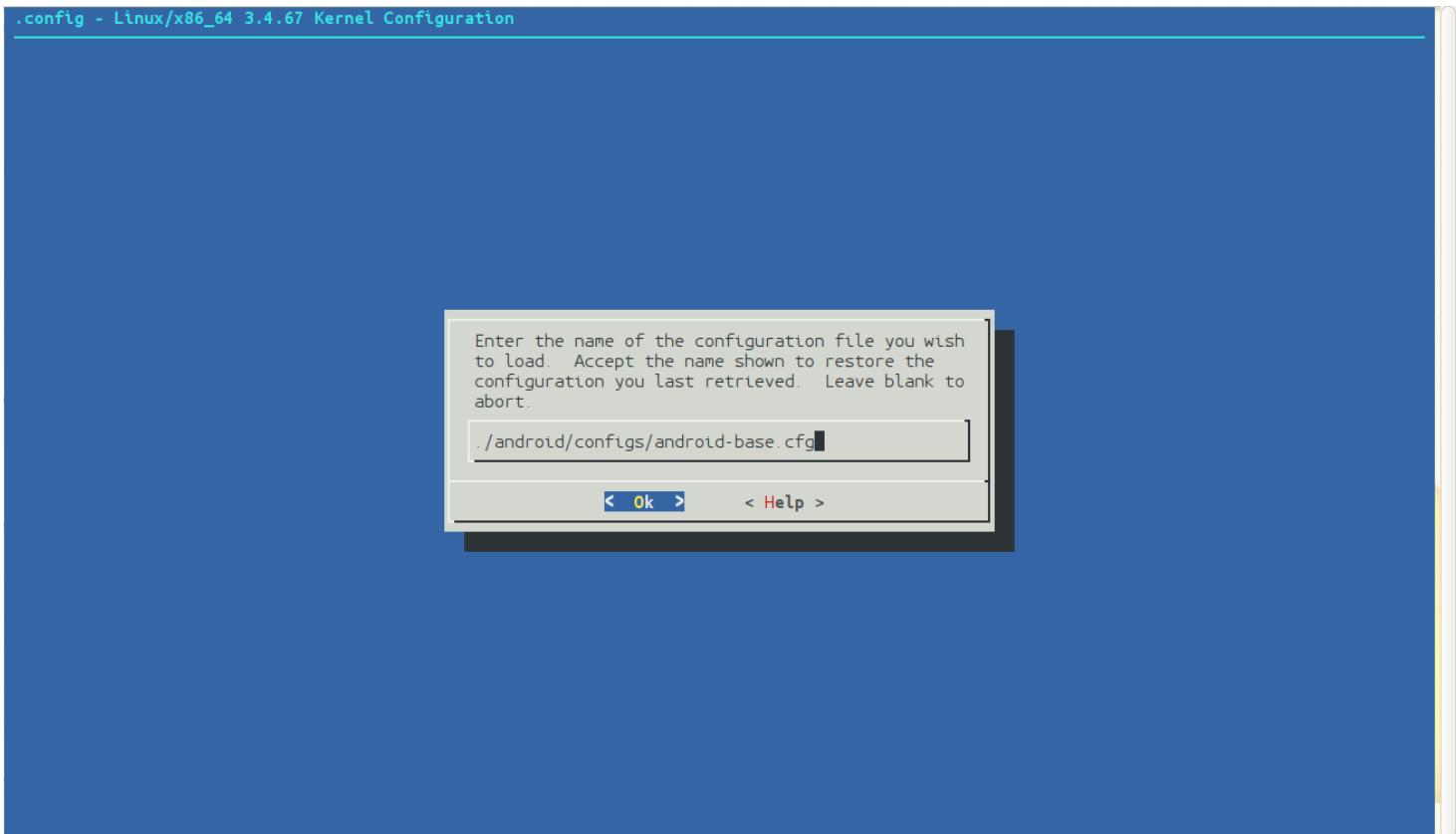
As seen in the ./security/ folder of the kernel's source code, Android supports the same Linux Security Modules (LSMs) that a desktop/server kernel would. By looking at any part of the Android kernel, the Android and Vanilla kernel are nearly the same. For illustration, the same filesystems are seen in Android's filesystem driver directory (./fs/) as in the Vanilla kernel.

To configure the kernel, do as you would for a Vanilla kernel - open a terminal in the source code directory and type "make <PREFERRED CONFIG TOOL>". I will use "make menuconfig" for my screenshots. Here is the most important step, load an alternate configuration file such as "./android/configs/android-base.cfg". Now, you can configure the kernel. However, be careful not to undo a feature/option that is important to Android (like ashmem). Also, you may see in the configuration tool (after loading the proper config file) that Android is an embedded Linux system. Keep this in mind when configuring the kernel. When configuring the Android kernel, it is very important to note that it must be cross-compiled.



NOTE: To know which features should not be changed, open the loaded configuration file in a text-editor (such as Gedit, Leafpad, etc.). The listed features should not be manipulated unless you have a specific purpose for doing so.

Once you have finished configuring the system, save the configuration as "./.config". Overwrite the file if needed (make a backup if the original "./.config" is important).



Before compiling, the host system must have JDK 6, Python2.7, and various developer's libraries installed on your system. Many developers recommend compiling the kernel on a 64-bit system (doing so on a 32-bit system is possible, but more difficult). Also, remember to set these variables before executing the "make" command -

Code:

```
export ARCH=arm  
export SUBARCH=arm  
export CROSS_COMPILE=arm-eabi-
```

The developer libraries I mentioned can be installed like this (for RedHat-based systems, use "yum" instead of "apt-get") -

Code:

```
sudo apt-get install git gnupg flex bison gperf build-essential zip curl libc6-dev  
libncurses5-dev:i386 x11proto-core-dev libx11-dev:i386 libreadline6-dev:i386 libgl1-  
mesa-glx:i386 libgl1-mesa-dev g++-multilib mingw32 tofrodos python-markdown libxml2-  
utils xsltproc zlib1g-dev:i386
```

Also, make sure that the proper cross-compiler is installed on your system (arm-eabi).

Making a symbolic link for this library may help on some systems -

Code:

```
sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so
```

NOTE: Some other libraries or links may be needed. If so, the compiler will give an error that a library or file is missing. Install or link as needed.

File Edit Package Settings Help

Reload Mark All Upgrades Apply Properties Quick filter Search arm compile

All	S	Package	Installed Version	Latest Version	Size	Description
Installed		binutils-arm-linux-gnueabi	2.23.52.20130913-0			GNU binary utilities, for arm-linux-gnueabi target
Installed (manual)		binutils-arm-linux-gnueabihf	2.23.52.20130913-0			GNU binary utilities, for arm-linux-gnueabihf target
New in repository		pkg-config-arm-linux-gnuea	4:4.7.2-1			manage compile and link flags for libraries for armel architecture
Not installed		pkg-config-arm-linux-gnuea	4:4.8.1-1			manage compile and link flags for libraries for armhf architecture
		gcj-4.8-arm-linux-gnueabihf	4.8.1-10ubuntu7cr			GCJ byte code and native compiler for Java(TM)
		dietlibc-dev	0.33~cvs20120325-			diet libc - a libc optimized for small size
		g++-4.7-arm-linux-gnueabi	4.7.3-1ubuntu1crc			GNU C++ compiler
		g++-4.7-arm-linux-gnueabihf	4.7.3-7ubuntu3crc			GNU C++ compiler
		g++-4.8-arm-linux-gnueabihf	4.8.1-10ubuntu7cr			GNU C++ compiler
		gccgo-4.8-arm-linux-gnueat	4.8.1-10ubuntu7cr			GNU Go compiler

GNU C++ compiler

[Get Screenshot](#) [Get Changelog](#) [Visit Homepage](#)

This is the GNU C++ compiler, a fairly portable optimizing compiler for C++.

This package contains C++ cross-compiler for armel architecture.

Sections Status Origin Custom Filters Search Results Architecture

68 packages listed, 3341 installed, 0 broken. 0 to install/upgrade, 0 to remove

If the configuration and compiling process went well, the Android kernel is ready for a device. The process of installing an Android kernel is a process that requires more knowledge in Android development and many more steps. So, I will not discuss the Android kernel that far.

Patches can be applied to the Android kernel's source code the same way as the Vanilla kernel, except the patches must be Android patches. The patches on Kernel.org will not work. They may work in some instances, but you must know what you are doing and only apply them to drivers/modules that are still Vanilla (unaltered from the original).

There are a few different kinds of Android kernels as seen on the Android kernel's Git page (<https://android.googlesource.com/?format=HTML>).

The **Goldfish kernel** (<https://android.googlesource.com/kernel/goldfish/>) is for emulated platforms like running a virtual Android system within a host system on a desktop computer.

The **MSM kernel** (<https://android.googlesource.com/kernel/msm/>) is for Qualcomm MSM chipsets.

The **OMAP kernel** is for the TI OMAP chipsets (<https://android.googlesource.com/kernel/omap/>).

Samsung hummingbird chipsets use the **Samsung kernel** (<https://android.googlesource.com/kernel/samsung/>).

The NVIDIA Tegra chipsets run on the **Tegra chipset kernel** (<https://android.googlesource.com/kernel/tegra/>).

The **Exynos kernel** is used by Samsung Exynos chipsets.

These various kernels mainly vary in their driver availability. For example, the Exynos kernel will have drivers for Exynos Samsung devices while the OMAP kernel does not.

Now, your understanding of Android should be larger than it was before. Feel free to send me more ideas on this article series or any Linux article in general. I hope to make this series continue for as long as I can. Mahalo!

The Linux Kernel: Intro to System Calls

Many GNU/Linux users have probably heard of systems calls. A system call is a special function/command that a program uses to communicate with the kernel of the operating system. The Linux kernel has a variety of system calls that it recognizes. Learning these system calls helps people to understand how GNU/Linux works. Even general/mainstream Linux users may find it interesting to know just how complicated the system is even though the user cannot see the complexity.

NOTE: Kernel calls is another name for system calls and so is syscall.

There are about six kinds of system calls (depending on how you want to classify them). These six are process control, information maintenance, communication, file management, memory management, and device management. "Information maintenance" is referring to system time, attributes of files and devices, and many other sets of information. "Communication" refers to networking, data transfer, attachment/detachment of remote devices.

When the Linux kernel receives a system call, it executes the command in kernel mode (privileged execution mode). This privileged mode is commonly called ring-0 (pronounced "ring zero").

NOTE: Some people get interrupts and system calls mixed up. A system call is a command while an interrupt is an event that causes the CPU to stop the current task and tend to the event. Hardware interrupts are called "interrupts" and software interrupts are called "traps" or "exceptions".

Some of you may be wondering, when an application is programmed, how does it get the code for the standard system calls. Well, the system calls come in the GNU C Library which is also called glibc. This is the library used for applications that run on systems using the Linux kernel and Hurd kernel or any GNU userland. Some derivatives are used in applications running on other kernels. For instance, after some major tweaking, glibc works on the NetBSD, OpenSolaris, FreeBSD kernel. FreeBSD and NetBSD typically use their own libc called "BSD libc". The modified glibc mentioned is used in the Debian system that uses the FreeBSD and NetBSD kernel (Debian GNU/FreeBSD and Debian GNU/NetBSD). Some other glibc derivatives and alternatives include

μClibc - This libc is used in mobile devices using the Linux kernel (except Android).

Bionic - Used in the Android OS. Bionic is based on BSD libc.

dietlibc - This is a lightweight libc for embedded systems.

Embedded glibc (EGLIBC) - The libc used in embedded systems is a tweaked/optimized version of the standard glibc.

klIBC (Kernel libc) - The Linux kernel uses klIBC while starting up. Once the kernel is loaded, it then uses glibc. However, not all distros use klIBC.

Newlib - Used in embedded systems.

These libraries provide various headers for C/C++ programming. The system calls are put in the code by importing a library as seen below. All of the system calls are not in one header, so an application only contains the system calls that it needs (unless there are some extra calls in the imported library that the program does not use).

Code:

```
#include <HEADER.h>
```

One reason why applications compiled for one operating system do not work on another is because the application uses different system calls. Wine is a compatibility layer (not an emulator) that allows Windows software to work on GNU/Linux and other Unix and Unix-like systems. This works because the Windows system calls are converted to the system calls that Linux recognizes (there are other mechanisms that make Windows applications work). If all systems used the same system calls, then some applications would be more cross-platform (some or many exceptions would exist). Think about source code. An application can be compiled on Linux, Solaris, and FreeBSD, but the binaries would only work on the operating system type on which the application was compiled.

