

# **Linux Kernel Crash Book**

## **Everything you need to know**

**Igor Ljubuncic aka Dedoimedo**

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

# Contents

<b>I LKCD</b>	<b>23</b>
<b>1 Introduction</b>	<b>23</b>
1.1 How does LKCD work? . . . . .	23
1.1.1 Stage 1 . . . . .	23
1.1.2 Stage 2 . . . . .	24
<b>2 LKCD Installation</b>	<b>25</b>
<b>3 LKCD local dump procedure</b>	<b>25</b>
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
<b>4 LKCD netdump procedure</b>	<b>31</b>
<b>5 Configure LKCD netdump server</b>	<b>31</b>
5.1 Required packages . . . . .	31
5.2 Configuration file . . . . .	31

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
<b>6</b>	<b>Configure LKCD client for netdump</b>	<b>34</b>
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
<b>7</b>	<b>Test functionality</b>	<b>37</b>
<b>8</b>	<b>Problems</b>	<b>38</b>
8.1	Unsuccessful netdump to different network segment . . . . .	39
<b>9</b>	<b>Conclusion</b>	<b>39</b>
<b>II</b>	<b>Kdump</b>	<b>41</b>
<b>10</b>	<b>Introduction</b>	<b>41</b>
10.1	Restrictions . . . . .	42
10.1.1	Kernel compilation . . . . .	42

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
<b>11 Kdump installation</b>	<b>43</b>
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
<b>12 Kdump packages &amp; files</b>	<b>49</b>
12.1 Kdump packages . . . . .	49
12.2 Kdump files . . . . .	50
<b>13 Kdump configuration</b>	<b>51</b>
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

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
<b>14 Test configuration</b>	<b>64</b>
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
<b>15 Simulate kernel crash</b>	<b>68</b>
<b>16 Kdump network dump functionality</b>	<b>69</b>
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
<b>17 Conclusion</b>	<b>72</b>
<b>III Crash Collection</b>	<b>73</b>

<b>18 Crash setup</b>	<b>73</b>
18.1 Prerequisites . . . . .	73
18.2 Kdump working crash installation . . . . .	73
18.3 Crash location . . . . .	74
18.4 Memory cores . . . . .	77
<b>19 Invoke crash</b>	<b>77</b>
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
<b>20 Running crash</b>	<b>81</b>
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
<b>21 Possible errors</b>	<b>87</b>
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
<b>22 Conclusion</b>	<b>90</b>
<b>IV Crash Analysis</b>	<b>91</b>

<b>23 Analyzing the crash report - First steps</b>	<b>91</b>
<b>24 Getting warmer</b>	<b>98</b>
24.1 Fedora example . . . . .	99
24.2 Another example, from the White Paper . . . . .	100
24.3 Kernel Page Error . . . . .	100
<b>25 Getting hot</b>	<b>102</b>
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
<b>26 Super geeky stuff</b>	<b>113</b>
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

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
<b>27</b>	<b>Next steps</b>	<b>144</b>
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
<b>28</b>	<b>Conclusion</b>	<b>154</b>
<b>V</b>	<b>Appendix</b>	<b>155</b>
<b>29</b>	<b>Kdump</b>	<b>155</b>
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

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
<b>30 Crash</b>	<b>168</b>
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
<b>31 Other tools</b>	<b>176</b>
31.1 gdb-kdump . . . . .	176
31.2 crosscrash . . . . .	177
<b>VI References</b>	<b>179</b>
<b>32 LKCD references</b>	<b>179</b>
<b>33 Kdump references</b>	<b>179</b>
<b>34 Crash references</b>	<b>179</b>
<b>35 Dedoimedo web articles</b>	<b>181</b>

## 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 configuration via 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

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

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

83	Debug kernel installation details . . . . .	144
84	kerneloops.org logo . . . . .	145
85	kerneloops.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

111	lcrash example . . . . .	175
112	gdb-kdump sample run . . . . .	177
113	crosscrash installation via YaST . . . . .	178

## 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

## 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  $\text{\LaTeX}$ . 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.

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

## About

[Dedoimedo](#) ([www.dedoimedo.com](http://www.dedoimedo.com)) is a website specializing in step-by-step tutorials intended 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!

## 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.

## **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).

## 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 gain 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.

## Errata

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

# Part I

# LKCD

## 1 Introduction

**LKCD** stands for **L**inux **K**ernel **C**h~~a~~s~~p~~ **D**ump. 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 SUSE<sup>1</sup> 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.

---

<sup>1</sup> LKCD is an older utility and may not work well with modern kernels.

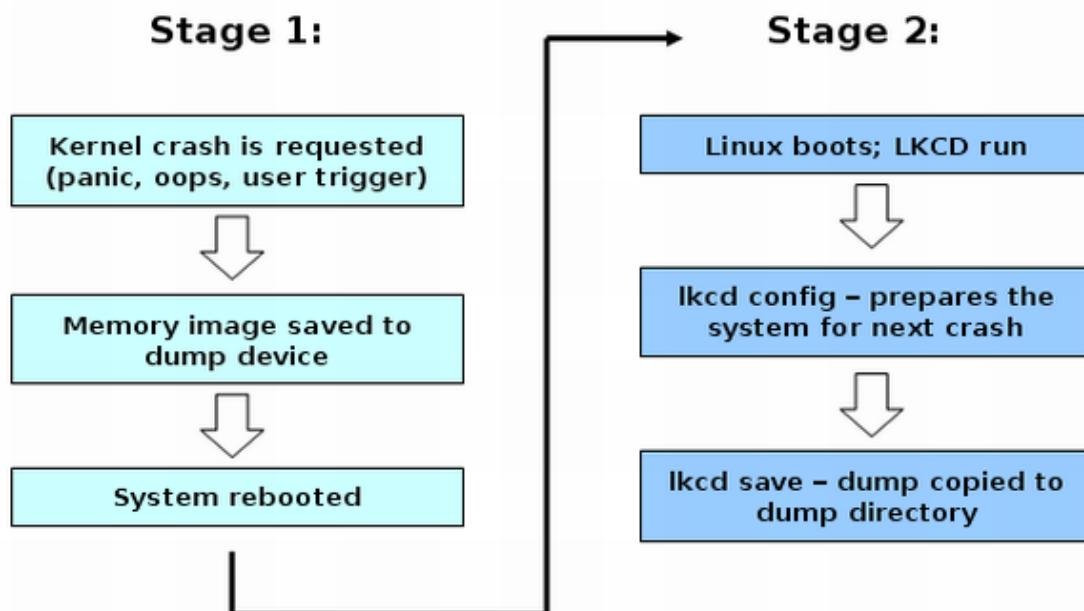
### 1.1.2 Stage 2

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 command 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



## 2 LKCD Installation

The LKCD installation requires kernel compilation. This is a lengthy and complex procedure 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.

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.

