

Linux Kernel Crash Book

Everything you need to know

Igor Ljubuncic aka Dedoimedo

[www.dedoimedo.com](http://www.dedoimedo.com/)

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Contents

I LKCD 23

1 Introduction 23

1.1 How does LKCD work? 23

1.1.1 Stage 1 23

1.1.2 Stage 2 24

2 LKCD Installation 25

3 LKCD local dump procedure 25

3.1 Required packages 25

3.2 Configuration file 25

3.2.1 Activate dump process (DUMP\_ACTIVE) 25

3.2.2 Configure the dump device (DUMP\_DEVICE) 25

3.2.3 Configure the dump directory (DUMPDIR) 26

3.2.4 Configure the dump level (DUMP\_LEVEL) 27

3.2.5 Configure the dump flags (DUMP\_FLAGS) 28

3.2.6 Configure the dump compression level (DUMP\_COMPRESS) . . 29

3.2.7 Additional settings 29

3.3 Enable core dump capturing 30

3.4 Configure LKCD dump utility to run on startup 30

4 LKCD netdump procedure 31

5 Configure LKCD netdump server 31

5.1 Required packages 31

5.2 Configuration file 31

2

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

5.2.1 Configure the dump flags (DUMP\_FLAGS) 32

5.2.2 Configure the source port (SOURCE\_PORT) 32

5.2.3 Make sure dump directory is writable for netdump user 32

5.3 Configure LKCD netdump server to run on startup 33

5.4 Start the server 33

6 Configure LKCD client for netdump 34

6.1 Configuration file 34

6.1.1 Configure the dump device (DUMP\_DEV) 34

6.1.2 Configure the target host IP address (TARGET\_HOST) 34

6.1.3 Configure target host MAC address (ETH\_ADDRESS) 35

6.1.4 Configure target host port (TARGET\_PORT) 35

6.1.5 Configure the source port (SOURCE\_PORT) 35

6.2 Enable core dump capturing 36

6.3 Configure LKCD dump utility to run on startup 37

6.4 Start the lkcd-netdump utility 37

7 Test functionality 37

8 Problems 38

8.1 Unsuccessful netdump to different network segment 39

9 Conclusion 39

II Kdump 41

10 Introduction 41

10.1 Restrictions 42

10.1.1 Kernel compilation 42

3

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

10.1.2 Hardware-specific configurations 42

10.2 How does Kdump work? 42

10.2.1 Terminology 42

10.2.2 Kexec 43

10.2.3 Kdump 43

11 Kdump installation 43

11.1 Standard (production) kernel 45

11.1.1 Under Processor type and features 45

11.1.2 Under Filesystems > Pseudo filesystems 47

11.1.3 Under Kernel hacking 47

11.1.4 Other settings 48

11.2 Crash (capture) kernel 49

12 Kdump packages & files 49

12.1 Kdump packages 49

12.2 Kdump files 50

13 Kdump configuration 51

13.1 Configuration file 51

13.1.1 Configure KDUMP\_KERNELVER 51

13.1.2 Configure KDUMP\_COMMANDLINE 52

13.1.3 Configure KDUMP\_COMMANDLINE\_APPEND 53

13.1.4 Configure KEXEC\_OPTIONS 54

13.1.5 Configure KDUMP\_RUNLEVEL 55

13.1.6 Configure KDUMP\_IMMEDIATE\_REBOOT 55

13.1.7 Configure KDUMP\_TRANSFER 56

13.1.8 Configure KDUMP\_SAVEDIR 56

13.1.9 Configure KDUMP\_KEEP\_OLD\_DUMPS 57

4

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

13.1.10 Configure KDUMP\_FREE\_DISK\_SIZE 58

13.1.11 Configure KDUMP\_DUMPDEV 58

13.1.12 Configure KDUMP\_VERBOSE 59

13.1.13 Configure KDUMP\_DUMPLEVEL 60

13.1.14 Configure KDUMP\_DUMPFORMAT 61

13.2 GRUB menu changes 62

13.3 Set Kdump to start on boot 63

14 Test configuration 64

14.1 Configurations 64

14.1.1 Kernel 64

14.1.2 GRUB menu 65

14.2 Load Kexec with relevant parameters 66

14.2.1 Possible errors 66

15 Simulate kernel crash 68

16 Kdump network dump functionality 69

16.1 Configuration file 69

16.1.1 Configure KDUMP\_RUNLEVEL 69

16.1.2 Configure KDUMP\_SAVEDIR 70

16.1.3 Kernel crash dump NFS example 71

17 Conclusion 72

III Crash Collection 73

5

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

18 Crash setup 73

18.1 Prerequisites 73

18.2 Kdump working crash installation 73

18.3 Crash location 74

18.4 Memory cores 77

19 Invoke crash 77

19.1 Old (classic) invocation 78

19.2 New invocation 78

19.3 Important details to pay attention to 80

19.4 Portable use 81

20 Running crash 81

20.1 Crash commands 83

20.1.1 bt - backtrace 83

20.1.2 log - dump system message buffer 84

20.1.3 ps - display process status information 84

20.2 Other useful commands 85

20.3 Create crash analysis file 85

20.4 Crash running in unattended mode 86

21 Possible errors 87

21.1 No debugging data available 87

21.2 vmlinux and vmcore do not match (CRC does not match) 89

21.3 No guarantee 90

22 Conclusion 90

IV Crash Analysis 91

6

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

23 Analyzing the crash report - First steps 91

24 Getting warmer 98

24.1 Fedora example 99

24.2 Another example, from the White Paper 100

24.3 Kernel Page Error 100

25 Getting hot 102

25.1 Backtrace 102

25.1.1 Call trace 104

25.1.2 Instruction pointer 105

25.1.3 Code Segment (CS) register 106

25.1.4 Privilege levels 106

25.1.5 Current Privilege Level (CPL) 106

25.1.6 Descriptor Privilege Level (DPL) &

Requested Privilege Level (RPL) 107

25.1.7 Fedora example, again 108

25.1.8 backtrace for all tasks 109

25.2 Dump system message buffer 110

25.3 Display process status information 111

25.4 Other useful information 113

26 Super geeky stuff 113

26.1 Kernel source 114

26.2 cscope 114

26.3 Disassemble the object 119

26.4 Trivial example 119

26.5 objdump 121

26.6 Moving on to kernel sources 123

7

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

26.6.1 What do we do now? 128

26.7 Intermediate example 129

26.7.1 Create problematic kernel module 129

26.7.2 Step 1: Kernel module 129

26.7.3 Step 2: Kernel panic 134

26.7.4 Step 3: Analysis 135

26.8 Difficult example 138

26.8.1 In-depth analysis 139

26.9 Alternative solution (debug kernel) 143

27 Next steps 144

27.1 kerneloops.org 145

27.2 Google for information 151

27.3 Crash analysis results 152

27.3.1 Single crash 153

27.3.2 Hardware inspection 153

27.3.3 Reinstallation & software changes 153

27.3.4 Submit to developer/vendor 153

28 Conclusion 154

V Appendix 155

29 Kdump 155

29.1 Architecture dependencies 155

29.2 Install kernel-kdump package manually 155

29.3 Install kexec-tools package manually 155

29.3.1 Download the package 156

29.3.2 Extract the archive 156

8

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

29.3.3 Make & install the package 156

29.3.4 Important note 156

29.4 SUSE & YaST Kdump module 157

29.5 SUSE (and openSUSE) 11.X setup 160

29.5.1 32-bit architecture 161

29.5.2 64-bit architecture 164

29.5.3 Other changes 166

30 Crash 168

30.1 Enable debug repositories 168

30.1.1 Enable repositories in CentOS 169

30.2 lcrash utility (for LKCD) 172

30.2.1 Kerntypes 173

30.3 lcrash demonstration 174

31 Other tools 176

31.1 gdb-kdump 176

31.2 crosscrash 177

VI References 179

32 LKCD references 179

33 Kdump references 179

34 Crash references 179

35 Dedoimedo web articles 181

9

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

List of Figures

1 LKCD stages 24

2 LKCD DUMPDIR directive change 27

3 LKCD netdump client source port configuration 36

4 LKCD netdump client successful configuration 37

5 Successful LKCD netdump procedure 38

6 LKCD netdump failure 39

7 Kernel compilation wizard 45

8 Kdump kernel version configuration 52

9 Kdump command line configuration 53

10 Kdump command line append configuration 54

11 Kdump options configurations 55

12 Kdump DUMPDEV configuration 59

13 Kdump DUMPLEVEL configuration 61

14 Console view of crash kernel dumping memory core 68

15 Contents of dumped memory core directory 69

16 Kdump SAVEDIR network configuration 71

17 Console view of network-based crash dump 71

18 Console view of network-based crash dump - continued 72

19 Installation of crash via software manager 74

20 openSUSE Kdump configurationvia YaST-Kdump module 75

21 CentOS Kdump configuration system-config-kdump utility 76

22 Generating crash dump files 77

23 Contents of a crash dump directory 77

24 New kdump invocation console output 79

25 Old crash invocation example on CentOS 5.4 79

26 New crash invocation example on CentOS 5.4 80

10

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

27 Crash debuginfo location on openSUSE 11.x 81

28 CRC match error 81

29 Crash working 82

30 Crash prompt 82

31 crash bt command example 83

32 crash log command example 84

33 crash ps command example 85

34 No debuginfo package on RedHat 87

35 No debuginfo package on openSUSE 88

36 Installing crash debug packages on CentOS 5.4 88

37 vmlinux and vmcore match problem on CentOS 89

38 CRC match error on openSUSE 89

39 No panic task found 90

40 Beginning crash analysis 92

41 Serious kernel problem example in Ubuntu 96

42 Kernel crash report in Fedora 97

43 Kernel crash report in Fedora, shown again 99

44 Backtrace of a crash dump 103

45 Example of a kernel crash with CPL 3 108

46 Fedora kernel crash example 109

47 Kernel crash log command output example 110

48 Kernel crash ps command output example 112

49 No panic task found on CentOS 5.4 112

50 bt command for wrong process 113

51 ps command output pointing at wrong process 113

52 Kernel source example on openSUSE 114

53 cscope installation via yum on CentOS 115

54 cscope loaded on CentOS 5.4 116

11

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

55 Find C symbol using cscope 117

56 cscope help menu 118

57 make cscope command example 118

58 cscope files 119

59 Compiling from sources with make 120

60 Kernel object is up to date 120

61 Object compiled with debug symbols 121

62 Disassembled object example 122

63 Memhog binary dumped with objdump 123

64 Failed kernel object compilation due to missing kernel config file 124

65 Editing Makefile 125

66 Makefile is missing 125

67 Successfully compiling kernel object 126

68 Kernel object is up to date 127

69 Disassembled kernel object 128

70 Basic kernel module 130

71 Basic example Makefile 131

72 Basic example make command output 132

73 modinfo example 133

74 lsmod example 133

75 Kernel module messages 134

76 Intermediate example crash summary 136

77 Intermediate example backtrace 137

78 Null pointer example crash report 139

79 Null pointer example crash log 140

80 Null pointer example disassembled object code 141

81 Null pointer example registers 142

82 Debug kernel installation 144

12

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

83 Debug kernel installation details 144

84 kernelops.org logo 145

85 kernelops.org example 146

86 kerneloops.org example - continued 147

87 kerneloops.org example code 148

88 Kernel crash report in Fedora 11 149

89 Kernel crash report in Fedora 12 150

90 Kernel crash report in Debian Lenny 151

91 Sample Google search 152

92 yast2-kdump package installation 157

93 Kdump YaST module 158

94 Kdump configuration via YaST module 159

95 Kdump configuration via YaST module - continued 160

96 Kdump startup configuration via YaST 161

97 boot.kdump chkconfig command 162

98 Runlevel configuration via YaST 163

99 Kdump GRUB syntax change 164

100 Failed memory reservation on a 64-bit machine 164

101 Kexec command line error 165

102 Wrong physical start value 165

103 Kdump working after reconfigured physical start value 166

104 debuginfo package missing 166

105 Available kernel debuginfo package in the repository 167

106 Kernel debuginfo package installation status 168

107 Enabling Debug repository in openSUSE 11.2 169

108 CentOS repository manager 170

109 Adding debug repository file 171

110 dwarfextract installation via YaST 173

13

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

111 lcrash example 175

112 gdb-kdump sample run 177

113 crosscrash installation via YaST 178

14

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

List of Tables

1 LKCD dump levels 28

2 LKCD dump flags 28

3 Kdump required packages 49

4 Kdump files 50

5 Kdump dump retention 57

6 Kdump verbosity configuration 60

7 Naming and file location differences between SUSE and RedHat 80

8 Kernel page error code 101

15

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Foreword

Writing books is not a new experience for me. I’ve been doing it since the age of 10.   
Most of these books gather proverbial dust on this or that hard disk, others are being   
pampered for limelight, others yet have been abandoned. There’s no better place to   
announce the demise of one project as at the birth of another. As you may have guessed,   
my super-extensive mother-of-all Linux topics book is not going to be published any time   
soon, as simple system administration no longer excites me. The single Apache chapter   
remains a proof-of-concept poetic demonstration, an orphan of what might have been.

Instead, I have started casting my eye toward more advanced, more complex topics. Like   
Linux crash analysis. This is a subject that has lots of unanswered mail threads and plain   
text documents scattered all over the place, inaccessible to almost everyone, save the   
tiny percentage of super geeks. Whether this should be so or not makes no difference.   
There comes a need, there comes a man with an idea, and that man writes a book.

My personal and professional interest in the last three years has taken me down the path   
of Linux kernel secrets, all the way into assembly code, where magic happens. I felt the   
desire to learn what happens in the heart of the system. Like most technical topics, there   
was some information to be found online, but it was cryptic, ambiguous, partial, nerdy,   
or just not there at all. Dedoimedo is a reflection of how things ought to be after all.   
I’m writing guides and tutorials and reviews the way I perceive the world - friendly and   
accessible toward normal human beings. In a way, every article is an attempt to make   
things a little clearer, a little more understandable. Step by step, nothing omitted, you   
know the mantra.

Linux kernel crash is no exception. If you’re familiar with my website, you know this book is just a compilation of seven in-depth tutorials already posted and available freely for everyone’s use. But there’s a difference between some HTML code, scattered around, and a beautiful stylish book written in LATEX. Not much difference, I admit, but still worth this fancy foreword.

This book is a product of several factors. First, my ego demands recognition, so I’m making the best effort of appearing smart in the posh circles. Nothing like a book to make you look wise and whatnot. Second, the book really makes sense, when you take the entire crash series into consideration. Starting with crash tools via collection all the way to analysis, plus some extras and general tips. It’s an entire world, really, and it belongs inside a single, comprehensive volume. Third, half a dozen Dedoimedo readers contacted me by mail, asking that I compile my crash material into one document. I did hint at a possible PDF given popular demand, so here we go.

16

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Linux Kernel Crash Book is 180 pages, 120 lovely screenshots and tons of excellent information. You won’t easily find better content on this subject, I guarantee you that. You get this book for free, no strings attached. There’s some copyright and disclaimer, mainly designed to protect my intellectual rights and hard work, but nothing draconian. Be fair and enjoy the knowledge shared with love and passion. If you happen to really like this book, think about donating a few bucks. An officially published book would probably cost between 20 and 40 dollars.

Now, this is no humble man’s marketing plot. I surely do not expect to get miraculously rich this way. If you take my Golf GTI donation scheme as an example, it’s a long way before my pockets swell with booty. However, like any egocentric human being, I love praise and recognition for my work. If you cannot or do not wish to donate money, then spread the word and lavish me with compliments. That will do, too.

I am also considering getting the book printed, whether through self-publishing or by insinuating my charm into an editor’s heart. So if you’re looking for talented fresh new blood to spatter the walls of glory, I’m your man. If you are a publisher and like my style and knowledge, don’t be shy, email me.

My readers, worry not! Even if this book goes pro, the online tutorials will always remain there, for free. The emphasis on always is within the Planck limits of time and space, excluding an occasional mega-meteor strike or a cosmic gamma ray burst event.

I guess that’s all. This book is waiting for you to read it. Enjoy!

Igor Ljubuncic aka Dedoimedo

February 2011

17

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

About

[Dedoimedo](http://www.dedoimedo.com) ([www.dedoimedo.com](http://www.dedoimedo.com/)) is a website specializing in step-by-step tutorials in-  
tended for human beings. Everything posted on my website is written in plain, down-to-  
Earth English, with plenty of screenshot examples and no steps ever skipped. You won’t easily find tutorials simpler or friendlier than mine.

Dedoimedo lurks under the name of Igor Ljubuncic, a former physicist, currently living the dream and working as a Linux Systems Expert, hacking the living daylight out of the Linux kernel. Few people have the privilege to work in what is essentially their hobby and passion and truly love it, so I’m most grateful for the beauty, freedom and infinite possibilities of the open-source world. I also hold a bunch of certifications of all kinds, but you can read more about those on my website.

Have fun!

18

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Copyright

Linux Kernel Crash Book is available under following conditions:

The book is free for personal and education purposes. Business organizations, companies   
and commercial websites can also use the book without additional charges, however they   
may not bundle it with their products or services. Said bodies cannot sell or lease the   
book in return for money or other goods. Modifications are not permitted without an   
explicit approval from the author. All uses must be accompanied with credits and a link   
to [www.dedoimedo.com.](http://www.dedoimedo.com./)

You may also mirror and hotlink to this book. You must credit me for any such use.

In all eventualities, Dedoimedo retains all rights, explicit and implicit, to the original material. The copyright section may change at any time, without prior notice. For any questions, please contact me by email.

19

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Disclaimer

I am not very fond of disclaimers, but they are a necessary part of our world. So here we go:

I must emphasize the purpose of this book is educational. It is not an official document   
and should not be treated as such. Furthermore, I cannot take any responsibility for   
errors, inaccuracies or damages resulting from the use of this book and its contents.

All of the material in this book has been carefully worded and prepared. However, if for some reason you may feel this book infringes on copyright or intellectual property of another work, please contact me with a detailed explanation pointing to the troublesome parts and I will try to sort the problem in the best way possible.

This book has also been posted as a series of articles on my website. For any news,   
changes or updates, you should always refer first to [www.dedoimedo.com.](http://www.dedoimedo.com./)

20

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Expectations

OK, so you got this book downloaded to your machine. What now? Are you going to   
use it daily? Is it going to make you any smarter? Will you be more proficient using   
Linux after reading this book? Will you become a hacker? Or perhaps a kernel expert?

Linux kernel crash analysis is not an everyday topic. It is very likely a niche topic, which will interest only system administrators and professionals dabbling in the kernel. This condition may stop you from reading the book, as you may not be either the person maintaining server boxes nor the code developer trying to debug his drivers.

However, you may also consider this book as a very extensive learning lesson in what goes behind the curtains of a typical Linux system. While you may not find immediate use to the contents presented in this book, the general knowledge and problem solving methods and tools you find here should serve you universally. Come the day, come the opportunity, you will find this book of value.

I have written the book in a simple, linear, step-by-step manner, trying to make it accessible even to less knowledgeable people. I am fully aware of the paradox in mixing words inexperienced users with kernel crash analysis, but it does not have to be so. Reading this book will provide you with the confidence and understanding of what makes your Linux box tick. However, it cannot replace hands-on experience and intuition gained from actual work with Linux systems.

Therefore, you may gains tons of knowledge, but you will not become a hacker, an expert   
or a posh consultant just by reading the contents of this book. In fact, this book may   
very well frustrate you. As simple as I tried to make it be, it’s still super-uber-ultra geeky.   
You could end spending hours rereading paragraphs, trying to figure out what’s going   
on, deciphering the crash analysis reports, and trying to replicate my examples. It is   
important that you do not get discouraged. Even if glory does not await you at the last   
page, I am convinced that by mastering this book you will gain valuable knowledge. For   
some of you, it will be an eye-opener and maybe a very useful business tool. For others,   
it will be a missing piece of the puzzle called Linux. Others yet might end waiting years   
for the reward to appear.

To wrap this philosophical speech, Linux Kernel Crash Book is a highly technical piece of education with immense practical applications. It is probably the most comprehensive guide on the subject you will currently find available on the market, free, paid, hobbyist, professional, or otherwise. It’s ideally suited for administrators and IT experts. It can also make home users happy, if they are willing to take the leap of faith.

Have fun.

21

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Errata

Here be fixes to errors, spelling mistakes and other issues found in the book.

22

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Part I

LKCD

1 Introduction

[LKCD](http://lkcd.sourceforge.net/index.html) stands for Linux Kernel Crash Dump. This tool allows the Linux system to write   
the contents of its memory when a crash occurs, so that they can be later analyzed for the root cause of the crash.

Ideally, kernels never crash. In reality, the crashes sometimes occur, for whatever reason. It is in the best interest of people using the plagued machines to be able to recover from the problem as quickly as possible while collecting as much data available. The most relevant piece of information for system administrators is the memory dump, taken at the moment of the kernel crash.

Note: This book part refers to a setup on SUSE1 9.X systems.

1.1 How does LKCD work?

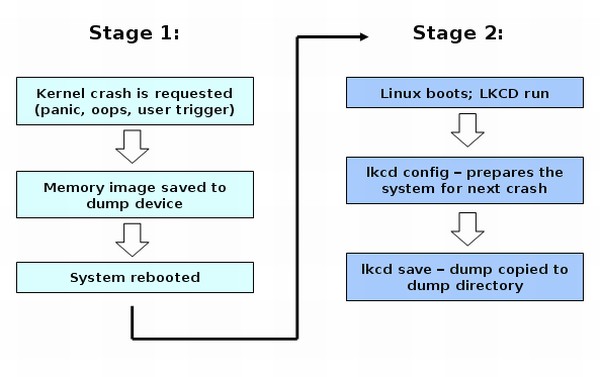
You won’t notice LKCD in your daily work. Only when a kernel crash occurs will LKCD kick into action. The kernel crash may result from a kernel panic or an oops or it may be user-triggered. Whatever the case, this is when LKCD begins working, provided it has been configured correctly. LKCD works in two stages:

1.1.1 Stage 1

This is the stage when the kernel crashes. Or more correctly, a crash is requested, either due to a panic, an oops or a user-triggered dump. When this happens, LKCD kicks into action, provided it has been enabled during the boot sequence. LKCD copies the contents of the memory to a temporary storage device, called the dump device, which is usually a swap partition, but it may also be a dedicated crash dump collection partition. After this stage is completed, the system is rebooted.

1 LKCD is an older utility and may not work well with modern kernels.

23



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

1.1.2 Stage 2

Once the system boots back online, LKCD is initiated. On different systems, this takes a different startup script. For instance, on a RedHat machine, LKCD is run by the /etc/rc.sysinit script.

Next, LKCD runs two commands. The first command is lkcd config, which we will   
review more intimately later. This commands prepares the system for the next crash.   
The second command is lkcd save, which copies the crash dump data from its temporary   
storage on the dump device to the permanent storage directory, called dump directory.

Along with the dump core, an analysis file and a map file are created and copied; we’ll talk about these separately when we review the crash analysis. A completion of this two-stage cycle signifies a successful LKCD crash dump.

Figure 1: LKCD stages

24

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

2 LKCD Installation

The LKCD installation requires kernel compilation. This is a lengthy and complex pro-  
cedure that takes quite a bit of time. It is impossible to explain how LKCD can be   
installed without showing the entire kernel compilation in detail. The kernel compilation   
is a delicate, complex process that merits separate attention; it will be presented in a   
dedicated tutorial on [www.dedoimedo.com.](http://www.dedoimedo.com./) Therefore, we will assume that we have a   
working system compiled with LKCD.

3 LKCD local dump procedure

3.1 Required packages

The host must have the lkcdutils package installed.

3.2 Configuration file

The LKCD configuration is located under /etc/sysconfig/dump. Back this up before   
making any changes! We will have to make several adjustments to this file before we can   
use LKCD.

3.2.1 Activate dump process (DUMP\_ACTIVE)

To be able to use LKCD when crashes occur, you must activate it.

DUMP\_ACTIVE="1"

3.2.2 Configure the dump device (DUMP\_DEVICE)

You should be very careful when configuring this directive. If you choose the wrong   
device, its contents will be overwritten when a crash is saved to it, causing data loss.

25

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Therefore, you must make sure that the DUMPDEV is linked to the correct dump device.   
In most cases, this will be a swap partition, although you can use any block device whose   
contents you can afford to overwrite. Accidentally, this section partially explains why the   
somewhat nebulous and historic requirement for a swap partition to be 1.5x the size of   
RAM.

What you need to do is define a DUMPDEV device and then link it to a physical block device; for example, /dev/sdb1. Let’s use the LKCD default, which calls the DUMPDEV directive to be set to /dev/vmdump.

DUMPDEV="/dev/vmdump"

Now, please check that /dev/vmdump points to the right physical device. Example:

ls -l /dev/vmdump

lrwxrwxrwx 1 root root 5 Nov 6 21:53 /dev/vmdump ->/dev/sda5

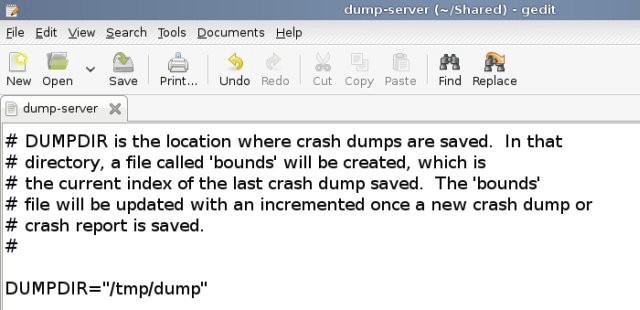
/dev/sda5 should be your swap partition or a disposable crash partition. If the symbolic link does not exist, LKCD will create one the first time it is run and will link /dev/vmdump to the first swap partition found in the /etc/fstab configuration file. Therefore, if you do not want to use the first swap partition, you will have to manually create a symbolic link for the device configured under the DUMPDEV directive.

3.2.3 Configure the dump directory (DUMPDIR)

This is where the memory images saved previously to the dump device will be copied and kept for later analysis. You should make sure the directory resides on a partition with enough free space to contain the memory image, especially if you’re saving all of it. This means 2GB RAM = 2GB space or more.

In our example, we will use /tmp/dump. The default is set to /var/log/dump.

26



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

DUMPDIR="/tmp/dump"

Figure 2: LKCD DUMPDIR directive change

3.2.4 Configure the dump level (DUMP\_LEVEL)

This directive defines what part of the memory you wish to save. Bear in mind your space restrictions. However, the more you save, the better when it comes to analyzing the crash root cause.