Winelib is a libc used to compile with source code that only works on Windows systems. However, Winelib makes the compiled program work on Unix and Unix-like systems. Beware, Winelib is not perfect and may not work with some programs. Also, Winelib only works with 32-bit software. Usually, to use Winelib, the make-file for the source code needs some tweaking.

In the next articles, all of the system calls will be discussed.

The Linux Kernel: System Calls A-E

Once a Linux user learns about the different system calls, then it becomes clear just how complex Linux can be in completing common tasks. Below, many system calls are listed and explained. Notice the double parenthesis after each one. These exist because in computer programming (most languages) functions and commands that are to be executed end in "(" with parameters within the parenthesis. Thus, all of the system calls are functions that are defined and programmed in the kernel's source code.

NOTE: Most of the system calls will be discussed across a few articles, but not all of them will be covered. Some of the obsolete ones will be listed, but most will not be mentioned.

accept() - This system call creates socket connections. The similar system call **accept4()** supports flags. This syscall supports various protocols such as IPv4/6, Appletalk, IPX, and others (including sockets for communications between processes).

access() - This checks the permissions of a file before the calling process can access the file. This syscall first ensures the specified file exists. If so, then the system call checks if the process/user may read, write, and/or execute the file.

acct() - Process accounting is turned on and off using this system call. Process accounting is record keeping for executed commands. This allows admins to be aware of all commands that were executed, who/what executed them, etc.

add_key() - This adds or updates a key in the kernel's key management facility.

adjtimex() - This system call updates the kernel's clock using an algorithm by David L. Mills'. This system call can also get various information like the amount of microseconds between ticks, current time, offset, precision, etc.

alarm() - An alarm is set which will send a signal to a process.

alloc_hugepages() - An old system call (no longer used) that allocated and freed huge pages (large chunks of memory).

bind() - Newly created sockets get an address (sometimes called a name) from this system call. When connecting to a server, the client uses **bind()** on its side of the connection (the initiated side) and the server will use **connect()** on its side of the socket.

brk() - Memory can be given to or taken from running processes using this system call. A "program-break" is the last part of a processes memory on RAM. More memory can be given to the process by allocating more memory at the program-break, and deallocated the program's memory removes memory at the program-break.

cacheflush() - Flush the data cache in the specified address.

capget() - Get the capabilities of threads.

`capset()` - Set the capabilities of threads.

NOTE: A thread's capabilities refers to its attributes and permissions such as permissions to access particular network ports, execute Root programs, etc. A complete list of the capabilities may be found here `/usr/include/linux/capabilities.h`

`chdir()` - Yes, this command users regularly use to change the current directory is a system call.

`chmod()` - Surprise! Another commonly used command is a system call. This one changes file permissions.

`chown()` - At this point, you may not be shocked; another system call that changes file ownership.

`chroot()` - This popular shell command is also a system call. This changes the root directory.

`clock_getres()` - This retrieves the clock's resolution. Resolution is another term for precision. This syscall only works on POSIX clocks.

`clock_nanosleep()` - This system call is like the commonly used "sleep" command, but this system call pauses threads at the nanosecond level.

`clone()` - Like `fork()`, the process is forked, but not with the same results as `fork()`. There are several differences between `fork()` and `clone()`, but `clone()` makes a child process that uses the parent process's memory space while `fork()` gives the child process its own memory space.

`close()` - After a program is done writing or reading a file, the file should be closed to release memory and a file descriptor for reuse. The `close()` system call performs the closing of the file. Some documentation may say `close()` closes a file descriptor. This is also true. Closing a file descriptor just means a file descriptor is freed.

`connect()` - Create a connection to a socket.

`creat()` - This system call creates a file. No, this is not a typo. The system call really is called "`creat()`" without the second "e".

`delete_module()` - Kernel modules are unloaded by this system call. If the specified module is being used or is needed by other modules or the kernel itself, then the syscall will leave the module alone.

`dup()` - File descriptors can be duplicated with this system call. File descriptors would need to be duplicated when a thread viewing a file forks or when a command's output is redirected.

`epoll_create()` - Create a new file descriptor for a new instance of epoll.

`epoll_ctl()` - This system call is used to perform various tasks on an epoll file descriptor.

`epoll_wait()` - This system call waits until an event is performed on a specified epoll file descriptor. This syscall is important when software should only perform some action after a particular event happens to an epoll file descriptor.

eventfd() - The Event-File-Descriptor system call creates a file descriptor that is used to notify software about events or to make some software wait on some event.

NOTE: An abbreviation for file descriptor is “fd”. So, system calls that end in “fd” may relate to file descriptors.

execve() - Have you ever wondered which system call (if any) causes executable to run? Well, this system call is the one that does so. execve() can also execute scripts that begin with a valid hashpling. If needed, this syscall will call the Linux Dynamic Linker (ld.so) to set and link required libraries to the executable.

_exit() - Processes/threads use this system call to close themselves. Yes, there is an underscore at the beginning of this syscall. If a thread calls _exit(), then only that thread closes.

FUN FACT: Programs close in one of three ways – kill signal, fatal error, or calling _exit(). Notice that only one out of three is a graceful way to close. In other words, programs either willing close (_exit()), crash (fatal error), or they are murdered (kill signal). Wow, software has a harsh life (^u^).

exit_group() - All of a process's threads and associated threads are closed with this syscall. This is a special form of _exit().

In the next article, we will continue discussing the other system calls.

The Linux Kernel: System Calls F-G

In this next article, we will continue discussing the system calls. Do not worry, this will not be as long as the configuration part of the kernel series.

faccessat() - The permissions for the specified file is checked, but this is performed using a directory file descriptor.

NOTE: In the most simplest terms, a file descriptor is a special number used to access a file.

posix_fadvise() - (commonly called fadvice(), although the actual call is posix_fadvise()) This system call is used to optimize data access. Specifically, this syscall plans ahead what file will be accessed and how to get the data. This speeds up data access for the kernel.

fallocate() - Disk space of a specified file is manipulated by this kernel call. Obviously, since every filesystem type (XFS, EXT4, NTFS, tmpfs, etc.) is different, this syscall does not work on all filesystems. fallocate() also works on some pseudo/virtual filesystems like tmpfs.

fchmod() - This syscall is the same thing as chmod(). The difference lies in the fact chmod() accepts a path name and fchmod() accepts a file descriptor instead.

fchmodat() - This system call changes a file's permissions and the file is specified using a file descriptor. fchmodat() is exactly like chmod() with the difference being their accepted input, a file descriptor and path, respectively. fchmod() and fchmodat() work a little differently from each other.

fchown() - Just like chown(), the owner of the specified file is changed. However, fchown() knows the file by its file descriptor, not its path.

fchownat() - This syscall is just like fchown().

fcntl() - Manipulate the specified file descriptor.

fgetxattr() - This syscall gets the value of the specified extended file attribute.

finit_module() - Using a file descriptor, an ELF-image is loaded into the kernel space.

flistxattr() - Given a file descriptor, this syscall will list the extended attributes owned by the specified file.

flock() - Create or remove an advisory lock on the specified file (the file must be open).

fork() - Child processes are commonly created using this kernel call. With this call, the child process gets its own PID and memory space. Many other attributes are not inherited.

free_hugepages() - Free huge pages (large chunks of memory).

fremovexattr() - When using a file descriptor, this syscall can remove an extended attribute.

fsetxattr() - With a known file descriptor, an extended attribute can be set.

fstat() - The status of a file can be read with this syscall when given a file descriptor.

fstatat() - With a directory file descriptor, a file's status can be read.

fstatfs() - The statistics of a filesystem can be retrieved with this kernel call. This system call is directed to the filesystem in question by using a file descriptor of any given file on that filesystem.

ftruncate() - With a given file descriptor, this kernel call will truncate the specified file to a desired length. This may mean cutting the file, thus losing data, or enlarging the file by adding null bytes. Null bytes are designated as a backslash zero (\0).

futex() - Futex stands for Fast User-space muTEX. With this syscall, threads and processes can adhere to the futex standard so the executing code can wait for shared resources. Any code that calls futex() must be written in non-portable assembly instructions.

get_kernel_syms() - All of the exported module and kernel symbols can be read with this system call.

NOTE: A symbol is a function or variable.

get_mempolicy() - To get the NUMA memory policy for a process, use this syscall.

get_robust_list() - The robust-futex list can be retrieved with this kernel call.

getcpu() - This syscall allows the calling thread to be found on a specific CPU and NUMA node. This is like the thread yelling "I am over here!".

getcwd(), getwd(), and get_current_dir_name() - GET Current Working Directory. These three syscalls give the same result, but they each function a little differently.

getdents() - This system call gets directory entries.

getgid() - The syscall retrieves the real GID of the calling process.

getegid() - The syscall retrieves the effective GID of the calling process.

FUN FACT: getgid() and getegid() are claimed to NEVER fail according to the man pages. Do we agree with that, or has someone found an exception?

getitimer() - This syscall gets the current value of one of the timers of a process. Each process has three timers - ITIMER_REAL (decrements in real time), ITIMER_VIRTUAL (decrements during execution), and ITIMER_PROF (decrements while either the process or system executes). These three timers are called "interval timers".

getpeername() - The name of a connected peer socket can be retrieved using this syscall.

getpagesize() - The size of a regular page in memory can be known with this kernel call.

NOTE: In simplest terms, a page in memory is analogous to a block on a magnetic hard-drive.

getpgid() - This syscall gets the PGID of the specified process by using its PID.

getpid() - The PID of the calling process is returned.

getppid() - The PID of the calling process's parent is returned.

getpriority() - The scheduling priority of the specified program is returned by this syscall.

getresuid() - The RUID, EUID, and the SUID of the calling process is returned.

getresgid()- The RGID, EGID, and the SGID of the calling process is returned.

NOTE: GID = Group ID. UID = User ID. R = Real. E = Effective. S = Set.

getrlimit() - This kernel call returns the resource limit of a process.

In the next article, we will continue discussing system calls. Enjoy!

The Linux Kernel: System Calls G-M

Okay, here we have more system calls to discuss.

`getrusage()` - The amount of resources used by the specified process is given by this syscall.

`getsid()` - This returns the session ID. "sid" stands for Session ID.

`getsockname()` - The address (name) of the socket is returned by this kernel call.

`getsockopt()` - The options for a specified socket is listed by `getsockopt()`.

`gettid()` - The TID of a thread can be seen with this syscall. TID stands for thread identification.

`gettimeofday()` - The current time and timezone can be seen with this call.

`getxattr()` - With a given inode, this system call retrieves the extended attributes associated with the inode.

NOTE: Extended attributes are attributes not normally supported by the filesystem.

`init_module()` - Kernel modules are loaded with this syscall. This system call loads the module into the kernel space and then performs other needed tasks to prepare the module for runtime.

`inotify_add_watch()` - Given a file-path, this syscall creates or modifies an inotify watch.

`inotify_init()` - After an inotify watch is created, it must be started via `inotify_init()`.

`inotify_rm_watch()` - When an inotify watch is no longer needed, this syscall removes the watch specified by a watch descriptor (wd). When watches are made with `inotify_init()`, the wd is given.

`io_cancel()` - Asynchronous IO tasks are canceled with this syscall.

`io_destroy()` - Instead of canceling asynchronous IO tasks, they can be destroyed, meaning all asynchronous IO tasks associated with the given identifier will be canceled.

`io_getevents()` - The asynchronous IO events listed in the completion queue can be seen with this syscall.

`io_setup()` - Asynchronous IO contexts are made using `io_setup()`.

`io_submit()` - To queue asynchronous IO blocks, use this kernel call.

`ioctl()` - With a file descriptor for a device-file, the device's parameters can be changed.

`ioperm()` - The IO permission of ports are set with this call.

`iopl()` - This kernel call changes the IO privilege level of the process that executed this call.