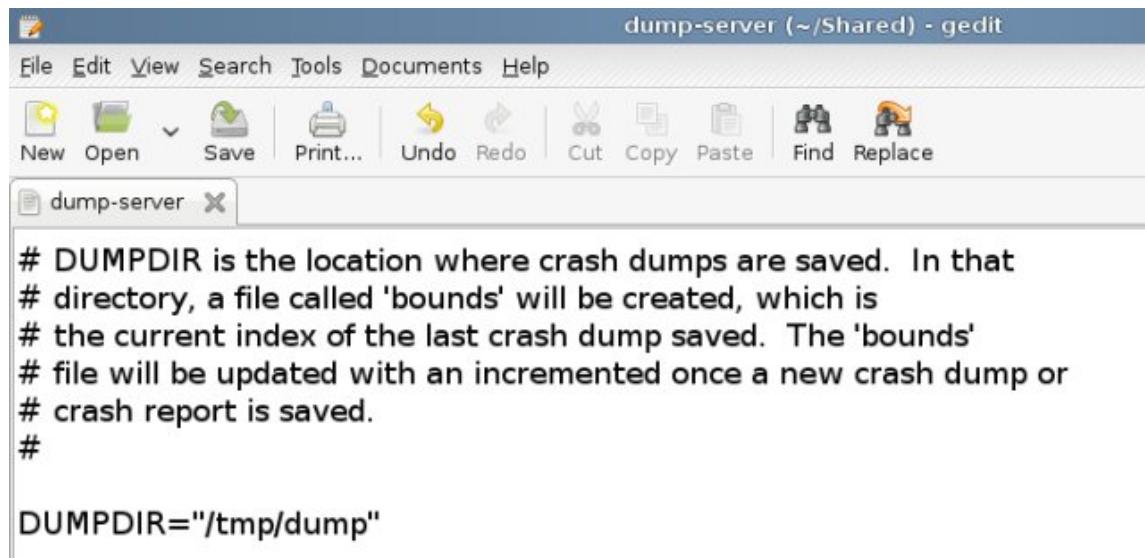
### 3.2.3 Configure the dump directory (DUMPDIR)

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`.

```
DUMPDIR="/tmp/dump"
```

**Figure 2:** LKCD DUMPDIR directive change



The screenshot shows a window titled "dump-server (~/Shared) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print..., Undo, Redo, Cut, Copy, Paste, Find, and Replace. The main text area contains the following configuration file content:

```
# DUMPDIR is the location where crash dumps are saved. In that
# directory, a file called 'bounds' will be created, which is
# the current index of the last crash dump saved. The 'bounds'
# file will be updated with an incremented once a new crash dump or
# crash report is saved.
#
DUMPDIR="/tmp/dump"
```

### 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.

**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"
```

### 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.

### 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

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.

### 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:

```
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 demonstrate:

```
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
```

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 configurations. 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/vmcore` device. However, network dump requires an active network interface. There are other considerations 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
```

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 neighborhood<sup>2</sup>, 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.

---

<sup>2</sup> 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.

```
SOURCE_PORT="6688"
```

**Figure 3:** LKCD netdump client source port configuration



The screenshot shows a window titled "dump-server (~/Shared) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print..., Undo, Redo, Cut, Copy, Paste, Find, and Replace. A tab labeled "dump-server" is open. The text in the editor is as follows:

```
# SOURCE_PORT is the port used on dumping machine to send dump data over
# the network. Default is 6688.

SOURCE_PORT=6688
```

## 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
```

### 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

```
kernel: netdump: local IP [REDACTED]
kernel: dump_netdev: source IP [REDACTED]
kernel: dump_netdev: target IP [REDACTED]
kernel: dump_netdev: target MAC [REDACTED]
kernel:
kernel: Network device eth0 successfully configured for
```

## 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

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

```
drwx----- 2 netdump root      72 2009-02-25 16:00 .
drwxrwxrwx  3 netdump dump     96 2009-02-25 15:57 ..
-rw-----  1 netdump root 645693440 2009-02-25 16:00 vmcore
```

## 8 Problems

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

## 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

```
kernel: SysRq : Starting crash dump
kernel: Reconfiguring memory bank information...
kernel: This may take a while....
kernel: done waiting: 0 cpus not responding
kernel: Dumping to network device netdump on CPU ? ...
kernel: network dump failed due to handshake failure
kernel: dump_dev_write failed !err -1
kernel: dump write header failed !err -1
kernel: dump update header failed ! error -1
kernel: NETDUMP END!
kernel: Dump Incomplete or failed!
```

## 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

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.

## 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 it 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.

**Note:** This book part refers to a setup on SUSE<sup>3</sup> 10.3 systems. The Appendix section contains additional information about SUSE 11.X and RedHat<sup>4</sup> 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 parameters 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:

---

<sup>3</sup> SUSE refers to both openSUSE and SUSE Linux Enterprise Server (SLES)

<sup>4</sup> RedHat refers to both CentOS and RedHat Enterprise Linux (RHEL)

- Standard (production) kernel - kernel we normally work with
- Crash (capture) kernel - kernel specially used for collecting crash dumps<sup>5</sup>

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.

---

<sup>5</sup> 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.

- 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.

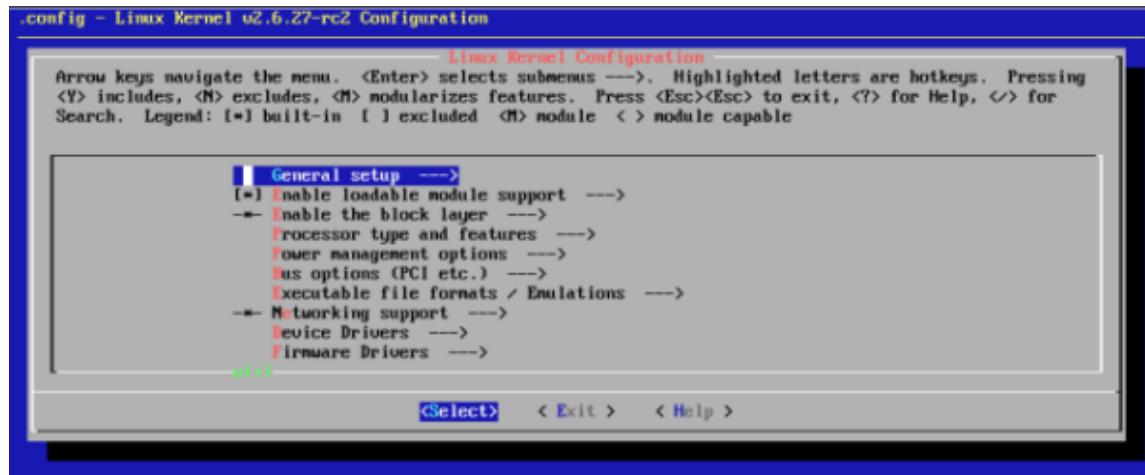
In this book, we will not go into details on kernel compilation. The compilation is a generic procedure that does not directly relate to Kdump and demands dedicated attention. We will talk about kernel compilation in a separate tutorial on [www.dedoimedo.com](http://www.dedoimedo.com). Here, we will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

**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](#) 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
```

**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 configure 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.