27

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Table 1: LKCD dump levels

|  |  |
| --- | --- |
| Value | Action |
| DUMP\_NONE (0) | Do nothing, just return if called |
| DUMP\_HEADER (1) | Dump the dump header and first 128K bytes out |
| DUMP\_KERN (2) | Everything in DUMP\_HEADER and kernel pages only |
| DUMP\_USED (4) | Everything except kernel free pages |
| DUMP\_ALL (8) | All memory |

3.2.5 Configure the dump flags (DUMP\_FLAGS)

The flags define what type of dump is going to be saved. For now, you need to know that there are two basic dump device types: local and network.

Table 2: LKCD dump flags

|  |  |
| --- | --- |
| Value | Action |
| 0x80000000 | Local block device |
| 0x40000000 | Network device |

Later, we will also use the network option. For now, we need local.

DUMP\_FLAGS="0x80000000"

28

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

3.2.6 Configure the dump compression level (DUMP\_COMPRESS)

You can keep the dumps uncompressed or use RLE or GZIP to compress them. It’s up   
to you.

DUMP\_COMPRESS="2"

I would call the settings above the "must-have" set. You must make sure these directives are configured properly for the LKCD to function. Pay attention to the devices you intend to use for saving the crash dumps.

3.2.7 Additional settings

There are several other directives listed in the configuration file. These other directives are all set to the the configuration defaults. You can find a brief explanation on each below. If you find the section inadequate, please email me and I’ll elaborate.

These include:

• DUMP\_SAVE="1" - Save the memory image to disk.

• PANIC\_TIMEOUT="5" - The timeout (in seconds) before a reboot after panic   
occurs.

• BOUNDS\_LIMIT ="10" - A limit on the number of dumps kept .

• KEXEC\_IMAGE="/boot/vmlinuz" - Defines what kernel image to use after re-  
booting the system; usually, this will be the same kernel used in normal production.

• KEXEC\_CMDLINE="root console=tty0" - Defines what parameters the kernel   
should use when booting after the crash; usually, you won’t have to tamper with this   
setting.

29

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

3.3 Enable core dump capturing

The first step we need to do is enable the core dump capturing. In other words, we need to sort of source the configuration file so the LKCD utility can use the values set in it. This is done by running the lkcd config command, followed by lkcd query command, which allows you to see the configuration settings.

lkcd config

lkcd query

The output is as follows:

Configured dump device: 0xffffffff

Configured dump flags: KL\_DUMP\_FLAGS\_DISKDUMP

Configured dump level: KL\_DUMP\_LEVEL\_HEADER| KL\_DUMP\_LEVEL\_KERN Configured dump compression method: KL\_DUMP\_COMPRESS\_GZIP

3.4 Configure LKCD dump utility to run on startup

To work properly, the LKCD must run on boot. On RedHat and SUSE machines, you can use the chkconfig utility to achieve this:

chkconfig boot.lkcd on

After the reboot, your machine is ready for crash dumping. We can begin testing the   
functionality. However, please note that disk-based dumping may not always succeed in   
all panic situations. For instance, dumping on hung systems is a best-effort attempt.   
Furthermore, LKCD does not seem to like the md RAID devices, presenting another   
problem into the equation. Therefore, to overcome the potentially troublesome situations

30

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

where you may end up with failed crash collections to local disks, you may want to consider using the network dumping option. Therefore, before we demonstrate the LKCD functionality, we’ll study the netdump option first.

4 LKCD netdump procedure

Netdump procedure is different from the local dump in having two machines involved in the process. One is the host itself that will suffer kernel crashes and whose memory image we want to collect and analyze. This is the client machine. The only difference from a host configured for local dump is that this machine will use another machine for storage of the crash dump.

The storage machine is the netdump server. Like any server, this host will run a service and listen on a port to incoming network traffic, particular to the LKCD netdump. When crashes are sent, they will be saved to the local block device on the server. Other terms used to describe the relationship between the netdump server and the client is that of source and target, if you will: the client is a source, the machine that generates the information; the server is the target, the destination where the information is sent. We will begin with the server configuration.

5 Configure LKCD netdump server

5.1 Required packages

The server must have the following two packages installed: lkcdutils and lkcdutils-  
netdump-server.

5.2 Configuration file

The configuration file is the same one, located under /etc/sysconfig/dump. Again, back   
this file up before making any changes. Next, we will review the changes you need to   
make in the file for the netdump to work. Most of the directives will remain unchanged,   
so we’ll take a look only at those specific to netdump procedure, on the server side.

31

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

5.2.1 Configure the dump flags (DUMP\_FLAGS)

This directive defines what kind of dump is going to be saved to the dump directory. Earlier, we used the local block device flag. Now, we need to change it. The appropriate flag for network dump is 0x40000000.

DUMP\_FLAGS="0x40000000"

5.2.2 Configure the source port (SOURCE\_PORT)

This is a new directive we have not seen or used before. This directive defines on which port the server should listen for incoming connections from hosts trying to send LKCD dumps. The default port is 6688. When configured, this directive effectively turns a host into a server - provided the relevant service is running, of course.

SOURCE\_PORT="6688"

5.2.3 Make sure dump directory is writable for netdump user

This directive is extremely important. It defines the ability of the netdump service to write to the partitions / directories on the server. The netdump server run as the netdump user. We need to make sure this user can write to the desired destination (dump) directory. In our case:

install -o netdump -g dump -m 777 -d /tmp/dump

You may also want to ls the destination directory and check the owner:group. It should be netdump:dump. Example:

32

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

ls -ld dump

drwxrwxrwx 3 netdump dump 96 2009-02-20 13:35 dump

You may also try getting away with manually chowning and chmoding the destination to see what happens.

5.3 Configure LKCD netdump server to run on startup

We need to configure the netdump service to run on startup. Using chkconfig to demon-  
strate:

chkconfig netdump-server on

5.4 Start the server

Now, we need to start the server and check that it’s running properly. This includes both   
checking the status and the network connections to see that the server is indeed listening   
on port 6688.

/etc/init.d/netdump-server start   
/etc/init.d/netdump-server status

Likewise:

netstat -tulpen | grep 6688

udp 0 0 0.0.0.0:6688 0.0.0.0:\* 479 37910 >> >> 22791/netdump-server

33

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Everything seems to be in order. This concludes the server-side configurations.

6 Configure LKCD client for netdump

Client is the machine (which can also be a server of some kind) that we want to collect kernel crashes for. When kernel crashes for whatever reason on this machine, we want it to send its core to the netdump server. Again, we need to edit the /etc/sysconfig/dump configuration file. Once again, most of the directives are identical to previous configura-  
tions. In fact, by changing just a few directives, a host configured to save local dumps can be converted for netdump.

6.1 Configuration file

6.1.1 Configure the dump device (DUMP\_DEV)

Earlier, we have configured our clients to dump their core to the /dev/vmdump device. However, network dump requires an active network interface. There are other consider-  
ations in place as well, but we will review them later.

DUMP\_DEV="eth0"

6.1.2 Configure the target host IP address (TARGET\_HOST)

The target host is the netdump server, as mentioned before. In our case, it’s the server machine we configured above. To configure this directive - and the one after - we need to go back to our server and collect some information, the output from the ifconfig command, listing the IP address and the MAC address. For example:

inet addr:192.168.1.3

HWaddr 00:12:1b:40:c7:63

34

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Therefore, our target host directive is set to:

TARGET\_HOST="192.168.1.3"

Alternatively, it is also possible to use hostnames, but this requires the use of hosts file, DNS, NIS or other name resolution mechanisms properly set and working.

6.1.3 Configure target host MAC address (ETH\_ADDRESS)

If this directive is not set, the LKCD will send a broadcast to the entire neighborhood2, possibly inducing a traffic load. In our case, we need to set this directive to the MAC address of our server:

ETH\_ADDRESS="00:12:1b:40:c7:63

6.1.4 Configure target host port (TARGET\_PORT)

We need to set this option to what we configured earlier for our server. This means port   
6688.

TARGET\_PORT="6688"

6.1.5 Configure the source port (SOURCE\_PORT)

Lastly, we need to configure the port the client will use to send dumps over network. Again, the default port is 6688.

2 Please note that the netdump functionality is limited to the same subnet that the server runs on. In our case, this means /24 subnet. We’ll see an example for this shortly.

35



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

SOURCE\_PORT="6688"

Figure 3: LKCD netdump client source port configuration

6.2 Enable core dump capturing

Perform the same steps we did during the local dump configuration: run the lkcd config and lkcd query commands and check the setup.

lkcd config

lkcd query

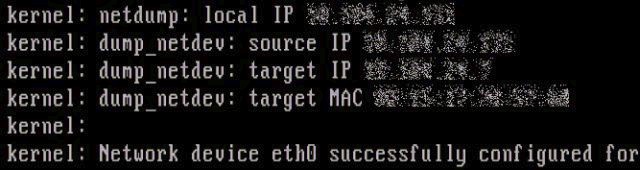
The output is as follows:

Configured dump device: 0xffffffff

Configured dump flags: KL\_DUMP\_FLAGS\_NETDUMP

Configured dump level: KL\_DUMP\_LEVEL\_HEADER| KL\_DUMP\_LEVEL\_KERN Configured dump compression method: KL\_DUMP\_COMPRESS\_GZIP

36



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

6.3 Configure LKCD dump utility to run on startup

Once again, the usual procedure:

chkconfig netdump-server on

6.4 Start the lkcd-netdump utility

Start the utility by running the /etc/init.d/lkcd-netdump script.

/etc/init.d/lkcd-netdump start

Watch the console for successful configuration message. If you see an image similar to the one below, it means you have successfully configured the client and can proceed to test the functionality.

Figure 4: LKCD netdump client successful configuration

7 Test functionality

To test the functionality, we will force a panic on our kernel. This is something you   
should be careful about doing, especially on your production systems. Make sure you

37



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

backup all critical data before experimenting. To be able to create panic, you will have to enable the System Request (SysRq) functionality on the desired clients, if it has not already been set:

echo 1 > /proc/sys/kernel/sysrq

And then force the panic:

echo c > /proc/sysrq-trigger

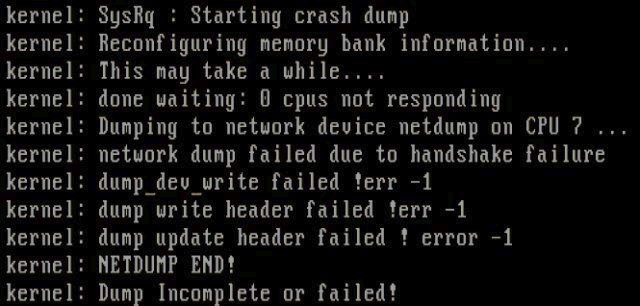
Watch the console. The system should reboot after a while, indicating a successful recovery from the panic. Furthermore, you need to check the dump directory on the netdump server for the newly created core, indicating a successful network dump. Indeed, checking the destination directory, we can see the memory core was successfully saved. And now we can proceed to analyze it.

Figure 5: Successful LKCD netdump procedure

8 Problems

You may encounter a few issues working with LKCD. Most notably, you may see config-  
uration and dump errors when trying to use the netdump functionality. Let’s review a   
typical case.

38



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

8.1 Unsuccessful netdump to different network segment

As mentioned before, the netdump functionality is limited to the same subnet. Trying   
to send the dump to a machine on a different subnet results in an error. This issue has   
no solution. Your best bet is to use a dedicated netdump server on the same 256-host   
subnet.

Figure 6: LKCD netdump failure

9 Conclusion

LKCD is a very useful application, although it has its limitations. On one hand, it provides   
with the critical ability to perform in-depth forensics on crashed systems post-mortem.   
The netdump functionality is particularly useful in allowing system administrators to save   
memory images after kernel crashes without relying on the internal hard disk space or   
the hard disk configuration. This can be particularly useful for machines with very large   
RAM, when dumping the entire contents of the memory to local partitions might be   
problematic. Furthermore, the netdump functionality allows LKCD to be used on hosts   
configured with RAID, since LKCD is unable to work with md partitions, overcoming the   
problem.

However, the limitation to use within the same network segment severely limits the   
ability to mass-deploy the netdump in large environments. It would be extremely useful

39

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

if a workaround or patch were available so that centralized netdump servers can be used without relying on specific network topography.

Lastly, LKCD is a somewhat old utility and might not work well on the modern kernels. In general, it is fairly safe to say it has been replaced by the more flexible Kdump, which we will review in the next Part.

40

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Part II

Kdump

10 Introduction

Linux kernel is a rather robust entity. It is stable and fault-tolerable and usually does not suffer irrecoverable errors that crash the entire system and require a reboot to restore to normal production. Nevertheless, these kinds of problems do occur from time to time. They are known as kernel crashes and are of utmost interest and importance to administrators in charge of these systems. Being able to detect the crashes, collect them and analyze them provides the system expert with a powerful tool in finding the root cause to crashes and possibly solving critical bugs.

In the previous part (I), we have learned how to setup, configure and use Linux Kernel Crash Dump (LKCD) utility. However, LKCD, being an older project, exhibited several major limitations in its functionality: LKCD was unable to save memory dumps to local RAID (md) devices and its network capability was restricted to sending memory cores to dedicated LKCD netdump servers only on the same subnet, provided the cores were under 4GB in size. Memory cores exceeding the 32-bit size barrier were corrupt upon transfer and thus unavailable for analysis. The same-subnet also proved impractical for large-scale operations with thousands of machines.

Kdump is a much more flexible tool, with extended network-aware capabilities. It aims to replace LKCD, while providing better scalability. Indeed, Kdump supports network dumping to a range of devices, including local disks, but also NFS areas, CIFS shares or FTP and SSH servers. This makes if far more attractive for deployment in large environments, without restricting operations to a single server per subnet.

In this part of the book, we will learn how to setup and configure Kdump for memory   
core dumping to local disks and network shares. We will begin with a short overview of   
basic Kdump functionality and terminology. Next, we will review the kernel compilation   
parameters required to use Kdump. After that, we will go through the configuration file   
and study each directive separately, step by step. We will also edit the GRUB menu   
as a part of the Kdump setup. Lastly, we will demonstrate the Kdump functionality,   
including manually triggering kernel crashes and dumping memory cores to local and   
network devices. In the Appendix section (V), you will also be able to learn about   
changes and new functionality added in later versions of Kdump, plus specific setups for   
openSUSE and CentOS.

41

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Note: This book part refers to a setup on SUSE3 10.3 systems. The Appendix section contains additional information about SUSE 11.X and RedHat4 5.X systems.

10.1 Restrictions

On one hand, this book will examine the Kdump utility in great detail. On the other, a number of Kdump-related topics will be only briefly discussed. It is important that you know what to expect.

10.1.1 Kernel compilation

I will not explain the Kernel compilation in this book, although I will explain the pa-  
rameters required for proper Kdump functionality. The kernel compilation is a delicate,   
complex process that merits separate attention; it will be presented in a dedicated tutorial   
on [www.dedoimedo.com.](http://www.dedoimedo.com./)

10.1.2 Hardware-specific configurations

Kdump can also run on the Itanium (ia64) and Power PC (ppc64) architectures. However,   
due to relative scarcity of these platforms in both the home and business use, I will focus   
on the i386 (and x86-64) platforms. The platform-specific configurations for Itanium and   
PPC machines can be found in the official Kdump documentation (see References (33)).

Now, let us begin.

10.2 How does Kdump work?

10.2.1 Terminology

To make things easier to understand, here’s a brief lexicon of important terms we will use in this book:

3 SUSE refers to both openSUSE and SUSE Linux Enterprise Server (SLES)

4 RedHat refers to both CentOS and RedHat Enterprise Linux (RHEL)

42

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

• Standard (production) kernel - kernel we normally work with

• Crash (capture) kernel - kernel specially used for collecting crash dumps5 Kdump has two main components - Kdump and Kexec.

10.2.2 Kexec

Kexec is a fastboot mechanism that allows booting a Linux kernel from the context of an already running kernel without going through BIOS. BIOS can be very time consuming, especially on big servers with numerous peripherals. This can save a lot of time for developers who end up booting a machine numerous times.

10.2.3 Kdump

Kdump is a new kernel crash dumping mechanism and is very reliable. The crash dump is captured from the context of a freshly booted kernel and not from the context of the crashed kernel. Kdump uses Kexec to boot into a second kernel whenever the system crashes. This second kernel, often called a crash or a capture kernel, boots with very little memory and captures the dump image.

The first kernel reserves a section of memory that the second kernel uses to boot. Kexec   
enables booting the capture kernel without going through BIOS hence the contents of   
the first kernel’s memory are preserved, which is essentially the kernel crash dump.

11 Kdump installation

There are quite a few requirements that must be met in order for Kdump to work.

5 I will sometimes use only partial names when referring to these two kernels. In general, if I do not   
specifically use the words crash or capture to describe the kernel, this means we’re talking about the   
production kernel.

43

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

• The production kernel must be compiled with a certain set of parameters required for kernel crash dumping.

• The production kernel must have the kernel-kdump package installed. The kernel-  
kdump package contains the crash kernel that is started when the standard kernel   
crashes, providing an environment in which the standard kernel state during the crash   
can be captured. The version of the kernel-dump package has to be identical to the   
standard kernel.

If the operating system comes with a kernel already compiled to run and use Kdump,   
you will have saved quite a bit of time. If you do not have a kernel built to support   
the Kdump functionality, you will have to do quite a bit of work, including a lengthy   
compilation and configuration procedure of both the standard, production kernel and the   
crash (capture) kernel.

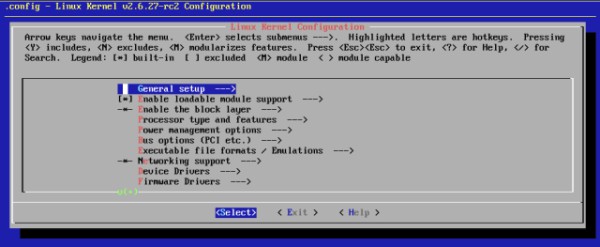
In this book, we will not go into details on kernel compilation. The compilation is a generic   
procedure that does not directly relate to Kdump and demands dedicated attention. We   
will talk about kernel compilation in a separate tutorial on [www.dedoimedo.com.](http://www.dedoimedo.com./) Here,   
we will take the compilation for granted and focus on the configuration.

Nevertheless, although we won’t compile, we will have to go through the list of kernel   
parameters that have to be configured so that your system can support the Kexec/Kdump   
functionality and collect crash dumps. These parameters need to be configured prior to   
kernel compilation.

The simplest way to configure kernel parameters is to invoke a kernel configuration wizard such as menuconfig or xconfig.

The kernel configuration wizard can be text (menuconfig ) or GUI driven (xconfig ). In   
both cases, the wizard contains a list of categories, divided into subcategories, which   
contain different tunable parameters. Just to give you an impression of what kernel   
compilation configuration looks like, for those of you who have never seen one:

44



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 7: Kernel compilation wizard

What you see above is the screenshot of a typical kernel configuration menu, ran inside the terminal. The wizard uses the text interface and is invoked by typing make menuconfig. Notice the category names; we will refer to them soon.

We will now go through the list of kernel parameters that need to be defined to enable Kdump/Kexec to function properly. For the sake of simplicity, this book part focuses on the x86 (x86\_64) architecture. For some details about other platforms and exceptions, please refer to the Appendix (V) and the official documentation.

11.1 Standard (production) kernel

The standard kernel can be a vanilla kernel downloaded from [The linux Kernel Archives](http://kernel.org/)   
or one of your favorite distributions. Whichever you choose, you will have to configure the kernel with the following parameters:

11.1.1 Under Processor type and features

Enable Kexec system call: This parameter tells the system to use Kexec to skip BIOS and boot (new) kernels. It is critical for the functionality of Kdump.

CONFIG\_KEXEC=y

45

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Enable kernel crash dumps: Crash dumps need to be enabled. Without this option, Kdump will be useless.

CONFIG\_CRASH\_DUMP=y

Optional: Enable high memory support (for 32-bit systems): You need to con-  
figure this parameter in order to support memory allocations beyond the 32-bit (4GB) barrier. This may not be applicable if your system has less than 4GB RAM or if you’re using a 64-bit system.

CONFIG\_HIGHMEM4G=y

Optional: Disable Symmetric Multi-Processing (SMP) support: Kdump can only work with a single processor. If you have only a single processor or you run your machine with SMP support disabled, you can safely set this parameter to (n).

CONFIG\_SMP=y

On the other hand, if your kernel must use SMP for whatever reason, you will want to set this directive to (y). However, you will have to remember this during the Kdump configuration. We will have to set Kdump to use only a single CPU. It is very important that you remember this!

To recap, you can either disable SMP during the compilation - OR - enable SMP but instruct Kdump to use a single CPU. This instruction is done by changing the Kdump configuration file. It is NOT a part of the kernel compilation configuration.

The configuration file change requires that one of the options be configured in a particular   
manner. Specifically, the directive below needs to be set in the Kdump configuration file   
under /etc/sysconfig/kdump AFTER the kernel has been compiled and installed.

46

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

KDUMP\_COMMANDLINE\_APPEND=”maxcpus=1 ”

11.1.2 Under Filesystems > Pseudo filesystems

Enable sysfs file system support: Modern kernel support (2.6 and above) this setting by default, but it does not hurt to check.

CONFIG\_SYSFS=y

Enable /proc/vmcore support: This configuration allows Kdump to save the memory dump to /proc/vmcore. We will talk more about this later. Although in your setup you may not use the /proc/vmcore as the dump device, for greatest compatibility, it is recommended you set this parameter to (y).

CONFIG\_PROC\_VMCORE=y

11.1.3 Under Kernel hacking

Configure the kernel with debug info: This parameter means the kernel will be built with debug symbols. While this will increase the size of the kernel image, having the symbols available is very useful for in-depth analysis of kernel crashes, as it allows you to trace the problems not only to problematic function calls causing the crashes, but also the specific lines in relevant sources. We will talk about this in great detail when we setup the crash, lcrash and gdb debugging utilities in the next part (III).

CONFIG\_DEBUG\_INFO=y

47

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

11.1.4 Other settings

Configure the start section for reserved RAM for the crash kernel6: This is a very important setting to pay attention to. To work properly, the crash kernel uses a piece of memory specially reserved to it. The start section for this memory allocation needs to be defined. For instance, if you intend to start the crash kernel RAM at 16MB7, then the value needs to be set to the following (in hexadecimal):

CONFIG\_PHYSICAL\_START=0x1000000

Configure kdump kernel so it can be identified: Setting this suffix allows kdump to select the right kernel for boot, since there may be several kernels under /boot on your system. In general, the rule of thumb calls for the crash kernel to be named the same as your production kernel, save for the -kdump suffix. You can check this by running the uname -r command in terminal, to see the kernel version you run and then check the files listed in the /boot directory.

CONFIG\_LOCALVERSION=”-kdump”

Please note that the above table is neither a holy bible nor rocket science. As always, it is   
quite possible that my observations are limited and apply only to a very specific, private   
setup. Therefore, please exercise discretion when using the above table for reference,   
taking into consideration the fact that you may not experience the same success as   
myself. That said, I have thoroughly tested the setup and it works flawlessly.

Now, your next step is to compile the kernel. I cannot dedicate the resource to cover   
the kernel compilation procedure at this point. However, if you’re using Kdump as a   
part of your production environment - rather than household hobby - there are pretty fair   
chances you will have dedicated support from vendors, which should provide you with

6 This parameter need special attention on openSUSE 11 and higher. Please refer to Appendix (V) for more details.

7 You may use other values that suit your operational needs. Make sure the allocation does not conflict with reserved memory used by the kernel or kernel modules.

48

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

the kernel already compiled for Kdump. I apologize for this evasion, but I must forgo the kernel compilation for another time.

Modern distributions, especially those forked off enterprise solutions, are configured to use Kdump. openSUSE 11.1 is a good example; you will only have to install the missing RPMs and edit the configuration file to get it to work. We will discuss openSUSE 11.1 some more later in the book.

11.2 Crash (capture) kernel

This kernel needs to be compiled with the same parameters as above, save one exception. Kdump does not support compressed kernel images as crash (capture) kernels8. There-  
fore, you should not compress this image. This means that while your production kernels will most likely be named vmlinuz, the Kdump crash kernels need to be uncompressed, hence named vmlinux, or rather vmlinux-kdump.

12 Kdump packages & files

12.1 Kdump packages

This is the list of required packages that must be installed on the system for Kdump to work. Please note that your kernel must be compiled properly for these packages to work as expected. It is very likely that you will succeed in installing them anyhow, however this is no guarantee that they will work.

Table 3: Kdump required packages

|  |  |
| --- | --- |
| Package name | Package info |
| kdump | Kdump package |
| kexec-tools | Kexec package |
| kernel-debuginfo9 | Crash analysis package (optional) |

8 This has changed in the more recent versions of Kdump. Please refer to the Appendix (V) for more   
details.

49

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

The best way to obtain these packages is from your software repositories. This guarantees   
you will be using the most compatible version of Kdump and Kexec. For example, on   
Debian-based systems, you can use the apt-get install command to fetch the necessary   
packages:

apt-get install <package name>

Likewise, please note that the production kernel also must have the kernel-kdump package installed. This package contains the crash kernel that is started when the standard kernel crashes, providing an environment in which the standard kernel state during the crash can be captured. The version of this package has to be identical to the production kernel. For details about how to obtain the kernel-kdump and kexec-tools packages not via the software repositories, please refer to the Appendix (V).

12.2 Kdump files

Here’s the list of the most important Kdump-related files: Table 4: Kdump files

|  |  |
| --- | --- |
| Path | Info |
| /etc/init.d/kdump10 | Kdump service |
| /etc/sysconfig/kdump11 | Kdump configuration file |
| /usr/share/doc/packages/kdump | Kdump documentation |

The Kdump installation also includes the GDB Kdump wrapper script (gdb-kdump),   
which is used to simplify the use of GDB on Kdump images. The use of GDB, as well as

9 The kernel-debuginfo package needs to match your kernel version - default, smp, etc.

11 The startup script has changed on the recent versions of SUSE systems.

11 The configuration file on RedHat-based systems is located under /boot/kdump.conf.

50

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

other crash analysis utilities requires the presence of the kernel-debuginfo package. On   
SUSE systems, the Kdump installation also includes the YaST module (yast2-kdump).

13 Kdump configuration

In the last section, we went through the kernel configuration parameters that need to be set for Kexec/Kdump to work properly. Now, assuming you have a functioning kernel that boots to the login screen and has been compiled with the relevant parameters, whether by a vendor or yourself, we will see what extra steps we need to take to make Kdump actually work and collect crash dumps.

We will configure Kdump twice: once for local dump and once for network dump, similarly   
to what we did with LKCD. This is a very important step, because LKCD is limited to   
network dumping only within the specific subnet of the crash machine. Kdump offers a   
much greater, more flexible network functionality, including FTP, SSH, NFS and CIFS   
support.