`ioprio_get()` - The priority and IO scheduling class of threads can be seen with this call. This call can

return this information for one or multiple threads.

ioprio_set() - The priority and IO scheduling class of a thread can be set with this call.

ipc() - To execute System V IPC system calls, the Linux kernel uses ipc() to start a System V IPC call. Not all architectures support ipc(). For instance, this call is not seen in ARM systems. Obviously, when making cross-platform software, do not use ipc().

kcmp() - With two PIDs, this system call can identify what kernel resources (if any) are shared between two processes.

kexec_load() - This syscall sets up a kernel to be executed after the next reboot. This is useful for running a diagnostic kernel after a system crash.

keyctl() - Changes to the key management facility of the kernel is made via keyctl().

kill() - A kill signal is sent to the specified process. As you may have noticed, many system calls require a pid, file descriptor, or some other low-level (closer to the hardware/inner-workings) identification to know on what to do work. Since kill is like other syscalls, that is why it uses a pid. Yes, in a command-line users are using a system call (kill) to perform a low-level operation on the system.

lgetxattr() - This is just like getxattr() except this system call is used on links to get the extended attributes of the link itself.

link() - Hard links are created using link().

listen() - This is the system call that listens for connections on a socket.

lookup_dcookie() - Using a cookie, the full path of a directory entry can be seen. A cookie is a directory entry identifier.

lremovexattr() - The extended attributes of a symbolic link can be removed. The extended attributes are untouched in the file to which the link points.

lseek() - A file's offset is changed. This syscall identifies the file based on file descriptor (fd).

lsetxattr() - This syscall sets extended attributes on links.

lstat() - This call provides information about the specified link.

madvise() - This system call is used by applications to give the kernel advice on how the application wishes to use memory. The kernel typically maps out memory as it is needed and as the kernel sees fit. The madvise() syscall comes from an application that can probably run more efficiently if its memory usage is managed in a particular way. Notice that the syscall is advice, meaning the kernel may disregard the application's request.

mbind() - This syscall sets the NUMA memory policy.

migrate_pages() - All of the memory pages belonging to the specified node will be moved to another node

More Linux Related Stuff on: <http://www.linux.org>

in memory.

`mincore()` - This kernel call checks to ensure the needed pages of memory (a page of memory is like a block of data on a hard-drive) exist where they are expected. If memory is accessed but is missing, a page fault will result, thus causing some severe errors. Thanks to this syscall, such errors can be prevented when the syscall is used.

`mkdir()` - This commonly known shell command is actually a system call that creates a directory. (You probably already knew that)

`mknod()` - This syscall can make a file, device file, or a named pipe.

`mlock()` - A specified portion of a processes virtual address space can be set to remain on the RAM and not go to the swap area. `mlockall()` is used to lock all of the virtual address space and `munlock()` and `munlockall()` can undo those syscalls.

`mmap()` - This maps memory for processes. `munmap()` unmaps the memory.

`mount()` - Here is another shell command that is actually a syscall. As you all may know, this mounts filesystems whether virtual (pseudo filesystems like `tmpfs`), real (`ext4`, `fat32`), network filesystems (`NFS`), files (iso files like DVD images).

`move_pages()` - This is another syscall that moves pages, but this syscall moves the memory page-by-page rather than in bulk or whole nodes.

`mprotect()` - This syscall changes the protection of the calling process's memory. In memory, "protection" is analogous (just like/equal) to permissions of files on hard-drives.

`mq_notify()` - Processes can "subscribe" to certain system messages (notifications). This system call allows processes to do so to specific types of notifications in the message queue.

`mq_open()` - POSIX message queues can be made or opened using this kernel call.

`mq_send()` - Messages are sent to the message queue using this system call.

`mq_unlink()` - Message queues can be deleted using this syscall.

In the next article, we will continue discussing system calls.

The Linux Kernel: System Calls M-R

Aloha, here are some more system calls to discuss. Remember, this is not a complete list. Some syscalls have become obsolete and others are platform specific.

mremap() - Memory REMAP is a syscall that remaps a virtual memory address. This means the kernel call gets a section of data and changes the size and location of that data's allocated area in memory.

msync() - As many people may know, when a file is edited (for example, a plain text file), the file is loaded to memory and changes take place there. To save the changes to the hard-drive, msync() synchronizes the file on RAM with the older file on the hard-drive.

nanosleep() - Like the sleep command commonly used in shell scripts, this command suspends execution on that thread. However, this command works on the nano-scale level.

nfsservctl() - This is the interface for the NFS daemon.

nice() - That commonly used and known command "nice" is another syscall.

uname() - Here is yet another syscall that is sometimes used by the user in a command-line or script.

open() - This syscall opens files.

NOTE: Sometimes, the calling process is referred to as a local process and the other processes are remote. For instance, if both Firefox and Thunderbird are running on the same machine, Firefox refers to Thunderbird as a remote process as does Thunderbird to Firefox. Each process views themselves as local.

pause() - This kernel call makes the calling process pause until one of two events take place. These two events are the death of the process (like a kill signal) or receiving a signal.

pciconfig_iobase() - This call is used to get information about IO regions on memory.

perf_event_open() - The system's performance is monitored when this syscall is executed.

personality() - This syscall creates the process's execution domain. In computing, a personality is the way an executable behaves. This refers to the different system calls and application binary interfaces (ABI).

perfmonctl() - This kernel call is the interface for the performance monitoring unit (PMU) of IA-64 CPUs.

pipe() - This kernel call makes a pipe () which is a form of interprocess communication. This sends data from one process to another, and data does not go to the sender.

pivot_root() - The root filesystem can be changed using this system call. pivot_root() is commonly used to change the root from initrd.

poll() - This syscall watches file descriptors for ones that are ready for IO operations.

pread() - With a given offset, pread() reads a file descriptor (fd).

pwrite() - With a given offset, pwrite() writes to a file descriptor (fd).

preadv() - This system call can read a file descriptor and fill many buffers. preadv() is like read() and readv() combined.

prlimit() - This is getrlimit() and setrlimit() combined into one syscall, so this one call gets and sets a process's resource limits.

process_vm_readv() - This kernel call gets data from a specified process (by pid) and gives it to the calling process.

process_vm_writev() - The calling process uses this syscall to send data to a remote process.

pselect() - This is like poll(), watching for many file descriptors for one to be free for IO operations.

ptrace() - A process can control and monitor another process (if permissions permit). The calling process is called the tracer and the process being monitored is called the tracee.

pwritev() - This syscall has both the features of writev() and pwrite().

query_module() - The information about a module can be received with this syscall.

quotactl() - Disk quotas are managed with this syscall.

read() - This system call gets data byte-by-byte from the specified file descriptor and places them in the buffer.

readahead() - Files are placed in the page cache by this syscall.

readlink() - Gets the full real pathname of the file the link points towards.

reboot() - Obviously, this syscall reboots the system. When CAD is used (Ctrl+Alt+Del), this kernel call is executed.

The Linux Kernel: System Calls R-S

There are still many more kernel calls, as you will see. Each one is important to the functioning of the kernel and system as a whole. Some of these syscalls are defined by the POSIX standard and are used by other systems (like *BSD). Some systems use a system call that has the same name and performs the same function. However, such calls may use different code even between architectures of the same operating system.

`recv()`, `recvfrom()`, `recvmsg()` - These three syscalls are nearly the same. They all receive messages from connected sockets, but these calls do so in a different way.

`recvmsg()` - Like the three calls mentioned previously, `recvmsg()` gets messages from sockets. However, this syscall can receive multiple messages at once, while the other calls get one at a time. The code used to make `recvmsg()` came from `recvmsg()`. (Notice the number of "m"s)

`remap_file_pages()` - This system call creates a new mapping on memory. Specifically, `remap_file_pages()` sets up a nonlinear mapping, meaning the pages are not placed in order on memory.

`removexattr()` - The extended attributes of files are removed with this syscall. The needed parameters include the path of the file and the name of the attribute.

`rename()` - This syscall renames a file.

`request_key()` - Keys can be retrieved from the kernel's key-ring by using this system call.

`restart_syscall()` - Sometimes, syscalls are temporarily paused by a stop signal (typically SIGSTOP). To resume such syscalls, use `restart_syscall()`.

`rmdir()` - Empty directories can be deleted with `rmdir()`.

`rt_sigqueueinfo()`, `rt_tgsigqueueinfo()` - A signal and data are sent to the specified process using one of these system calls. Both of these calls are the same, but they differ in the accepted parameters. `rt_sigqueueinfo()` needs to know the tgid (Thread Group ID) while `rt_tgsigqueueinfo()` needs to know both the tgid and tid (Thread ID).

`sigaction()` - Signals sent to processes may need to be modified. This kernel call allows the calling process to change the desired result of a signal sent to a process.

`sigpending()` - This syscall allows the calling process to view pending signals.

`sigprocmask()` - This kernel call allows the calling process to view its masked signals.

NOTE: Masked signals are signals that are blocked.

`sigsuspend()` - This syscall is used to pause a process.

More Linux Related Stuff on: <http://www.linux.org>

`sched_get_priority_max()`, `sched_get_priority_min()` - Every scheduling policy (or scheduling algorithm) has a set priority range. These two syscalls return the maximum and minimum priority numbers (respectively) accepted by a policy.

`sched_setaffinity()`, `sched_getaffinity()` - The CPU affinity (CPU pinning) of a thread can be set or viewed with these syscalls, respectively. CPU affinity assigns a thread or process to a processor. For instance, on systems with multiple processors, processes and threads may not be processed by many CPU chips at once. Instead, code may stay with one CPU.

`sched_setparam()`, `sched_getparam()` - These syscalls allow the parameters of a schedule to be set and viewed for a process specified by its PID.

`sched_setscheduler()`, `sched_getscheduler()` - With a given PID, a processes scheduling policy (algorithm) can be set or viewed.

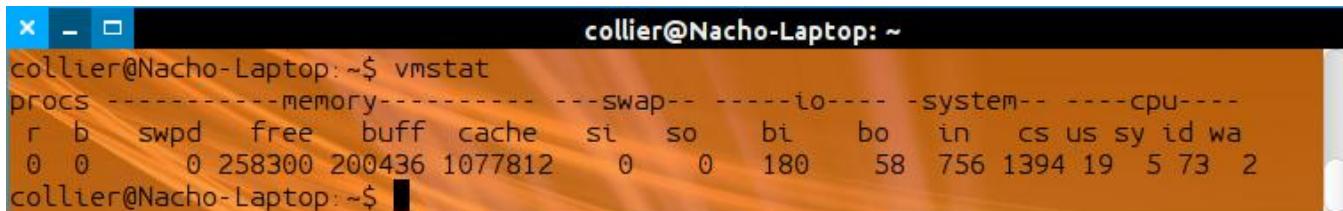
`sched_yield()` - The calling process will be placed at the end of the processor's task queue.

`select()` - This is another syscall used to monitor multiple file descriptors so that an IO task can be performed on the next available descriptor.

`send()`, `sendto()`, and `sendmsg()` - These syscalls perform nearly the same task, but they each have a slightly different method of functioning. `send()` is the same as `write()` except that the system call accepts flags while `write()` cannot. These send syscalls all use sockets, but different arguments.

`sendfile()` - This syscall copies data from one file descriptor to another. This is a faster way to copy files since this action is performed within the kernel. Most tasks completed in the kernel space complete faster than they do in the userspace.

FUN FACT: Are you wondering how many system calls are being made on your system right now? To figure out how many system calls are made per second system wide (on all processors), use the `vmstat` command and look at the "sy" column. For my system at the time of executing `vmstat`, I had five system calls running.



```
collier@Nacho-Laptop: ~$ vmstat
procs      memory          swap      io      system      cpu
 r b   swpd   free   buff   cache   si   so   bi   bo   in   cs us sy id wa
 0 0     0 258300 200436 1077812    0    0   180    58   756 1394 19  5 73  2
collier@Nacho-Laptop: ~$
```

`sendmmsg()` - More than one message can be sent down a socket using this kernel call. Most message-sending syscalls can only send one message down a socket at a time.

`set_mempolicy()` - This syscall is used by the calling process to change their NUMA-memory policy.

`set_thread_area()` - This syscall writes an entry on the local storage array of a thread (TLS = Thread Local Storage).

`set_tid_address()` - A pointer is created by this kernel call that points to the specified TID (Thread ID).

`setdomainname()` - This syscall sets the domain name and saves it as an array with each character in their own field. This value can be retrieved with `getdomainname()`.

`setfsgid()` - This kernel call changes the FileSystem Group ID (FSGID) which is a GID used when accessing network filesystems.

`setfsuid()` - `setfsuid()` is a lot like `setfsgid()` except that `setfsuid()` changes the User ID (UID).

`setgid()` - The calling process is given a Group ID.

`setgroups()` - The supplementary Group IDs are set for the calling process.

`sethostname()` - The hostname is set in the form of an array with one character per field.

`setns()` - A thread can be given a namespace by using this syscall.

`setpgid()` - The GID of a process is set with this kernel call.

`setpriority()` - The schedule priority of a process is set.

`setreuid()`, `setregid()` - These syscalls set the real and effective User or Group IDs.