```
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
```

#### 11.1.4 Other settings

**Configure the start section for reserved RAM for the crash kernel<sup>6</sup>:** 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 16MB<sup>7</sup>, 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

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

<sup>7</sup> 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.

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) kernels<sup>8</sup>. Therefore, 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 *vmlinuz*, or rather *vmlinuz-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-debuginfo <sup>9</sup>	Crash analysis package (optional)

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

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/kdump <sup>10</sup>	Kdump service
/etc/sysconfig/kdump <sup>11</sup>	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

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

<sup>11</sup> The startup script has changed on the recent versions of SUSE systems.

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

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!

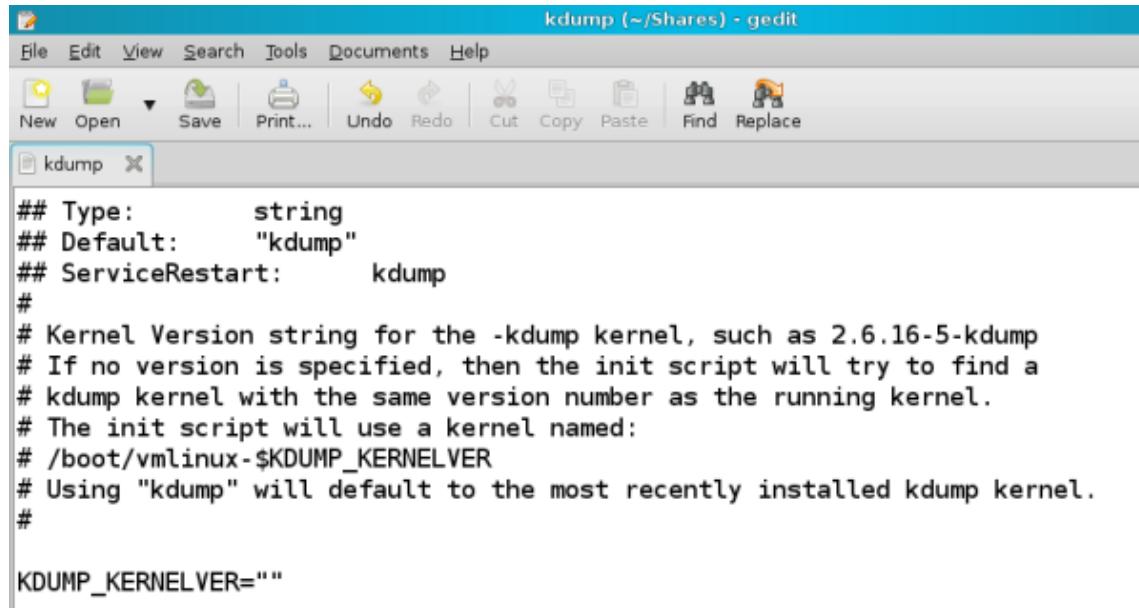
#### 13.1.1 Configure KDUMP\_KERNELVER

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.

```
KDUMP_KERNELVER=""
```

**Figure 8:** Kdump kernel version configuration

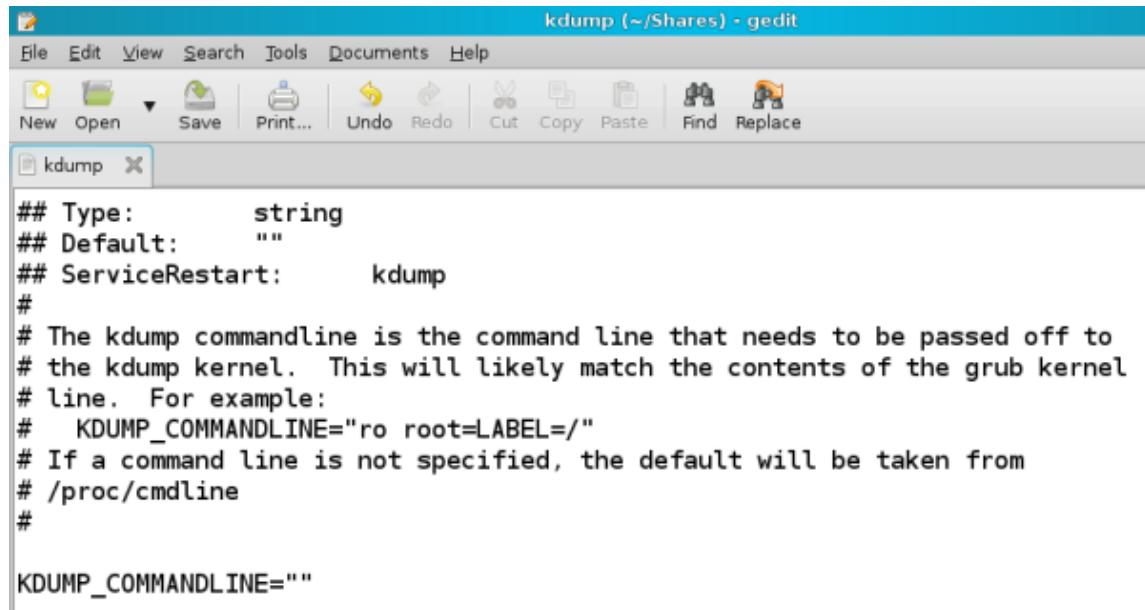


```
kdump (~/Shares) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
kdump x
## Type:      string
## Default:   "kdump"
## ServiceRestart:    kdump
#
# Kernel Version string for the -kdump kernel, such as 2.6.16-5-kdump
# If no version is specified, then the init script will try to find a
# kdump kernel with the same version number as the running kernel.
# The init script will use a kernel named:
# /boot/vmlinux-$KDUMP_KERNELVER
# Using "kdump" will default to the most recently installed kdump kernel.
#
KDUMP_KERNELVER=""
```

### 13.1.2 Configure KDUMP\_COMMANDLINE

This setting 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=""
```

**Figure 9:** Kdump command line configuration

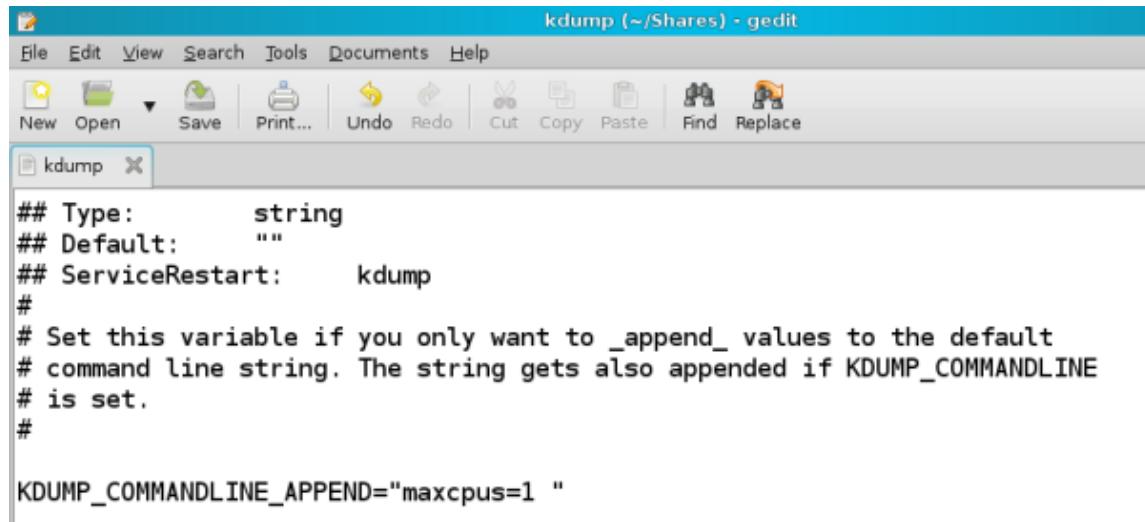
The screenshot shows a Gedit text editor window titled "kdump (~/Shares) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print..., Undo, Redo, Cut, Copy, Paste, Find, and Replace. The main text area displays the following configuration code:

```
## Type:      string
## Default:   ""
## ServiceRestart:    kdump
#
# The kdump commandline is the command line that needs to be passed off to
# the kdump kernel. This will likely match the contents of the grub kernel
# line. For example:
#   KDUMP_COMMANDLINE="ro root=LABEL=/"
# If a command line is not specified, the default will be taken from
# /proc/cmdline
#
KDUMP_COMMANDLINE=""
```

