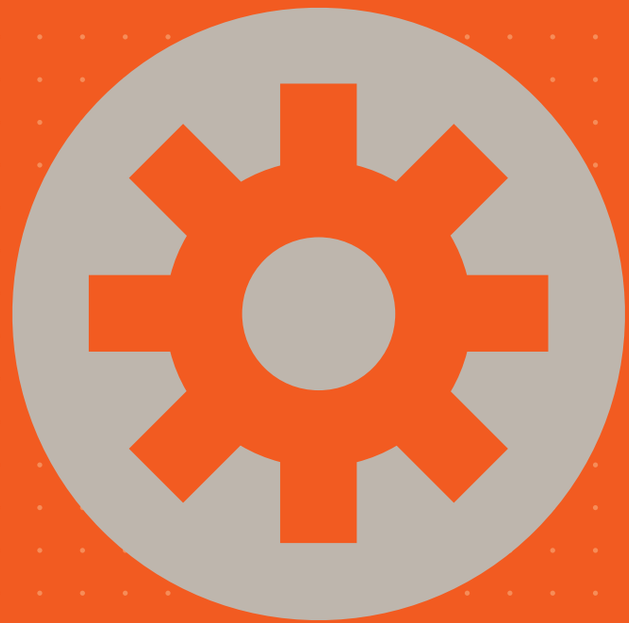
odm.ubuntu.com

ubuntu



Credits

The content for this booklet is sourced from the community sites:
wiki.ubuntu.com
help.ubuntu.com
smackerelofopinion.blogspot.com

Contents

Introduction

Jon Melamut
VP, Professional &
Engineering Services

"Canonical is focused on making Ubuntu an outstanding consumer and enterprise-ready product; Canonical is engaged in a wide range of activities to drive down the costs, perceived complexity, and time it takes to ship hardware that fully supports Linux. In addition to working with component manufacturers and BIOS vendors, Canonical is committed to ensuring that ODMs (Original Design Manufacturers) are as confident undertaking system integration work on Ubuntu as they are with Windows.

The demand for ensuring that devices are ready to go to market with Linux have never been higher. Yet many of our partners have learned that an ecosystem focused on shipping Windows exclusively for 15+ years finds working with Linux unfamiliar, non-standard, thereby leading to process inefficiencies. Similarly as ARM based devices become more mainstream at the heart of web-centric computing, the industry needs support to enhance its readiness to bring these low cost and low power devices to the market.

For any OEM (Original Equipment Manufacturer) product manager today, Canonical's Professional & Engineering Services group offers a standardised range of services to make sure that high-quality Ubuntu-based products ship on time. We strongly believe that initiating these projects on a one-off basis can result in delays and unnecessary complexity. Our services include adapting installation methods for individual ODMs, quality assurance and in factory support as necessary.

Canonical understands that keeping a manufacturing line running is essential to maintaining profitable margins for OEMs & ODMs. Canonical with its partners have brought tens of millions devices pre-installed with Ubuntu to market on time.

We believe that by selecting Ubuntu as your Linux platform of choice and Canonical as your full service system integrator, will allow your suppliers, partners, and internal technical teams to focus on achieving the full potential of Linux in client and server computing."

Getting started

Using printk

The simplest, and probably most effective way to debug the kernel is via printk(). This enables one to print messages to the console, and it very similar to printf(). Note that printk() can slow down the execution of code which can alter the way code runs, for example, changing the way race conditions occur.

Changing the ring buffer size

The internal kernel console message buffer can sometimes be too small to capture all of the printk messages, especially when debug code generates a lot of printk messages.To increase the internal buffer, use the kernel boot parameter:

log_buf_len=N

where *N* is the size of the buffer in bytes, and must be a power of *2*.