13.1 Configuration file

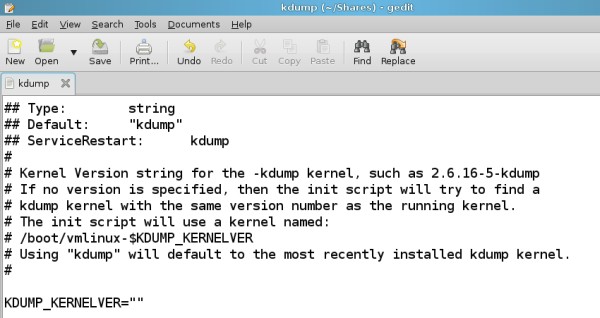
The configuration file for Kdump is /etc/sysconfig/kdump. We will start with the basic, local dump functionality. Later, we will also demonstrate a crash dump over network. You should save a backup before making any changes!

13.1.1 Configure KDUMP\_KERNELVER

This setting refers to the CONFIG\_LOCALVERSION kernel configuration parameter that we reviewed earlier. We specified the suffix -kdump, which tells our system to use kernels with -kdump suffix as crash kernels. Like the short description paragraph specifies, if no value is used, the most recently installed Kdump kernel will be used. By default, crash kernels are identified by the -kdump suffix.

In general, this setting is meaningful only if non-standard suffices are used for Kdump kernels. Most users will not need touch this setting and can leave it at the default value, unless they have very specific needs that require certain kernel versions.

51



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

KDUMP\_KERNELVER=””

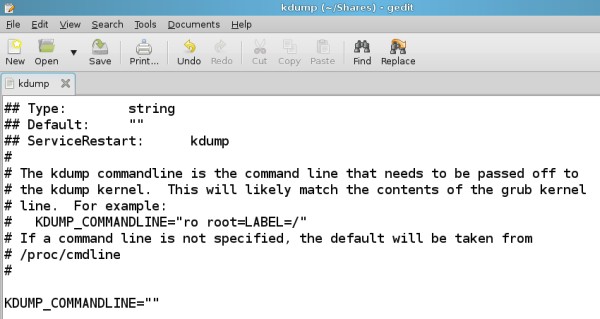
Figure 8: Kdump kernel version configuration

13.1.2 Configure KDUMP\_COMMANDLINE

This settings tells Kdump the set of parameters it needs to boot the crash kernel with.   
In most cases, you will use the same set as your production kernel, so you won’t have to   
change it. To see the current set, you can issue the cat command against /proc/cmdline.   
When no string is specified, this is the set of parameters that will be used as the default.   
We will use this setting when we test Kdump (or rather, Kexec) and simulate a crash   
kernel boot.

KDUMP\_COMMANDLINE=””

52



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

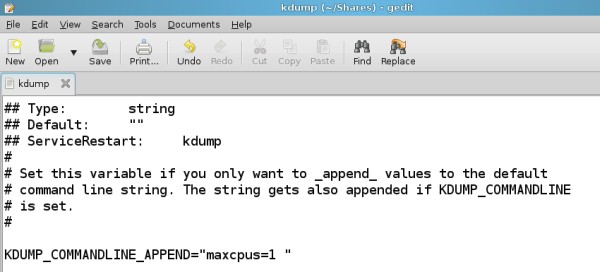
Figure 9: Kdump command line configuration

13.1.3 Configure KDUMP\_COMMANDLINE\_APPEND

This is a very important directive. It is extremely crucial if you use or have to use an SMP kernel. We have seen earlier during, the configuration of kernel compilation parameters, that Kdump cannot use more than a single core for the crash kernel. Therefore, this parameter is a MUST if you’re using SMP. If the kernel has been configured with SMP disabled, you can ignore this setting.

KDUMP\_COMMANDLINE\_APPEND=”MAXCPUS=1 ”

53



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 10: Kdump command line append configuration

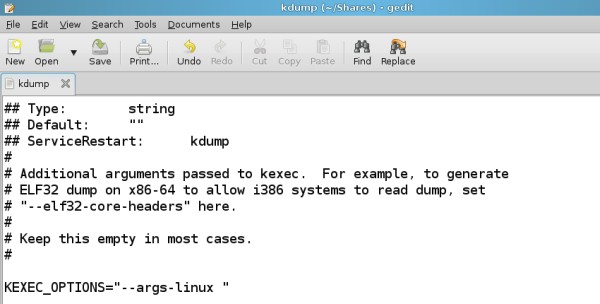
13.1.4 Configure KEXEC\_OPTIONS

As we’ve mentioned earlier, Kexec is the mechanism that boots the crash kernel from the context of the production kernel. To work properly, Kexec requires a set of arguments. The basic set used is defined by the /proc/cmdline. Additional arguments can be specified using this directive. In most cases, the string can be left empty. However, if you receive strange errors when starting Kdump, it is likely that Kdump on your particular kernel version cannot parse the arguments properly. To make Kdump interpret the additional parameters literally, you may need to add the string -args-linux.

You should try both settings and see which one works for you. If you’re interested, you can Google for “-args-linux” and see a range of mailing list threads and bug entries revolving around this subject. Nothing decisive, so trial is your best choice here. We’ll discuss this some more later on.

KDUMP\_OPTIONS=”--args-linux ”

54



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 11: Kdump options configurations

13.1.5 Configure KDUMP\_RUNLEVEL

This is another important directive. If defines the runlevel into which the crash kernel should boot. If you want Kdump to save crash dumps only to a local device, you can set the runlevel to 1. If you want Kdump to save dumps to a network storage area, like NFS, CIFS or FTP, you need the network functionality, which means the runlevel should be set to 3. You can also use 2, 5 and s. If you opt for runlevel 5 (not recommended), make sure the crash kernel has enough memory to boot into the graphical environment. The default 64MB is most likely insufficient.

KDUMP\_RUNLEVEL=”1”

13.1.6 Configure KDUMP\_IMMEDIATE\_REBOOT

This directive tells Kdump whether to reboot out of the crash kernel once the dump is   
complete. This directive is ignored if the KDUMP\_DUMPDEV parameter (see below)   
is not empty. In other words, if a dump device is used, the crash kernel will not be

55

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

rebooted until the transfer and possibly additional post-processing of the dump image to   
the destination directory are completed. You will most likely want to retain the default   
value.

KDUMP\_IMMEDIATE\_REBOOT=”yes”

13.1.7 Configure KDUMP\_TRANSFER

This setting tells Kdump what to do with the dumped memory core. For instance,   
you may want to post-process it instantly. KDUMP\_TRANSFER requires the use of   
a non-empty KDUMP\_DUMPDEV directive. Available choices are /proc/vmcore and   
/dev/oldmem. This is similar to what we’ve seen with LKCD utility. Normally, either   
/proc/vmcore or /dev/oldmem will point out to a non-used swap partition.

For now, we will use only the default setting, which is just to copy the saved core image to   
KDUMP\_SAVEDIR. We will talk about the DUMPDEV and SAVEDIR directives shortly.   
However, we will study the more advanced transfer options only when we discuss crash   
analysis utilities.

KDUMP\_TRANSFER=””

13.1.8 Configure KDUMP\_SAVEDIR

This is a very important directive. It tells us where the memory core will be saved. Currently, we are talking about local dump, so for now, our destination will point to a directory on the local filesystem. Later on, we will see a network example. By default, the setting points to /var/log/dump.

KDUMP\_SAVEDIR=”file:///var/log/dump”

56

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

We will change this to:

KDUMP\_SAVEDIR=”file:///tmp/dump”

Please pay attention to the syntax. You can also use the absolute directory paths inside   
the quotation marks without prefix, but this use is discouraged. You should specify what   
kind of protocol is used, with file:// for local directories, nfs:// for NFS storage and   
so on. Furthermore, you should make sure the destination is writable and that is has   
sufficient space to accommodate the memory cores. The KDUMP\_SAVEDIR directive   
can be used in conjunction with KDUMP\_DUMPDEV, which we will discuss a little later   
on.

13.1.9 Configure KDUMP\_KEEP\_OLD\_DUMPS

This settings defines how many dumps should be kept before rotating. If you’re short on space or are collecting numerous dumps, you may want to retain only a small number of dumps. Alternatively, if you require a backtrace as long and thorough as possible, increase the number to accommodate your needs. The default value is 5:

KEEP\_OLD\_DUMPS=5

To keep an infinite number of old dumps, set the number to 0. To delete all existing dumps before writing a new one, set the number to -2. Please note the somewhat strange values, as they are counterintuitive.

Table 5: Kdump dump retention

|  |  |
| --- | --- |
| Value | Dumps kept |
| 0 | all (infinite number) |
| -2 | none |

57

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

13.1.10 Configure KDUMP\_FREE\_DISK\_SIZE

This value defines the minimum free space that must remain on the target partition,   
where the memory core dump destination directory is located, after accounting for the   
memory core size. If this value cannot be met, the memory core will not be saved, to   
prevent possible system failure. The default value is 64MB. Please note it has nothing to   
do with the memory allocation in GRUB. This is an unrelated, purely disk space setting.

KDUMP\_FREE\_DISK=”64GB”

13.1.11 Configure KDUMP\_DUMPDEV

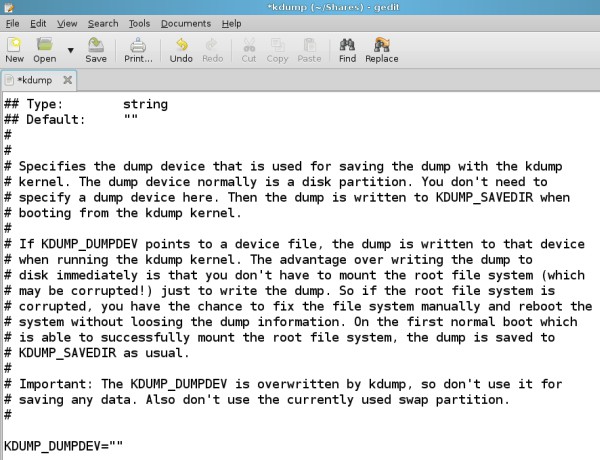
This is a very important directive. We have mentioned it several times before. KDUMP\_   
DUMPDEV does not have to be used, but you should carefully consider whether you   
might need it. Furthermore, please remember that this directive is closely associated with   
several other settings, so if you do use it, the functionality of Kdump will change.

First, let’s see when it might be prudent to use KDUMP\_DUMPDEV : Using this directive   
can be useful if you might be facing filesystem corruption problems. In this case, when   
a crash occurs, it might not be possible to mount the root filesystem and write to the   
destination directory (KDUMP\_SAVEDIR). Should that happen, the crash dump will fail.   
Using KDUMP\_DUMPDEV allows you to write to a device or a partition in raw mode,   
without any consideration to underlying filesystem, circumventing any filesystem-related   
problems.

This also means that there will be no KDUMP\_IMMEDIATE\_REBOOT ; the directive will also be ignored, allowing you to use the console to try to fix system problems manually, like check the filesystem, because no partition will be mounted and used. Kdump will examine the KDUMP\_DUMPDEV directive and if it’s not empty, it will copy the contents from the dump device to the dump directory (KDUMP\_SAVEDIR).

On the other hand, using KDUMP\_DUMPDEV increases the risk of disk corruption in   
the recovery kernel environment. Furthermore, there will be no immediate reboot, which   
slows down the restoration to production. While such a solution is useful for small scale   
operations, it is impractical for large environments. Moreover, take into account that   
the dump device will always be irrecoverably overwritten when the dump is collected,   
destroying data present on it. Secondly, you cannot use an active swap partition as the   
dump device.

58



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

KDUMP\_DUMPDEV=””

Figure 12: Kdump DUMPDEV configuration

13.1.12 Configure KDUMP\_VERBOSE

This is a rather simple, administrative directive. It tells how much information is output to   
the user, using bitmask values in a fashion similar to the chmod command. By default, the   
Kdump progress is written to the standard output (STDOUT) and the Kdump command   
line is written into the syslog. If we sum the values, we get command line (1) + STDOUT

(2) = 3. See below for all available values:

59

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

KDUMP\_VERBOSE=3

Table 6: Kdump verbosity configuration

|  |  |
| --- | --- |
| Value | Action |
| 1 | Kdump command line written to syslog |
| 2 | Kdump progress written to STDOUT |
| 4 | Kdump command line written to STDOUT |
| 8 | Kdump transfer script debugged |

13.1.13 Configure KDUMP\_DUMPLEVEL

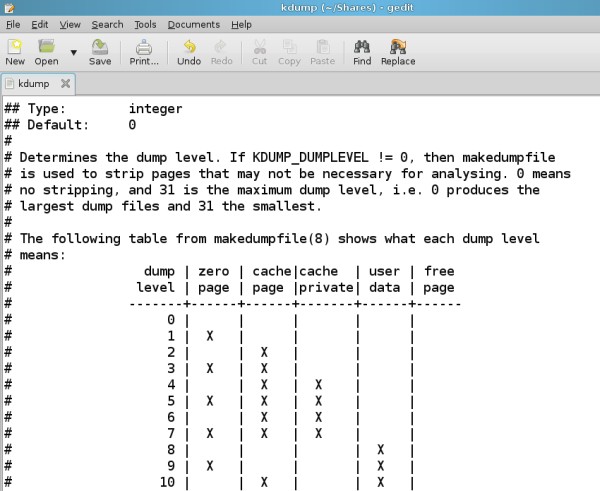
This directive defines the level of data provided in the memory dump. Values range from

0 to 32. Level 0 means the entire contents of the memory will be dumped, with no detail omitted. Level 32 means the smallest image. The default value is 0.

KDUMP\_DUMPLEVEL=”0”

You should refer to the configuration file for exact details about what each level offers and plan accordingly, based on your available storage and analysis requirements. You are welcome to try them all. I recommend using 0, as it provides most information, even though it requires hefty space.

60



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 13: Kdump DUMPLEVEL configuration

13.1.14 Configure KDUMP\_DUMPFORMAT

This setting defines the dump format. The default selection is ELF, which allows you   
to open the dump with gdb and process it. You can also use compressed, but you can   
analyze the dump only with the crash utility. We will talk about these two tools in great   
detail in the next part. The default and recommended choice is ELF, even though the   
dump file is larger.

KDUMP\_DUMPLEVEL=”ELF”

61

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

13.2 GRUB menu changes

Because of the way it works, Kdump requires a change to the kernel entry in the GRUB   
menu. As you already know, Kdump works by booting from the context of the crashed   
kernel. In order for this feature to work, the crash kernel must have a section of mem-  
ory available, even when the production kernel crashes. To this end, memory must be   
reserved.

In the kernel configurations earlier, we declared the offset point for our memory reserva-  
tion. Now, we need to declare how much RAM we want to give our crash kernel. The   
exact figure will depend on several factors, including the size of your RAM and possibly   
other restrictions. If you read various sources online, you will notice that two figures are   
mostly used: 64MB and 128MB. The first is the default configuration and should work.   
However, if it proves unreliable for whatever reason, you may want to try the second   
value. Test-crashing the kernel a few times should give you a good indication whether   
your choice is sensible or not.

Now, let us edit the GRUB configuration file12. First, make sure you backup the file before any changes.

cp /boot/grub/menu.lst /boot/grub/menu.lst-backup

Open the file for editing. Locate the production kernel entry and append the following:

crashkernel=XM@YM

YM is the offset point we declared during the kernel compilation - or has been configured for us by the vendor. In our case, this is 16M. XM is the size of memory allocated to the crash kernel. Like I’ve mentioned earlier, the most typical configuration will be either 64M or 128MB. Therefore, the appended entry should look like:

12 If you’re using GRUB2, the editing of the configuration file must be done via scripts and not manually.   
Please refer to [www.dedoimedo.com](http://www.dedoimedo.com/) for a complete GRUB2 tutorial.

62

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

crashkernel=64M@16M

A complete stanza inside the menu.list file:

title Some Linux

root (hd0,1) kernel /boot/vmlinuz root=/dev/sda1

resume=/dev/sda5 splash=silent crashkernel=64M@16M

13.3 Set Kdump to start on boot

We now need to enable Kdump on startup. This can be done using chkconfig or sysv-  
rc-conf utilities on RedHat- or Debian-based distros, respectively. For a more detailed   
tutorial about the usage of these tools, please take a look at [this](http://www.dedoimedo.com/computers/linux-system-utilities.html) tutorial online.

For example, using the chkconfig utility13:

chkconfig dump on

Changes to the configuration file require that the Kdump service be restarted. However,   
the Kdump service cannot run unless the GRUB menu change has been affected and the   
system rebooted. You can easily check this by trying to start the Kdump service:

/etc/init.d/kdump start

If you have not allocated the memory or if you have used the wrong offset, you will get an error. Something like this:

13 The service name has changed in SUSE 11 and above; please refer to Appendix (V) for more details.

63

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

/etc/init.d/kdump start   
Loading kdump failed

Memory for crashkernel is not reserved

Please reserve memory by passing "crashkernel=X@Y"

parameter to the kernel Then try loading kdump kernel

If you receive this error, this means that the GRUB configuration file has not been edited   
properly. You will have to make the right changes, reboot the system and try again.   
Once this is done properly, Kdump should start without any errors. We will mention this   
again when we test our setup. This concludes the configurations section. Now, let’s test   
it.

14 Test configuration

Before we start crashing our kernel for real, we need to check that our configuration really works. This means executing a “dry” run with Kexec. In other words, configure Kexec to load with desired parameters and boot the crash (capture) kernel. If you successfully pass this stage, this means your system is properly configured and you can test the Kdump functionality with a real kernel crash.

Again, if your system comes with the kernel already compiled to use Kdump, you will have   
saved a lot of time and effort. Basically, the Kdump installation and the configuration   
test are completely unnecessary. You can proceed straight away to using Kdump.

14.1 Configurations

14.1.1 Kernel

First, let’s quickly check that our kernel has been compiled with relevant parameters14:

14 This configuration is relevant for SUSE-based systems. On RedHat-based systems, the kernel config-  
uration is located under /boot/config.

64

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

zcat /proc/config.gz

If everything is as expected, we can proceed on to the next step. Please note that /proc/config.gz is not available for all distributions.

14.1.2 GRUB menu

Next, you need to make sure your production kernel is configured to allocate memory to the crash kernel. This means that the crashkernel=XM@YM string has to be appended to the relevant GRUB kernel entry and that you’re using the correct offset, as specified in the kernel parameters. As we’ve seen earlier, the memory allocation requires a reboot to take effect. Then, try to start the Kdump service:

/etc/init.d/kdump start

If you have not allocated the memory or used the wrong offset, you will get an error. Something like this:

/etc/init.d/kdump start   
Loading kdump failed

Memory for crashkernel is not reserved

Please reserve memory by passing "crashkernel=X@Y"

parameter to the kernel Then try loading kdump kernel

The error is quite descriptive and rather self-explanatory. You will have to edit the GRUB configuration file, reboot and try again. Once you do it properly, Kdump should start without any errors.

65

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

14.2 Load Kexec with relevant parameters

Our first step is to load Kexec with desired parameters into the existing kernel. Usually, you will want Kdump to run with the same parameters your production kernel booted with. So, you will probably use the following configuration to test Kdump:

/usr/local/sbin/kexec -l /boot/vmlinuz-‘uname -r‘

--initrd=/boot/initrd-‘uname -r‘

--command-line=‘cat /proc/cmdline‘

Then, execute Kexec (it will load the above parameters):

/usr/local/sbin/kexec -e

Your crash kernel should start booting. As said before, it will skip BIOS, so you should see the boot sequence in your console immediately. If this step completes successfully without errors, you are on the right path. I would gladly share a screenshot here, but it would look just like any other boot, so it’s useless. The next step would be to load the new kernel for use on panic. Reboot and then test:

/usr/local/sbin/kexec -p

14.2.1 Possible errors

At this stage, you may encounter a possible error. Something like this:

66

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

kexec\_load failed: Cannot   
assign requested address   
entry = 0x96550 flags = 1   
nr\_segments = 4

segment[0].buf = 0x528aa0   
segment[0].bufsz = 2044   
segment[0].mem = 0x93000   
segment[0].memsz = 3000   
segment[1].buf = 0x521880   
segment[1].bufsz = 7100   
segment[1].mem = 0x96000   
segment[1].memsz = 9000

segment[2].buf = 0x2aaaaaf1f010   
segment[2].bufsz = 169768   
segment[2].mem = 0x100000   
segment[2].memsz = 16a000   
segment[3].buf = 0x2aaaab11e010   
segment[3].bufsz = 2f5a36   
segment[3].mem = 0xdf918000

If this happens, this means you have one of the three following problems:

1. You have not configured the production kernel properly and Kdump will not work.   
 You will have to go through the installation process again, which includes compiling   
 the kernel with relevant parameters.

2. The Kexec version you are using does not match the kernel-kdump package. Make   
 sure the right packages are selected. You should check the installed versions of the   
 two packages - kernel-kdump and kexec-tools. Refer to the [official](http://lse.sourceforge.net/kdump/) website for details.

3. You may be missing -args-linux in the configuration file, under KEXEC\_OPTIONS.

Once you successfully solve this issue, you will be able to proceed with testing. If the crash kernel boots without any issues, this means you’re good to go and can start using Kdump for real.

67



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

15 Simulate kernel crash

We can begin the real work here. Like with LKCD, we will simulate a crash and watch   
magic happen. To manually crash the kernel, you will have to enable the System Request   
(SysRq) functionality (A.K.A. magic keys), if it has not already been enabled on your   
system(s), and then trigger a kernel panic. Therefore, first, enable the SysRq:

echo 1 > /proc/sys/kernel/sysrq

Then, crash the kernel:

/echo c > /proc/sysrq-trigger

Now watch the console. The crash kernel should boot up. After a while, you should   
see Kdump in action. Let’s see what happens in the console. A small counter should   
appear, showing you the progress of the dump procedure. This means you have most   
likely properly configured Kdump and it’s working as expected. Wait until the dump   
completes. The system should reboot into the production kernel when the dump is   
complete.

Figure 14: Console view of crash kernel dumping memory core

Indeed, checking the destination directory, you should see the vmcore file15.

15 On more modern versions of openSUSE, the contents of the directory include additional files. Please refer to Memory cores (18.4) in the Crash Collection part and Appendix (V) for more details.

68



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 15: Contents of dumped memory core directory

This concludes the local disk dump configuration. Now, we will see how Kdump handles network dump.

16 Kdump network dump functionality

Being able to send kernel crash dumps to network storage makes Kdump attractive for   
deployment in large environments. It also allows system administrators to evade local   
disk space limitations. Compared to LKCD, Kdump is much more network-aware; it   
is not restricted to dumping on the same subnet and there is no need for a dedicated   
server. You can use NFS areas or CIFS shares as the archiving destination. Best of all,   
the changes only affect the client side. There is no server-side configuration.

16.1 Configuration file

To make Kdump send crash dumps to network storage, only two directives in the configu-  
ration file need to be changed for the entire procedure to work. The other settings remain identical to local disk functionality, including starting Kdump on boot, GRUB menu ad-  
dition, and Kexec testing. The configuration file is located under /etc/sysconfig/kdump. As always, before effecting a change, backup the configuration file.

16.1.1 Configure KDUMP\_RUNLEVEL

To use the network functionality, we need to configure Kdump to boot in runlevel 3. By   
default, runlevel 1 is used. Network functionality is achieved by changing the directive.

KDUMP\_RUNLEVEL=3

69

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

16.1.2 Configure KDUMP\_SAVEDIR

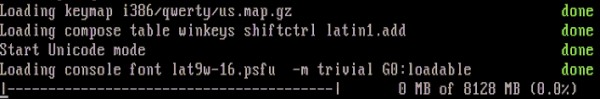
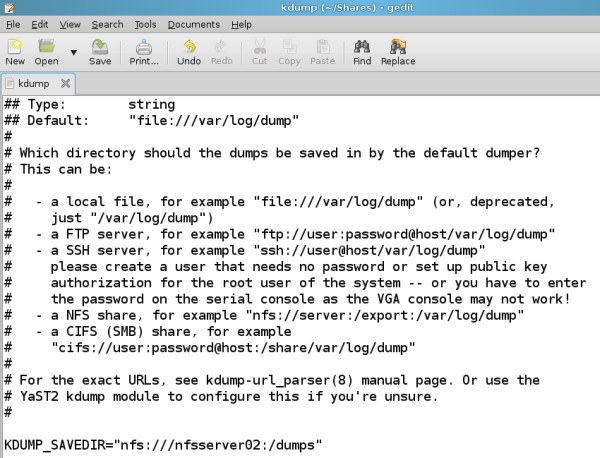
The second step is to configure the network storage destination. We can no longer use the local file. We need to use either an NFS area, a CIFS share or an SSH or an FTP server. In this book, we will configure an NFS area, because it seems the most sensible choice for sending crash dumps to. The configuration of the other two is very similar, and just as simple. The one thing you will have to pay attention to is the notation. You need to use the correct syntax:

KDUMP\_SAVEDIR=”nfs:///<server>:/<dir>

<server> refers to the NFS server, either by name or IP address. If you’re using a   
name, you need to have some sort of a name resolution mechanism in your environment,   
like hosts file or DNS. <dir> is the exported NFS directory on the NFS server. The   
directory has to be writable by the root user. In our example, the directive takes the   
following form:

KDUMP\_SAVEDIR=”nfs:///nfsserver02:/dumps

70



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 16: Kdump SAVEDIR network configuration

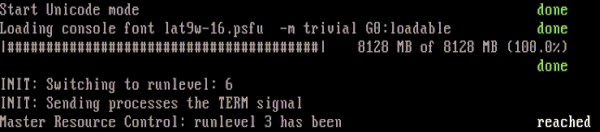
These are the two changes required to make Kdump send memory dumps to a NFS storage area in the case of a kernel crash. Now, we will test the functionality.

16.1.3 Kernel crash dump NFS example

Like the last time, we will trigger a kernel crash using the Magic Keys and observe the   
progress in the console. You should a progress bar, showing the percentage of memory   
core dumped (copied) to the network area. After a while, the process will complete and   
the crash kernel will reboot. If you get to see output similar to the two screenshots below,   
this means you have most likely successfully configured Kdump network functionality.

Figure 17: Console view of network-based crash dump

71



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 18: Console view of network-based crash dump - continued

This concludes the long and thorough configuration and testing of Kdump. If you have successfully managed all the stages so far, this means your system is ready to be placed into production and collect memory cores when kernel panic situations occur. Analyzing the cores will provide you with valuable information that should hopefully help you find and resolve the root causes leading to system crashes.

17 Conclusion

Kdump is a powerful, flexible Kernel crash dumping utility. The ability to execute a crash kernel in the context of the running production kernel is a very useful mechanism. Similarly, the ability to use the crash kernel in virtually all runlevels, including networking and the ability to send cores to network storage using a variety of protocols significantly extends our ability to control the environment.