### 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 "

**Figure 10:** Kdump command line append configuration

The screenshot shows a Gedit text editor window titled "kdump (~/Shares) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print..., Undo, Redo, Cut, Copy, Paste, Find, and Replace. The main text area contains the following configuration code:

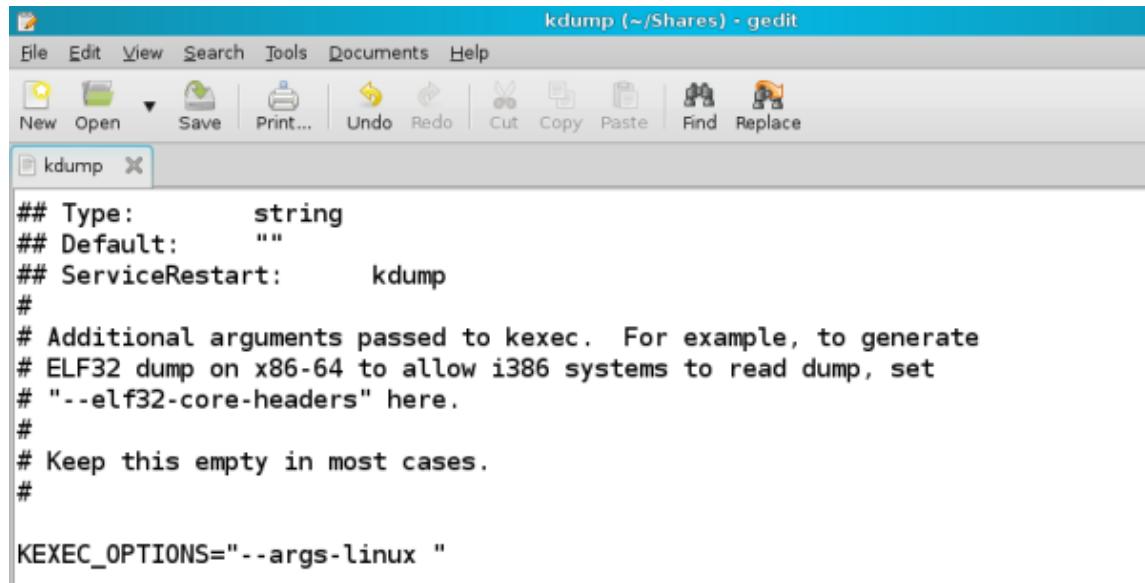
```
## Type:      string
## Default:   ""
## ServiceRestart:    kdump
#
# Set this variable if you only want to _append_ values to the default
# command line string. The string gets also appended if KDUMP_COMMANDLINE
# is set.
#
KDUMP_COMMANDLINE_APPEND="maxcpus=1 "
```

### 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 "

**Figure 11:** Kdump options configurations

The screenshot shows a window titled "kdump (~/Shares) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print..., Undo, Redo, Cut, Copy, Paste, Find, and Replace. The main text area contains the following configuration code:

```
## Type:      string
## Default:   ""
## ServiceRestart:    kdump
#
# Additional arguments passed to kexec. For example, to generate
# ELF32 dump on x86-64 to allow i386 systems to read dump, set
# "--elf32-core-headers" here.
#
# Keep this empty in most cases.
#
KEXEC_OPTIONS="--args-linux "
```

### 13.1.5 Configure KDUMP\_RUNLEVEL

This is another important directive. It 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

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"
```

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 it 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 setting 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

### 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.



**Figure 12:** Kdump DUMPDEV configuration

```
*kdump (~/Shares) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
* *kdump X
## Type:      string
## Default:   ""
#
#
# Specifies the dump device that is used for saving the dump with the kdump
# kernel. The dump device normally is a disk partition. You don't need to
# specify a dump device here. Then the dump is written to KDUMP_SAVEDIR when
# booting from the kdump kernel.
#
# If KDUMP_DUMPDEV points to a device file, the dump is written to that device
# when running the kdump kernel. The advantage over writing the dump to
# disk immediately is that you don't have to mount the root file system (which
# may be corrupted!) just to write the dump. So if the root file system is
# corrupted, you have the chance to fix the file system manually and reboot the
# system without loosing the dump information. On the first normal boot which
# is able to successfully mount the root file system, the dump is saved to
# KDUMP_SAVEDIR as usual.
#
# Important: The KDUMP_DUMPDEV is overwritten by kdump, so don't use it for
# saving any data. Also don't use the currently used swap partition.
#
KDUMP_DUMPDEV=""
```

### 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:

```
KDUMP_VERBOSE=3
```

**Table 6:** Kdump verbosity configuration

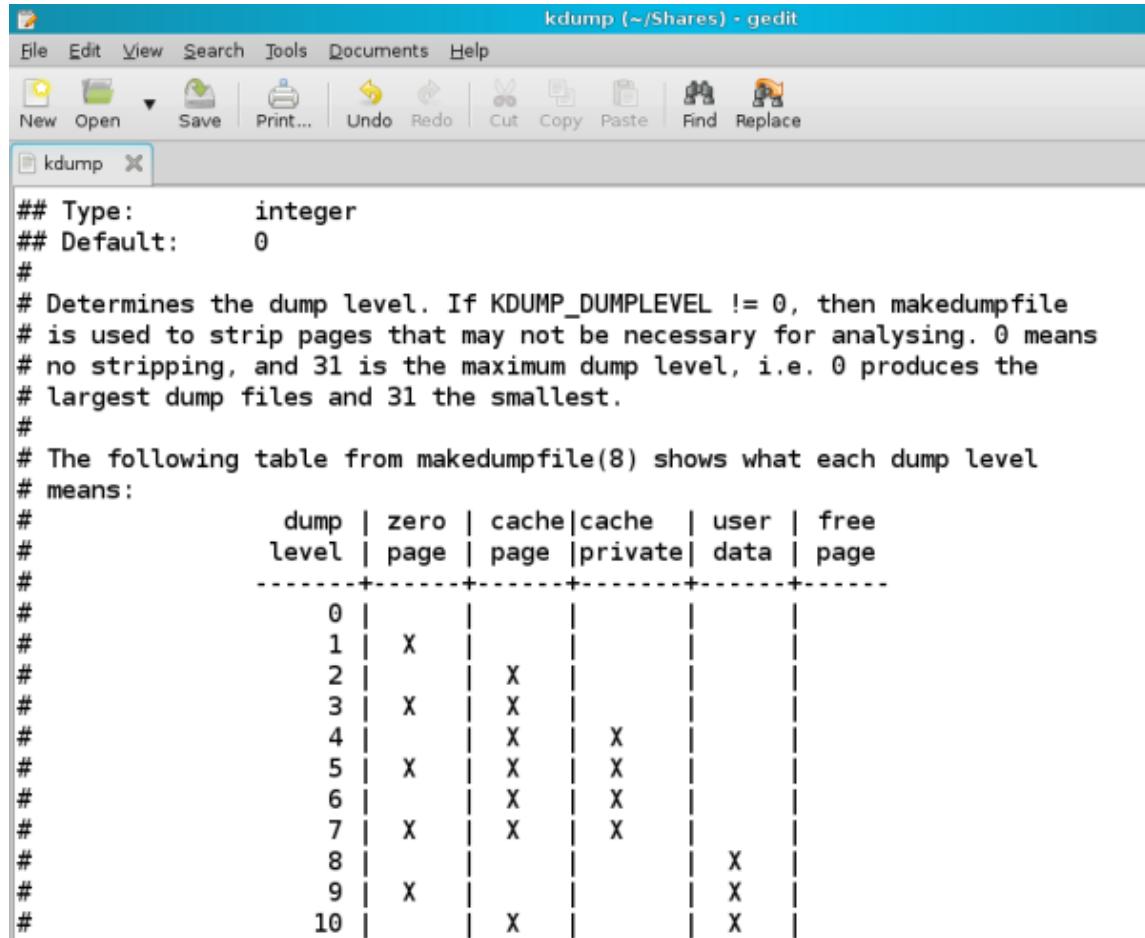
Value	Action
1	Kdump command line written to syslog
2	Kdump progress written to STDOUT
4	Kdump command line written to STDOUT
8	Kdump transfer script debugged

### 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.

**Figure 13:** Kdump DUMPLLEVEL configuration