`setresuid()`, `setresgid()` - These two syscalls are like their equivalent kernels calls above, but with the additional ability to set the Saved-User-ID (SUID) or Saved-Group-ID (SGID).

`setrlimit()` - A resource limit is set with this kernel call.

There are some more syscalls to be discussed. We are almost done.

The Linux Kernel: System Calls S-T

Aloha and welcome to the next system call article. Most of these calls in this article will probably be more easily understood than most we have discussed.

`setsid()` - The process group ID and session ID of the calling process are set to the PID of the calling process.

`setsockopt()` - Options for a specified socket are set using this syscall.

`settimeofday()` - The timezone and time are set via `settimeofday()`.

`setuid()` - The User ID (UID) is set with this call.

`setup()` - This deprecated syscall was once used to prepare devices and filesystems on the system and mount the root filesystem.

`setxattr()` - Extended attributes are set using this kernel call.

`shutdown()` - Many of you may think this system call shuts down the system. Well, guess what? It does not. Rather, this system call closes a socket or at least part of the socket.

NOTE: `reboot()` is the system call that reboots or powers off the system.

`sigaction()` - Signals can be viewed and changed with this syscall.

`signalfd()` - Signals can be accepted by a process via a file descriptor. However, such a file descriptor must be created first using this syscall.

`sigpending()` - The calling thread can view pending signals coming to it using this syscall.

`sigprocmask()` - A process's blocked signals can be changed via `sigprocmask()`.

`socket()` - The commonly discussed sockets are created with this syscall. A socket is a named pipe with additional abilities. The sockets commonly discussed in these syscall articles are sometimes called Unix Domain Sockets.

`socketpair()` - A pair of sockets are created using this system call.

`splice()` - Sockets can be spliced either for input or output.

`stat()` - This syscall returns the "status" of a file. The "status" is information such as the number of blocks owned by the file, the IDs of the owning group and owner, the storage device's ID, file size, number of hardlinks, and a few other pieces of information.

`statfs()` - Use this kernel call to get the "status" of a filesystem. Such status includes the number of free and total blocks, filesystem type, and other information pertaining to the filesystem itself.

stime() - This syscall sets the time in seconds since January 1st, 1970 (epoch).

subpage_prot() - Pages can be divided into subpages on memory. This system call allows permissions to be set to specific subpages, but only on PowerPC processors.

NOTE: Remember that blocks are to hard-drives as pages are to memory.

swapon() and swapoff() - These kernel calls turn the swap area on and off, respectively. Turning swap off may be done when the admin is changing its size on a live/active system or for various other reasons.

symlink() - Shortcuts (or soft-links) are made using this system call.

sync() - When files are changed, the edits are held in memory. The changes are written to the hard-drive after the sync() call is executed.

syncfs() - Like sync(), syncfs() causes changes to be written to the storage unit. However, only one file is changed while sync() tells all modifications to be written.

sync_file_range() - Like syncfs(), not all synchronizations are executed. However, only a portion of a single file is synchronized with the memory.

NOTE: When you make changes to a file (like opening a text file in Gedit), the changes are held in the buffer cache on memory.

sysfs() - Information about the current/present filesystems can be viewed with this syscall.

sysinfo() - An overview of the systems information (system statistics or sys stats) can be viewed with sysinfo(). Specifically, this data contains various memory space info, time since booting, buffer data, and some other helpful information.

syslog() - Kernel messages are viewed by syslog() which is also the system call that gives syslogd (a daemon) the data it places into logs.

tee() - The "tee" command used in shells is a system call that splits a pipe into two data pipes.

tkill() - Many users are aware that kill() kills a process, but few people know that individual threads can be killed. Such a task can be done with the syscall tkill().

time() - The current system time will be returned by the syscall in the form of seconds since January 1st, 1970.

NOTE: January 1st, 1970 is commonly referred to as "The Epoch". Now, when you read about some software (like a syscall) returning time in seconds since the Epoch, you know that that is what it means.

timer_create() - This syscall creates a per-process timer, which is a timer for each process.

timer_delete() - Timers can be deleted via timer_delete().

NOTE: Timers are identified by "timerid".

More Linux Related Stuff on: <http://www.linux.org>

`timer_getoverrun()` - This syscall is used for expiration notices (like stating a backup is three days late). This call calculates the time interval between the time the timer was due up to the time `timer_getoverrun()` is called. This time range is the overrun.

`timer_gettime()` and `timer_settime()` - These syscalls allow the time on timers to be set or viewed.

The Linux Kernel: System Calls T-Z

Here is the last set of standard syscalls. By the end of the article, you will know what I mean by standard and non-standard kernel calls.

times() - The various time data of processes are returned with this call. The "time data" includes user time, system time, children's user time, and the children's system time.

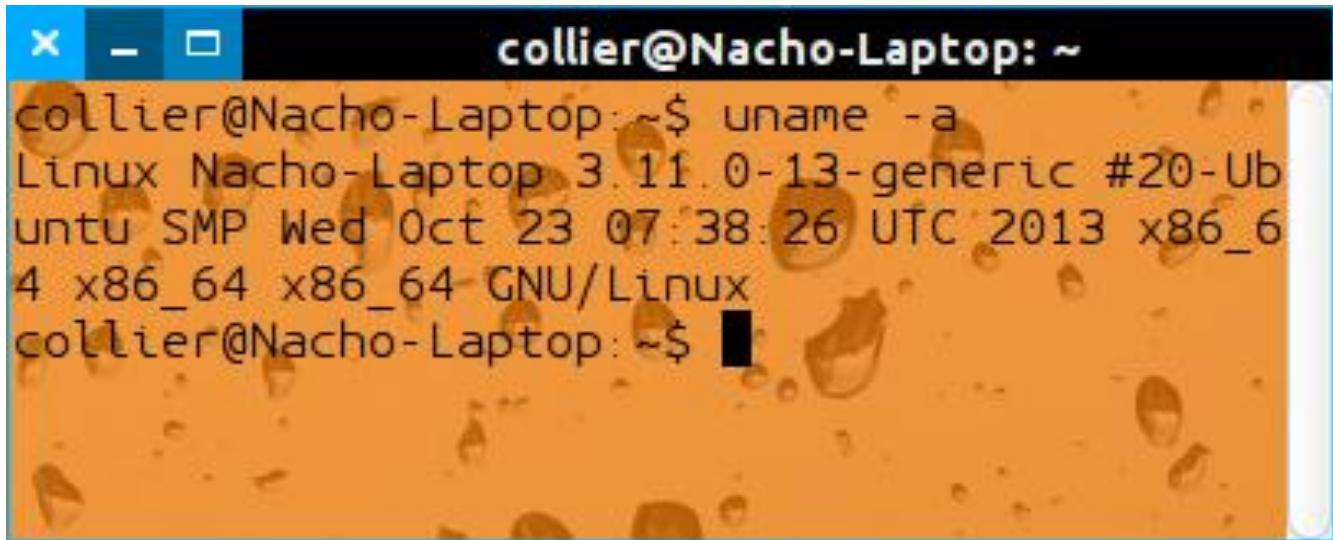
truncate() - A file (specified by its path) is resized to a particular size. If the file is larger than the desired size, then the extra data is lost. If the file is smaller than the needed size, then null bits are added to reach the final file size.

umask() - When files are created, permissions must be set on the file. The umask() kernel call tells the process what permissions to set on the file it is creating.

umount() - This is the kernel call used to unmount filesystems.

umount2() - This syscall is the same as umount(). However, this system call accepts flags as arguments.

uname() - This syscall is a commonly known and used command in the command-line. Uname() returns various information about the active kernel.

A screenshot of a terminal window titled "collier@Nacho-Laptop: ~". The window contains the command "uname -a" followed by its output: "Linux Nacho-Laptop 3.11.0-13-generic #20-Ubuntu SMP Wed Oct 23 07:38:26 UTC 2013 x86_64 x86_64 GNU/Linux".

```
collier@Nacho-Laptop:~$ uname -a
Linux Nacho-Laptop 3.11.0-13-generic #20-Ubuntu SMP Wed Oct 23 07:38:26 UTC 2013 x86_64 x86_64 GNU/Linux
collier@Nacho-Laptop:~$
```

unlink() - This syscall is used to delete soft links, sockets, pipes, and device files. The remove() syscall is used for files and rmdir() is needed to delete directories, but these calls do not work on soft links, sockets, pipes, or device files. That is why unlink() is needed. As you know, sockets and pipes are used by processes quite often, so unlink() is made to wait until the processes are done using the object it needs to remove. unlinkat() is the same as unlink() except for some minor differences in its functioning and the fact that it accepts flags.

unshare() - Sometimes, processes may share various resources (like virtual memory) after they fork. It may not always be desirable that the processes share so many resources. The unshare() syscall helps separate the processes more by making them get their own resources rather than sharing.

uselib() - As many GNU/Linux users know, many programs need various shared libraries (like the one under /lib/, /lib32/, /lib64/, /libx32/ ,and elsewhere). The uselib() system call is used by processes to load the needed library. This syscall loads libraries by path.

ustat() - The calling process can view the statistics of mounted filesystems using this syscall.

utime() - This system call is used to change the access and modification times associated with a file. The system call goes by inode to find the file. This kernel call is commonly used after a file is access or modified.

utimensat() - This kernel call is just like utime(), but the difference lies in the precision. utime() is precise down to the microsecond while utimensat() can handle nanoseconds.

vfork() - This syscall is like fork(), but with some differences. vfork() gives the child process its own page tables while fork() makes the parent and child processes share. The parent process still shares many of the attributes with the child process.

vhangup() - A hang-up is simulated in the terminal. This is needed so users will have a fresh terminal (tty) when they login after someone else has been logged into the terminal.

vm86() - This is a kernel call used to initiate a virtual 8086 mode which is commonly needed to run dosemu. The older (and obsolete) version of this system call has been renamed to vm86old(). If dosemu is able to run on your system, then your kernel has the vm86() system call. If not, then your platform cannot support this platform-specific kernel call.

vmsplice() - This syscall splices pages of memory to a pipe.

wait() - The calling process will be suspended until one of its children processes exit or are killed.

wait3() and **wait4()** - These syscalls are deprecated, so waitid() is used instead.

waitid() - This syscall makes the parent process pause until one of its child processes have a particular state change (see the NOTE a few lines below). waitid() is more precise than waitpid() since a specific state change can be the trigger for the parent to resume execution.

waitpid() - A child process can be specified by PID. Then, the parent process (which is also the calling process of waitpid()) will be paused until the specified child process experiences a state change of any kind.

NOTE: A "state change" refers to one of the following events - the termination of a process, a process resumes execution, or a process is stopped.

write() - This syscall writes data from a buffer to the specified file descriptor.

Now, we have discussed most of the official syscalls. The only ones I did not discuss were most of the obsolete or platform specific syscalls (although I mentioned some). In a future article, you will learn about some of the non-standard syscalls (calls not in vanilla kernels). Enjoy!

We are almost done discussing system calls. As you have probably noticed, system calls are very important in managing some common tasks (like time and system logs). Other system calls (like tee()) are used like commands in a terminal. Then, there are the system calls that only the kernel can use (like setup()). All of these kernel call work together to give the user a functioning system.