Changing debug levels

One can specify the type of printk() log level by prepending the 1st printk() argument with one of the following:

KERN_EMERG / system is unusable */*
KERN_ALERT / action must be taken immediately */*
KERN_CRIT / critical conditions */*
KERN_ERR / error conditions */*
KERN_WARNING / warning conditions */*
KERN_NOTICE / normal but significant condition */*
KERN_INFO / informational */*
KERN_DEBUG / debug-level messages */*

For example:

printk(KERN_DEBUG "example debug message\n");

If one does not specify the log level then the default log level of KERN_WARNING is used. For example, enable all levels of console message:

echo 7 > /proc/sys/kernel/printk

To view console messages at boot, remove the quite and splash boot parameters from the kernel boot line in grub. This will disable the splash screen and re-enable console messages.

Early Printk Statements

The earlyprintk kernel option supports debug output via the VGA, serial port and USB debug port.

The USB debug port is of interest - most modern systems seem to provide a debug port capability which allows one to send debug over USB to another machine. To check if your USB controller has this capability, use:

sudo lspci -vvv | grep "Debug port"

and look for a string such as "Capabilities: [58] Debug port: BAR=1 offset=00a0".

To select this mode of earlyprintk debugging use:

earlyprintk=dbgp

for the default first port, or select the *Nth* debug enabled port using:

earlyprintk=dbgpN

You also needs to build a kernel with the following config option enabled:

CONFIG_EARLY_PRINTK_DBGP=y

To capture the USB debug using (e.g.) */dev/ ttyUSB0* with minicom see: *Documentation/x86/ earlyprintk.txt*

Another way to get debug out is just using the boot option:

earlyprintk=vga

however this has the problem that the messages are eventually overwritten by the real console.

Finally, for old legacy serial ports on their machine (which is quite unlikely nowadays with newer hardware), one can use:

earlyprintk=serial,ttySn,baudrate

where *ttySn* is the *nth tty* serial port. You can also append the "*,keep*" option to not disable the *earlyprintk* once the real console is up and running.

So, with earlyprintk, there is some chance of being able to get some form of debug out to a device to allow one to debug kernel problems that occur early in the initialisation phase.

Serial Console

Serial console enables one to dump out console messages over a serial cable. Most modern PCs do not have legacy serial ports, so instead, one can use a USB serial dongle instead. To do so, one needs to enable

USB serial support as a kernel build configuration:

CONFIG_USB_SERIAL_CONSOLE=y
CONFIG_USB_SERIAL=y

and enable the appropriate driver, e.g.:

CONFIG_USB_SERIAL_PL2303=y

and boot this kernel with

console=tty console=ttyUSB0,9600n8

Console Messages

Kernel Oops messages general contain a fair amount of information. Unfortunately the stack dump can can scroll off the top of the 25 line Virtual Console. To capture more of a Oops, try the following:

chvt 1
setfont /usr/share/consolefonts/Uni1-VGA8.psf.gz

Another trick is to rebuild the kernel to only capture the initial Oops information. To do this, modify *dump_stack* in *arch/x86/kernel/dumpstack_*.c* and comment out the call to *show_trace()*

Kernel Messages

Slowing down kernel messages on boot

If a machine hangs during the kernel boot process and you would like to be able to see all the kernel messages but unfortunately they scroll off the console too quickly. You can slow down kernel console messages at boot time using by building the kernel with the following option enabled:

CONFIG_BOOT_PRINTK_DELAY=y

And boot the machine with the following kernel boot parameter:

boot_delay=N

where *N = msecs* delay between each console message.

Kernel panic during suspend

To stop console messages from being suspended use the kernel parameter:

no_console_suspend=1

Boot with this option, *chvt 1 (to console #1)*, and suspend using *pm-suspend*.

Kernel Oops page fault error codes

The x86 Linux kernel Oops messages provide normally just enough information to help debug critical bugs.

kernel BUG at kernel/signal.c:1599!
Unable to handle kernel NULL pointer
dereference at virtual address 00000000
pc = 84427f6a
**pde = 00000000*
Oops: 0001 [#1]

The 4 digit value after the "Oops:" message dumps out the page fault error code in hexadecimal which in turn can help you deduce what caused the oops. The page fault error code is encoded as follows:

bit 0 - 0 = no page found, 1 = protection fault
bit 1 - 0 = read access, 1 = write access
bit 2 - 0 = kernel-mode access, 1 = user mode access
bit 3 - 0 = n/a, 1 = use of reserved bit detected
bit 4 - 0 = n/a, 1 = fault was an instruction fetch

So, in the above example, the Oops error code was *0x0001* which means it was a page protection fault, read access in kernel mode.