```

## Type:      integer
## Default:   0
#
# Determines the dump level. If KDUMP_DUMPLLEVEL != 0, then makedumpfile
# is used to strip pages that may not be necessary for analysing. 0 means
# no stripping, and 31 is the maximum dump level, i.e. 0 produces the
# largest dump files and 31 the smallest.
#
# The following table from makedumpfile(8) shows what each dump level
# means:
#
#      dump | zero | cache|cache | user | free
#      level | page | page |private| data | page
#      -----+-----+-----+-----+-----+
#      0   |       |       |       |       |       |
#      1   |   X  |       |       |       |       |
#      2   |       |   X  |       |       |       |
#      3   |   X  |   X  |       |       |       |
#      4   |       |   X  |       |   X  |       |
#      5   |   X  |   X  |       |   X  |       |
#      6   |       |   X  |       |   X  |       |
#      7   |   X  |   X  |       |   X  |       |
#      8   |       |       |       |       |   X  |
#      9   |   X  |       |       |       |   X  |
#     10  |       |   X  |       |       |   X  |

```

### 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\_DUMPFORMAT="ELF"**

## 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 memory 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 reservation. 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 file<sup>12</sup>. 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:

<sup>12</sup>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.

```
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](#) tutorial online.

For example, using the *chkconfig* utility <sup>13</sup>:

```
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:

---

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

```
/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 parameters<sup>14</sup>:

---

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

```
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.

## 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:

```
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](#) 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.

## 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

```
Setting flag_elf64 to true  
[ 28 %]
```

Indeed, checking the destination directory, you should see the *vmcore* file<sup>15</sup>.

<sup>15</sup>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.

**Figure 15:** Contents of dumped memory core directory

```
drwxr-xr-x  2 root root    4096 2009-03-05 15:17 .
drwxr-x---  3 root root    4096 2009-03-05 15:17 ..
-rw-----  1 root root 260005768 2009-03-05 15:18 vmcore
```

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 configuration 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 addition, 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
```

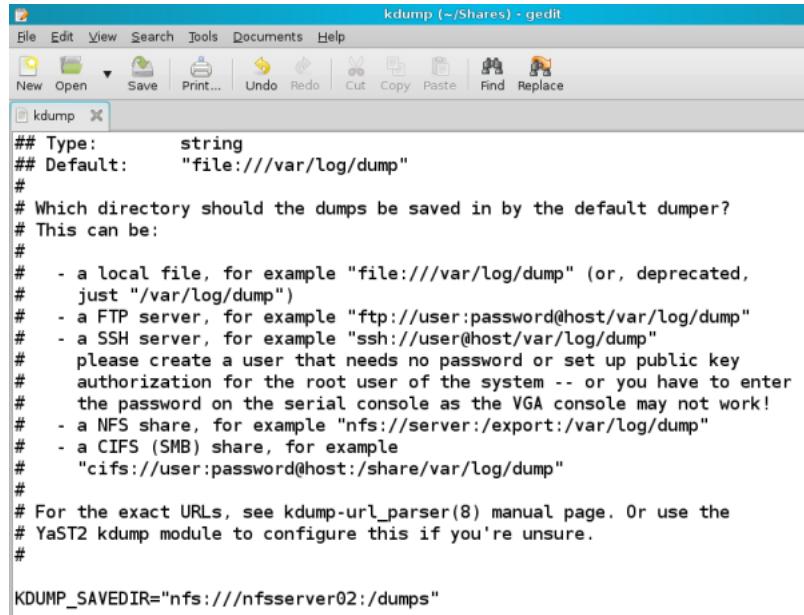
### 16.1.2 Configure KDUMP\_SAVEDIR

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"
```

**Figure 16:** Kdump SAVEDIR network configuration


```

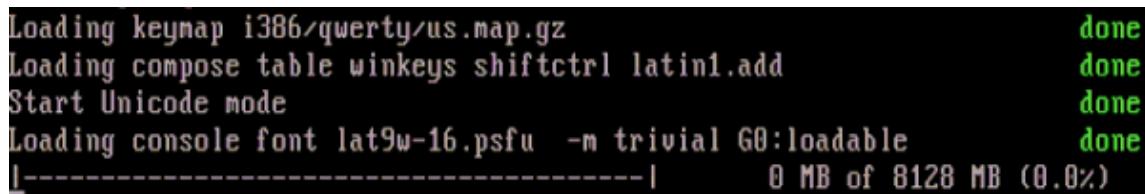
## Type:      string
## Default:   "file:///var/log/dump"
#
# Which directory should the dumps be saved in by the default dumper?
# This can be:
#
# - a local file, for example "file:///var/log/dump" (or, deprecated,
#   just "/var/log/dump")
# - a FTP server, for example "ftp://user:password@host/var/log/dump"
# - a SSH server, for example "ssh://user@host/var/log/dump"
#   please create a user that needs no password or set up public key
#   authorization for the root user of the system -- or you have to enter
#   the password on the serial console as the VGA console may not work!
# - a NFS share, for example "nfs://server:/export:/var/log/dump"
# - a CIFS (SMB) share, for example
#   "cifs://user:password@host::share/var/log/dump"
#
# For the exact URLs, see kdump-url_parser(8) manual page. Or use the
# YaST2 kdump module to configure this if you're unsure.
#
KDUMP_SAVEDIR="nfs:///nfsserver02:/dumps"

```

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 see 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


```

Loading keymap i386/qwerty/us.map.gz          done
Loading compose table winkeys shiftctrl latin1.add  done
Start Unicode mode                            done
Loading console font lat9w-16.psfu -m trivial G0:loadable  done
|-----| 0 MB of 8128 MB (0.0%)

```

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

```
Start Unicode mode                                         done
Loading console font lat9w-16.psfu -m trivial G0:loadable   done
| #####| 8128 MB of 8128 MB (100.0%)                      done
INIT: Switching to runlevel: 6
INIT: Sending processes the TERM signal
Master Resource Control: runlevel 3 has been reached
```

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.

## 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 simple and straightforward. You can use either yum, zypper or apt to obtain the package very easily.