SNEAK PEEK: After syscalls, we will discuss the different types of ID, like PID, GUID, and many other that I hope you never heard about. (I have also been getting some ideas from the readers that email me {I give thanks to those readers.})

Non-Standard Syscalls

There is another set of kernel calls called the non-standard syscalls or unimplemented system calls. These kernel calls are not part of the vanilla kernel, or at least not in recent versions.

afs_syscall() - This is a seldomly used system call for OpenAFS. OpenAFS is an open-source alternative to the Andrew FileSystem, both of which are distributed filesystems. This syscall managed the Input/Output of OpenAFS. Most developers now use ioctl() instead.

break() - No info available. I cannot find any information about this syscall. If anyone knows anything, please share with us.

getpmsg() - This kernel call is the same as getmsg(). However, getpmsg() offered calling processes greater control than getmsg(). Both of these syscalls allowed a process to get a message from a stream.

gtty(), stty() - This system call is mostly obsolete due to ioctl(). gtty() may still be used for backwards compatibility, but in general, developers may no longer see this syscall. It is (or was) used to control terminal devices through the device files (/dev/tty*). stty() is an equivalent of gtty().

idle() - This kernel call makes the CPU idle.

lock() - This system call locks or unlocks access to registers.

madvice1() - This syscall is not used in the vanilla kernel, but is used in the Android-Linux kernels. madvice1() is an alias to madvice(), both of which advise the kernel on how to allocate and manage memory for the calling process.

mpx() - This kernel call creates and manages multiplexed files.

phys() - mmap() is now used instead of phys().

prof() - This syscall is a profiler which is used to measure the performance of the kernel.

putpmsg() - This is equivalent to putmsg(). Both of these syscalls are used to put messages on the message stream.

security() - Some Linux Security Modules (LSMs) use (or had used) this kernel call to define system calls for security purposes. Most LSMs now use socket use socketcall().

tuxcall() - This call comes from a TUX module and is sent to the kernel. The call asks the kernel to perform some task for the module. A TUX module is basically a server application/daemon in the form of a Linux module. Imagine an Apache server being a kernel module; that is essentially how TUX works.

vserver() - This is a virtualization system call that is used by this specialized kernel ([More Linux Related Stuff on: <http://www.linux.org>](http://linux-vserver.org>Welcome_to_Linux-VServer.org).</p></div><div data-bbox=)

Last Note on System Calls

This last "syscall" does not belong with the others, but I will mention it here. On some syscall tables (I will explain that in a moment), people may see a ni(), sys_ni(), ni_syscall(), or something of that manner. This is a syscall place holder. To better understand this, it is important to know that syscalls are assigned a syscall number. For instance, here is a syscall table for the v3.10 Android's Linux kernel (https://android.googlesource.com/kernel/.../v3.10-rc1-0-gd132r/arch/m32r/kernel/syscall_table.S). Notice that each syscall has a number associated with it. For instance, write() is syscall "6". When syscalls are given new numbers or are removed, the ni() syscall is a null kernel call that reserves that syscall number for later or private use. These syscall tables may also be called "system call maps". The Linux kernel (when executing) stores the syscall map in the "%eax" register. These numbers are probably only important to syscall developers or assembly programmers.

Linux IDs

Linux and many other operating systems use various identifiers. An identifier is a token or number used to designate some object or entity. Such an object or entity may be a user, group, process, etc. The Linux kernel prefers to use identifiers when referring to some object. For instance, the "kill" command is a system call that stops a process, but does so after being given the process's identifier. This syscall (as well as many other kernel calls) will only accept an identifier and not a name. The Linux kernel has a variety of types of identifiers as you will see.

Group ID (GID) - Every group is given a GID. Many Unix systems have different GID maximums (GIDs are not unlimited). Many distros have limit of 0-65535 or 0-4294967295. This limit is due to how GIDs are stored. As many of you know, Linux is written in C/C++ and Assembly. A GID is (depending on the system) an unsigned 16-bit integer or unsigned 32-bit integer. Every user must belong to at least one group. The group a user primarily belongs to is defined in /etc/passwd. Additional groups are listed in /etc/group.

Process ID (PID) - Every process whether it be a daemon, GUI application, kernel, etc. has a process ID. The kernel always has a PID of 0 and init (or an alternative) has a PID of 1. When a process is no longer running, its PID can be reused by a new process. There is a limit to the number of available PIDs, but the limit is so high that users would run out of memory and/or CPU resources before the PID limit can be reached.

Parent Process ID (PPID) - A process has a PPID which is the same number as the PID belonging to the processes parent. For example, if I open Xterm which has a PID of 123 and I start Firefox in Xterm by typing "firefox", then Firefox's PPID will be 123 which is Xterm's PID. The PPID is a way of indicating which process (the parent) started the child process.

User ID (UID) - Every user has a UID. Like GIDs, the number of UID's is limited by the type of integer that defines a UID. Again like GIDs, many distros have limit of 0-65535 or 0-4294967295 as an unsigned 16-bit integer or unsigned 32-bit integer, respectively. The Root user always has a UID of 0. The UIDs are stored in /etc/passwd.

Effective User ID (EUID) - When a process or user is trying to access or create some object, the kernel checks the EUID to see if the user or process has permissions to perform the desired action. In most cases, the EUID is the exact same thing as the UID. However, the EUID may differ from the UID. Assume a regular user uses a command that requires Root privileges. The user's permissions are temporarily upgraded (if allowed) to perform the special function (like "sudo"). Users cannot change their UID while being logged in. This is why the EUID is needed, so users can change their EUID when needed. Remember, the UID is used to identify the user/process and EUID is used to identify the permissions. In other words, the UID 0 is Root and an EUID of 0 is Root's file permissions.

FUN FACT: The "whoami" command shows who you are based on your Effective User ID, not your User ID.

Real User ID (RUID) - The Real User ID is like EUID, but is related to sending signals to processes. For instance, a process can only send signals to other processes with the same RUID. When a user starts a program, the program has the same RUID as the user. The RUID is the same as the UID, hence the name

"Real User ID". The RUID is not changed. Sometimes the EUID may need to change to the user's ID, and the EUID gets this info from the RUID. The RUID has the same limits as the UID (being an unsigned 32-bit integer).

Saved/Set User ID (SUID) - Remember how the Effective User ID (EUID) changes? Well, sometimes it may need to change to have the same value as the RUID temporarily. It would be inefficient to go through the process of re-elevating privileges again, so the EUID will go back to the ID saved in the SUID. For illustration, a process or user may perform some action that needs elevated privileges. The EUID is changed, but then something needs to be done under the RUID and then change back. Due to security reasons, a user/process cannot assign itself privileges back. This is where SUID comes into play. When, the EUID needs to switch back to the elevated privileges, the ID is retrieved from the Saved User ID (or Set User ID). However, such elevated privileges can be revoked even from the SUID. By default, the SUID will have the same value as the EUID.

NOTE: Saved IDs and Set IDs are the same.

Real Group ID (RGID) - The Real Group ID is like the Real User ID, but this applies to groups.

Effective Group ID (EGID) - This is just like an Effective User ID, but this is for groups.

Saved Group ID (SGID) - The SGID uses the same concepts as Saved/Set User IDs.

Session ID (SID) - A Session ID is used for communication purposes so only the authenticated processes can communicate together on that particular session only. This helps to prevent hijackers and malware from intercepting a data transaction. The only processes that can communicate on that "conversation" are those with the session ID for that particular communication session. Session IDs are common between HTTP connections.

Process Group ID (PGID) - A Process Group ID is the ID given to a set of processes that group together. The PGID is the same as the PID of the process that formed/declared the group via the setsid() kernel call. This is kind of like groups (as in GID), but PGID are not defined/preset in a file like Group IDs (/etc/passwd and /etc/group). Process groups are needed for inter-process communication purposes. Process groups prevent race conditions as well as other obstacles that may harm communication.

Filesystem User ID (FSUID) - When a filesystem operation is to be performed on a file or the filesystem itself, the FSUID is used rather than the EUID. Effective User IDs (EUID) are used for the data and FSUIDs are needed to modify the file itself (like change permissions and such).

Universally Unique ID (UUID) - This 16-octet number has a variety of uses. Many Linux users may have seen a storage unit's UUID. UUIDs are needed to provide some kind of general unique ID that is permanent (unless changed by the user or programmer). For example, Gparted can be used to change a memory card's or hard-drive's UUID. Mac addresses are not the same as UUIDs, although they seem the same. A Mac address is made up of six sets of hexadecimal pairs, like this - ##:#:#:#:#:#. An online UUID generator can be found here (<http://www.uuidgenerator.net/>). Or, you can do it locally as a fan of mine suggested on Google Plus - cat /proc/sys/kernel/random/uuid

Partition ID - The different types of filesystems have a unique ID. This ID is usually used by partition formatters to format a filesystem. For example, swap partitions have a partition ID of 82. However, not

More Linux Related Stuff on: <http://www.linux.org>

every filesystem uses a unique ID; xiafs, ext2, ext3, Reiserfs, and others share ID 83.

USB ID - Different USB devices also have an ID. The different IDs are listed here (<http://www.linux-usb.org/usb.ids>). Other types of hardware (like PCI devices) have ID of their own, but I will not bore you with all of the different kinds of hardware IDs since I am sure you get the point - PCI IDs, SCSI IDs, etc.

Descriptors

If you have read the part of the Linux kernel series that discusses system calls, then you are probably wondering a lot about file descriptors. In short terms, a file descriptor is some special indicator (in the form of an integer) used to access files. The descriptors are stored on an array that contains the file descriptor and various information about the file. The file descriptor may not point to a file, but it may point to a socket, pipe, directory, device, etc. Remember, in Linux, everything is a file. We will take a moment to try to thoroughly understand file descriptors.

Only the kernel can access the file descriptor table or index (which ever you prefer to call it). The file descriptor index keeps track of a file's location (i.e. /some/path), the process/user accessing the file, the processes PID, the file's type (i.e. pipe, directory, regular file, etc.), and some other information.

Many kernel calls usually access files using a file descriptor, just like processes are referred to by their PIDs and other IDs. Closed files do not have file descriptors just like how processes are only given a PID when they are open/executing. Files cannot get a file descriptor until they are opened/accessible.

The "lsof" command Lists Open Files by checking the list of file descriptors. Users can use lsof to view the file descriptor index. Users can also see what files particular processes have open. Users can also see the file descriptors processes have open via /proc/. In a process's proc folder (like /proc/2010/), users can see the file descriptors that the process is using by looking in the "fd" folder. For instance, to see the file descriptors that are being used by process 2010, type "ls /proc/2010/fd/".

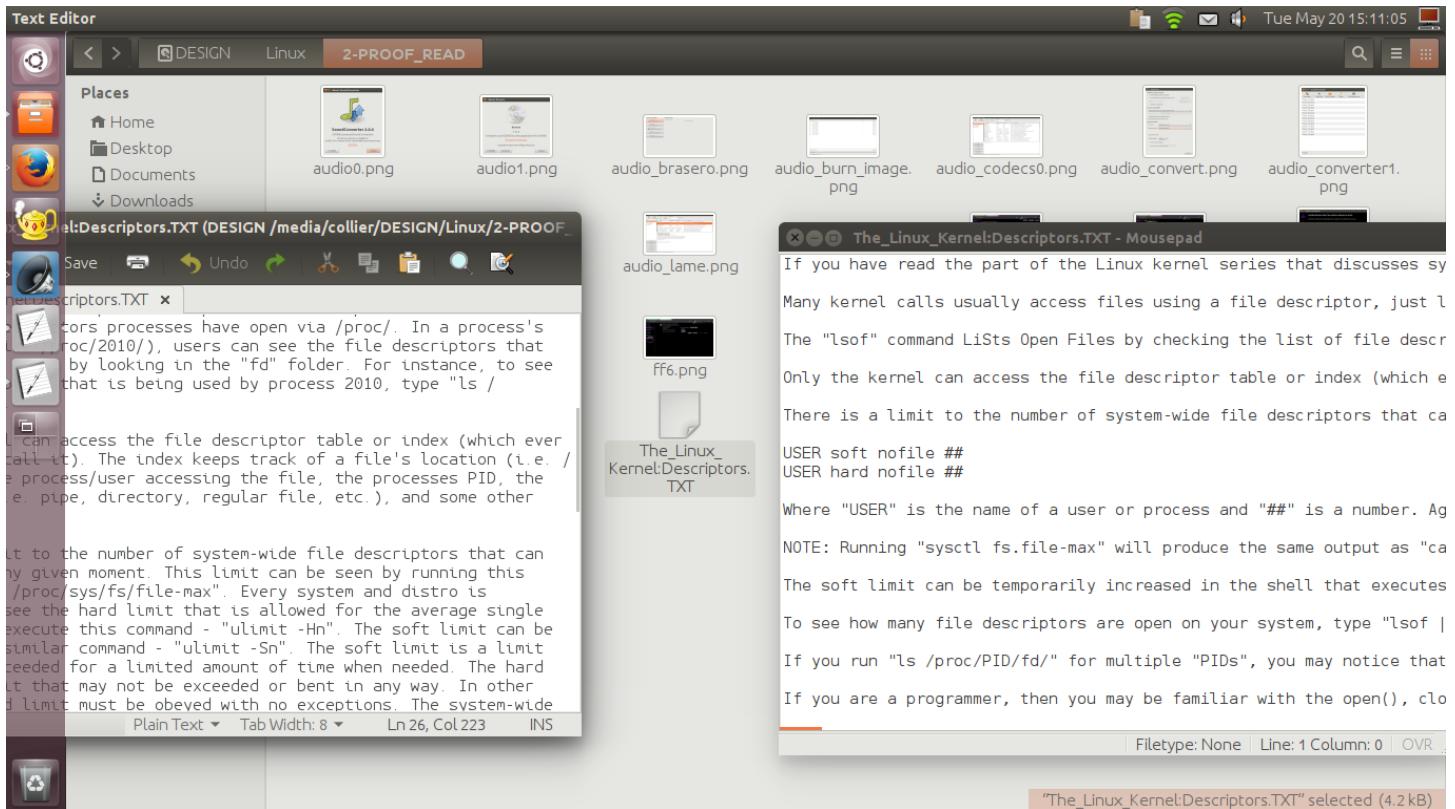
```
collier@Nacho-Linux:~$ sudo lsof | head -25
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
      Output information may be incomplete.
COMMAND  PID  TID    USER   FD   TYPE   DEVICE SIZE/OFF NODE NAME
init     1      1    root    cwd   DIR      8,1    4096    2 /
init     1      1    root   rtd   DIR      8,1    4096    2 /
init     1      1    root   txt   REG      8,1  265848  262211 /sbin/init
init     1      1    root   mem   REG      8,1   47712  1314836 /lib/x86_64-linux-gnu/libnss_files-2.19.so
init     1      1    root   mem   REG      8,1   47760  1314846 /lib/x86_64-linux-gnu/libnss_nis-2.19.so
init     1      1    root   mem   REG      8,1   97296  1314830 /lib/x86_64-linux-gnu/libnsl-2.19.so
init     1      1    root   mem   REG      8,1   39824  1314832 /lib/x86_64-linux-gnu/libnss_compat-2.19.so
init     1      1    root   mem   REG      8,1   14664  1314778 /lib/x86_64-linux-gnu/libdl-2.19.so
init     1      1    root   mem   REG      8,1  252032  1314864 /lib/x86_64-linux-gnu/libpcre.so.3.13.1
init     1      1    root   mem   REG      8,1  141574  1314881 /lib/x86_64-linux-gnu/libpthread-2.19.so
init     1      1    root   mem   REG      8,1  1845024 1314761 /lib/x86_64-linux-gnu/libc-2.19.so
init     1      1    root   mem   REG      8,1   31792  1314889 /lib/x86_64-linux-gnu/librt-2.19.so
init     1      1    root   mem   REG      8,1   39328  1314801 /lib/x86_64-linux-gnu/libjson-c.so.2.0.0
init     1      1    root   mem   REG      8,1  134296  1323367 /lib/x86_64-linux-gnu/libselinux.so.1
init     1      1    root   mem   REG      8,1  281552  1314775 /lib/x86_64-linux-gnu/libdbus-1.so.3.7.6
init     1      1    root   mem   REG      8,1   38920  1314823 /lib/x86_64-linux-gnu/libneth-dbus.so.1.0.0
init     1      1    root   mem   REG      8,1   96280  1314825 /lib/x86_64-linux-gnu/libneth.so.1.0.0
init     1      1    root   mem   REG      8,1  149120  1314737 /lib/x86_64-linux-gnu/ld-2.19.so
init     1      0    root   0u   CHR      1,3      0t0   1029 /dev/null
init     1      1u   root   1u   CHR      1,3      0t0   1029 /dev/null
init     1      2u   root   2u   CHR      1,3      0t0   1029 /dev/null
init     1      3r   root   3r   FIFO     0,8      0t0   8557 pipe
init     1      4w   root   4w   FIFO     0,8      0t0   8557 pipe
init     1      5r   root   5r   0000     0,9      0    7335 anon_inode
collier@Nacho-Linux:~$
```