Specifically, in comparison to the older LKCD utility, it offers improved functionality on all levels, including a more robust mechanism and better scalability. Kdump can use local RAID (md) devices if needed. Furthermore, it has improved network awareness and can work with a number of protocols, including NFS, CIFS, FTP, and SSH. The memory cores are no longer limited by the 32-bit barrier.

We will talk about the post-processing of the memory cores in the next part.

72

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Part III

Crash Collection

In this part, you will learn how to use the crash utility to open the dumped memory cores, collected at the time of kernel crashes, and read the information contained therein. Please note that this part focuses mainly on being able to use and process the crash dumps. We will focus on the crash analysis more deeply later on.

Like the Kdump setup, this part of the book is mainly intended for power users and system administrators, but if you wish to enrich your Linux knowledge, you’re more than welcome to use the material. Some of the steps will require in-depth familiarity with the functionality of the Linux operating system, which will not be reviewed here.

We will also briefly mention the older lcrash utility, which you may want to run against memory cores collected using LKCD. However, since the two are somewhat obsolete, we will not focus too much on their use. For more details about lcrash and gdb-kdump, please take a look at the Appendix (V).

Note: This part of the book focuses on both SUSE 10.X and 11.X and RedHat 5.X   
systems.

18 Crash setup

18.1 Prerequisites

You must have Kdump setup properly and working.

18.2 Kdump working crash installation

crash can be found in the repositories of all major distros. The installation is fairly simply   
and straightforward. You can use either yum, zypper or apt to obtain the package very   
easily.

73



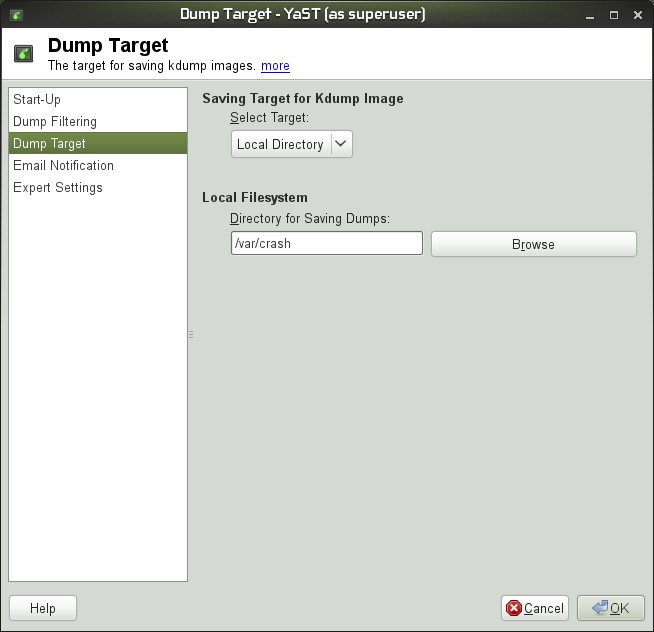
[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 19: Installation of crash via software manager

18.3 Crash location

The default crash directory is /var/crash. You can change the path to anything you   
want, provided there’s enough space on the target device. In general, you should choose

74



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

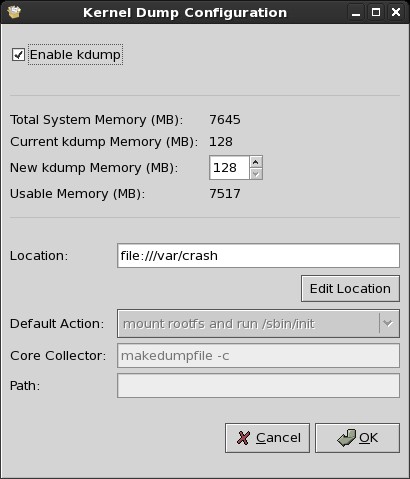
a disk or a partition that is equal or exceeds the size of your physical memory. You can   
change this path either using GUI tools or manually editing the Kdump configuration file:

• /etc/sysconfig/kdump on openSUSE.

• /etc/kdump.conf on CentOS (RedHat).

Figure 20: openSUSE Kdump configurationvia YaST-Kdump module

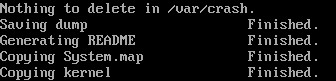
75



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 21: CentOS Kdump configuration system-config-kdump utility

76



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

18.4 Memory cores

Memory cores are called vmcore and you will find them in dated directories inside the crash directory. On older versions of Kdump, the directories would only contain the vmcore file. Newer versions also copy the kernel and System map file into the directory, making the core processing easier.

Figure 22: Generating crash dump files

Figure 23: Contents of a crash dump directory

19 Invoke crash

The crash utility can be invoked in several ways. First, there is some difference between older and newer versions of Kdump16, in terms of what they can do and how they process the memory cores. Second, the crash utility can be run manually or unattended. Let’s first review the differences between the older and newer versions.

16 The older version refers to SUSE 10.X systems. The newer versions refers to SUSE 11.x and RedHat

5.X systems.

77

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

19.1 Old (classic) invocation

The old invocation is done like this:

crash <System map> <vmlinux> vmcore

<System map> is the absolute path to the System map file, which is normally located under /boot. This file must match the version of the kernel used at the time of the crash. The System map file is a symbol table used by the kernel. A symbol table is a look-up between symbol names and their addresses in memory. A symbol name may be the name of a variable or the name of a function. The System.map is required when the address of a symbol name is needed. It is especially useful for debugging kernel panics and kernel oopses, which is what we need here.

For more details, you may want to read:

• [System.map on Wikipedia](http://en.wikipedia.org/wiki/System.map)

• [The Linux Kernel HOWTO - Systemmap](http://www.faqs.org/docs/Linux-HOWTO/Kernel-HOWTO.html#systemmap)

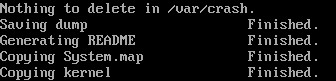
<vmlinux> is the uncompressed version of the kernel that was running when the memory core was collected. vmcore is the memory core.

The System map and vmlinux files remain in the /boot directory and are not copied into   
the crash directory. However, they can be manually copied to other machines, allowing   
portable use of crash against memory cores collected on other systems and/or kernels.

19.2 New invocation

The newer versions of Kdump can work with compressed kernel images. Furthermore,   
they copy the System map file and the kernel image into the crash directory, making the   
use of crash utility somewhat simpler. Finally, there are two ways you can process the   
cores.

78



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 24: New kdump invocation console output

You can use the old way. Here’s an example on CentOS 5.4:

crash \

/boot/System.map-2.6.18-164.10.1.el5 \   
/boot/vmlinuz-2.6.18-164.10.1.el5 \   
vmcore

Figure 25: Old crash invocation example on CentOS 5.4

Notice the use of vmlinuz kernel image, as opposed to vmlinux previously required. Alter-  
natively, you can use only the debug information under /usr/lib/debug. The information   
is extracted during the installation of kernel-debuginfo packages matching the kernel that   
was running at the time of the kernel crash. The syntax for CentOS and openSUSE is   
somewhat different.

openSUSE:

crash \

/usr/lib/debug/boot/<kernel>.debug \   
vmcore

79



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

CentOS (RedHat):

crash \

/usr/lib/debug/lib/modules/<kernel>/vmlinux \   
vmcore

Figure 26: New crash invocation example on CentOS 5.4

For more information, please consider reading the following articles:

• [Crashdump Debugging - openSUSE](http://en.opensuse.org/Crashdump_Debugging)

• [Kdump - openSUSE](http://en.opensuse.org/Kdump)

I must emphasize that the topic of how gdb and crash find the debuginfo of binaries can be a little confusing, so you may also want to spend a week or three and read the long documentation on gdb:

• [Debugging with GDB](http://sourceware.org/gdb/current/onlinedocs/gdb/index.html)

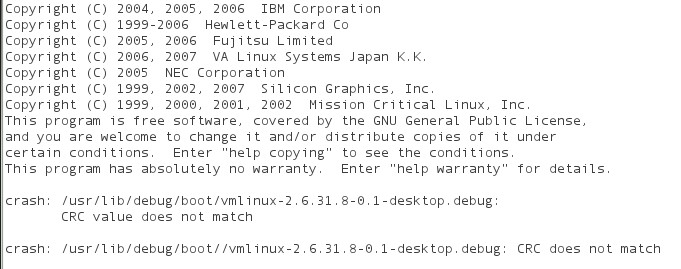
19.3 Important details to pay attention to

Now, since SUSE and RedHat use somewhat different syntax, things can be a little confusing. Therefore, please note the following table of comparison:

Table 7: Naming and file location differences between SUSE and RedHat

|  |  |  |
| --- | --- | --- |
|  | SUSE | RedHat |
| System map | System-map | System.map |
| Debug info | /usr/lib/debug/boot/ | /usr/lib/debug/lib/modules/ |

80



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 27: Crash debuginfo location on openSUSE 11.x

19.4 Portable use

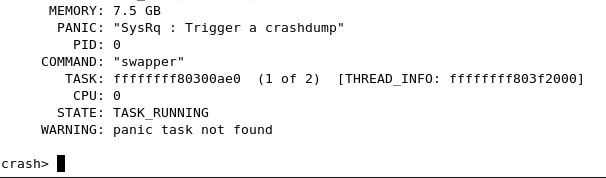
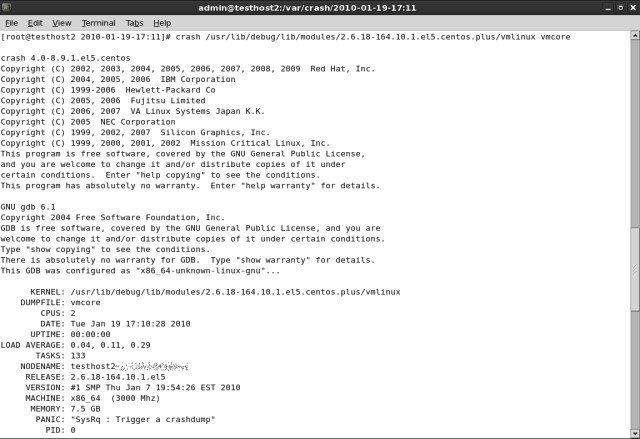
To process cores on other machines, you can either copy the System map and the ker-  
nel or just the debug information file. Newer versions of Kdump and crash will work with compressed kernel images. The debug info must match the kernel version exactly, otherwise you will get a CRC match error:

Figure 28: CRC match error

20 Running crash

All right, now that we know the little nuances, let’s run crash. Kdump is working and   
doing its magic in the background. We will not discuss Kdump-related issues here. Please   
refer to the previous book part (II) for more details. If you get the crash prompt after   
invoking the crash command, either using the old or new syntax, then everything is ok.

81

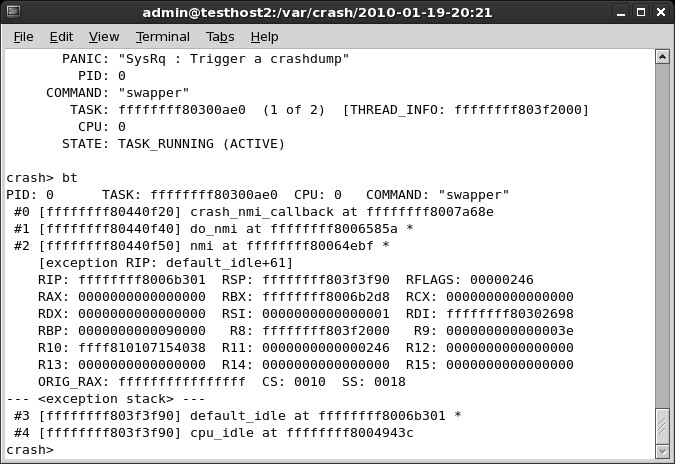


[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 29: Crash working

Figure 30: Crash prompt

82



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

20.1 Crash commands

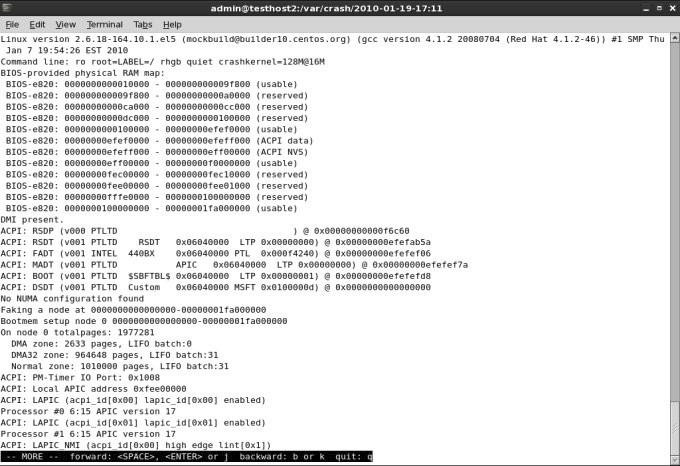
Once crash is running and you’re staring at the crash prompt, it’s time to try some crash   
commands. In this part, we will not focus too much on the commands or understanding   
their output. For now, it’s a brief overview of what we need. crash commands are listed   
in superb detail in the [White Paper](http://people.redhat.com/anderson/crash_whitepaper/). In fact, the document is pretty much everything you   
will need to work with crash. Here’s a handful of important and useful commands you   
will need:

20.1.1 bt - backtrace

Display a kernel stack backtrace. If no arguments are given, the stack trace of the current context will be displayed.

Figure 31: crash bt command example

83



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

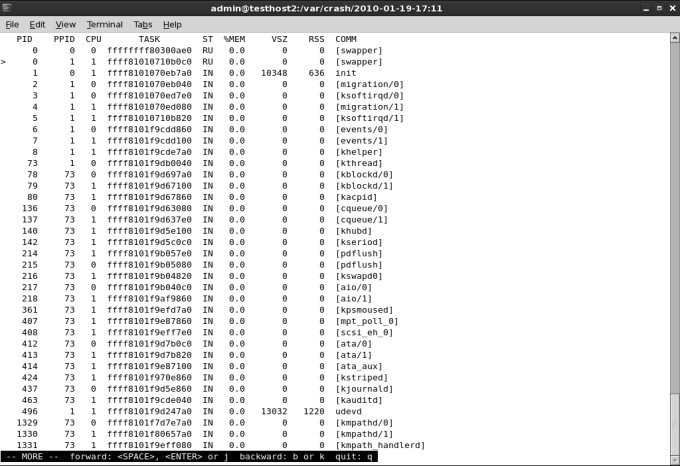
20.1.2 log - dump system message buffer

This command dumps the kernel log\_buf contents in chronological order. Figure 32: crash log command example

20.1.3 ps - display process status information

This command displays process status for selected, or all, processes in the system. If no arguments are entered, the process data is displayed for all processes.

84



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 33: crash ps command example

And there are many other commands. The true study begins here. We will review the usage of these commands, as well as many others in the next part. There, we will examine several simulated, study cases, as well as real crashes on production systems.

20.2 Other useful commands

You will also want to try help and h (command line history).

20.3 Create crash analysis file

Processed command output can be sent to an external file. You merely need to use the   
redirection symbol (>) and specify a filename. This contrasts the usage of the lcrash

85

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

utility, which specifically requires -w flag to write to files17.

20.4 Crash running in unattended mode

Now that we know how to run crash commands and produce analysis files, why not do that entirely unattended? This can be done by specifying command line input from a file. Commands can be sent to crash in two ways:

crash -i inputfile

Or using redirection:

crash < inputfile

In both cases, the crash inputfile is a text file with crash commands one per line. For   
the crash utility to exit, you will also need to include the exit command at the end.   
Something like:

bt

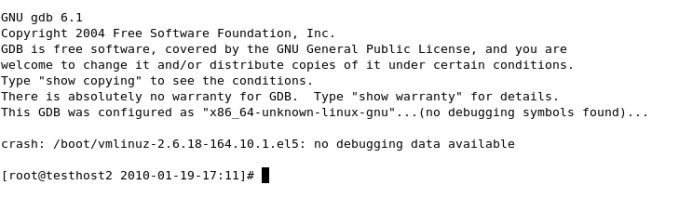
log   
ps

exit

Thus, the complete, unattended analysis takes the form of:

17 See Appendix (V) for more details.

86



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

crash <debuginfo> vmcore < inputfile > outputfile

Or perhaps:

crash <System map> <vmlinux> vmcore < inputfile > outputfile

So there we are! It’s all good. But, you may encounter problems ...

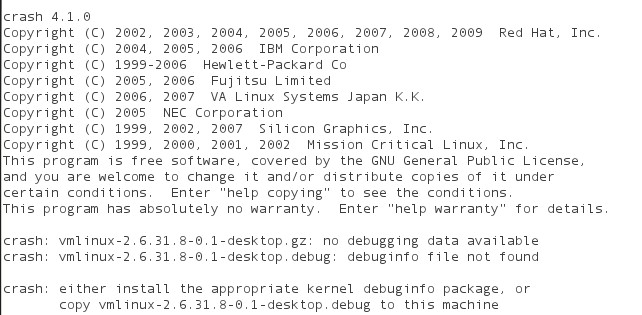
21 Possible errors

21.1 No debugging data available

After running crash, you may see this error:

Figure 34: No debuginfo package on RedHat

87



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

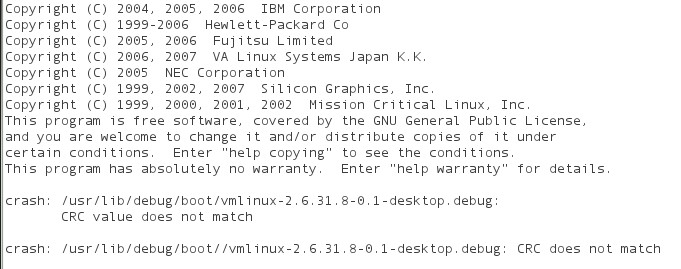
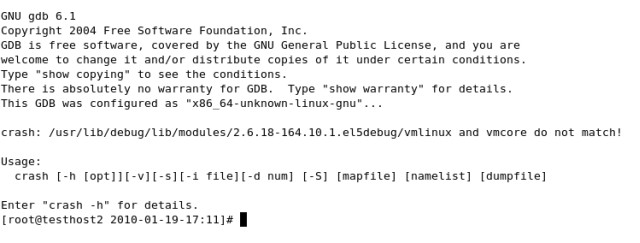
Figure 35: No debuginfo package on openSUSE

This means you’re probably missing the debuginfo packages. You should start your   
package manager and double-check. If you remember, I’ve repeatedly stated that having   
the debuginfo packages installed is a prerequisite for using Kdump and crash correctly18.

Figure 36: Installing crash debug packages on CentOS 5.4

18 The procedure how to enable debug repositories is explained in the Appendix (V).

88



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

21.2 vmlinux and vmcore do not match (CRC does not match)

You may also get this error:

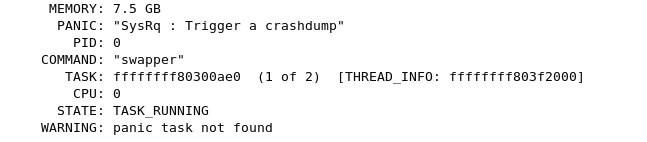
Figure 37: vmlinux and vmcore match problem on CentOS

On SUSE, it may look like this:

Figure 38: CRC match error on openSUSE

If you see the following messages: vmlinux and vmcore do not match! or CRC does not   
match, this means you have invoked crash against the wrong version of debuginfo, which   
does not match the vmcore file. Remember, you must use the exact same version!

89



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

21.3 No guarantee

There could be additional problems. Your dump may be invalid or incomplete. The   
header may be corrupt. The dump file may be in an unknown format. And even if   
the vmcore has been processed, the information therein may be partial or missing. For   
example, crash may not be able to find the task of the process causing the crash:

Figure 39: No panic task found

There’s no guarantee it will all work. System crashes are quite violent and things might   
not go as smoothly as you may desire, especially if the crashes are caused by hardware   
problems. For more details about possible errors, please consult the White Paper.

22 Conclusion

In this part, we have learned how to use the crash utility to open and process dumped memory cores. We focused on subtle differences in the setup on RedHat and SUSE, as well as different invocation methods and syntax used by these operating systems. Next, we learned about the crash functionality and the basic commands. Now we will perform the detailed analysis of collected cores.

90

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Part IV

Crash Analysis

We have learned how to configure our systems for kernel crash dumping, using LKCD and Kdump, both locally and across the network. We have learned how to setup the crash dumping mechanism on both CentOS and openSUSE, and we reviewed the subtle differences between the two operating systems. Next, we mastered the basic usage of the crash utility, using it to open the dumped memory core and process the information contained therein. But we did not yet learn to interpret the output.

In this part, we will focus on just that; read the vmcore analysis, understand what the entries mean and perform a basic investigation19. Then, we will slowly examine more complex problems. We will even write our own kernel module, make it faulty on purpose, and then use it to generate a crash. Afterwards, we will use the kernel crash report to find and solve the bug in our source code. Finally, we will derive an efficient methodology for handling kernel crash problems in the future

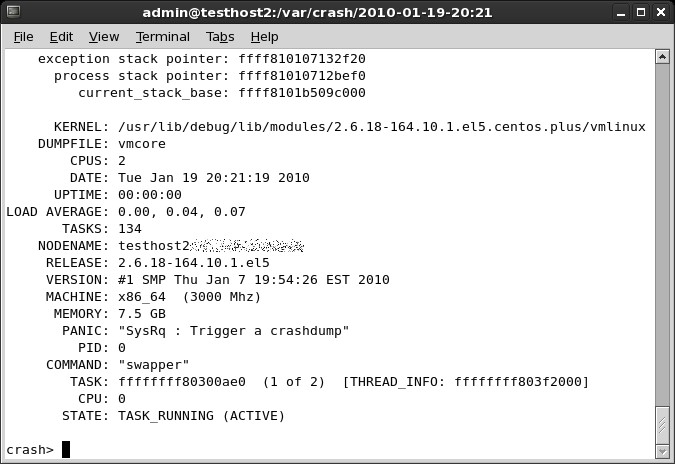
Note: Operating systems used to demonstrate the functionality are openSUSE 11.X and CentOS 5.X.

23 Analyzing the crash report - First steps

Once you launch crash, you will get the initial report information printed to the console. This is where the analysis of the crash begins.

19 You MUST read the other parts in other to fully understand how crash works. Without mastering the basic concepts, including Kdump and crash functionality, you will not be able to follow this part of the book efficiently.

91



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 40: Beginning crash analysis

92

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

crash 4.0-8.9.1.el5.centos

Copyright (C) 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009 Red Hat, Inc.

Copyright (C) 2004, 2005, 2006 IBM Corporation

Copyright (C) 1999-2006 Hewlett-Packard Co

Copyright (C) 2005, 2006 Fujitsu Limited

Copyright (C) 2006, 2007 VA Linux Systems Japan K.K. Copyright (C) 2005 NEC Corporation

Copyright (C) 1999, 2002, 2007 Silicon Graphics, Inc.

Copyright (C) 1999, 2000, 2001, 2002 Mission Critical Linux, Inc. This program is free software, covered by the GNU General Public License, and you are welcome to change it and/or distribute   
copies of it under certain conditions. Enter "help copying"   
to see the conditions. This program has absolutely no warranty. Enter "help warranty" for details.

NOTE: stdin: not a tty

GNU gdb 6.1

Copyright 2004 Free Software Foundation, Inc.

GDB is free software,covered by the GNU General Public

License, and you are welcome to change it and/or distribute

copies of it under certain conditions. Type "show copying" to see the conditions. There is absolutely no warranty for GDB. Type "show warranty" for details. This GDB was configured as "x86\_64-unknown-linux-gnu"...

bt: cannot transition from exception stack to current process stack:

exception stack pointer: ffff810107132f20   
 process stack pointer: ffff81010712bef0

current\_stack\_base: ffff8101b509c000

93

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

KERNEL: /usr/lib/debug/lib/modules/

2.6.18-164.10.1.el5.centos.plus/vmlinux DUMPFILE: vmcore

CPUS: 2

DATE: Tue Jan 19 20:21:19 2010 UPTIME: 00:00:00

LOAD AVERAGE: 0.00, 0.04, 0.07   
 TASKS: 134

NODENAME: testhost2@localdomain   
RELEASE: 2.6.18-164.10.1.el5

VERSION: #1 SMP Thu Jan 7 19:54:26 EST 2010 MACHINE: x86\_64 (3000 Mhz)

MEMORY: 7.5 GB

PANIC: "SysRq : Trigger a crashdump"   
 PID: 0

COMMAND: "swapper"

TASK: ffffffff80300ae0 (1 of 2)

[THREAD\_INFO: ffffffff803f2000]   
CPU: 0

STATE: TASK\_RUNNING (ACTIVE)

Let’s walk through the report. The first thing you see is some kind of an error:

bt: cannot transition from exception stack

to current process stack:

exception stack pointer: ffff810107132f20   
 process stack pointer: ffff81010712bef0

current\_stack\_base: ffff8101b509c000

The technical explanation for this error is a little tricky. Quoted from the crash util-  
ity mailing list [thread](http://www.mail-archive.com/crash-utility@redhat.com/msg01699.html) about changes in the crash utility 4.0-8.11 release, we learn the   
following information:

94

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

If a kdump NMI issued to a non-crashing x86\_64 cpu was received while running in   
schedule(), after having set the next task as "current" in the cpu’s runqueue, but prior to   
changing the kernel stack to that of the next task, then a backtrace would fail to make   
the transition from the NMI exception stack back to the process stack, with the error   
message "bt: cannot transition from exception stack to current process stack". This   
patch will report inconsistencies found between a task marked as the current task in a   
cpu’s runqueue, and the task found in the per-cpu x8664\_pda "pcurrent" field (2.6.29   
and earlier) or the per-cpu "current\_task" variable (2.6.30 and later). If it can be safely   
determined that the runqueue setting (used by default) is premature, then the crash   
utility’s internal per-cpu active task will be changed to be the task indicated by the   
appropriate architecture specific value.

What does this mean? It’s a warning that you should heed when analyzing the crash report. It will help us determine which task structure we need to look at to troubleshoot the crash reason. For now, ignore this error. It’s not important to understanding what the crash report contains. You may or may not see it.

Now, let’s examine the code below this error.

KERNEL: specifies the kernel running at the time of the crash. DUMPFILE: is the name of the dumped memory core.   
CPUS: is the number of CPUs on your machine.   
DATE: specifies the time of the crash.

TASKS: indicates the number of tasks in the memory at the time of the crash. Task is a set of program instructions loaded into memory.

NODENAME: is the name of the crashed host.

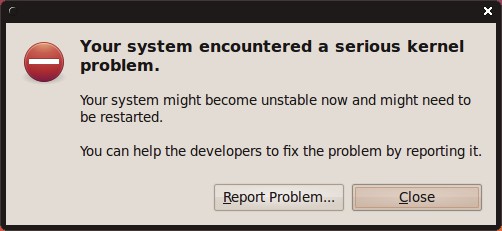
RELEASE: and VERSION: specify the kernel release and version. MACHINE: specifies the architecture of the CPU.