**Figure 19:** Installation of crash via software manager

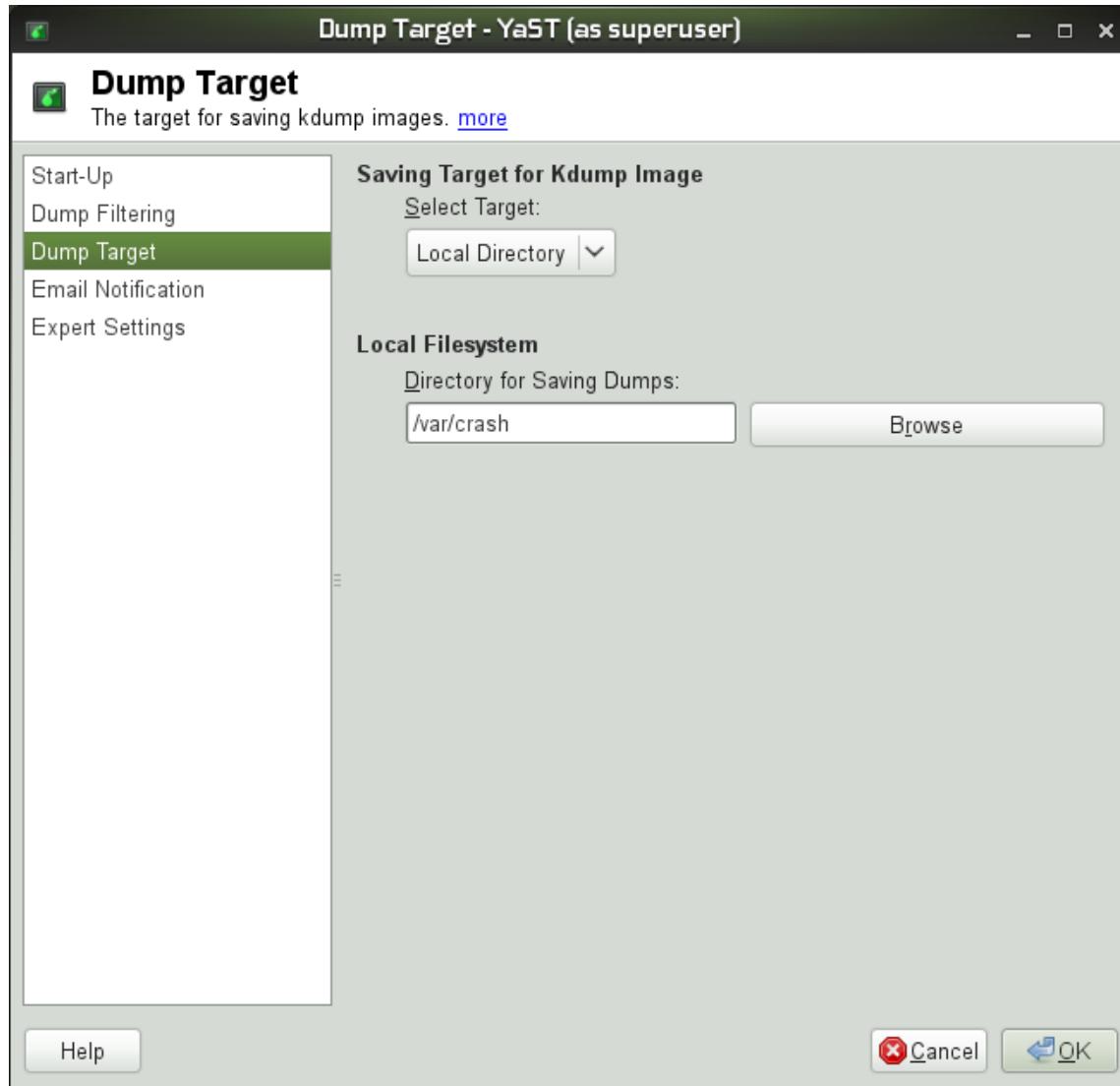
### 18.3 Crash location

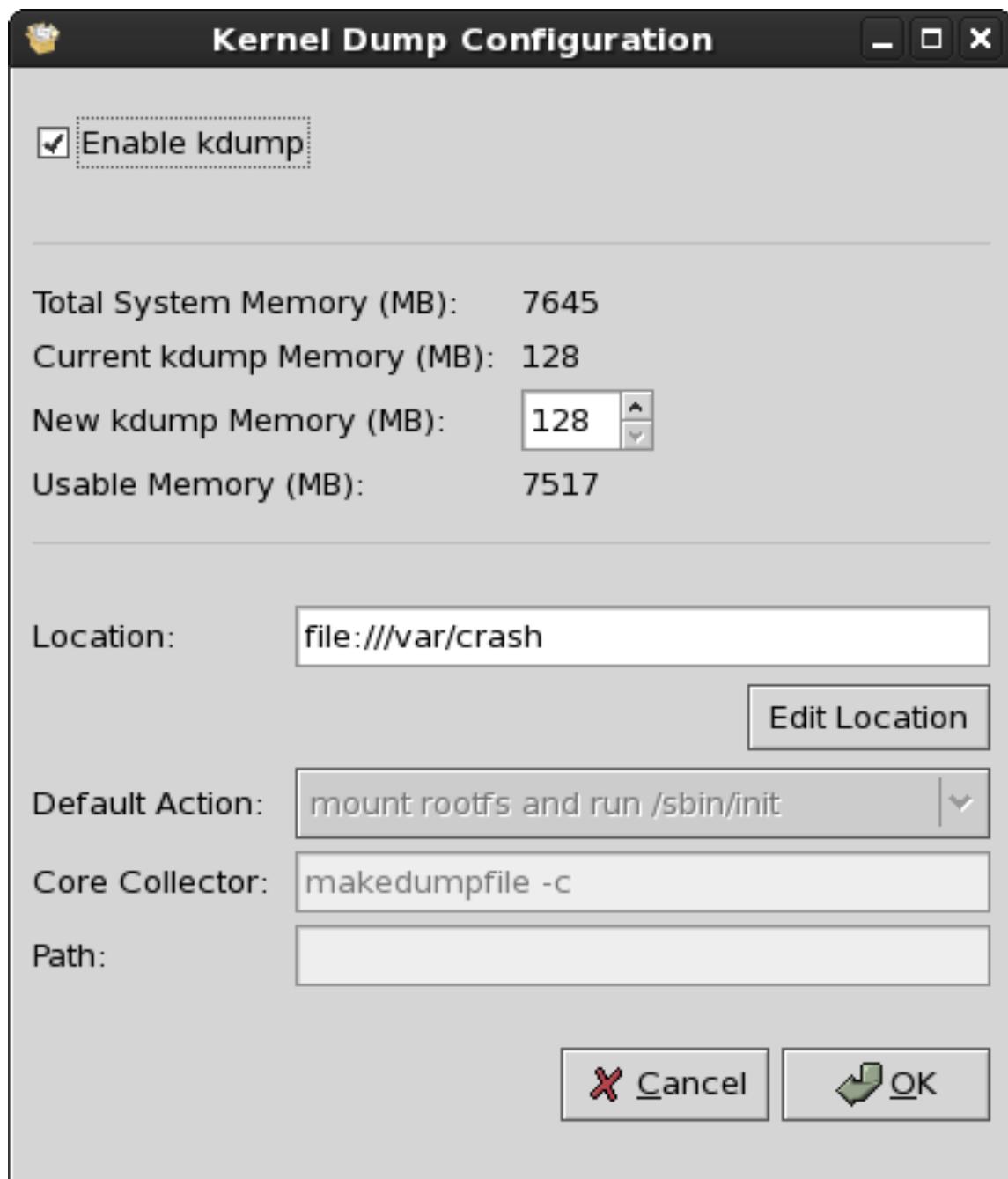
The default crash directory is `/var/crash`. You can change the path to anything you want, provided there's enough space on the target device. In general, you should choose