If you run "ls /proc/PID/fd/" for multiple "PIDs", you may notice that file descriptors "0", "1", and "2" are under most or all PID folders. "0" is standard in (stdin), "1" is standard out (stdout), and "2" is standard

error (stderr).

```
collier@Nacho-Linux:~$ ls /proc/
1 1079 15 1929 2038 2157 23 2482 3048 392 53 7 891      bus      ioports      mounts      timer_list
10 1081 150 1988 2041 2159 2302 25 3057 41 533 701 895      cgroups    ipmi        mtrr        timer_stats
1003 11 1502 1994 2043 2166 2305 2509 3089 42 539 707 9      cmdline    irq         net         tty
10063 12 151 1997 2050 2167 2321 2543 3097 43 54 710 902      consoles   kallsyms   pagetypeinfo  uptime
10270 1254 152 2 2055 2170 2327 26 31 44 556 727 903      cpuinfo    kcore       partitions  version
1033 1255 153 20 2057 2177 2331 2613 32 45 57 728 904      crypto     key-users  sched_debug  version_signature
10354 1258 1553 2005 2058 2186 2335 2677 33 46 58 743 907      devices    kmsg        schedstat   vmallocinfo
10377 1276 1575 2008 2060 22 2354 27 34 467 59 75 959      diskstats kpagecount  self        vmstat
10383 1292 16 2010 2083 2248 2384 28 3436 47 60 76 9681      dma        kpageflags slabinfo
10428 1298 17 2019 2091 2262 2390 2871 356 48 600 8 9695      driver     latency_stats loadavg
10435 13 170 2022 2099 2272 24 2875 36 49 601 814 973      execdomains fb          locks        stat
10436 1344 171 2026 21 2276 2432 29 361 5 606 8191 98      filesystems mdstat      swaps
10500 1397 18 2029 2115 2277 2440 297 363 50 607 820 99      meminfo    fs          sys
10513 14 19 2030 2120 2279 2453 3 37 51 61 822 8526 8526      misc       interrupts sysrq-trigger
10530 1403 1907 2033 2154 2286 2458 302 38 52 62 8526 8526      modules    iomem
10578 1462 1925 2034 2155 2293 2464 3047 39 527 63 8808 buddyinfo      schedstat  stat
                                         map_files  mounts      numa_maps  pagemap      personality
                                         mounts      numa_maps  pagemap      sched      sessionid  statm
                                         numad      numa_maps  pagemap      stack      smaps      status
                                         oom_score  numa_maps  pagemap      syscalls   uid_map   timers
                                         oom_score_adj      sched      stack      syscalls   wchan
                                         root
collier@Nacho-Linux:~$ ls /proc/2010/fd/
0 10 12 14 16 18 2 21 23 25 27 29 30 32 34 36 38 4 41 43 45 47 49 50 52 54 56 7 9
1 11 13 15 17 19 20 22 24 26 28 3 31 33 35 37 39 40 42 44 46 48 5 51 53 55 6 8
collier@Nacho-Linux:~$
```

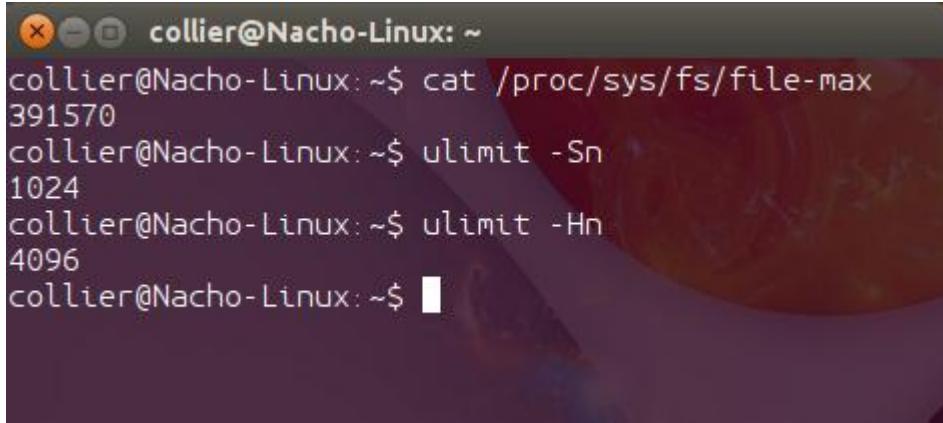
Processes can share file descriptors. For instance, a process may need to share a file descriptor with another process. Also, when a process is forked, the child process has the same file descriptors as the parent process.



There is a limit to the number of system-wide file descriptors that can be active at any given moment. This limit can be seen by running this command - "cat /proc/sys/fs/file-max". Every system and distro is different. To see the hard limit that is allowed for the average single user/process, execute this command - "ulimit -Hn". The soft limit can be viewed with a similar command - "ulimit -Sn". The soft limit is a limit they may be exceeded for a limited amount of time when needed. The hard limit is a limit that may not be exceeded or bent in any way. In other words, the hard limit must be obeyed with no exceptions. The system-wide file descriptor limit can be changed via this command - "sysctl -w fs.file-max=#", where "#" is the desired number. Then, using the same number, add this line to /etc/sysctl.conf - "fs.file-max = #". Reboot the system to apply the changes or run "sysctl -p" if you prefer not to reboot. Obviously, making such changes must be performed as Root. To change the hard and soft limits of a particular user, add or edit these lines in the /etc/security/limits.conf file -

```
USER soft nofile ##
USER hard nofile ##
```

Where "USER" is the name of a user or process and "##" is a number. Again, perform this action as Root.

A screenshot of a terminal window titled "collier@Nacho-Linux: ~". The window contains the following text:

```
collier@Nacho-Linux: ~$ cat /proc/sys/fs/file-max
391570
collier@Nacho-Linux: ~$ ulimit -Sn
1024
collier@Nacho-Linux: ~$ ulimit -Hn
4096
collier@Nacho-Linux: ~$
```

NOTE: Running "sysctl fs.file-max" will produce the same output as "cat /proc/sys/fs/file-max".

The soft limit can be temporarily increased in the shell that executes this command - "ulimit -n ###".

To see how many file descriptors are open on your system, type "lsof | wc -l". To see how many file descriptors are currently being actively used, look at the first number of this output - "cat /proc/sys/fs/file-nr".

If you are a programmer, then you may be familiar with the open(), close(), read(), and write() commands which are commonly used in C/C++, Python, and other languages. These commands interact with the file descriptors. open() and close() open and close a file descriptor, respectively. Without running these two commands the file descriptor is never made or left open, again respectively.

Hopefully, this article has helped readers to understand that file descriptors are basically PIDs for files. This knowledge should help make system calls easier to understand.

PulseAudio and Other Sound Components

Many Linux users have probably heard of PulseAudio, but many users may not completely understand why it is need or specifically how it works. PulseAudio is a part of the sound system and helps to make all of the sound components work together as you will see.

NOTE: This article will explain how the sound system works. This article does not explain installation or command-line parameters.

There are many ways to send sound from one location to another. Sound may come from a process/application (like Clementine, VLC, or a video game), the network (like a meeting with a friend over the Internet), from a microphone, or other source. Then, the sound may be sent to a file, over the network, to a speaker, to headphones, etc. How can Linux properly and accurately ensure sound is sent from point A to point B? The answer is PulseAudio (PulseAudio.org). PulseAudio is a daemon that acts as a proxy server for sound. All sound goes through PulseAudio. PulseAudio will then determine where the sound is meant to go. When a process makes a sound (like your music or video game), the application does not send sound straight to the speaker. Rather, it is sent to PulseAudio. If PulseAudio were to be killed with a kill command, then you would lose your sound. However, PulseAudio tends to restart itself on many systems, or users may have an alternative to PulseAudio.

Once PulseAudio receives the sound, what happens next? Well, once PulseAudio has a sound destined for the speaker or headphones, PulseAudio will send the sound to Advanced Linux Sound Architecture (ALSA) (www.alsa-project.org). ALSA is a sound API for the Linux kernel. ALSA runs in kernel-space and contains many sound drivers and features. Users can install various applications that allow them to control the audio system for various reasons (like JACK (<http://jackaudio.org/>)). AlsaMixer is an interface to ALSA used to configure sound and volume (<http://alsa.opensrc.org/Alsamixer>).

After ALSA makes any modifications to the sound (due to mixers and such), the drivers perform their low-level operations with the hardware. The sound is then sent to the desired destination. If PulseAudio sends sound over the network, then the client's ALSA process will perform such operations.

Some users have the Open Sound System (OSS) instead of ALSA. However, ALSA is the newest and most recommended sound-interface/sound-architecture for the kernel.

Sound can be sent via Bluetooth (like for headphones) thanks to the "bluetooth-alsa" software (<http://bluetooth-alsa.sourceforge.net/>).

As with anything, there are exceptions. Processes can send sound directly to ALSA, thus bypassing PulseAudio. Alternately, processes can send data to ALSA's own sound server. However, most systems use a "PulseAudio to ALSA to hardware" approach.

This is a little off topic, but I am sure some of you have thought of this while reading this article. So, I will briefly inform you on some related software. GStreamer is a multimedia-framework that provides many codecs and helps link various sound components. Phonon is like a Qt/KDE alternative to Gstreamer (both of which have entirely different code and teams of coders).