MEMORY: is the size of the physical memory on the crashed machine. And now come the interesting bits:

PANIC: specifies what kind of crash occurred on the machine. There are several types that you can see.

SysRq (System Request) refers to Magic Keys, which allow you to send instructions directly to the kernel. They can be invoked using a keyboard sequence or by echoing letter commands to /proc/sysrq-trigger, provided the functionality is enabled. We have discussed this in the Kdump part.

95



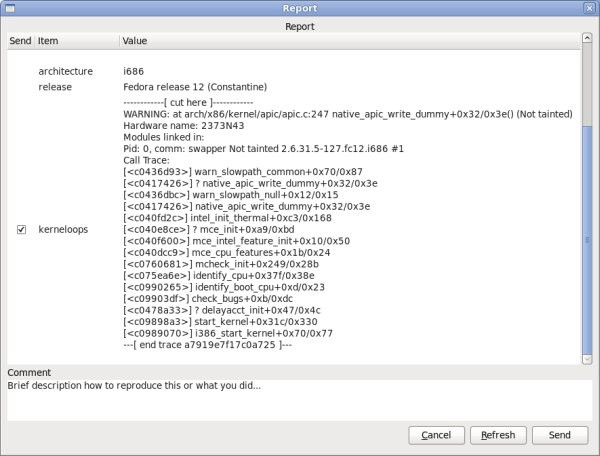
[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Oops is a deviation from the expected, correct behavior of the kernel. Usually, the oops   
results in the offending process being killed. The system may or may not resume its   
normal behavior. Most likely, the system will enter an unpredictable, unstable state,   
which could lead to kernel panic if some of the buggy, killed resources are requested later   
on.

For example, in my Ubuntu [Karmic](http://www.dedoimedo.com/computers/ubuntu-9-10.html) and Fedora [Constantine](http://www.dedoimedo.com/computers/fedora-12.html) reviews, we’ve seen evidence   
of kernel crashes. However, the system continued working. These crashes were in fact   
oopses.

Figure 41: Serious kernel problem example in Ubuntu

96



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 42: Kernel crash report in Fedora

We will discuss the Fedora case later on.

Panic is a state where the system has encountered a fatal error and cannot recover. Panic can be caused by trying to access non-permitted addresses, forced loading or unloading of kernel modules, or hardware problems.

In our first, most benign example, the PANIC: string refers to the use of Magic Keys. We deliberately triggered a crash.

PANIC: "SysRq : Trigger a crashdump"

97

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

PID: is the process ID of the ... process that caused the crash.   
COMMAND: is the name of the process, in this case swapper.

COMMAND: “swapper”

swapper, or PID 0 is the scheduler. It’s the process that delegates the CPU time between runnable processes and if there are no other processes in the runqueue, it takes control. You may want to refer to swapper as the idle task, so to speak.

There’s one swapper per CPU, which you will soon see when we start exploring the crash in greater depth. But this is not really important. We will encounter many processes with different names.

TASK: is the address in memory for the offending process. We will use this information   
later on. There’s a difference in the memory addressing for 32-bit and 64-bit architectures.

CPU: is the number of the CPU (relevant if more than one) where the offending process was running at the time of the crash. CPU refers to CPU cores and not just physical CPUs. If you’re running your Linux with hyperthreading enabled, then you will also be counting separate threads as CPUs. This is important to remember, because recurring crashes on just one specific CPU might indicate a CPU problem.

If you’re running your processes with affinity set to certain CPUs (taskset), then you   
might have more difficulty pinpointing CPU-related problems when analyzing the crash   
reports.

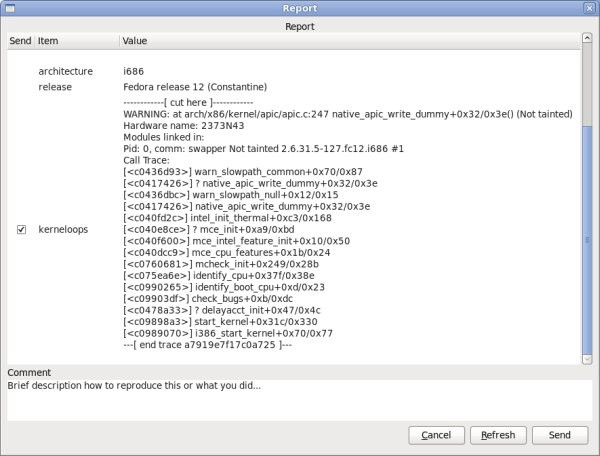
You can examine the number of your CPUs by running cat /proc/cpuinfo.

STATE: indicates the process state at the time of the crash. TASK\_RUNNING refers to runnable processes, i.e. processes that can continue their execution. Again, we will talk more about this later on.

24 Getting warmer

We’ve seen one benign example so far. Just an introduction. We will take a look at several more examples, including real cases. For now, we know little about the crash, except that the process that caused it. We will now examine several more examples and try to understand what we see there.

98



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

24.1 Fedora example

Let’s go back to Fedora case. Take a look at the screenshot below. While the information   
is arranged somewhat differently than what we’ve seen earlier, essentially, it’s the same   
thing.

Figure 43: Kernel crash report in Fedora, shown again

But there’s a new piece of information:

Pid: 0, comm: swapper Not tainted.

99

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Let’s focus on the Not tainted string for a moment. What does it mean? This means that the kernel is not running any module that has been forcefully loaded. In other words, we are probably facing a code bug somewhere rather than a violation of the kernel. You can examine your running kernel by executing:

cat /proc/sys/kernel/tainted

So far, we’ve learned another bit of information. We will talk about this later on.

24.2 Another example, from the White Paper

Take a look at this example:

MEMORY: 128MB

PANIC: "Oops: 0002" (check log for details)

PID: 1696

COMMAND: "insmod"

What do we have here? A new piece of information. Oops: 0002. What does this   
mean? This is the kernel page error code. We will now elaborate what it is and how it   
works.

24.3 Kernel Page Error

The four digits are a decimal code of the Kernel Page Error. Reading O’Reilly’s Under-  
standing Linux Kernel, Chapter 9: Process Address Space, Page Fault Exception Handler, pages 376-382, we learn the following information:

100

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

• If the first bit is clear (0), the exception was caused by an access to a page that is not present; if the bit is set (1), this means invalid access right.

• If the second bit is clear (0), the exception was caused by read or execute access; if set (1), the exception was caused by a write access.

• If the third bit is clear (0), the exception was caused while the processor was in Kernel mode; otherwise, it occurred in User mode.

• The fourth bit tells us whether the fault was an Instruction Fetch. This is only valid for 64-bit architecture. Since our machine is 64-bit, the bit has meaning here.

Table 8: Kernel page error code

|  |  |  |
| --- | --- | --- |
|  | Value | |
| Bit | 0 | 1 |
| 0 | No page found | Invalid access20 |
| 1 | Read or Execute | Write |
| 2 | Kernel mode | User mode |
| 3 | Not instruction fetch | Instruction fetch |

Therefore, to understand what happened, we need to translate the decimal code into   
binary and then examine the four bits, from right to left. In our case, decimal 2 is binary

10. Looking from right to left, bit 1 is zero, bit 2 is lit, bit 3 and 4 are zero. Notice the binary count, starting from zero. In other words:

0002 (dec) → 0010 (binary) → Not instruction fetch | Kernel mode | Write | Invalid access

20 Sometimes, invalid access is also referred to as protection fault.

101

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

This is quite interesting. Seemingly incomprehensible information starts to feel very   
logical indeed. Therefore, we have a page not found during a write operation in Kernel   
mode; the fault was not an Instruction Fetch. Of course, it’s a little more complicated   
than that, but still we’re getting a very good idea of what’s going on. Well, it’s starting   
to get interesting, isn’t it? Looking at the offending process, insmod, this tells us quite   
a bit. We tried to load a kernel module. It tried to write to a page it could not find,   
meaning protection fault, which caused our system to crash. This might be a badly   
written piece of code.

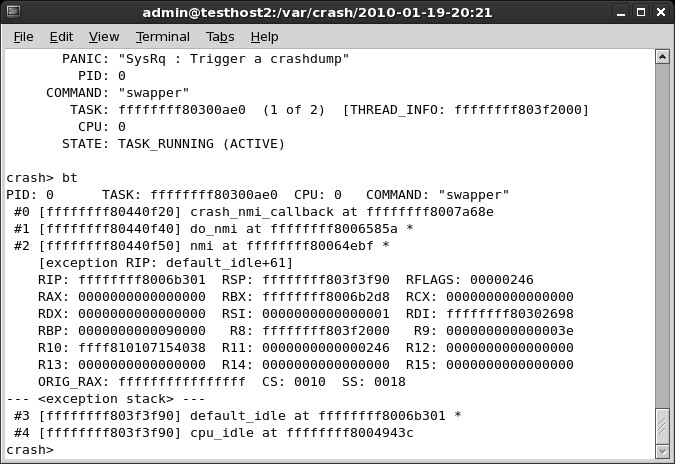
OK, so far, we’ve seen quite a bit of useful information. We learned about the basic identifier fields in the crash report. We learned about the different types of Panics. We learned about identifying the offending process, deciding whether the kernel is tainted and what kind of problem occurred at the time of the crash. But we have just started our analysis. Let’s take this to a new level.

25 Getting hot

25.1 Backtrace

In the previous part, we learned about some basic commands. It’s time to put them to good use. The first command we want is bt - backtrace. We want to see the execution history of the offending process, i.e. backtrace.

102



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 44: Backtrace of a crash dump

103

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

PID: 0 TASK: ffffffff80300ae0 CPU: 0 COMMAND: "swapper"

#0 [ffffffff80440f20] crash\_nmi\_callback at ffffffff8007a68e #1 [ffffffff80440f40] do\_nmi at ffffffff8006585a \*   
#2 [ffffffff80440f50] nmi at ffffffff80064ebf \*   
[exception RIP: default\_idle+61]

RIP: ffffffff8006b301 RSP: ffffffff803f3f90 RFLAGS: 00000246 RAX: 0000000000000000 RBX: ffffffff8006b2d8

RCX: 0000000000000000

RDX: 0000000000000000 RSI: 0000000000000001 RDI: ffffffff80302698

RBP: 0000000000090000 R8: ffffffff803f2000

R9: 000000000000003e

R10: ffff810107154038 R11: 0000000000000246

R12: 0000000000000000

R13: 0000000000000000 R14: 0000000000000000

R15: 0000000000000000

ORIG\_RAX: ffffffffffffffff CS: 0010 SS: 0018

--- <exception stack> ---

#3 [ffffffff803f3f90] default\_idle at ffffffff8006b301 \* #4 [ffffffff803f3f90] cpu\_idle at ffffffff8004943c

25.1.1 Call trace

The sequence of numbered lines, starting with the hash sign (#) is the call trace. It’s a list of kernel functions executed just prior to the crash. This gives us a good indication of what happened before the system went down.

104

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

#0 [ffffffff80440f20] crash\_nmi\_callback at ffffffff8007a68e #1 [ffffffff80440f40] do\_nmi at ffffffff8006585a \*   
#2 [ffffffff80440f50] nmi at ffffffff80064ebf \*   
[exception RIP: default\_idle+61]

RIP: ffffffff8006b301 RSP: ffffffff803f3f90 RFLAGS: 00000246 RAX: 0000000000000000 RBX: ffffffff8006b2d8

RCX: 0000000000000000

RDX: 0000000000000000 RSI: 0000000000000001 RDI: ffffffff80302698

RBP: 0000000000090000 R8: ffffffff803f2000

R9: 000000000000003e

R10: ffff810107154038 R11: 0000000000000246

R12: 0000000000000000

R13: 0000000000000000 R14: 0000000000000000

R15: 0000000000000000

ORIG\_RAX: ffffffffffffffff CS: 0010 SS: 0018

--- <exception stack> ---

#3 [ffffffff803f3f90] default\_idle at ffffffff8006b301 \* #4 [ffffffff803f3f90] cpu\_idle at ffffffff8004943c

25.1.2 Instruction pointer

The first really interesting line is this one:

[exception RIP: default\_idle+61]

We have exception RIP: default\_idle+61. What does this mean? First, let’s discuss RIP. RIP is the instruction pointer21. It points to a memory address, indicating the progress of program execution in memory. In our case, you can see the exact address in the line just below the bracketed exception line:

21 On 32-bit architecture, the instruction pointer is called EIP.

105

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

[exception RIP: default\_idle+61]

RIP: ffffffff8006b301 RSP: ffffffff803f3f90 RFLAGS: 00000246

For now, the address itself is not important. The second part of information is far more   
useful to us. default\_idle is the name of the kernel function in which the RIP lies. +61   
is the offset, in decimal format, inside the said function where the exception occurred.

25.1.3 Code Segment (CS) register

The code between the bracketed string down to — <exception stack> — is the   
dumping of registers. Most are not useful to us, except the CS (Code Segment) register.

CS: 0010

Again, we encounter a four-digit combination. In order to explain this concept, I need to deviate a little and talk about Privilege levels.

25.1.4 Privilege levels

Privilege level is the concept of protecting resources on a CPU. Different execution threads   
can have different privilege levels, which grant access to system resources, like memory   
regions, I/O ports, etc. There are four levels, ranging from 0 to 3. Level 0 is the most   
privileged, known as Kernel mode. Level 3 is the least privileged, known as User mode.

Most modern operating systems, including Linux, ignore the intermediate two levels, using only 0 and 3. The levels are also known as Rings. A notable exception of the use of levels was IBM OS/2 system.

25.1.5 Current Privilege Level (CPL)

Code Segment (CS) register is the one that points to a segment where program instruc-  
tions are set. The two least significant bits of this register specify the Current Privilege Level (CPL) of the CPU. Two bits, meaning numbers between 0 and 3.

106

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

25.1.6 Descriptor Privilege Level (DPL) &

Requested Privilege Level (RPL)

Descriptor Privilege Level (DPL) is the highest level of privilege that can access the resource and is defined. This value is defined in the Segment Descriptor. Requested Privilege Level (RPL) is defined in the Segment Selector, the last two bits. Mathemat-  
ically, CPL is not allowed to exceed MAX(RPL,DPL), and if it does, this will cause a general protection fault. Now, why is all this important, you ask?

Well, for instance, if you encounter a case where system crashed while the CPL was 3, then this could indicate faulty hardware, because the system should not crash because of a problem in the User mode. Alternatively, there might be a problem with a buggy system call. Just some rough examples.

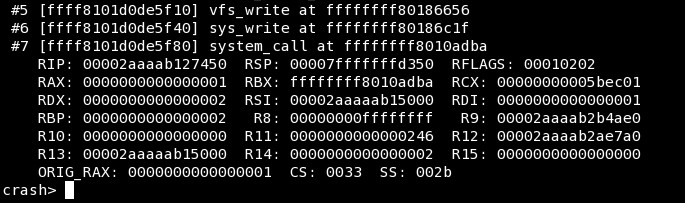
For more information, please consider referring to O’Reilly’s Understanding Linux Kernel,   
Chapter 2: Memory Addressing, Page 36-39. You will find useful information about Seg-  
ment Selectors, Segment Descriptors, Table Index, Global and Local Descriptor Tables,   
and of course, the Current Privilege Level (CPL). Now, back to our crash log:

CS: 0010

As we know, the two least significant bits specify the CPL. Two bits means four levels, however, levels 1 and 2 are ignored. This leaves us with 0 and 3, the Kernel mode and User mode, respectively. Translated into binary format, we have 00 and 11.

The format used to present the descriptor data can be confusing, but it’s very simple. If the right-most figure is even, then we’re in the Kernel mode; if the last figure is odd, then we’re in the User mode. Hence, we see that CPL is 0, the offending task leading to the crash was running in the Kernel mode. This is important to know. It may help us understand the nature of our problem. Just for reference, here’s an example where the crash occurred in User mode, collected on a SUSE machine:

107



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 45: Example of a kernel crash with CPL 3

But that’s just geeky talk. Back to our example, we have learned many useful, important details. We know the exact memory address where the instruction pointer was at the time of the crash. We know the privilege level.

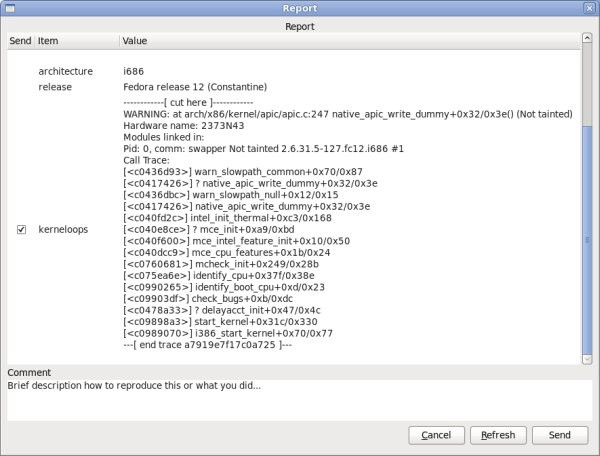
More importantly, we know the name of the kernel function and the offset where the RIP was pointing at the time of the crash. For all practical purposes, we just need to find the source file and examine the code. Of course, this may not be always possible, for various reasons, but we will do that, nevertheless, as an exercise.

So, we know that crash\_nmi\_callback() function was called by do\_nmi(), do\_nmi() was called by nmi(), nmi() was called by default\_idle(), which caused the crash. We can examine these functions and try to understand more deeply what they do. We will do that soon. Now, let’s revisit our Fedora example one more time.

25.1.7 Fedora example, again

Now that we understand what’s wrong, we can take a look at the Fedora example again   
and try to understand the problem. We have a crash in a non-tainted kernel, caused by   
the swapper process. The crash report points to native\_apic\_write\_dummy function.

108



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 46: Fedora kernel crash example

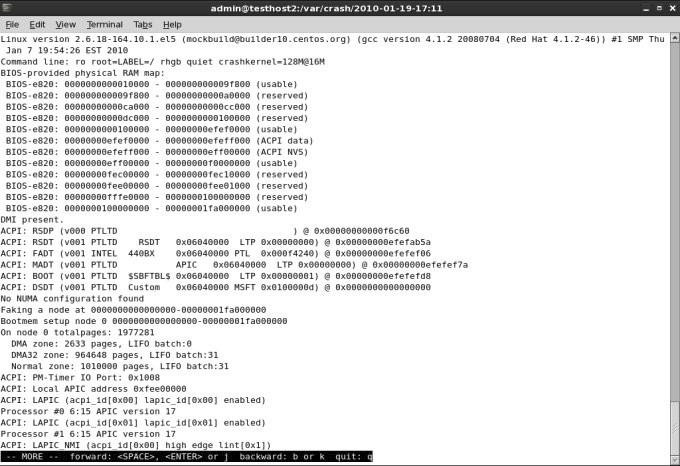
Then, there’s also a very long call trace. Quite a bit of useful information that should help us solve the problem. We will see how we can use the crash reports to help developers fix bugs and produce better, more stable software. Now, let’s focus some more on crash and the basic commands.

25.1.8 backtrace for all tasks

By default, crash will display backtrace for the active task. But you may also want to   
see the backtrace of all tasks. In this case, you will want to run [foreach](http://people.redhat.com/anderson/crash_whitepaper/help_pages/foreach.html).

foreach bt

109



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

25.2 Dump system message buffer

log - dump system message buffer: This command dumps the kernel log\_buf con-  
tents in chronological order.

Figure 47: Kernel crash log command output example

The kernel log bugger (log\_buf) might contains useful clues preceding the crash, which   
might help us pinpoint the problem more easily and understand why our system went   
down. The log command may not be really useful if you have intermittent hardware   
problems or purely software bugs, but it is definitely worth the try. Here’s our crash log,   
the last few lines:

110

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

ide: failed opcode was: 0xec

mtrr: type mismatch for f8000000,400000 old: uncachable new: write-combining

ISO 9660 Extensions: Microsoft Joliet Level 3 ISO 9660 Extensions: RRIP\_1991A

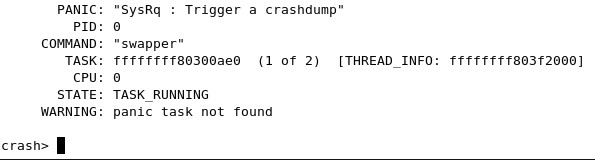
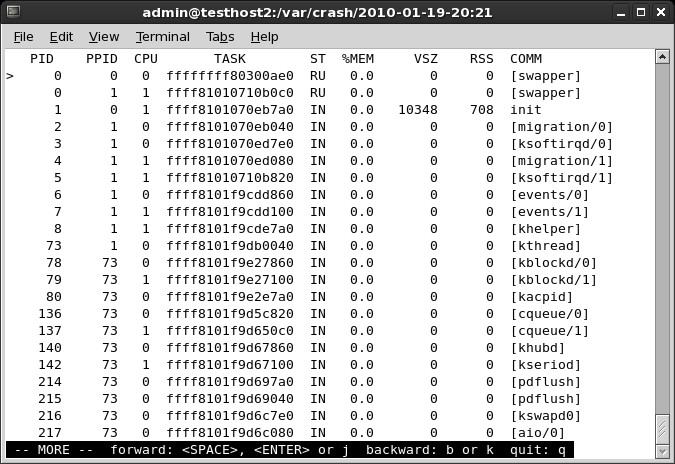
SysRq : Trigger a crashdump

And there’s the SysRq message. Useful to know. In real cases, there might be something far more interesting.

25.3 Display process status information

ps - display process status information This command displays process status for   
selected, or all, processes in the system. If no arguments are entered, the process data   
is displayed for all processes. Take a look at the example below. We have two swapper   
processes! As I told you earlier, each CPU has its own scheduler. The active task is   
marked with >.

111



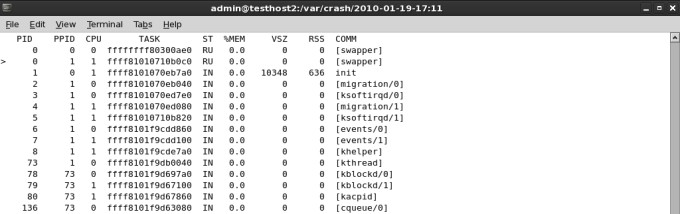
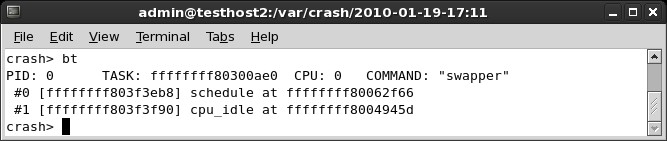
[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 48: Kernel crash ps command output example

The crash utility may load pointing to a task that did not cause the panic or may not be able to find the panic task. There are no guarantees. If you’re using virtual machines, including VMware or Xen, then things might get even more complicated.

Figure 49: No panic task found on CentOS 5.4

112



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 50: bt command for wrong process

In this case, the pointer in the ps output marks the "wrong" process: Figure 51: ps command output pointing at wrong process

Using backtrace for all processes (with foreach) and running the ps command, you should be able to locate the offending process and examine its task.

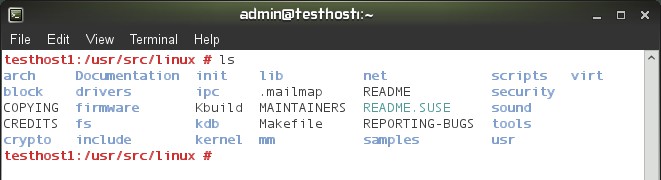
25.4 Other useful information

A few more items you may need: bracketed items are kernel threads; for example, init and udevd are not. Then, there’s memory usage information, VSZ and RSS, process state, and more.

26 Super geeky stuff

Note: This section is impossibly hard. Too hard for most people. Very few people are   
skilled enough to dabble in kernel code and really know what’s going on in there. Trying

113



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

to be brave and tackle the possible bugs hidden in crash cores is a noble attempt, but   
you should not take this lightly. I have to admit that although I can peruse crash reports   
and accompanying sources, I still have a huge deal to learn about the little things and   
bits. Don’t expect any miracles. There’s no silver-bullet solution to crash analysis!

Now, time to get ultra-serious. Let’s say you may even want to analyze the C code for   
the offending function. Needless to say, you should have the C sources available and   
be able to read them. This is not something everyone should do, but it’s an interesting   
mental exercise. Source code. All right, you want examine the code. First, you will have   
to obtain the sources.

26.1 Kernel source

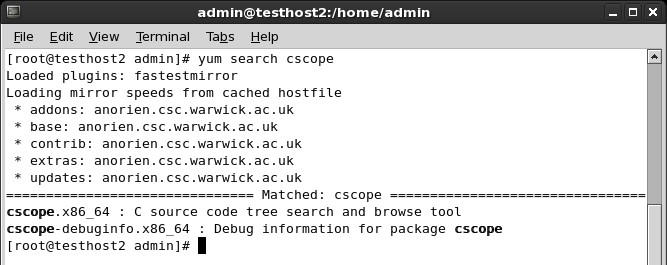
Some distributions make the sources readily available. For example, in openSUSE, you   
just have to download the kernel-source package. With CentOS, it is a little more difficult,   
but doable. You can also visit the [Linux Kernel Archive](http://kernel.org/) and download the kernel matching   
your own, although some sources may be different from the ones used on your system, since some vendors make their own custom changes.

Once you have the sources, it’s time to examine them. Figure 52: Kernel source example on openSUSE

26.2 cscope

You could browse the sources using the standard tools like find and grep, but this can   
be rather tedious. Instead, why not let the system do all the hard work for you. A very   
neat utility for browsing C code is called [cscope](http://cscope.sourceforge.net/). The tool runs from the command line

114



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

and uses a vi-like interface. By default, it will search for sources in the current directory, but you can configure it any which way. cscope is available in the repositories:

Figure 53: cscope installation via yum on CentOS

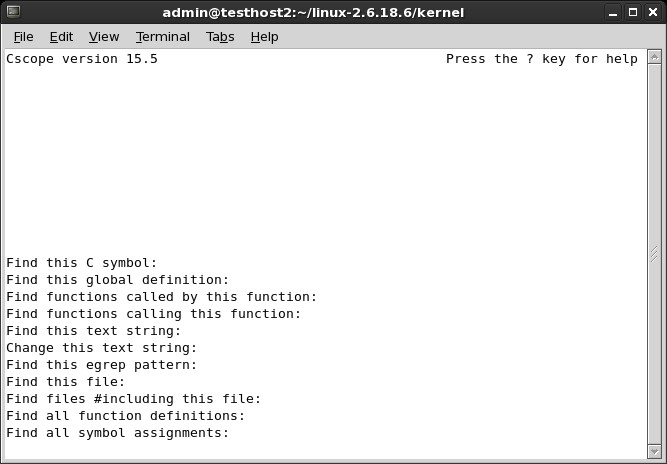
Now, in the directory containing sources22, run cscope:

cscope -R

This will recursively search all sub-directories, index the sources and display the main interface. There are other uses as well; try the man page or -help flag.