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 configuration via YaST-Kdump module



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

## 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

```
Nothing to delete in /var/crash.  
Saving dump                                Finished.  
Generating README                            Finished.  
Copying System.map                          Finished.  
Copying kernel                             Finished.
```

**Figure 23:** Contents of a crash dump directory



The screenshot shows a terminal window with the title bar "admin@testhost1:~". The menu bar includes "File", "Edit", "View", "Terminal", and "Help". The command "testhost1:/var/crash/2010-01-22-19:46 # ls -l" is run, displaying the following file listing:

```
total 700736  
-rw-r--r-- 1 root root      183 2010-01-22 19:46 README.txt  
-rw-r--r-- 1 root root  1512503 2010-01-22 19:46 System.map-2.6.31.8-0.1-desktop  
-rw-r--r-- 1 root root 737740284 2010-01-22 19:46 vmcore  
-rw-r--r-- 1 root root  5187339 2010-01-22 19:46 vmlinuz-2.6.31.8-0.1-desktop.gz
```

## 19 Invoke crash

The crash utility can be invoked in several ways. First, there is some difference between older and newer versions of Kdump<sup>16</sup>, 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.

<sup>16</sup>The older version refers to SUSE 10.X systems. The newer versions refers to SUSE 11.x and RedHat 5.X systems.

## 19.1 Old (classic) invocation

The old invocation is done like this:

```
crash <System map> <vmlinuz> 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](#)
- [The Linux Kernel HOWTO - Systemmap](#)

<**vmlinuz**> 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 vmlinuz 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.

**Figure 24:** New kdump invocation console output

```
Nothing to delete in /var/crash.  
Saving dump                                Finished.  
Generating README                            Finished.  
Copying System.map                          Finished.  
Copying kernel                             Finished.
```

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. Alternatively, 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
```

**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](#)
- [Kdump - openSUSE](#)

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](#)

### 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

	<b>SUSE</b>	<b>RedHat</b>
System map	System-map	System.map
Debug info	/usr/lib/debug/boot/	/usr/lib/debug/lib/modules/

**Figure 27:** Crash debuginfo location on openSUSE 11.x

```
admin@testhost1:~$ ls -l
total 109644
-rw-r--r-- 1 root root 112272112 2009-10-27 13:17 vmlinu...-0.1-debug
admin@testhost1:~$
```

## 19.4 Portable use

To process cores on other machines, you can either copy the System map and the kernel 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

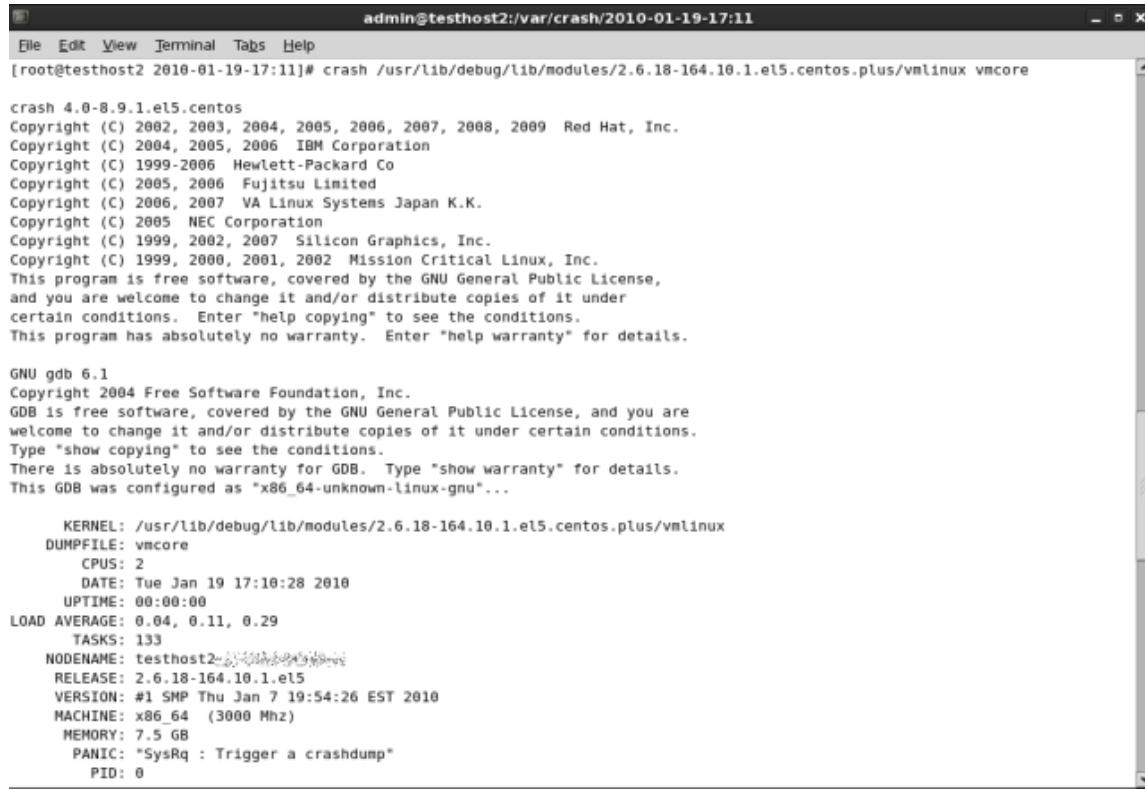
```
Copyright (C) 2004, 2005, 2006 IBM Corporation
Copyright (C) 1999-2006 Hewlett-Packard Co
Copyright (C) 2005, 2006 Fujitsu Limited
Copyright (C) 2006, 2007 VA Linux Systems Japan K.K.
Copyright (C) 2005 NEC Corporation
Copyright (C) 1999, 2002, 2007 Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 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.

crash: /usr/lib/debug/boot/vmlinu-2.6.31.8-0.1-desktop.debug:
      CRC value does not match

crash: /usr/lib/debug/boot//vmlinu-2.6.31.8-0.1-desktop.debug: CRC does not match
```

## 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.

**Figure 29:** Crash working


The screenshot shows a terminal window titled "admin@testhost2:/var/crash/2010-01-19-17:11". The command entered is "crash /usr/lib/debug/lib/modules/2.6.18-164.10.1.el5.centos.plus/vmlinux vmcore". The output displays the crash program's license information, which is a modified version of the GNU General Public License. It includes details about the kernel version (4.0-8.9.1.el5.centos), the GDB version (6.1), and the system configuration (CPU count: 2, memory: 7.5 GB, release: 2.6.18-164.10.1.el5). The output ends with a panic message: "PANIC: SysRq : Trigger a crashdump".

```

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.

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"...

KERNEL: /usr/lib/debug/lib/modules/2.6.18-164.10.1.el5.centos.plus/vmlinux
DUMPFILE: vmcore
CPUS: 2
DATE: Tue Jan 19 17:18:28 2010
UPTIME: 00:00:00
LOAD AVERAGE: 0.04, 0.11, 0.29
TASKS: 133
NODENAME: testhost2
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

```

**Figure 30:** Crash prompt

```

MEMORY: 7.5 GB
PANIC: "SysRq : Trigger a crashdump"
PID: 0
COMMAND: "swapper"
TASK: ffffff80300ae0 (1 of 2) [THREAD_INFO: ffffff803f2000]
CPU: 0
STATE: TASK_RUNNING
WARNING: panic task not found

crash> █