A lot of Oops error codes are *0x0000*, which means a page was not found by a read access in kernel mode.

For more information, consult *arch/x86/mm/fault.c*

Serial Console in VirtualBox

In some debug scenerios it can be helpful to debug the kernel running inside a virtual machine. This is useful for some classes of non-hardware specific bugs, for example generic kernel core problems or debugging file system drivers.

You can capture Linux console messages running inside VirtualBox by setting it the VirtualBox serial log to */tmp/vbox* and running a serial *tty* communications program such as minicom, and configure it to communicate with a named pipe *tty* called *unix#/tmp/vbox*

Boot with virtualised kernel boot line:

console=tty console=ttyS0,9600

and minicom will capture the console messages

Using Mainline Kernels

The mainline kernels archive is located at the URL below, there is a directory for each mainline build:

http://kernel.ubuntu.com/~kernel-ppa/mainline

The tagged releases are found under a directory matching their tag name and which kernel configuration they were built with (*<tag>-<series>*). Daily releases are found in the daily sub-directory named for the date they were made.Each build directory contains the header and image .deb files for the i386 and amd64

architectures, generic flavour.

Installing Mainline Kernels

To use the mainline kernel as-is you only only need to download and install the *image*.deb package that corresponds to your architecture, however if you need to build any external modules you also need the correct **header*.deb* and **source*.deb* packages.

To install, download the common headers, architecture specific headers, and the architecture specific image.

Once you have those downloaded they will need to be installed using *dpkg*:

```
sudo dpkg -i *.deb
```

When this process completes you should have a new entry on your boot menu representing the mainline kernel.

Uninstalling Mainline Kernels

If you would like to uninstall a mainline kernel, first use:

```
dpkg -l | grep "linux\-[a-z]*\-"
```

to find the exact name of the kernel packages you want to uninstall, and then do:

```
sudo apt-get remove KERNEL_PACKAGES_TO_REMOVE
```

Remember that several packages belong to one kernel version: common headers, architecture specific headers and the architecture specific image.

Debugging Common Issues

Debugging Hotkeys

If gnome-settings-daemon or gnome-power-manager is running, stop it first with killall gnome-settings-daemon gnome-power-manager; these daemons grab some X events exclusively and prevent them from being seen with xev.

Run xev to test whether a keypress event is seen:

```
xev | sed -n 's/^.*state \([0-9].*\), keycode *\([0-9]\+\)
*\(.*\), .*$/keycode \2 = \3, state = \1/p'
```

If it reports a keypress event and if the keycode is right, it's a desktop issue.

In some cases the keybindings may be wrong, perhaps due to a legacy (i.e., *pre-evdev*) keymap. You can check your keymap using *gconf-editor* and looking under */apps/gnome_settings_daemon/* keybindings. Bindings without sensible key names are probably bugs.

For audio volume control hotkeys, gnome-soundproperties may be misconfigured. You can either examine with gconf-editor '*/desktop/gnome/sound*' or do '*gconftool --recursive-list /desktop/gnome/sound*' to get the current settings; the particular configuration items are '*default_mixer_tracks*' and '*default_mixer_device*'.

If the key code is wrong, or there is no keypress event, or the key only works once and then the desktop gets "stuck", exercise the "Fixing broken keys" section in */usr/share/doc/udev/README keymap.txt*

If that was successful, file a bug against udev ("*ubuntubug udev*") and attach your newly created keymap and rule.

If udev's keymap tool shows a correct key symbol, look up the symbolic name in */usr/include/linux/input.h*. If it is mapped to a code over 255 (over 0x0ff), then it is outside X's range. In this case, if it is important to have the key mapped, the key should be remapped to an appropriate value < 256.