22 By default, the sources are located under /usr/src/linux.

115

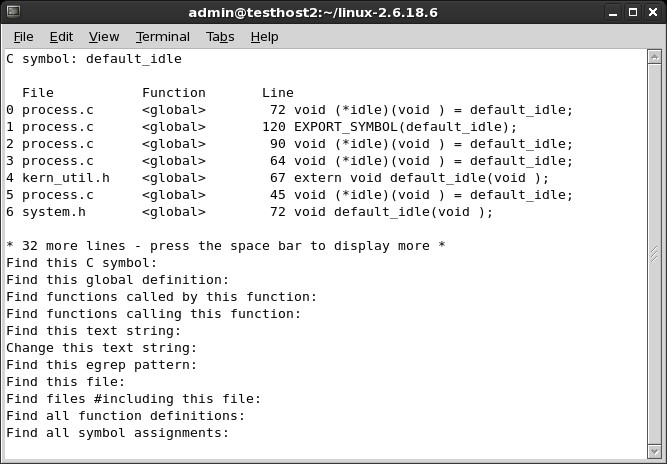


[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 54: cscope loaded on CentOS 5.4

Now, it’s time to put the tool to good use and search for desired functions. We will begin with Find this C symbol. Use the cursor keys to get down to this line, then type the desired function name and press Enter. The results will be displayed:

116



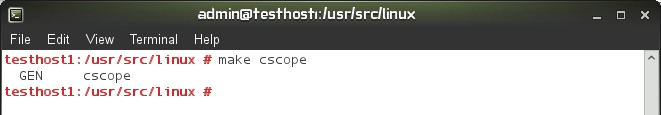
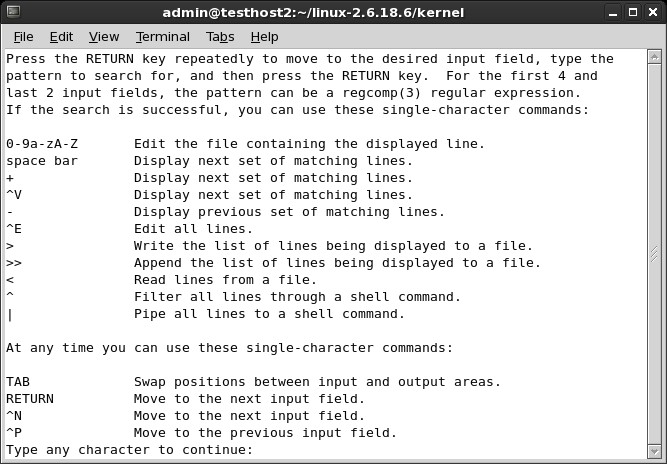
[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 55: Find C symbol using cscope

Depending on what happened, you may get many results or none. It is quite possible that there is no source code containing the function seen in the crash report. If there are too many results, then you might want to search for the next function in the call trace by using the Find functions called by this function option. Use Tab to jump between the input and output section. If you have official vendor support, this is a good moment to turn the command over and let them drive.

If you stick with the investigation, looking for other functions listed in the call trace can help you narrow down the C file you require. But there’s no guarantee and this can be a long, tedious process. Furthermore, any time you need help, just press ? and you will get a basic usage guide:

117



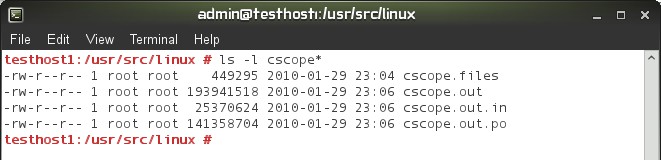
[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 56: cscope help menu

In the kernel source directory, you can also create the cscope indexes, for faster searches in the future, by running make cscope.

Figure 57: make cscope command example

118



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 58: cscope files

26.3 Disassemble the object

Assuming you have found the source, it’s time to disassemble the object compiled from this source. First, if you’re running a debug kernel, then all the objects have been compiled with the debug symbols. You’re lucky. You just need to dump the object and burrow into the intermixed assembly-C code. If not, you will have to recompile the source with debug symbols and then reverse-engineer it.

This is not a simple or a trivial task. First, if you use a compiler that is different than the one used to compile the original, your object will be different from the one in the crash report, rendering your efforts difficult if not impossible.

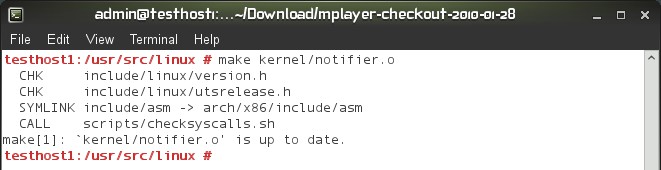
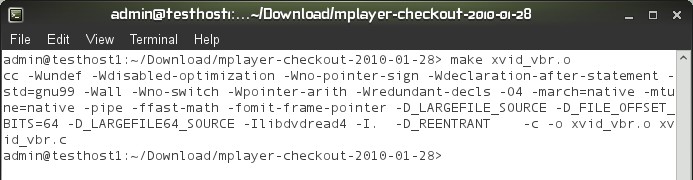
26.4 Trivial example

I call this example trivial because it has nothing to do with the kernel. It merely demon-  
strates how to compile objects and then disassemble them. Any source will do. In our   
case, we’ll use [MPlayer](http://www.mplayerhq.hu/design7/dload.html), a popular open-source media player as our scapegoat. Download   
the MPlayer source code, run ./configure, make. After the objects are created, delete one of them, then recompile it.

Run make <object name>, for instance:

make xvid\_bvr.o

119



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 59: Compiling from sources with make

Please note that make has no meaning without a Makefile, which specifies what needs to be done. But we have a Makefile. It was created after we ran ./configure. Otherwise, all this would not really work. Makefile is very important. We will see a less trivial example soon. Now, if you do not remove the existing object, then you probably won’t be able to make it. make compares timestamps on sources and the object, so unless you change the sources, the recompile of the object will fail.

Figure 60: Kernel object is up to date

Now, here’s another simple example, and note the difference in the size of the created object, once with the debug symbols and once without:

120



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 61: Object compiled with debug symbols

If you don’t have a Makefile, you can invoke gcc manually using all sorts of flags. You   
will need kernel headers that match the architecture and the kernel version that was used   
to create the kernel where the crash occurred, otherwise your freshly compiled objects   
will be completely different from the ones you may wish to analyze, including functions   
and offsets.

26.5 objdump

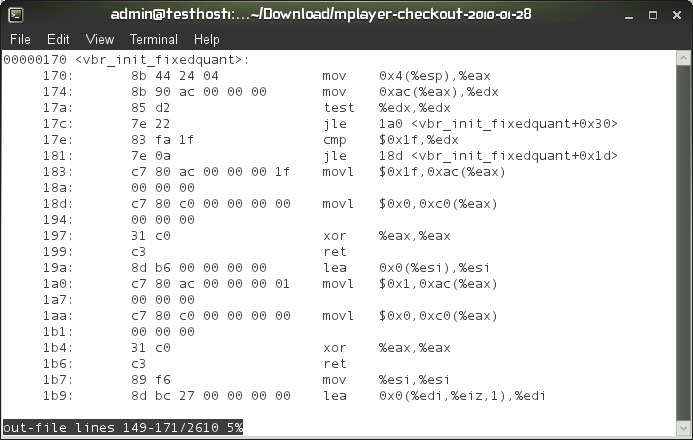
A utility you want to use for disassembly is [objdump](http://linux.die.net/man/1/objdump). You will probably want to use the   
utility with -S flag, which means display source code intermixed with assembly instruc-  
tions. You may also want -s flag, which will display contents of all sections, including empty ones. -S implies -d, which displays the assembler mnemonics for the machine instructions from objfile; this option only disassembles those sections which are expected to contain instructions. Alternatively, use -D for all sections.

Thus, the most inclusive objdump would be:

objdump -D -S <compiled object with debug symbols> > <output file>

It will look something like this:

121

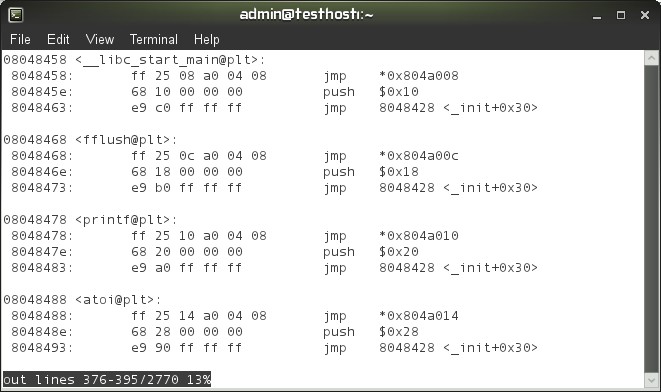


[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 62: Disassembled object example

And an even better example, the memhog dump:

122



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 63: Memhog binary dumped with objdump

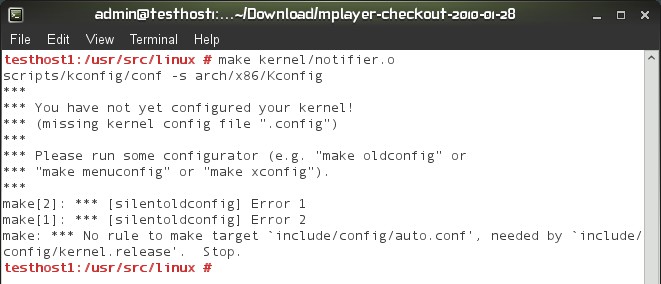
26.6 Moving on to kernel sources

Warming up. Once you’re confident practicing with trivial code, time to move to kernel.   
Make sure you do not just delete any important file. For the sake of exercise, move or   
rename any existing kernel objects you may find lurking about. Then, recompile them.   
You will require the .config file used to compile the kernel. It should be included with   
the sources. Alternatively, you can dump it. On openSUSE, under /proc/config.gz.

zcat /proc/config.gz > .config

On RedHat machines, you will find the configuration files also under /boot. Make sure you use the one that matches the crashed kernel and copy it over into the source directory. If needed, edit some of the options, like CONFIG\_DEBUG\_INFO. Without the .config file, you won’t be able to compile kernel sources:

123



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 64: Failed kernel object compilation due to missing kernel config file

You may also encounter an error where the Makefile is supposedly missing, but it’s there. In this case, you may be facing a relatively simply problem, with the wrong $ARCH environment variable set. For example, i585 versus i686 and x86-64 versus x86\_64. Pay attention to the error and compare the architecture to the $ARCH variable. In the worst case, you may need to export it correctly. For example:

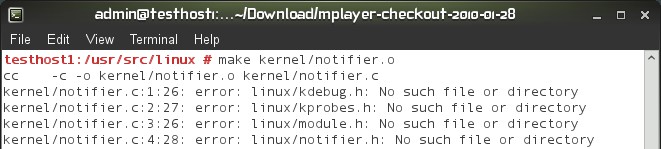
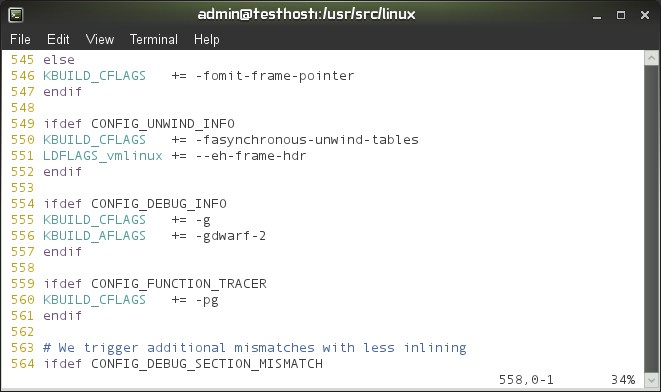
export ARCH=x86\_64

As a long term solution, you could also create symbolic links under /usr/src/linux from   
the would-be bad architecture to the right one. This is not strictly related to the analysis   
of kernel crashes, but if and when you compile kernel sources, you may encounter this   
issue. Now, regarding the CONFIG\_DEBUG\_INFO variable; it should be set to 1 in your   
.config file. If you recall the Kdump part, this was a prerequisite we asked for, in order   
to be able to successfully troubleshoot kernel crashes. This tells the compiler to create   
objects with debug symbols.

Alternatively, export the variable in the shell, as CONFIG\_DEBUG\_INFO=1.

CONFIG\_DEBUG\_INFO=1

124



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Then, take a look at the Makefile. You should see that if this variable is set, the object will be compiled with debug symbols (-g). This is what we need. After that, once again, we will use objdump.

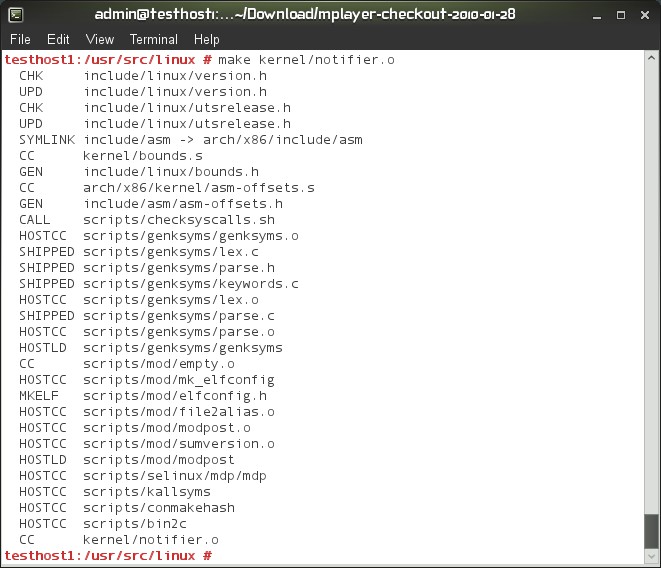
Figure 65: Editing Makefile

Now, Makefile might really be missing. In this case, you will get a whole bunch of errors related to the compilation process.

Figure 66: Makefile is missing

But with the Makefile in place, it should all work smoothly.

125

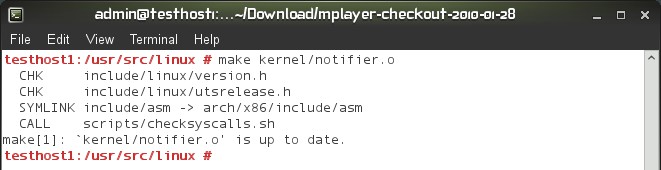


[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 67: Successfully compiling kernel object

And then, there’s the object up to date example again. If you do not remove an existing one, you won’t be able to compile a new one, especially if you need debug symbols for later disassembly.

126

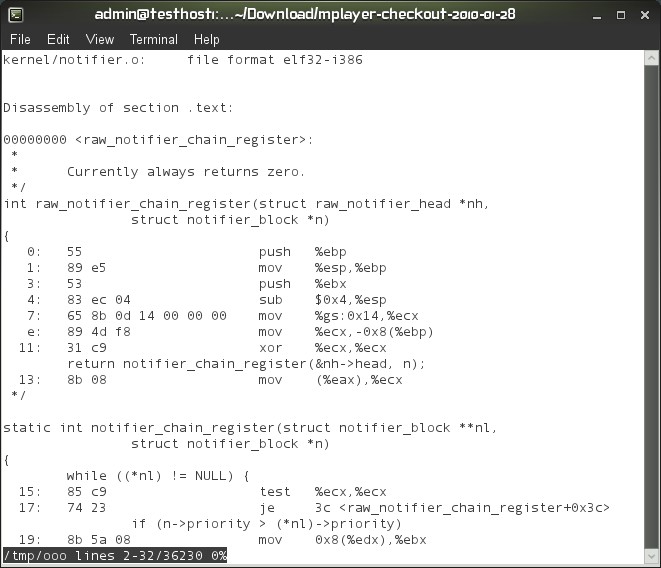


[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 68: Kernel object is up to date

Finally, the disassembled object:

127



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 69: Disassembled kernel object

26.6.1 What do we do now?

Well, you look for the function listed in the exception RIP and mark the starting address. Then add the offset to this number, translated to hexadecimal format. Then, go to the line specified. All that is left is to try to understand what really happened. You’ll have an assembly instruction listed and possibly some C code, telling us what might have gone wrong. It’s not easy. In fact, it’s very difficult. But it’s exciting and you may yet succeed, finding bugs in the operating system. What’s more fun than that?

Above, we learned about the compilation and disassembly procedures, without really

128

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

doing anything specific. Now that we know how to go about compiling kernel objects and dissecting them into little bits, let’s do some real work.

26.7 Intermediate example

We will now try something more serious. Grab a proof-of-concept code that crashes the   
kernel, compile it, examine the crash report, then look for the right sources, do the whole   
process we mentioned above, and try to read the alien intermixed assembly and C code.

Of course, we will be cheating, cause we will know what we’re looking for, but still, it’s   
a good exercise. The most basic non-trivial example is to create a kernel module that   
causes panic. Before we panic our kernel, let’s do a brief overview of the kernel module   
programming basics.

26.7.1 Create problematic kernel module

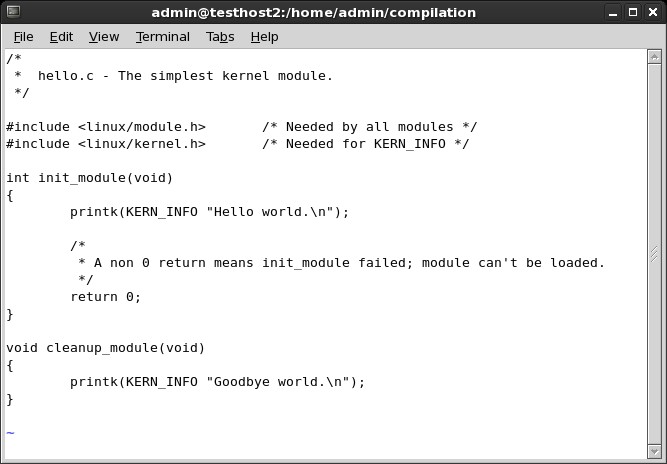
This exercise forces us to deviate from the crash analysis flow and take a brief look at the C programming language from the kernel perspective. We want to crash our kernel, so we need kernel code. While we’re going to use C, it’s a little different from everyday stuff. Kernel has its own rules.

We will have a sampling of kernel module programing. We’ll write our own module and   
Makefile, compile the module and then insert it into the kernel. Since our module is   
going to be written badly, it will crash the kernel. Then, we will analyze the crash report.   
Using the information obtained in the report, we will try to figure out what’s wrong with   
our sources.

26.7.2 Step 1: Kernel module

We first need to write some C code. Let’s begin with hello.c. Without getting too   
technical, here’s the most basic of modules, with the init and cleanup functions. The   
module does not nothing special except print messages to the kernel logging facility.

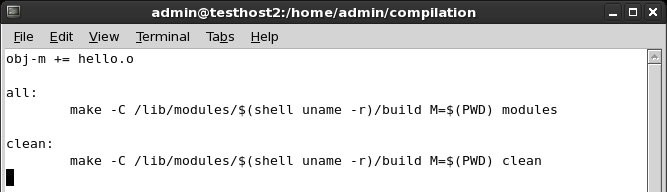
129



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 70: Basic kernel module

130



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

1 /∗

2 ∗ hello .c − The simplest kernel module .

3 ∗/

4

5 #i n c l u d e < l i n u x / module . h> /∗ Needed by all modules ∗/

6 #i n c l u d e < l i n u x / k e r n e l . h> /∗ Needed for KERN\_INFO ∗/

7

8 int init\_module(void)

9 {

10 printk (KERN\_INFO " Hello world .\n") ;

11

12 /∗

13 ∗ A non 0 return means init\_module failed ; module can’t be

loaded.

14 ∗/

15 return 0;

16 }

17

18 void cleanup\_module(void)

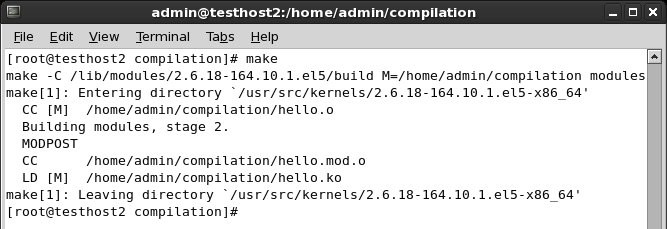
19 {

20 printk (KERN\_INFO "Goodbye world .\n") ;

21 }

We need to compile this module, so we need a Makefile: Figure 71: Basic example Makefile

131



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

1 obj−m += hello . o   
2

3 all :

4 make −C / l i b /modules/$( s h e l l uname −r )/ build M=$(PWD)

modules   
5

6 clean:

7 make −C / l i b /modules/$( s h e l l uname −r )/ build M=$(PWD) clean

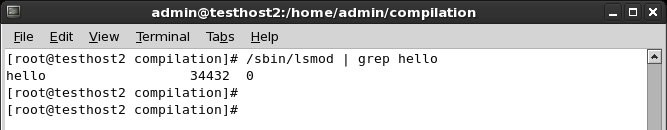
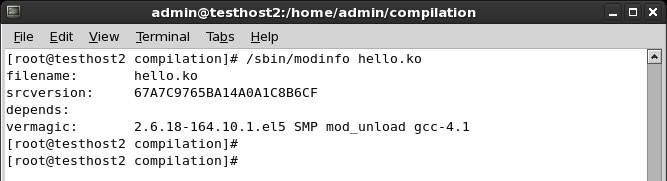
Now, we need to make the module. In the directory containing your hello.c program and the Makefile, just run make. You will see something like this:

Figure 72: Basic example make command output

Our module has been compiled. Let’s insert it into the kernel. This is done using the insmod command. However, a second before we do that, we can examine our module and see what it does. Maybe the module advertises certain bits of information that we might find of value. Use the modinfo command for that.

/sbin/modinfo hello.ko

132



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 73: modinfo example

In this case, nothing special. Now, insert it:

/sbin/insmod hello.ko

If the module loads properly into the kernel, you will be able to see it with the lsmod command:

/sbin/lsmod | grep hello

Figure 74: lsmod example

Notice that the use count for our module is 0. This means that we can unload it from   
the kernel without causing a problem. Normally, kernel modules are used for various

133



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

purposes, like communicating with system devices. Finally, to remove the module, use the rmmod command:

/sbin/rmmod hello

If you take at a look at /var/log/messages, you will notice the Hello and Goodbye messages, belonging to the init\_module and cleanup\_module functions:

Figure 75: Kernel module messages

That was our most trivial example. No crash yet. But we have a mechanism of inserting code into the kernel. If the code is bad, we will have an oops or a panic.

26.7.3 Step 2: Kernel panic

We’ll now create a new C program that uses the panic system call on initialization. Not very useful, but good enough for demonstrating the power of crash analysis. Here’s the code, we call it kill-kernel.c.

134

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

1 /∗

2 ∗ kill−kernel.c − The simplest kernel module to crash kernel .

3 ∗/

4

5 #i n c l u d e < l i n u x / module . h> /∗ Needed by all modules ∗/

6 #i n c l u d e < l i n u x / k e r n e l . h> /∗ Needed for KERN\_INFO ∗/

7

8 int init\_module(void)

9 {

10 printk (KERN\_INFO " Hello world . Now we crash .\n") ;

11 panic ("Down we go , panic called ! ") ;

12

13 return 0;

14 }

15

16 void cleanup\_module(void)

17 {

18 printk (KERN\_INFO "Goodbye world .\n") ;

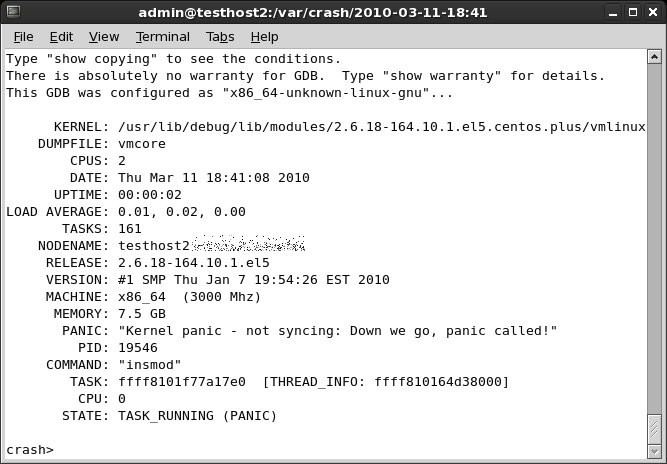
19 }

When inserted, this module will write a message to /var/log/messages and then panic.   
Indeed, this is what happens. Once you execute the insmod command, the machine will   
freeze, reboot, dump the kernel memory and then reboot back into the production kernel.

26.7.4 Step 3: Analysis

Let’s take a look at the vmcore.

135

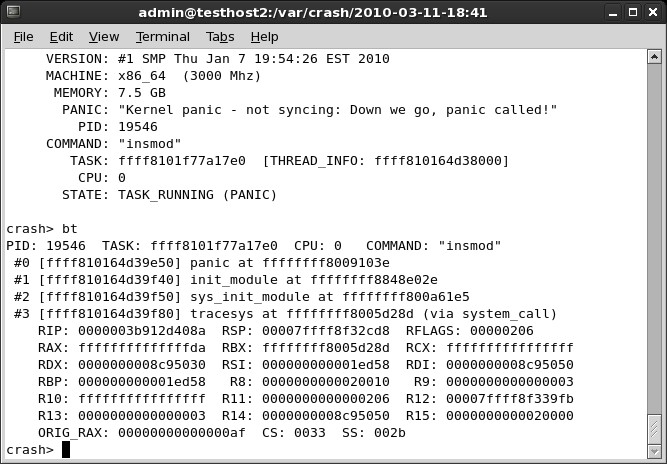


[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 76: Intermediate example crash summary

And the backtrace:

136



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 77: Intermediate example backtrace

What do we have here? First, the interesting bit, the PANIC string:

"Kernel panic - not syncing: Down we go, panic called!"

That bit looks familiar. Indeed, this is our own message we used on panic. Very infor-  
mative, as we know what happened. We might use something like this if we encountered an error in the code, to let know the user what the problem is. Another interesting piece is the dumping of the CS register - CS: 0033. Seemingly, we crashed the kernel in user mode. As I’ve mentioned before, this can happen if you have hardware problems or if there’s a problem with a system call. In our case, it’s the latter. Well, that was easy -  
and self-explanatory. So, let’s try a more difficult example.