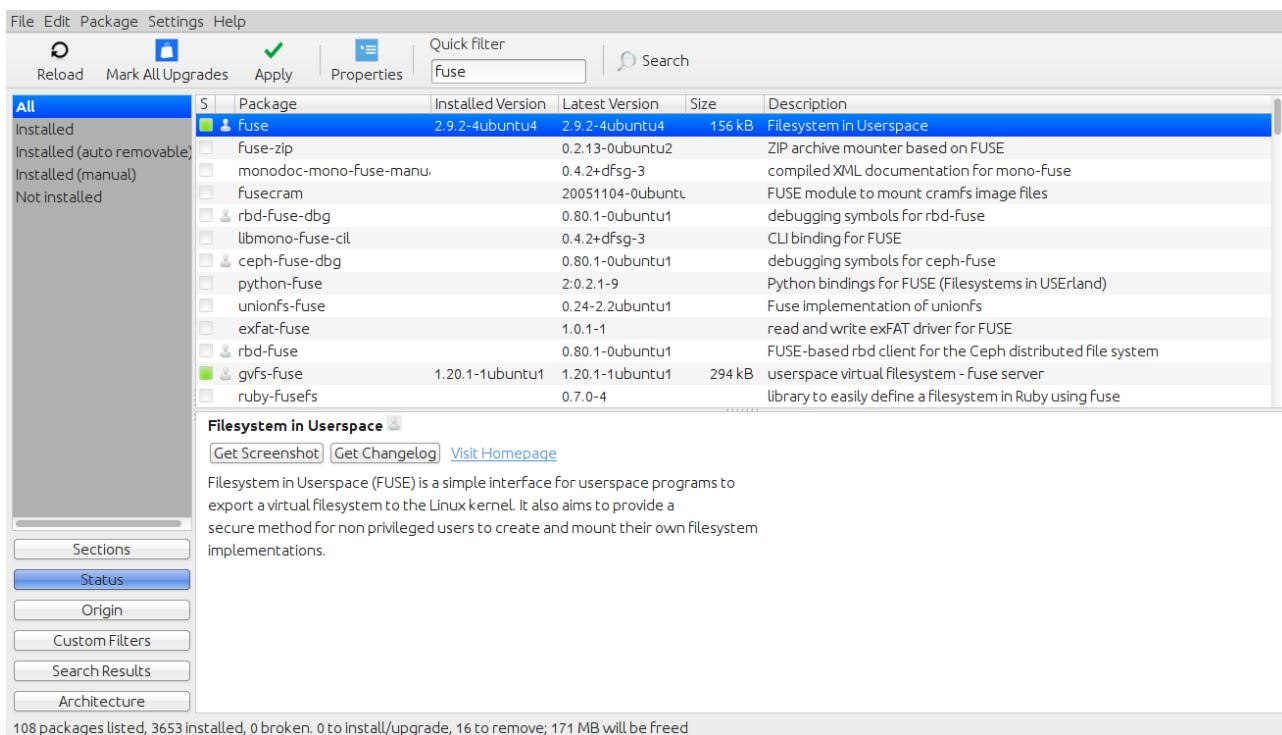
FUSE

NOTE: For those of you that enjoy learning about filesystems, you may enjoy this link - <http://www.linux.org/threads/filesystem-article-index.6028/>

Linux Kernel - <http://www.linux.org/threads/linux-kernel-reading-guide.5384/>

The Filesystem in Userspace (FUSE) is a special part of the Linux kernel that allows regular users to make and use their own filesystems without needing to change the kernel or have Root privileges. The filesystems used in FUSE are virtual filesystems. Not all virtual filesystems use FUSE. The code for FUSE itself is in the kernel, but the filesystem is in userspace. However, typical filesystems exist in the kernelspace.

FUSE allows users to make their own filesystems, modify filesystems, or use a special filesystem temporarily. FUSE can also be used to add extra features and abilities to a system or software. For example, [GVFS \(GNOME Virtual FileSystem\)](#) is a filesystem that allows applications to access remote files as if they were local. FUSE makes this possible. Also, at the time this article was written, the Linux kernel did not natively support exFAT (also called FAT64). If users wish to access a storage unit with a exFAT/FAT64 filesystem, users can mount the filesystem in FUSE. However, FUSE may need some extra packages to be installed to run some filesystems or virtual filesystems. For instance, if a user wanted to use exFAT like in the example above, then the user would need to install the "exfat-fuse" package which extends FUSE's abilities. The "exfat-fuse" package is a FUSE driver. Many drivers/extensions are available for FUSE.



FUSE is hosted at <http://fuse.sourceforge.net/>. FUSE is open-source freeware that anyone may obtain and use. FUSE is stable, secure, and reliable. However, it is more efficient to use a "real" filesystem if possible. FUSE is compatible and works on Solaris, FreeBSD, Darwin, GNU/Hurd, OS X, Opensolaris, and others. Some operating systems do not support FUSE. Such systems can use a fork of FUSE or use an alternative.

More Linux Related Stuff on: <http://www.linux.org>

For example, NetBSD uses PUFFS and Windows uses "fuse4win".

FUSE not only mounts virtual filesystems, but also "real" filesystems like ZFS and exFAT. FUSE can also mount files like ISOs, CD track files, and compressed files (zip, gzip, tar, etc.). FUSE's abilities extend to network filesystems like HTTP-FS, aptfs (apt repos), and others. FUSE can also be used to transfer files to Apple devices like the ipod and iphone (iFUSE). Amazingly, FUSE can also be used to convert FLAC to MP3 via the MP3 filesystem (mp3fs).

Users can program their own filesystems and use them in FUSE without changing the Linux kernel or needing Root privileges. Users program filesystems in a similar way programs are coded. If users wished to program a virtual filesystem using Ruby, then the programmer would need the "ruby-fusefs" package. FUSE bindings exist for many programming languages including Perl, Python, C/C++. C#, and many others. The complete list of language bindings are here (<http://sourceforge.net/apps/mediawiki/fuse/index.php?title=LanguageBindings>).

With FUSE, these virtual filesystems are not formatted. Rather, all a user needs to do to initiate a FUSE filesystem is to mount the specified FUSE filesystem to an empty directory that a user has permissions to use.

Now that many of you know a lot about FUSE, it is time to explain how it works. FUSE is a Linux module that acts as a "middle-man" or mediator between the FUSE filesystem and the Linux-kernel's VFS module. The VFS module can only be accessed by privileged users or processes. Since all users may access FUSE and FUSE may access VFS, that is how the permissions work to allow any user to use a FUSE filesystem. As for the FUSE filesystem itself, the code contains a structure variable that contains a pointer to functions in the FUSE filesystem code that respond to various actions that a process may try to use. In simple terms, the structure variable is a set of code that says a "read" syscall must have a particular set of parameters and is handled by a specific function defined in the FUSE filesystem's code.

Sample FUSE structure variable code -

Code:

```
struct fuse_operations {
    int (*mknod) (const char *, mode_t, dev_t);
    int (*getdir) (const char *, fuse_dirh_t, fuse_dirfil_t);
    int (*mkdir) (const char *, mode_t);
    int (*rmdir) (const char *);
    int (*readlink) (const char *, char *, size_t);
    int (*symlink) (const char *, const char *);
    int (*unlink) (const char *);
    int (*rename) (const char *, const char *);
    int (*link) (const char *, const char *);
    int (*chmod) (const char *, mode_t);
    int (*chown) (const char *, uid_t, gid_t);
    int (*truncate) (const char *, off_t);
    int (*utime) (const char *, struct utimbuf *);
    int (*open) (const char *, struct fuse_file_info *);
    int (*read) (const char *, char *, size_t, off_t, struct fuse_file_info *);
    int (*write) (const char *, const char *, size_t, off_t, struct fuse_file_info *);
    int (*statfs) (const char *, struct statfs *);
    int (*flush) (const char *, struct fuse_file_info *);
    int (*release) (const char *, struct fuse_file_info *);
    int (*fsync) (const char *, int, struct fuse_file_info *);
    int (*getattr) (const char *, struct stat *);
    int (*setxattr) (const char *, const char *, const char *, size_t, int);
```

More Linux Related Stuff on: <http://www.linux.org>

```

int (*getxattr) (const char *, const char *, char *, size_t);
int (*listxattr) (const char *, char *, size_t);
int (*removexattr) (const char *, const char *);
};

```

So, if a process using a FUSE filesystem performs some task that uses the write() syscall, then the filesystem code will execute the code for write(). When you write code for a FUSE filesystem, you could program it that when a process writes to a file that a copy of the file is to be made first.

Here is a table to help summarize the parts of a FUSE filesystem's code -

Import Headers – all C/C++ code imports headers and other programming language import some kind of library.

Declare Variables – Any variables that are used a lot in the code should be declared near the topic of the source code so the programmers can easily find and change global variables.

Syscall Declaration – A variable structure titled “fuse_operations” declares a syscall and needed parameters.

Syscall Functions – Next, programmers would write code for the action or actions that should occur when a particular syscall is declared. Developers may have a function for open(), write(), read(), and many other syscalls or needed features.

Main() - Obviously, if the FUSE filesystem is coded in C/C++, you will have a “int main()” which is where the code “starts” after the functions and variables are set.

When the filesystem “executes”, FUSE will be a mediator and communicate with the kernel on behalf of the FUSE filesystem.

Below is sample FUSE filesystem code that came from FUSE's Sourceforge page (<http://fuse.sourceforge.net/helloworld.html>).

Code:

```

/*
 * FUSE: Filesystem in Userspace
 * Copyright (C) 2001-2007 Miklos Szeredi <miklos@szeredi.hu>

This program can be distributed under the terms of the GNU GPL.
See the file COPYING.

gcc -Wall hello.c `pkg-config fuse --cflags --libs` -o hello
 */

#define FUSE_USE_VERSION 26

#include <fuse.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>

static const char *hello_str = "Hello World!\n";
static const char *hello_path = "/hello";

```

More Linux Related Stuff on: <http://www.linux.org>

```

static int hello_getattr(const char *path, struct stat *stbuf)
{
    int res = 0;

    memset(stbuf, 0, sizeof(struct stat));
    if (strcmp(path, "/") == 0) {
        stbuf->st_mode = S_IFDIR | 0755;
        stbuf->st_nlink = 2;
    } else if (strcmp(path, hello_path) == 0) {
        stbuf->st_mode = S_IFREG | 0444;
        stbuf->st_nlink = 1;
        stbuf->st_size = strlen(hello_str);
    } else
        res = -ENOENT;

    return res;
}

static int hello_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
                        off_t offset, struct fuse_file_info *fi)
{
    (void) offset;
    (void) fi;

    if (strcmp(path, "/") != 0)
        return -ENOENT;

    filler(buf, ".", NULL, 0);
    filler(buf, "..", NULL, 0);
    filler(buf, hello_path + 1, NULL, 0);

    return 0;
}

static int hello_open(const char *path, struct fuse_file_info *fi)
{
    if (strcmp(path, hello_path) != 0)
        return -ENOENT;

    if ((fi->flags & 3) != O_RDONLY)
        return -EACCES;

    return 0;
}

static int hello_read(const char *path, char *buf, size_t size, off_t offset,
                     struct fuse_file_info *fi)
{
    size_t len;
    (void) fi;
    if(strcmp(path, hello_path) != 0)
        return -ENOENT;

    len = strlen(hello_str);
    if (offset < len) {
        if (offset + size > len)
            size = len - offset;
        memcpy(buf, hello_str + offset, size);
    }
}

```

```
    } else
        size = 0;

    return size;
}

static struct fuse_operations hello_oper = {
    .getattr    = hello_getattr,
    .readdir    = hello_readdir,
    .open       = hello_open,
    .read       = hello_read,
};

int main(int argc, char *argv[])
{
    return fuse_main(argc, argv, &hello_oper, NULL);
}
```

Virtualization

Many of us have heard of virtualization, guest operating systems, hypervisors, and many other related terms. Virtualization allows users to run an operating system (OS) inside the "real" OS as if it were an application, or virtualize some device or network. But, what exactly is virtualization and how does it all work?

NOTE: This article discusses virtualization as in operating system virtualization. This article does not discuss virtualization as seen in Java's virtual machine, runtime environments, and similar systems.

Virtualization is the creation of a system or component that is not truly real. In other words, virtualization makes something seem genuine. The virtual object is not exactly real, but it is not entirely fake either. In computer technology, virtualization can be used to make "semi-real" networks, devices, computer systems, etc. Users may make a **virtual device** that the operating system thinks is a genuine physical device. The user knows the device is entirely software, but the computer cannot see the difference between a virtual device and a physical device. There is also **OS virtualization** which is where an operating system (called the **guest**) runs inside (or on top) of the OS on the hardware (called the **host**). The host is running on the physical computer while the guest system thinks it is running on bare-metal hardware. The guest runs as it would on real hardware, but it cannot see or access the physical hardware. Instead, it thinks it is running on the hardware that the virtualization software is virtualizing.