If the events are reported by more than one input device then report a kernel bug (Ubuntu linux package) because it should only send the event on one device.

If not found with keymap, use acpi_listen to determine whether the key is coming through as an ACPI event instead of a keypress if there is an ACPI event but no keypress, this is a bug in the kernel (ubuntu-bug linux) for not translating the ACPI event to an input event.

If there is neither an ACPI event nor an input event, this is probably also a kernel bug.

Debugging Suspend

Suspending from text mode

The first step for debugging suspend is to determine if the issue occurs when triggered using the pm-suspend command. If possible you should reboot the system with the *no_console_suspend* boot parameter. See *DebuggingKernelBoot* for instructions on how to modify boot parameters:

http://wiki.ubuntu.com/DebuggingKernelBoot

You should then switch to VT1 by pressing Ctrl-Alt-F1. Login at the prompt there and run the following commands:

setfont /usr/share/consolefonts/Uni1-VGA8.psf.gz
sudo pm-suspend

This will select a much smaller font so that you can see more messages should they come out, and then initiate the suspend.

Please report whether you got any additional messages. Digital photos of the screen are a sensible way to get this into the bug.

Enabling Suspend Debugging

If the previous step produces nothing useful, then you will need to try enabling kernel suspend debugging. Details can be found at: https://wiki.ubuntu.com/DebuggingKernelSuspend

Debugging Sound Problems

It is often the case that a muted channel is the problem, even though the description may not sound immediately relevant. In this regard, muted Surround or Center channels are common culprits. So unmute and raise the volume of one channel at a time and check whether sound is then produced by a running sound application. Open a terminal window and launch alsamixer. Then unmute as described above.

Checking sound device assignment

Most sound applications output to card0 by default. In some cases, other audio devices (like a USB MIDI Keyboard) might be recognized as a soundcard and take card0, bumping your real soundcard to card1. To see which devices are connected to which cards, do the following:

cat /proc/asound/cards

You can manipulate the device number assignment by modifying */etc/modprobe.d/sound.conf* options and s*lots=,snd-usb-audio*

Checking permissions and resources

Make sure that all users needing access to the Sound Device can “Use audio devices” in the “User Privileges” tab of users-admin (System->Administration->Users and Groups).

Test different “Sound Servers”: Go to System > Preferences > Sound (“Multimedia Systems Selector” in earlier editions of Ubuntu). From there, you can test the different options. In some scenarios several different sound servers may be installed, and only one may work. This is probably the origin of the problem if you cannot play audio with xine or rhythmbox, but you can with xmms or helix/realplayer.

If the application sounds work, but the system sounds do not (login, logout, error sounds...) try removing the .asoundrc* files from your own directory (e.g. with ‘rm .asoundrc*’). It should make the system sounds work without a reboot.

Debugging X Freezes

Symptoms

- X stops responding to input (sometimes mouse cursor can still move, but clicking has no effect)
- The screen displays but does not update. Sometimes there is screen corruption too, but usually there isn’t.
- Often, X cannot be killed; only a reboot clears the state
- The system operates fine over SSH but not on the graphical console
- Error messages such as “GPU lockup” are (sometimes) present in your dmesg output

Non-Symptoms

- A backtrace appears in Xorg.0.log - most of the time this indicates a crash, not a freeze.
- X seems to be working, but the monitor appears to just be “off”
- The caps lock key blinks - this indicates a kernel failure, not X
- X CPU or memory load is high, making system laggy or freeze up. This usually indicates a client application error.
- Screen still updates (look at clock), but can’t be interacted with - probably is an input bug, not a GPU freeze
- System freezes for a period but then comes back. Real freezes never come back.

Typical X Freeze Problems

Problem: New hardware freezes
First test newer kernels, then test newer X components:

- *https://launchpad.net/~xorg-edgers/+archive/ppa*

Problem: Freezes occur when idle and screensaver is set to random settings
A lot of freezes occur in the 3D code, and go unnoticed by users that don’t otherwise use 3D stuff, except when an OpenGL screensaver activates via Random setting.