137

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

For more information about writing kernel modules, including benevolent purposes, please   
consult the [Linux Kernel Module Programming Guide](http://tldp.org/LDP/lkmpg/2.6/html/).

26.8 Difficult example

Now another, a more difficult example. We panicked our kernel with ... panic. Now, let’s try some coding malpractice and create a NULL pointer testcase. We will now create a classic NULL pointer example, the most typical problem with programs. NULL pointers can lead to all kinds of unexpected behavior, including kernel crashes. Our program, called null-pointer.c, now looks like this:

1 /∗

2 ∗ null−pointer.c

kernel.

3 ∗/   
4

− A not so simple kernel module to crash

5 #i n c l u d e < l i n u x / module . h>   
6 #i n c l u d e < l i n u x / k e r n e l . h>   
7

8 char ∗ p=NULL ;   
9

10 int init\_module(void)

11 {

/∗ Needed by all modules ∗/   
/∗ Needed for KERN\_INFO ∗/

12 printk (KERN\_INFO "We is gonna KABOOM now!\n") ;

13

14 ∗p = 1;

15 return 0;

16 }

17

18 void cleanup\_module(void)

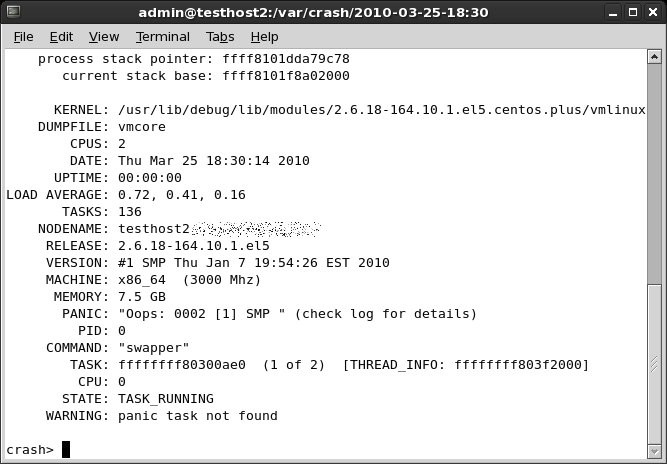
19 {

20 printk (KERN\_INFO "Goodbye world .\n") ;

21 }

We declare a NULL pointer and then dereference it. Not a healthy practice. I guess   
programmers can explain this more eloquently than I, but you can’t have something   
pointing to nothing get a valid address of a sudden. In kernel, this leads to panic.

138



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Indeed, after making this module and trying to insert it, we get panic. Now, the sweet   
part.

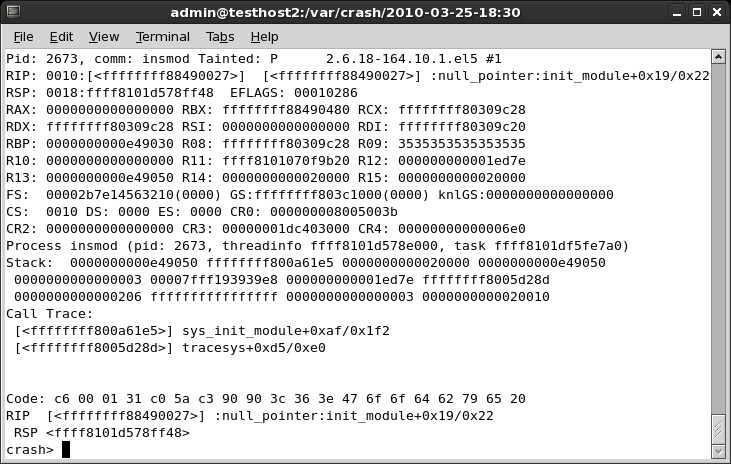
26.8.1 In-depth analysis

Looking at the crash report, we see a goldmine of information: Figure 78: Null pointer example crash report

Let’s digest the stuff:

PANIC: "Oops: 0002 [1] SMP " (check log for   
details)

139



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

We have an Oops on CPU 1. 0002 translates to 0010 in binary, meaning no page was found during a write operation in kernel mode. Exactly what we’re trying to achieve. We’ve also referred to the log. More about that soon.

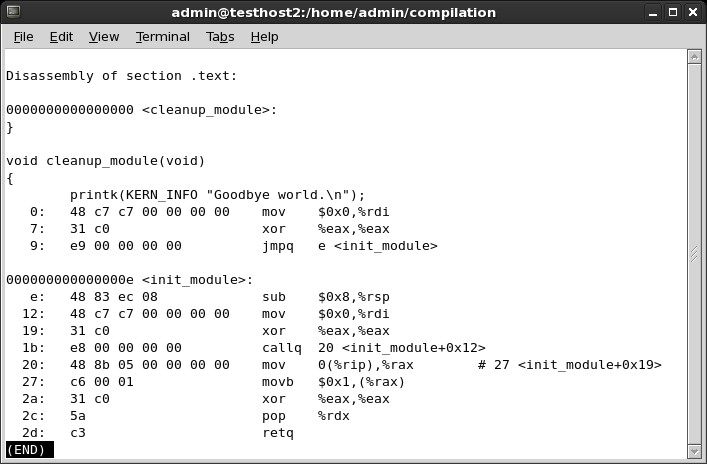
WARNING: panic task not found

There was no task, because we were just trying to load the module, so it died before   
it could run. In this case, we will need to refer to the log for details. This is done by   
running log in the crash utility, just as we’ve learned. The log provides us with what we   
need:

Figure 79: Null pointer example crash log

The RIP says null\_pointer:init\_module+0x19/0x22. We’re making progress here.   
We know there was a problem with NULL pointer in the init\_module function. Time

140



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

to disassemble the object and see what went wrong. There’s more useful information, including the fact the kernel was Tainted by our module, the dumping of the CS register and more. We’ll use this later. First, let’s objdump our module.

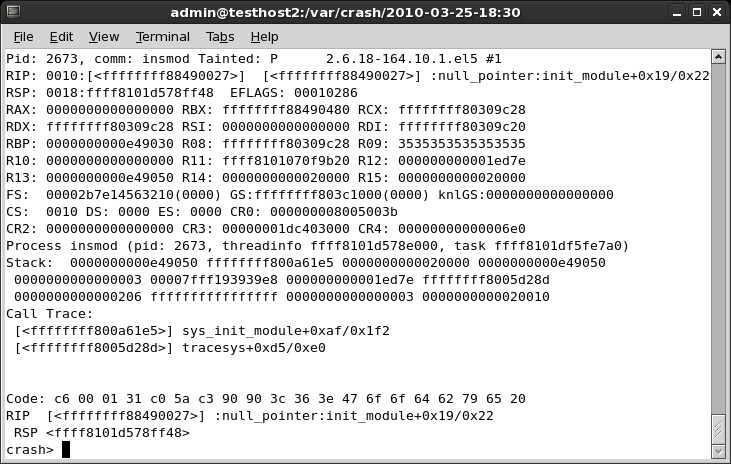
objdump -d -S null-pointer.ko > /tmp/whatever

Looking at the file, we see the Rain Man code:

Figure 80: Null pointer example disassembled object code

The first part, the cleanup is not really interesting. We want the init\_module. The   
problematic line is even marked for us with a comment: # 27 <init\_module+0x19>.

141



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

27: c6 00 01 movb $0x1,(%rax)

What do we have here? We’re trying to load (assembly movb) value 1 ($0x1) into the RAX register (%rax). Now, why does it cause such a fuss? Let’s go back to our log and see the memory address of the RAX register:

Figure 81: Null pointer example registers

RAX register is: 0000000000000000. In other words, zero. We’re trying to write to   
memory address 0. This causes the page fault, resulting in kernel panic. Problem solved!

Of course, in real life, nothing is going to be THAT easy, but it’s a start. In real life,   
you will face tons of difficulties, including missing sources, wrong versions of GCC and   
all kinds of problems that will make crash analysis very, very difficult. Remember that!

In some cases, you will see a problem that is not immediately apparent from looking at   
the sources. This means you will need to work your way through the vmcore, carefully

142

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

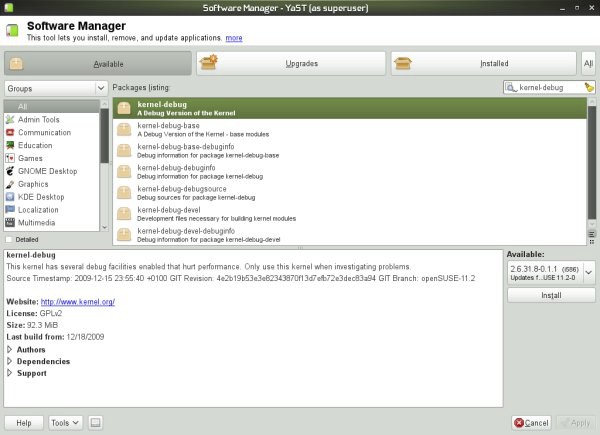
tracing the execution. In a nutshell, you will execute whatis command against the function listed in exception RIP to see what kind of object it is and what arguments it takes. Then, you will run bt -f command to show all stack data in a frame and focus on the last thing pushed on the stack. After that, you will use stack command to dump the complete contents of the data structure at the given address and work your way through the structure chain, trying to pinpoint the failing/buggy bit of code.

For more information, please take a look at the [case study](http://people.redhat.com/anderson/crash_whitepaper/#EXAMPLES) shown in the crash White   
Paper. Again, it’s easier when you know what you’re looking for. Any example you encounter online will be several orders of magnitude simpler than your real crashes, but it is really difficult demonstrating an all-inclusive, abstract case. Still, I hope my two examples are thorough enough to get you started.

26.9 Alternative solution (debug kernel)

If you have time and space, you may want to download and install a debug kernel for   
your kernel release. Not for everyday use, of course, but it could come handy when   
you’re analyzing kernel crashes. While it is big and bloated, it may offer additional,   
useful information that can’t be derived from standard kernels. Plus, the objects with   
debug symbols might be there, so you won’t need to recompile them, just dump them   
and examine the code.

143



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 82: Debug kernel installation

Figure 83: Debug kernel installation details

27 Next steps

So the big question is, what do crash reports tell us? Well, using the available information, we can try to understand what is happening on our troubled systems.

First and foremost, we can compare different crashes and try to understand if there’s any   
common element. Then, we can try to look for correlations between separate events,   
environment changes and system changes, trying to isolate possible culprits to our crashes.

144



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Combined with submitting crash reports to vendors and developers, plus the ample use   
of Google and additional resources, like mailing lists and forums, we might be able to   
narrow down our search and greatly simply the resolution of problems. Kernel crash bug   
reporting

When your kernel crashes, you may want to take the initiative and submit the report to   
the vendor, so that they may examine it and possibly fix a bug. This is a very important   
thing. You will not only be helping yourself but possibly everyone using Linux anywhere.   
What more, kernel crashes are valuable. If there’s a bug somewhere, the developers will   
find it and fix it.

27.1 kerneloops.org

[kerneloops homepage](http://www.kerneloops.org/) - <http://www.kerneloops.org/>  
Figure 84: kernelops.org logo

kerneloops.org is a website dedicated to collecting and listing kernel crashes across the various kernel releases and crash reasons, allowing kernel developers to work on identifying most critical bugs and solving them, as well as providing system administrators, engineers and enthusiasts with a rich database of crucial information.

145



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 85: kernelops.org example

Remember the Fedora 12 kernel crash report? We had that native\_apic\_write\_dummy? Well, let’s see what kerneloops.org has to say about it.

146

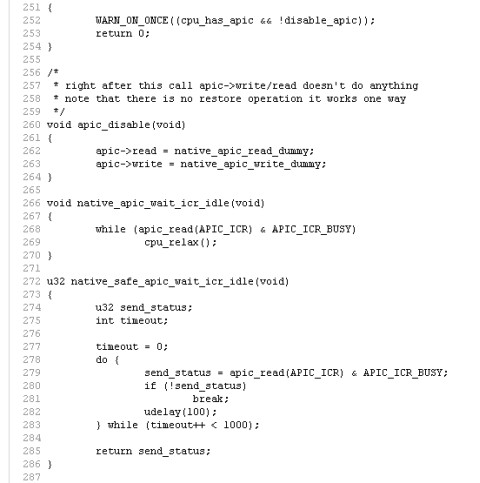


[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 86: kerneloops.org example - continued

As you can see, quite a lot. Not only do you have all sorts of useful statistics, you can actually click on the exception link and go directly to source, to the problematic bit of code and see what gives. This is truly priceless information!

147



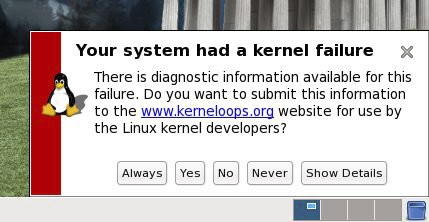
[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 87: kerneloops.org example code

As we mentioned earlier, some modern Linux distributions have an automated mechanism for kernel crash submission, both anonymously and using a Bugzilla account.

For example, Fedora 12 uses the Automatic Bug Reporting Tool (ABRT), which collects   
crash data, runs a report and then sends it for analysis with the developers. For more   
details, you may want to read the [Wiki](https://fedorahosted.org/abrt/wiki). Beforehand, Fedora 11 used kerneloops utility,

148



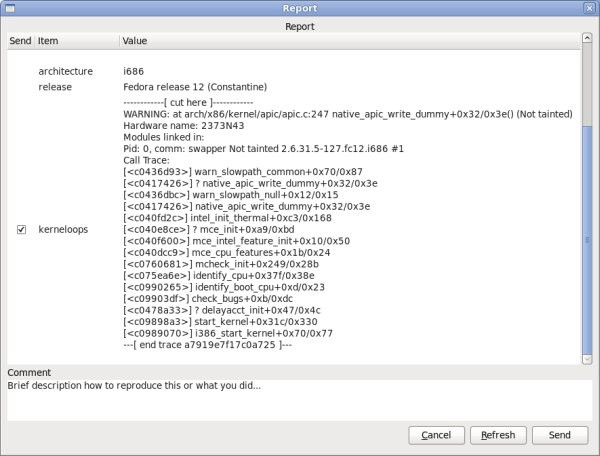
[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

which sent reports to, yes, you guessed it right, kerneloops.org. Now, some screenshots;   
here’s an example of a live submission in [Fedora 11](http://www.dedoimedo.com/computers/fedora-11.html):

Figure 88: Kernel crash report in Fedora 11

And more recently in [Fedora 12](http://www.dedoimedo.com/computers/fedora-12.html):

149



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 89: Kernel crash report in Fedora 12

And here’s [Debian 5.03 Lenny](http://www.dedoimedo.com/computers/debian.html):

150



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 90: Kernel crash report in Debian Lenny

Hopefully, all these submissions help make next releases of Linux kernel and the specific distributions smarter, faster, safer, and more stable.

27.2 Google for information

Sounds trivial, but it is not. If you’re having a kernel crash, there’s a fair chance some-  
one else saw it too. While environments differ from one another, there still might be   
some commonality for them all. Then again, there might not. A site with 10 database   
machines and local logins will probably experience different kinds of problems than a   
10,000-machine site with heavy use of autofs and NFS. Similarly, companies working   
with this or that hardware vendor are more likely to undergo platform-specific issues that   
can’t easily be find elsewhere.

The simplest way to search for data is to paste the exception RIP into the search box and look for mailing list threads and forum posts discussing same or similar items. Once again, using the Fedora case an an example:

151



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 91: Sample Google search

27.3 Crash analysis results

And after you have exhausted all the available channels, it’s time to go through the   
information and data collected and try to reach a decision/resolution about the problem   
at hand.

We started with the situation where our kernel is experiencing instability and is crashing. To solve the problem, we setup a robust infrastructure that includes a mechanism for kernel crash collection and tools for the analysis of dumped memory cores. We now understand what the seemingly cryptic reports mean.

The combination of all the lessons learned during our long journey allows us to reach a   
decision what should be done next. How do we treat our crashing machines? Are they   
in for a hardware inspection, reinstallation, something else? Maybe there’s a bug in the   
kernel internals? Whatever the reason, we have the tools to handle the problems quickly   
and efficiently. Finally, some last-minute tips, very generic, very generalized, about what   
to do next:

152

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

27.3.1 Single crash

A single crash may seem as too little information to work with. Don’t be discouraged. If you can, analyze the core yourself or send the core to your vendor support. There’s a fair chance you will find something wrong, either with software at hand, the kernel or the hardware underneath.

27.3.2 Hardware inspection

Speaking of hardware, kernel crashes can be caused by faulty hardware. Such crashes   
usually seem sporadic and random in reason. If you encounter a host that is experiencing   
many crashes, all of which have different panic tasks, you may want to considering   
scheduling some downtime and running a hardware check on the host, including memtest,   
CPU stress, disk checks, and more. Beyond the scope of this book, I’m afraid.

The exact definition of what is considered many crashes, how critical the machine is, how much downtime you can afford, and what you intend to do with the situation at hand is individual and will vary from one admin to another.

27.3.3 Reinstallation & software changes

Did the software setup change in any way that correlates with the kernel crashes? If so, do you know what the change is? Can you reproduce the change and the subsequent crashes on other hosts? Sometimes, it can be very simple; sometimes, you may not be able to easily separate software from the kernel or the underlying hardware.

If you can, try to isolate the changes and see how the system responds with or without them. If there’s a software bug, then you might be just lucky enough and have to deal with a reproducible error. Kernel crashes due to a certain bug in software should look pretty much the same. But there’s no guarantee you’ll have it that easy.

Now, if your system is a generic machine that does not keep any critical data on local disks, you may want to consider wiping the slate clean - start over, with a fresh installation that you know is stable. It’s worth a try.

27.3.4 Submit to developer/vendor

Regardless of what you discovered or you think the problem is, you should send the kernel   
crash report to the relevant developer and/or vendor. Even if you’re absolutely sure you

153

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

know what the problem is and you’ve found the cure, you should still leave the official fix in the hands of people who do this kind of work for a living.

I have emphasized this several times throughout the book, because I truly believe this   
is important, valuable and effective. You can easily contribute to the quality of Linux   
kernel code by submitting a few short text reports. It’s as simple and powerful as that.

28 Conclusion

We worked carefully and slowly through the kernel crash analysis series. In this last part,   
we have finally taken a deep, personal look at the crash internals and now have the   
right tools and the knowledge to understand what’s bothering our kernel. Using this new   
wealth of information, we can work on making our systems better, smarter and more   
stable.

154

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Part V

Appendix

29 Kdump

This section contains a few more details about Kdump. Namely, it provides instruc-  
tions how to install kexec-tools and kernel-kdump packages manually, and how to use the friendly and simple YaST Kdump module to configure and setup Kdump in SUSE. Furthermore, it also covers changes introduced in openSUSE 11 and above.

29.1 Architecture dependencies

The settings listed in this book are only valid for the i386 and x86\_64 platforms. Itanium   
and PPC require some changes. The best place to look for details is the official doc-  
umentation under /usr/share/doc/packages/kdump. Likewise, please check References

(33) further below.

29.2 Install kernel-kdump package manually

The simplest way of installing the package is via the official distro repositories. However, if this package is missing, your kernel is probably not configured to use Kdump in the first place, so the chance of encountering this situation is slim. Still, if you did have to compile the kernel manually, then you will have to install this package after the kernel has been built and booted into.

29.3 Install kexec-tools package manually

It is possible that you will have to manually download and install the kexec-tools package, especially if you do not have a vendor-ready kernel image. The best way to install the package is via the official repositories. However, if the package is not available that way, then to obtain kexec-tools, you will have to do the following:

155

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

29.3.1 Download the package

You can look for the package either on the [official site](http://lse.sourceforge.net/kdump/) or download a version from   
[kernel.org](http://www.kernel.org/pub/linux/kernel/people/horms/kexec-tools/), whichever suits you best.

29.3.2 Extract the archive

The kexec-tools package comes archives. You will first need to extract the package:

tar zxvf kexec-tools.tar.gz

29.3.3 Make & install the package

To be able to compile your system will have to have the compilation tools installed,   
including make, gcc, kernel-source, and kernel-headers. You can obtain these from the   
repositories relevant to your distribution. For instance, on Debian-based distros, these   
tools are obtained very easily by installing build-essential package (sudo apt-get install   
build-essential ).

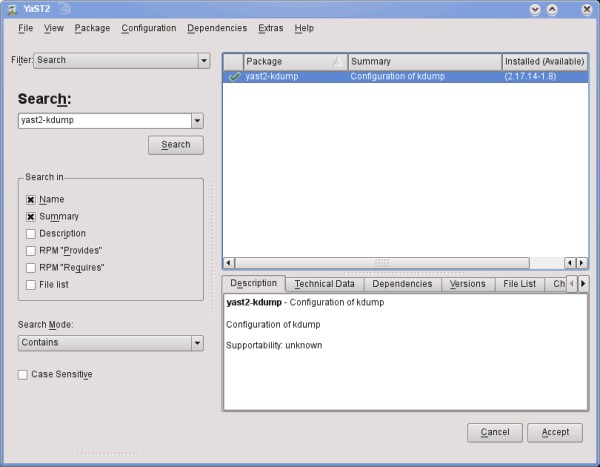
cd kexec-tools-<your-version>   
make

make install

29.3.4 Important note

Please make sure you download the right package that matches your Kdump version. Otherwise, when you try to run Kexec, you are likely to see strange errors, similar to Possible errors (14.2.1) we have seen earlier during the Kdump testing.

156



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

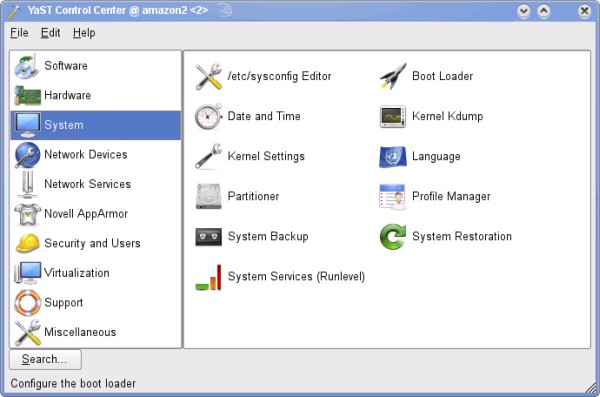
29.4 SUSE & YaST Kdump module

openSUSE (but also SLES) comes with a very handy YaST Kdump module (yast-kdump),   
which allows you to administer the Kdump configuration using YaST. On one hand, this   
makes the setup much easier. On the other, you will probably not understand the Kdump   
functionality as thoroughly as when using the command-line and working directly against   
the configuration file.

Nevertheless, I thought it would be useful to mention this. Indeed, you can see a number of screenshots taken on an openSUSE 11.1 machine, demonstrating the installation and the use of the yast-kdump package.

Figure 92: yast2-kdump package installation

157



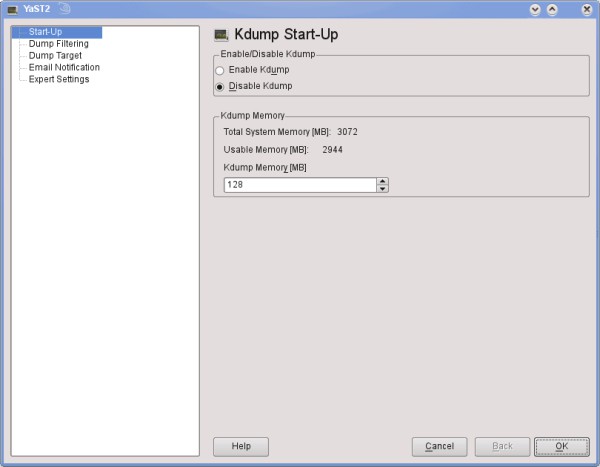
[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

After the installation, you can find the module in the System sub-menu. It’s called Kernel   
Kdump.

Figure 93: Kdump YaST module

After launching the application, you can start managing the configuration, just like we   
did before. The main difference is getting used to the layout, as the options are now   
dispersed across a number of windows. Personally, I find this approach more difficult   
to understand and manage. However, you should be aware of its existence and use it if   
needed.

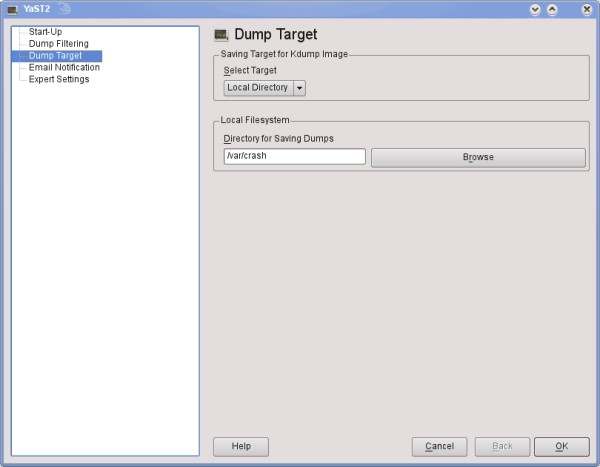
158



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 94: Kdump configuration via YaST module

159



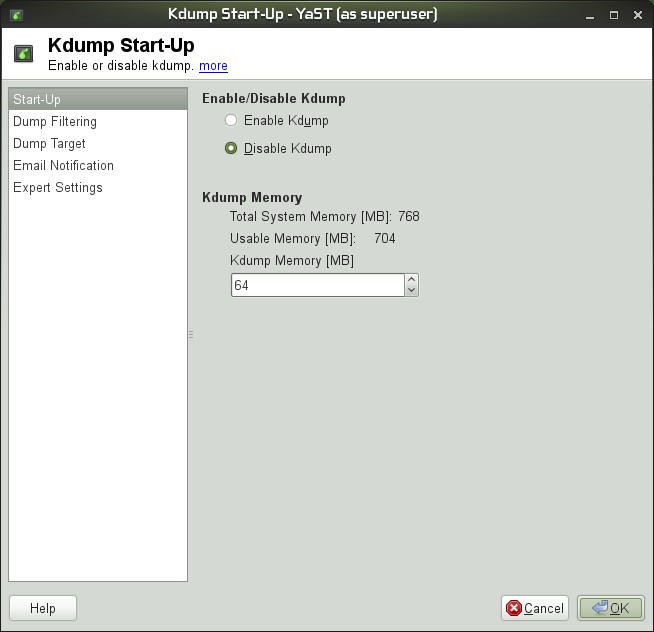
[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 95: Kdump configuration via YaST module - continued

29.5 SUSE (and openSUSE) 11.X setup

While the bulk of the book part above explains the Kdump setup in detail, some things   
have changed from SUSE 10.3 to the more recent 11.x versions. This section elaborates   
on the differences in the Kdump setup on openSUSE 11.x. Kdump works pretty much   
without any problems. Still, you may encounter a few odd issues here and there. I would   
like to help you understand these potential problems and provide you with the right tools   
to overcome them.