Virtualization is not emulation. Emulation is the process of pretending (or imitating) some other device or system. Think about some game emulators like ZSNES (which mimics Super Nintendo). They behave like the real console just as ZSNES acts like the Super Nintendo consoles. The emulator performs the functions of the real device, but it does not truly act like the device. Emulators copy the behavior of real devices enough to complete the same tasks they perform. For instance, Super Nintendo games work on the console and SNES ROMs (files containing an SNES game) work on ZSNES. On both systems, the game functions perfectly, or at least nearly perfect on the emulator. However, emulators do not act or appear exactly like the real system. For instance, most game emulators support the ability to save the game's current state/progress while consoles usually use a different way to save the game (like checkpoints). Also, emulators may provide the user with additional functionality. In contrast, virtualization is nearly real. Virtualization is faster than emulation in nearly all cases.

A **virtual machine** is a piece of software that represents a computer system. Guest operating systems run inside of virtual machines (like Oracle's VirtualBox). The guest OS thinks the virtual machine is real hardware. The hypervisor creates and manages the virtual machine. Multiple hypervisors and guest systems can be run on a host system as long as there is enough "real" hardware to satisfy the needs of the guests. A guest can also be run in a guest.

NOTE: Usually, when many guests are run, a single hypervisor manages all of the virtual machines containing guests.

Virtual machines can represent hardware of an architecture that is very different from the host. This is possible because of the way the guest's system calls are mapped (or "translated"). Users can run 32-bit guests on 64-bit hosts. Operating systems designed for one processor type can be run on a host that uses an entirely different CPU. However, not all processors, BIOS chips, and hypervisors support certain guest

architectures. For instance, some 64-bit hosts cannot support a 64-bit guest (usually due to limitations in the CPU and/or BIOS). Because of this fact, be sure to understand the virtualization capabilities of your BIOS, CPU, hypervisor, and host kernel when choosing the right hypervisor+guest+host combination for your needs. To run a particular guest, you may need a different hypervisor, host OS, or entirely different physical hardware.

FUN FACT: Under very special conditions, a 64-bit guest can run on a 32-bit host. This is possible if the physical hardware is 64-bit and has a CPU that offers certain virtualization extensions.

Hypervisors may work at different levels of the host system depending on the particular hypervisor's abilities. KVM (Kernel-based Virtual Machine) is a hypervisor in the Linux kernel (or as a module) that runs virtual machines in separate userspaces. QEMU and Oracle's VirtualBox (VBox) are hypervisors that run in the host's userspace as an application. Some hypervisors may work at the hardware level. Such hypervisors may exist as firmware or act as a compatibility layer under the OS kernels (such as Xen which is a hypervisor in the form of a microkernel). Many hypervisors exist, each with their advantages and disadvantages as well as supported features. Not all hardware supports the same hypervisors. Some hypervisors require special CPU commands (instruction sets and extensions). Computers with a CPU that lacks virtualization extensions will only be able to use hypervisors that run as applications or use paravirtualization (discussed below).

NOTE: Processors with virtualization instructions/extensions can help speed-up virtualization. This is known as hardware-assisted virtualization.

Virtual machines often use virtual devices, and they may use **virtual networks**. Many guest systems use a file as a hard-drive (**device virtualization**). ISO files are examples of virtual optical discs (like CDs and DVDs), and “.img” files are often used as virtual floppies. Many virtual machines allow users to use ISO files on their host system as discs on the guest. Guests can form a virtual network with other systems like other guests, the host, and external systems. Some hypervisors allow the host's directories to be mounted in the guest as a drive. This helps users transfer files between systems.

Another type of virtualization exists called **paravirtualization** (in contrast to full virtualization). This is a form of virtualization where the guest knows it is running in a virtual machine. Thus, the guest will modify its behavior slightly to enhance its performance. The guest does so by using a special set of system calls referred to as "hypercalls". Not all operating systems support hypercalls. Operating systems that support hypercalls either have the hypercalls built into the kernel or as a driver. Hypercalls use the paravirtualization application-programming-interface (para-API). Hypercalls come from the kernel and are sent to the hypervisor while the guest's userspace still sends system calls to the guest's kernel. Using paravirtualization (also called para-virt or PV) does not require the processor to have virtualization extensions. The Xen hypervisor supports paravirtualization.

NOTE: Hypercalls are to hypervisors as system calls are to kernels. (Analog from <http://wiki.xen.org/wiki/Hypercall>)

Guest's userspace >>(syscalls)>> Guest's kernel >>(hypercalls)>> Hypervisor

Instead of hypervisors, **OS-level virtualization** can be used. In this virtualization process, the kernel runs multiple sets of userspaces. Hypercalls and virtual machines are not used since the userspaces interact with the kernel directly. The userspaces communicate with the kernel using the typical system calls like “normal” systems. There is only one kernel running on the physical hardware. Since there is only one

More Linux Related Stuff on: <http://www.linux.org>

kernel that interacts with the userspaces directly, each userspace must be compatible with the kernel. This means OS-level virtualization is only possible among systems that use the same kernel. For instance, users running a Linux kernel in this fashion can only run userspaces of various Linux distros. Each userspace instance is referred to as a "container" in contrast to the term "virtual machine". The machine can act as a server that uses one IP address with each userspace using its own hostname or domain-name. Alternately, the userspaces can each be set to use a different port. Either way, only one IP address is used, thus saving IP addresses for other uses.

There is also another form of OS-level virtualization. LXC is a hypervisor that allows each userspace to have its own Linux kernel. This is possible because LXC divides the hardware resources for each kernel (via cgroups) as if the hardware were multiple computers. Such systems do not use the concept of "host and guest". Technically speaking, each OS is its own host.

NOTE: OS-level virtualization is not the same as multibooting.

In summary, operating systems can be run in another operating system by using virtualization or paravirtualization. Virtualization also allows multiple operating systems to run on the same piece of hardware simultaneously. Hypervisors manage and create virtual machines that contain the guest OS. All this works together to allow users to take advantage of other operating systems to perform tasks that may not be easily performed using their usual OS. Understanding the basics of virtualization can help users with creating virtual systems.

Xen

For those of you that understand virtualization, you may want to have an understanding of the most popular hypervisor for Linux - Xen. Xen is an open-source hypervisor that is well supported by Linux. Some other operating systems support Xen as well. Knowing the basics of Xen can help you understand hypervisors better and know why Xen is so popular.

Xen is a popular hypervisor that supports OS virtualization and paravirtualization (PV). Xen is a "Type 1" hypervisor meaning that it runs on the hardware. When the computer boots-up, the bootloader (typically GRUB) loads Xen (Xen is a microkernel). Then, Xen sets up specialized virtual machines called "Domains". Domains are low-level virtual machines that do not have a host, and only one domain can access the hardware directly. Xen first sets up the primary domain. This domain is known by many names such as "Domain Zero", "Dom0", "Domain0", "Host Domain", or "The Control Domain". Xen cannot function without the control domain. The control domain provides the drivers that are used by itself and other domains. The control domain also contains the controls and settings for Xen as well as Xen itself. The other domains are called "Guest Domains", "Unprivileged Domains", or "DomU". The guest domains use the drivers provided by Domain0. The guest domains are not permitted to access the hardware directly. Instead, they must allow Dom0 to manage the hardware. (<http://www.xenproject.org/>) (<http://wiki.xen.org>)

NOTE: With a virtualization structure like this, it is hard to specifically say which system is the host. Some may argue that Xen is the host system since it is a layer between the hardware and operating systems. As for the other view, Domain0 could be considered the host since Xen depends on it and unprivileged domains need Domain0 for hardware access. Either way, the unprivileged domains act like guests, so they are often called guest domains. Some manuals may refer to the hardware itself as the host.

Most Linux distros contain **PV-enabled kernels**; PV is enabled in the vanilla kernel. PV-enabled kernels contain the **paravirt_ops** (pv-ops) framework which is a kernel infrastructure that provides **hypervcalls** and other components needed for paravirtualization. Thus, paravirtualization does not require virtualization support from the CPU. Just as the userspace sends system calls to the kernel, the Linux kernel sends hypercalls to Xen. Hypercalls usually have a naming scheme like this "__HYPERVERSOR_*", where the asterisk represents the rest of the hypercall's name.
(<http://xenbits.xen.org/docs/unstable/hypercall/index.html>)

NOTE: Other hypervisors recognize and use the hypercalls from the paravirt_ops framework.

Xen can send the virtual operating systems "**Virtual Interrupts**". These virtual interrupts go to the kernel of the virtual OS. The names of the interrupts begin with (VIRQ_). "Event Channels" provide the framework for event notifications. These notifications are Xen's equivalent to hardware interrupts.

Xen Cloud Platform (XCP) is an ISO file that comes with Xen and a Linux distro set as Dom0. In other words, this is a complete Xen system that you install from the ISO. Otherwise, install a Linux distro and use the package manager to install Xen. If you are interested in installing Xen, then I recommend you go to Xen's official website and read the installation instructions.
(http://wiki.xenproject.org/wiki/Xen_Project_Beginners_Guide)

Often times, Xen systems use a Logical Volume Manager (LVM) for local **storage**, but this is not required. Xen can also use network storage. Xen also uses "Xen Virtual Block Devices" (VBD or XVD). These are files where the file itself is a virtual hard-drive. Such files use the ".dsk" file extension. The first virtual

More Linux Related Stuff on: <http://www.linux.org>

Xen hard-drive is designated as /dev/xvda and /dev/xvdb is the second (just change the last letter). The virtual hard-drives are treated as IDE drives, but number of maximum partitions is fifteen. Xen Virtual Block Devices are commonly used with paravirtualized domains.

NOTE: Domain0 must be installed on the physical hardware, but all of the guest domains may be installed in XVDs.

You may be wondering, “**why install domains in an XVD file?**”. Well, doing so allows the virtual OS to be copied as a file. Admins can copy the “.dsk” file and easily and quickly install preconfigured systems or make backups. Having a complete OS in a single file makes backups and installation very easy. Since Xen is a hypervisor, nearly any system running Xen would be able to use the OS in the “.dsk” file. This also makes hardware requirements flexible because as long as Xen is supported and Domain0 has the needed drivers, a particular virtual OS will be able to run on any machine. If an OS becomes corrupted, the file can be copied over the top of the corrupted Domain's image.

LINK: This site offers some pre-made Xen images as well as images for other hypervisors.
<http://stacklet.com/downloads/images/public>

XenServer is an open-source Xen system offered by **Citrix**. Users can purchase support, but the software itself is free. XenServer has an interesting piece of software called **XenMotion**. XenMotion allows a currently running guest domain to be moved from one physical machine to another host machine that both share the same storage. This allows admins to repair servers without having the system down.
(<http://xenserver.org/>)

Xen supports various operating systems including Windows. However, Windows can only be in a guest domain, never in the control domain. NetBSD and OpenSolaris are two examples of operating systems that can function as Domain0.

"**xencommons**" is the init script that each operating system domain must have properly installed to run the required Xen services.

Various interfaces exist for managing a Xen system. Users can use Xen's command-line tools or install a separate interface. Many management tools exist including graphical interfaces. "libvirt" is an example of a hypervisor management API that many management tools use. "libvirt" is a library written in C, but various programming bindings allow programmers to implement libvirt in management tools that are programmed in other computer languages.

(http://wiki.xen.org/wiki/Xen_Management_Tools)(http://wiki.xen.org/wiki/XCP_Management_Tools)

Xen uses the Hierarchical Protection Domains (**CPU Protection Rings**) security feature offered by most (or all) processors. A CPU can execute code under different security modes. Most CPUs have four rings (security modes) designated "Ring 0" through "Ring 3". Ring 0 is the most privileged and allows complete direct-access to the hardware. In contrast, Ring 3 has very few privileges and must request permission from the user to perform special tasks. Xen runs in Ring 0 while Dom0 and the guest domains all run in Ring 1. User applications run in Ring 3, which is the default for most operating systems. Xen does not assign any process to Ring 2.

Xen is a powerful hypervisor with many features and uses. Having a general understanding of Xen can

More Linux Related Stuff on: <http://www.linux.org>

help admins decide which hypervisor to use. Understanding Xen also helps users gain a better understanding of virtualization and Linux.