Problem: Freezes when screensaver or video player changes DPMS settings
You can manually invoke and control Display Power Management (DPMS) using the xset command line tool:

sleep 1; xset s activate or sleep 1; xset dpms force off

Problem: Log shows “[mi] EQ overflowing” and X freezes
This message indicates that the server has noticed that the GPU is locked up. This is a particularly common failure-mode for the nouveau driver.

Problem: Log shows something about ring buffers and I830WaitLpRing (-intel only)

A WaitLpRing bug is generally a GPU hang, which can be caused by sending the GPU a bad instruction or address.

Debugging Wireless

Useful Commands
sudo lshw
This command lists detailed hardware information. The option “-businfo” lists information about any SCSI, IDE, IDE devices and their bus addresses along with the class of each device.The configuration line will tell you if there is a driver loaded for your device, except devices using orinoco drivers. If you do not see a driver listed here, then there is not one loaded and assigned to the device, and it will not show up in iwconfig output or the nework-admin gui.

lspci -v | grep Ethernet
This command lists information about devices on the pci bus. Adding the -n option to lspci makes the output include the numerical PCI vendor and device ID’s.This command shows the revision of the card (in above example the revision of the card is B5 not 01). Using the -n option you can find the PCI ID (168C:0013) of the card and find the correct driver to use with ndiswrapper.

sudo lsusb -v
This command lists information about devices on the USB bus.

ssudo lsmod
This command lists kernel modules that are loaded and running.

rfkill list
This command prints information detailing whether there are software or hardware blocks on your rf devices.

sudo iwconfig
This command prints information about a wireless interface and allows you to configure the network interface from the command line.

Access Point: If you see all zeros here or nothing then you are not connected/associated to your router. When you are connected it will show the mac address of the router here.

sudo iwlist <ath0> scan
This command will give you more detailed information from the wireless interface such as a scan of all available routers with in range.A completed scan your device and driver are probably working properly. Some devices, such as orinoco cards, do not support scanning so this command may not work for you.

sudo dhclient <ath0>
dhclient deals with DHCP if your router is running as a DHCP server.

Debugging USB Problems

Basic Information

To get a list of currently attached USB devices (including hubs) use the following command:

sudo lsusb -v

Another variation, which results in a slightly more condensed format but will also show which driver currently is used for the devices can be received by:

cat /proc/bus/usb/devices

If there is no file like that, the usbfs needs to be mounted:

sudo mount -t usbfs none /proc/bus/usb

Getting USB Tracedata

Recent kernels have a facility called usbmon which can be used to gather information about traffic on the USB bus(es). To use it, first mount the debug filesystem:

sudo mount -t debugfs none /sys/kernel/debug

If there is no */sys/kernel/debug/usb/usbmon/* directory after this, the usbmon module must be loaded. Otherwise there are several file in the usbmon directory.

The file names consist of a number and a letter. The number relates to the USB bus (with 0 being sort of the master, which relates to all buses). The letter is either s, u or t. The s file contains a generic event overview. The t and u files will stream trace data. With t being the older format, while u is currently preferred.

Now, in order to gather debug data, one can either use the master file 0u (but that contains data from all devices) or find out the bus on which a device shows up (with one of the basic information sources) and then using the trace file for that bus. If, for example the device is on bus 2, the following command will write trace data into a file:

sudo cat /sys/kernel/debug/usb/usbmon/2u >bus2data.txt

To end tracing, just kill the command (CTRL-C).

Debugging Firmware with FWTS

Download the images 32 bit or 64 bit from:

https://wiki.ubuntu.com/HardwareEnablementTeam/Documentation/FirmwareTestSuiteLive

Unzip the image using gunzip, e.g.

gzip -d fwts-live--*.img.gz*

Then Insert a USB stick into your machine, and unmount it. Copy the live image to the USB stick

sudo dd if=fwts-live-natty-amd64-usb-hdd-20111004-1.img of=/dev/sdb && sync

Lastly remove the USB stick. Now is ready to insert the USB stick into the machine you want to test and boot the machine.