160



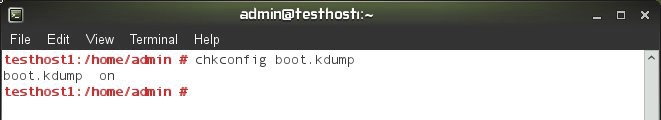
[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

29.5.1 32-bit architecture

On 32-bit openSUSE 11.2 (Gnome), which I used for the setup, the configuration of Kdump was pretty straightforward. I downloaded and installed the required packages using YaST and then launched the YaST Kernel Kdump menu.

Figure 96: Kdump startup configuration via YaST

161



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Kdump service

Kdump service has changed. It is no longer called kdump, it’s called boot.kdump and is   
invoked during the boot. This means that you will have to adjust the usage of chkconfig   
for enabling/disabling Kdump. The new Kdump startup scripts make it more similar to   
LKCD.

Figure 97: boot.kdump chkconfig command

You can also use the System Services module in YaST:

162



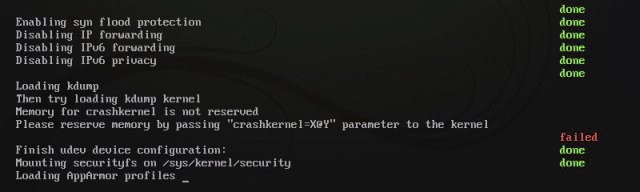
[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 98: Runlevel configuration via YaST

Memory allocation syntax

The memory allocation syntax has also changed. Although you can use the old crashk-  
ernel=XM@YM syntax just like before, you will notice the default written in the GRUB   
menu.lst configuration file is slightly different. The new syntax specifies a range rather   
just the starting point for the allocation. It’s nothing cardinal, but worth paying attention   
to.

163



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 99: Kdump GRUB syntax change

29.5.2 64-bit architecture

Memory allocation

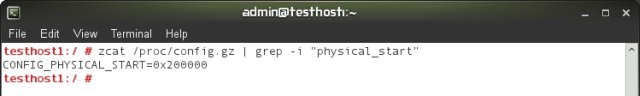
Similarly, installing and configuring Kdump on 64-bit openSUSE 11.2 takes as much effort as doing that on the 32-bit machine. However, when I tried to dump the memory, instead of booting into the crash kernel via the Kexec mechanism, the system simply got stuck. I realized the default allocating is incorrect. There are several ways you can ascertain this. First, when the system boots, you can hit Escape button to switch to verbose mode and then watch the console for Kdump error messages.

Figure 100: Failed memory reservation on a 64-bit machine

Alternatively, you can run Kexec and see if it throws any errors. Just execute:

kexec -p

164



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 101: Kexec command line error

We’ve seen this kind of message before, and it tells us that the memory has not been reserved properly. Either you have used a bad offset or none at all. The thing is, by default openSUSE, both 32-bit and 64-bit are configured to use the 16MB offset. You can check this value under /proc/config.gz, which contains the list of all parameters the kernel has been compiled with.

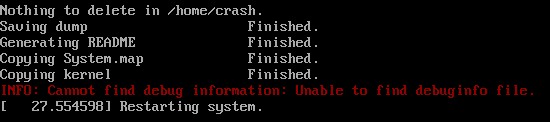
Unfortunately, while 16MB works for 32-bit systems, it is incorrect for the 64-bit architec-  
ture. Furthermore, the CONFIG\_PHYSICAL\_START value set under /proc/config.gz is incorrect. On my 64-bit openSUSE, it shows:

CONFIG\_PHYSICAL\_START=0x200000

Figure 102: Wrong physical start value

If you translate this into decimal, it’s only 2MB, below the 16MB value, an impossible   
allocation, when it should really read 0x2000000 or 32MB. Indeed, changing it to 32MB   
solves the problem. Of course, making the right choice from the start would be even   
better.

165



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 103: Kdump working after reconfigured physical start value

29.5.3 Other changes

Uncompressed kernel images

The new Kdump can worked with compressed kernels, so you will no longer require vm-  
linux under your /boot directory. Furthermore, the crash mechanism has also undergone some changes, allowing you to process memory cores in several different ways23.

debuginfo package missing

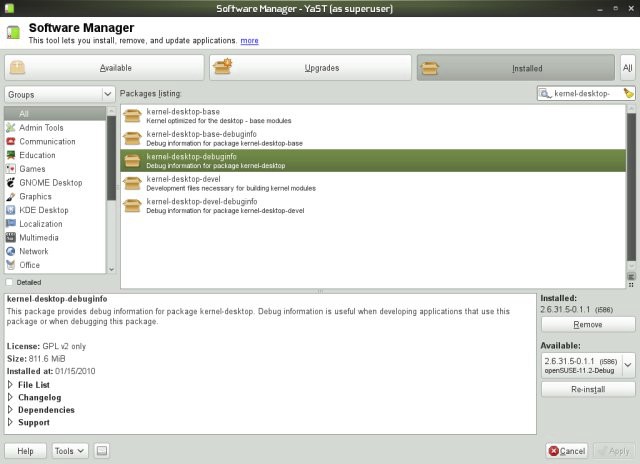
One more problem I’ve encountered is that there is no debuginfo package for the latest   
kernel available. This means you will not be able to process your cores. We have   
talked about this earlier in the Crash Collection part (III); for now, you should carefully   
inspect what your running kernel version is and what debug packages are available in the   
repositories.

Figure 104: debuginfo package missing

Available package in the repositories:

23 See Crash Collection (III).

166

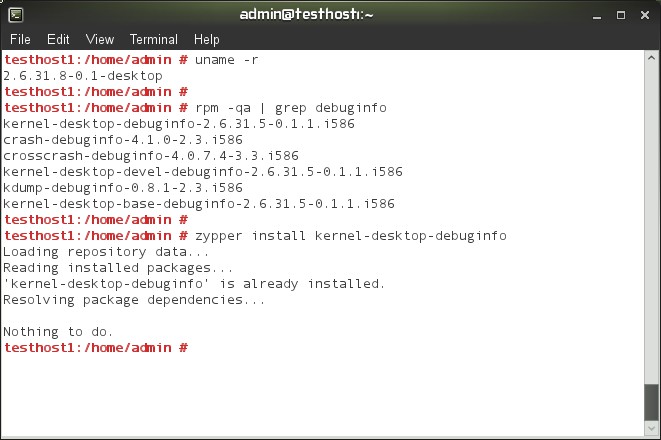


[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 105: Available kernel debuginfo package in the repository

Now, I may be mistaken, but here’s what it looks like:

167



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 106: Kernel debuginfo package installation status

The running kernel is at version 31.8-0.1, but the debuginfo is only at version 31.5-0.1.1.   
For 64-bit systems, there’s kernel-desktop-debuginfo version 31.8-0.1.1, but not 31.8-0.1,   
which again, poses a problem, as the two do not match. I did not let zypper get in the   
way, so I did a manual check in the Update repository, looking for the RPM package that   
matches my running kernel and could not find it, in either 32-bit or 64-bit directories. I   
hope this gets sorted soon24.

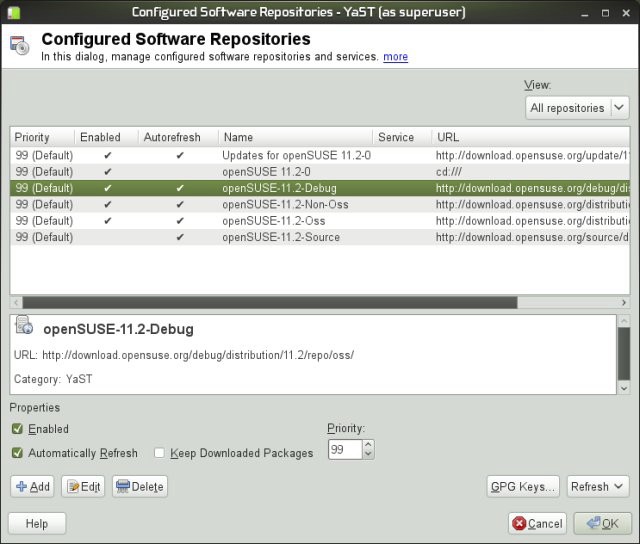
30 Crash

30.1 Enable debug repositories

A necessary part of the crash analysis procedure is to have the right debug package   
installed, namely the debuginfo package for your running kernel. In commercial versions

24 No change at the time this book was written.

168



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

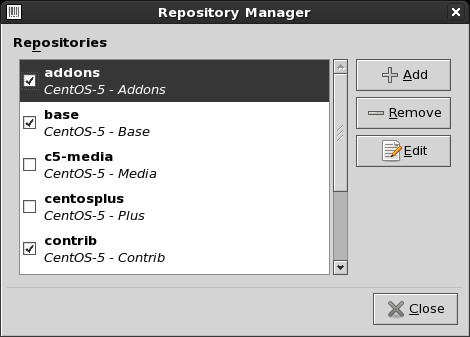
of SUSE and RedHat, debug repositories are enabled by default. However, in openSUSE, the repositories are available, but disabled, whereas in CentOS, they are missing entirely; you will have to add them manually.

Figure 107: Enabling Debug repository in openSUSE 11.2

30.1.1 Enable repositories in CentOS

Now, let’s take a look at CentOS package management, which uses Pirut front-end for the yum package manager.

169

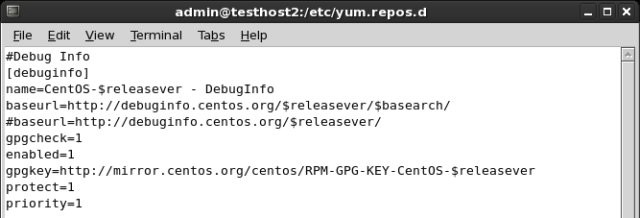


[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 108: CentOS repository manager

The default repository list does not have the Debug repository either included or enabled.   
We’ll need to add it manually, by hand. Go to [CentOS Wiki Additional Resource](http://wiki.centos.org/AdditionalResources/Repositories/DebugInfo) page and   
copy the text from the code box into a text editor. Save the file as Centos-Debug.repo under /etc/yum.repos.d.

170



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 109: Adding debug repository file

#Debug Info

[debuginfo]

name=CentOS-$releasever - DebugInfo

baseurl=<http://debuginfo.centos.org/$releasever/$basearch/>  
#baseurl=<http://debuginfo.centos.org/$releasever/>  
gpgcheck=1

enabled=0

gpgkey=[http://mirror.centos.org/centos/RPM-GPG-KEY-](http://mirror.centos.org/centos/rpm-gpg-key-/)  
CentOS-$releasever

protect=1

priority=1

Please pay attention to the two baseurl lines. The official CentOS documentation lists   
the second, shorter string, currently commented in the image and the code section above,   
as the right URL for the repository. It does not work25. However, commenting it out and   
enabling the first line, which is commented out by default, solves the problem and you   
have the Debug repository enabled. The default version, which does not work:

25 This may change at any time.

171

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

#baseurl=<http://debuginfo.centos.org/$releasever/$basearch/>  
baseurl=<http://debuginfo.centos.org/$releasever/>

This is how it ought to be:

baseurl=<http://debuginfo.centos.org/$releasever/$basearch/>  
#baseurl=<http://debuginfo.centos.org/$releasever/>

Next, run yum (or Pirut)26, after the packages are indexed, you will have debuginfo   
available, including kernel-debuginfo packages that are mandatory for crash analysis.   
You can also manually download RPM files from the [repository](http://debuginfo.centos.org/), but this is a tedious   
work and you may miss dependencies.

30.2 lcrash utility (for LKCD)

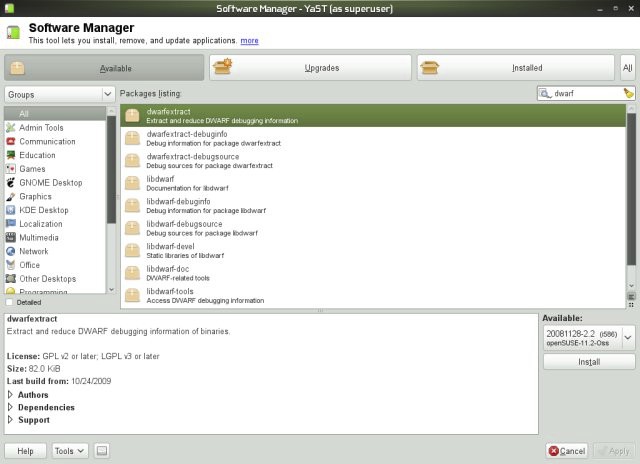
lcrash is an older utility that you may want to use with memory cores collected us-  
ing LKCD. In general, you will need not use the tool manually, because the lkcd sav e   
command that is invoked after the memory core is dumped invokes in turn lcrash and   
processes the core. lcrash requires System map and Kerntypes files to process the cores:

lcrash <System map> <Kerntypes> <core>

<System map> is usually found under /boot. <Kerntypes> is usually found under   
/boot. This file lists kernel structures and is required for the analysis of the cores.   
<core> is the name of LKCD saved core. LKCD cores are named dump.X, where X is   
a sequential number, from 0 to 9. The cores are rotated after 10 collected dumps.

26 You will need to toggle enabled=0 to enabled=1 in the file or run the Repository Manager and select the Debug repository before you can start searching and installing packages.

172



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

30.2.1 Kerntypes

If your running kernel does not have the Kerntypes file, you may be able to create one. You will need to make sure your kernel has been compiled with the -g option. You can verify this under /proc/config.gz, CONFIG\_DEBUG\_INFO=y. We did mention this as a prerequisite for crash dump collection.

Next, you will require the dwarfextract utility and run it against the kernel that matches   
the one used to collect the core and extract the kernel structures. dwarfextract is a   
tool to postprocess debuginfo. The tools removes duplicate type information caused by   
linking different compilation units. Currently, the tool has only been used to work on the   
debuginfo of the kernel package. Further functionality has been [requested](https://features.opensuse.org/305572) in the future.

Figure 110: dwarfextract installation via YaST

The usage is as follows:

173

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

dwarfextract vmlinux <Kerntypes>

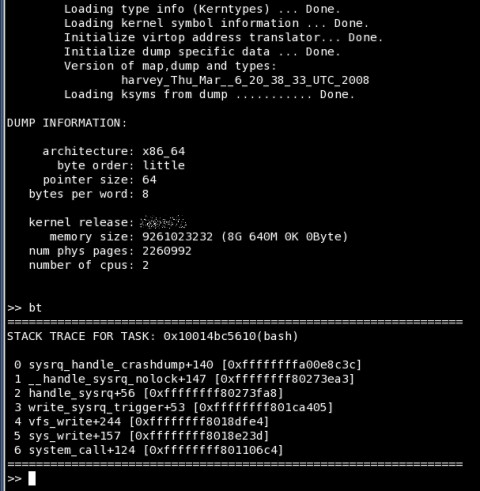
You can name the file anything you want. Just make sure to use the correct name and path when you invoke the lcrash utility.

30.3 lcrash demonstration

And that’s all. If your system is setup correctly, lcrash should load:

For more details, please consult the [official](http://lkcd.sourceforge.net/doc/index.html) documentation. You may also want to read   
the somewhat [older](http://www.faqs.org/docs/Linux-HOWTO/Linux-Crash-HOWTO.html) howto on faqs.org. Furthermore, there’s a very detailed guide in   
PDF format is available (direct link): [lcrash HOWTO](http://lkcd.sourceforge.net/doc/lcrash.pdf). And that’s all. If your system is   
setup correctly, lcrash should load:

174



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 111: lcrash example

175

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

31 Other tools

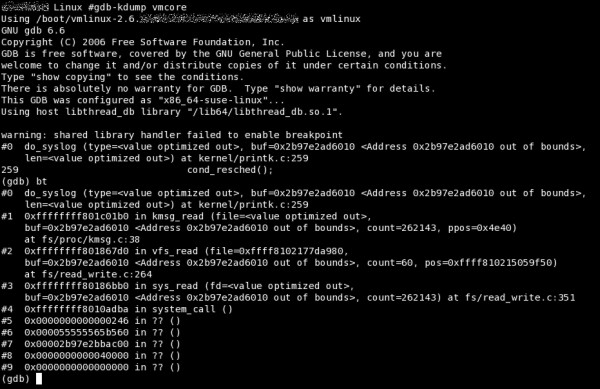
31.1 gdb-kdump

gdb-kdump is a helper script that you can use instead of crash, although you will experi-  
ence a limited subset of commands and functions. gdb-kdump can automatically search   
and processes the latest core, uncompress kernels, and run basic commands like bt, btpid   
and dmesg. gdb-kdump is run against the vmcore file. By default, it will look for the   
same kernel used in the vmcore under /boot. If it does not find it, it will complain, but   
you can solve the problem by either copying or symlinking the vmlinux file.

gdb-kdump vmcore

Here’s a sample output:

176



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

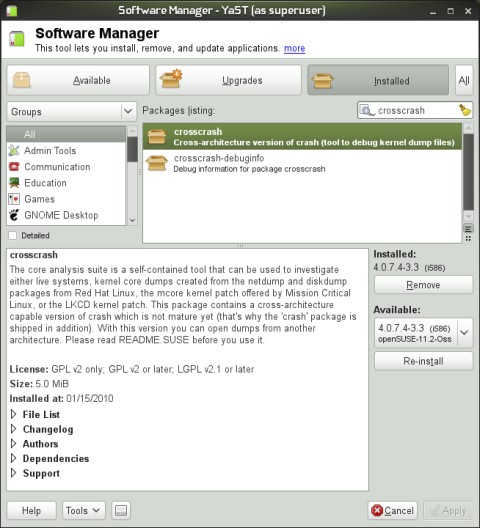
Figure 112: gdb-kdump sample run

gdb-kdump usage is beyond the scope of this book. We will talk more about gdb in a   
dedicated tutorial on [www.dedoimedo.com.](http://www.dedoimedo.com./)

31.2 crosscrash

Another interesting tool you might be interested in is crosscrash. Like gdb-kdump, it’s meant to facilitate the reading and analysis of memory core files, without forcing the users to remember the subtle differences between kernel releases, tools and formats. crosscrash is still a new technology, so it may not work as expected. However, you should know about it and test once in a while, to see if it suits your needs.

177



[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Figure 113: crosscrash installation via YaST

178

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

Part VI

References

All of the references are listed as they appear in the book, in the chronological order. The   
links are also fully parsed so that you can use them if you print this book. This section   
also includes a number of links to Dedoimedo articles mentioned here. For updates,   
as well as the complete listing of 500+ reviews, guides and tutorials, you should visit   
[www.dedoimedo.com.](http://www.dedoimedo.com./)

32 LKCD references

1. [LKCD official site](http://lkcd.sourceforge.net/index.html)

<http://lkcd.sourceforge.net/index.html>

2. [Kernel panic](http://en.wikipedia.org/wiki/Kernel_panic)

[http://en.wikipedia.org/wiki/Kernel\_panic](http://en.wikipedia.org/wiki/kernel_panic/)

3. [Linux kernel oops](http://en.wikipedia.org/wiki/Linux_kernel_oops)

[http://en.wikipedia.org/wiki/Linux\_kernel\_oops](http://en.wikipedia.org/wiki/linux_kernel_oops/)

33 Kdump references

1. [Kdump official site](http://lse.sourceforge.net/kdump/)

<http://lse.sourceforge.net/kdump/>

2. [Kdump official documentation](http://www.mjmwired.net/kernel/Documentation/kdump/)

[http://www.mjmwired.net/kernel/Documentation/kdump/](http://www.mjmwired.net/kernel/documentation/kdump/)

3. [Debugging Linux kernel using Kdump](http://www.linuxsymposium.org/2006/kdump_slides.pdf)

[http://www.linuxsymposium.org/2006/kdump\_slides.pdf](http://www.linuxsymposium.org/2006/kdump_slides.pdf/)

34 Crash references

1. [System.map on Wikipedia](http://en.wikipedia.org/wiki/System.map)

[http://en.wikipedia.org/wiki/System.map](http://en.wikipedia.org/wiki/system.map/)

179

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

2. [The Linux Kernel HOWTO - Systemmap](http://www.faqs.org/docs/Linux-HOWTO/Kernel-HOWTO.html#systemmap)

[http://www.faqs.org/docs/Linux-HOWTO/Kernel-HOWTO.html#systemmap](http://www.faqs.org/docs/linux-howto/kernel-howto.html#systemmap/)

3. [Crashdump Debugging - openSUSE](http://en.opensuse.org/Crashdump_Debugging)

[http://en.opensuse.org/Crashdump\_Debugging](http://en.opensuse.org/crashdump_debugging/)

4. [Kdump - openSUSE](http://en.opensuse.org/Kdump)

[http://en.opensuse.org/Kdump](http://en.opensuse.org/kdump/)

5. [Crash White Paper](http://people.redhat.com/anderson/crash_whitepaper/)

<http://people.redhat.com/anderson/crash_whitepaper/>

6. [Official LKCD documentation](http://lkcd.sourceforge.net/doc/index.html)

<http://lkcd.sourceforge.net/doc/index.html>

7. [Linux Crash HOWTO](http://www.faqs.org/docs/Linux-HOWTO/Linux-Crash-HOWTO.html)

[http://www.faqs.org/docs/Linux-HOWTO/Linux-Crash-HOWTO.html](http://www.faqs.org/docs/linux-howto/linux-crash-howto.html)

8. [lcrash HOWTO](http://lkcd.sourceforge.net/doc/lcrash.pdf)

[http://lkcd.sourceforge.net/doc/lcrash.pdf](http://lkcd.sourceforge.net/doc/lcrash.pdf/)

9. [Crash utility notes - Transition from exception stack](http://www.mail-archive.com/crash-utility@redhat.com/msg01699.html)

<http://www.mail-archive.com/crash-utility@redhat.com/msg01699.html>

10. O’Reilly’s Understanding Linux Kernel, Chapter 9: Process Address Space, Page   
 Fault Exception Handler, pages 376-382

11. O’Reilly’s Understanding Linux Kernel, Chapter 2: Memory Addressing, Page 36-39

12. [Linux Kernel Archive](http://kernel.org/)

<http://kernel.org/>

13. [cscope](http://cscope.sourceforge.net/)

<http://cscope.sourceforge.net/>

14. [MPlayer source download](http://www.mplayerhq.hu/design7/dload.html)

<http://www.mplayerhq.hu/design7/dload.html>

15. [objdump man page](http://linux.die.net/man/1/objdump)

[http://linux.die.net/man/1/objdump](http://linux.die.net/man/1/objdump/)

16. [Linux Kernel Module Programming Guide](http://tldp.org/LDP/lkmpg/2.6/html/)   
 [http://tldp.org/LDP/lkmpg/2.6/html/](http://tldp.org/ldp/lkmpg/2.6/html/)

180

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

17. [Crash Whitepaper case study](http://people.redhat.com/anderson/crash_whitepaper/#EXAMPLES)

[http://people.redhat.com/anderson/crash\_whitepaper/#EXAMPLES](http://people.redhat.com/anderson/crash_whitepaper/#examples/)

18. [Fedora ABRT wiki](https://fedorahosted.org/abrt/wiki)

[https://fedorahosted.org/abrt/wiki](https://fedorahosted.org/abrt/wiki/)

19. [CentOS Debug repository](http://debuginfo.centos.org/repository)

[http://debuginfo.centos.org/repository](http://debuginfo.centos.org/repository/)

35 Dedoimedo web articles

1. [Collecting and analyzing Linux kernel crashes - LKCD](http://www.dedoimedo.com/computers/lkcd.html)   
 <http://www.dedoimedo.com/computers/lkcd.html>

2. [Collecting and analyzing Linux kernel crashes - Kdump](http://www.dedoimedo.com/computers/kdump.html)   
 <http://www.dedoimedo.com/computers/kdump.html>

3. [How to enable debug repositories in CentOS Linux](http://www.dedoimedo.com/computers/centos-debug.html)

<http://www.dedoimedo.com/computers/centos-debug.html>

4. [Kdump on openSUSE 11.2](http://www.dedoimedo.com/computers/kdump-opensuse.html)

<http://www.dedoimedo.com/computers/kdump-opensuse.html>

5. [Kdump on CentOS 5.4](http://www.dedoimedo.com/computers/kdump-centos.html)

<http://www.dedoimedo.com/computers/kdump-centos.html>

6. [Collecting and analyzing Linux kernel crashes - crash](http://www.dedoimedo.com/computers/crash.html)   
 <http://www.dedoimedo.com/computers/crash.html>

7. [Analyzing Linux kernel crash dumps with crash - The one tutorial that has it all](http://www.dedoimedo.com/computers/crash-analyze.html)   
 <http://www.dedoimedo.com/computers/crash-analyze.html>

8. [openSUSE 11.2 KDE](http://www.dedoimedo.com/computers/opensuse-11sp2.html)

<http://www.dedoimedo.com/computers/opensuse-11sp2.html>

9. [openSUSE 11.2 Gnome](http://www.dedoimedo.com/computers/opensuse-11sp2-gnome.html)

<http://www.dedoimedo.com/computers/opensuse-11sp2-gnome.html>

10. [CentOS 5.3](http://www.dedoimedo.com/computers/centos.html)

<http://www.dedoimedo.com/computers/centos.html>

181

[www.dedoimedo.com](http://www.dedoimedo.com/) all rights reserved

11. [Ubuntu 9.10 Karmic Koala](http://www.dedoimedo.com/computers/ubuntu-9-10.html)

<http://www.dedoimedo.com/computers/ubuntu-9-10.html>

12. [Fedora 11 Leonidas](http://www.dedoimedo.com/computers/fedora-11.html)

<http://www.dedoimedo.com/computers/fedora-11.html>

13. [Fedora 12 Constantine](http://www.dedoimedo.com/computers/fedora-12.html)

<http://www.dedoimedo.com/computers/fedora-12.html>

14. [Debian 5.03 Lenny](http://www.dedoimedo.com/computers/debian.html)

<http://www.dedoimedo.com/computers/debian.html>

15. [RedHat Linux Enterprise 6 Beta](http://www.dedoimedo.com/computers/rh6-beta.html)

<http://www.dedoimedo.com/computers/rh6-beta.html>

16. [Linux system utilities](http://www.dedoimedo.com/computers/linux-system-utilities.html)

<http://www.dedoimedo.com/computers/linux-system-utilities.html>

17. [GRUB bootloader - Full tutorial](http://www.dedoimedo.com/computers/grub.html)

<http://www.dedoimedo.com/computers/grub.html>

18. [GRUB 2 bootloader - Full tutorial](http://www.dedoimedo.com/computers/grub-2.html)

<http://www.dedoimedo.com/computers/grub-2.html>

182