```

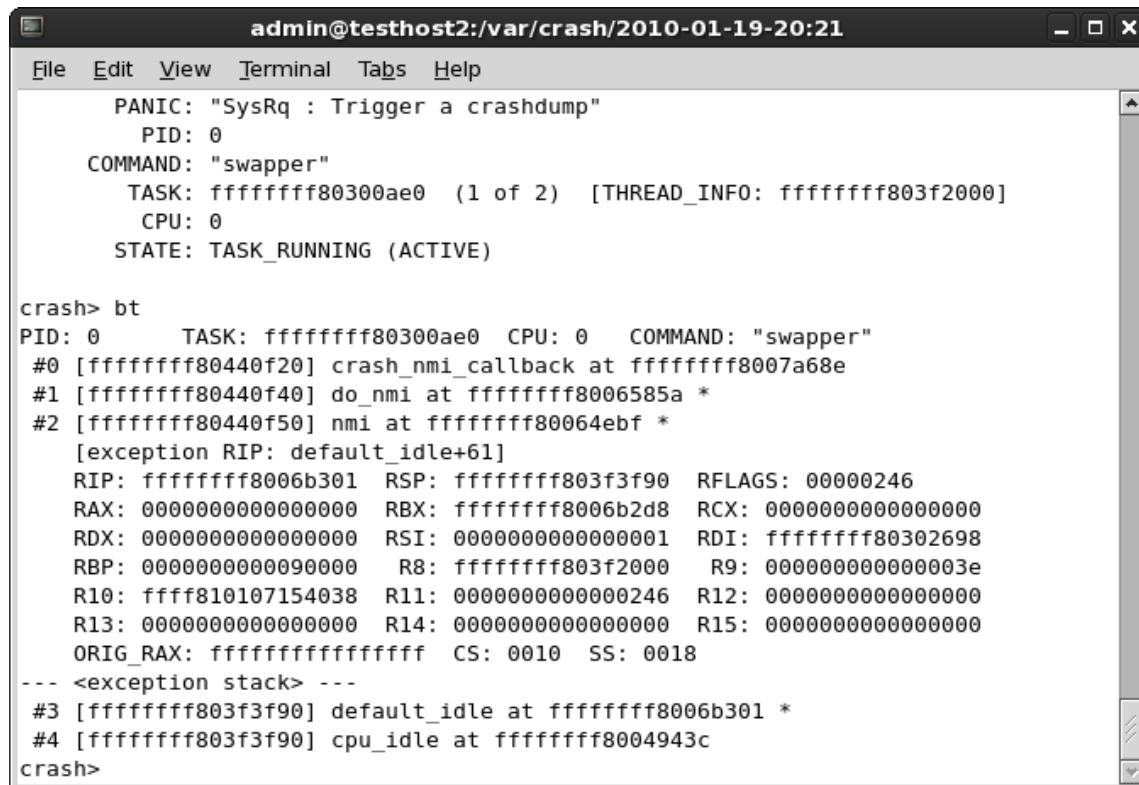
## 20.1 Crash commands

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](#). 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



```
PANIC: "SysRq : Trigger a crashdump"
PID: 0
COMMAND: "swapper"
TASK: ffffffff80300ae0 (1 of 2) [THREAD_INFO: ffffffff803f2000]
CPU: 0
STATE: TASK_RUNNING (ACTIVE)

crash> bt
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: ffffffff8006b301  CS: 0010  SS: 0018
--- <exception stack> ---
#3 [ffffffff803f3f90] default_idle at ffffffff8006b301 *
#4 [ffffffff803f3f90] cpu_idle at ffffffff8004943c
crash>
```

### 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

```

admin@testhost2:/var/crash/2010-01-19-17:11
File Edit View Terminal Tabs Help
Linux version 2.6.18-164.10.1.el5 (mockbuild@builder10.centos.org) (gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)) #1 SMP Thu Jan 7 19:54:26 EST 2010
Command line: ro root=LABEL=/ rhgb quiet crashkernel=128M@16M
BIOS-provided physical RAM map:
BIOS-e820: 00000000000010000 - 0000000000009f800 (usable)
BIOS-e820: 0000000000009f800 - 00000000000a0000 (reserved)
BIOS-e820: 00000000000ca000 - 00000000000cc000 (reserved)
BIOS-e820: 00000000000dc000 - 00000000000100000 (reserved)
BIOS-e820: 0000000000100000 - 00000000001effef0000 (usable)
BIOS-e820: 0000000000effef0000 - 0000000000effeff0000 (ACPI data)
BIOS-e820: 0000000000effff0000 - 0000000000eff00000 (ACPI NVS)
BIOS-e820: 0000000000effff0000 - 0000000000f000000 (usable)
BIOS-e820: 0000000000fec00000 - 0000000000fec10000 (reserved)
BIOS-e820: 0000000000fee00000 - 0000000000fee01000 (reserved)
BIOS-e820: 0000000000fffe0000 - 00000000010000000 (reserved)
BIOS-e820: 00000000010000000 - 0000000001fa000000 (usable)
DMI present.
ACPI: RSDP (v000 PTLTD ) @ 0x000000000000f6c68
ACPI: RSDT (v001 PTLTD RSDT 0x06040000 LTP 0x00000000) @ 0x000000000efefab5a
ACPI: FADT (v001 INTEL 440BX 0x060400000 PTL 0x000f4240) @ 0x000000000efefef06
ACPI: MADT (v001 PTLTD APIC 0x06040000 LTP 0x00000000) @ 0x000000000efefef7a
ACPI: BOOT (v001 PTLTD SSBFTBLS 0x06040000 LTP 0x00000001) @ 0x000000000efefef08
ACPI: DSDT (v001 PTLTD Custom 0x06040000 MSFT 0x0100000d) @ 0x0000000000000000
No NUMA configuration found
Faking a node at 0000000000000000-000000001fa00000
Bootmem setup node 0 0000000000000000-000000001fa000000
On node 0 totalpages: 1977281
    DMA zone: 2633 pages, LIFO batch:0
    DMA32 zone: 964648 pages, LIFO batch:31
    Normal zone: 1010000 pages, LIFO batch:31
ACPI: PM-Timer IO Port: 0x1008
ACPI: Local APIC address 0xfee00000
ACPI: LAPIC (acpi_id[0x00] lpic_id[0x00] enabled)
Processor #0 6:15 APIC version 17
ACPI: LAPIC (acpi_id[0x01] lpic_id[0x01] enabled)
Processor #1 6:15 APIC version 17
ACPI: LAPIC NMI (acpi id[0x00] high edge lint[0x1])
-- MORE -- forward: <SPACE>, <ENTER> or j backward: b or k quit: q

```

### 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.

**Figure 33:** crash ps command example

PID	PPID	CPU	TASK	ST	%MEM	VSZ	RSS	COMM
0	0	0	fffff80300ae0	RU	0.0	0	0	[swapper]
0	1	1	ffff81010710b0c0	RU	0.0	0	0	[swapper]
1	0	1	ffff810107eb7a0	IN	0.0	10348	636	init
2	1	0	ffff810107eb040	IN	0.0	0	0	[migration/0]
3	0	1	ffff810107e0ed7e0	IN	0.0	0	0	[ksoftirqd/0]
4	1	1	ffff810107eb0d80	IN	0.0	0	0	[migration/1]
5	1	1	ffff81010710b0820	IN	0.0	0	0	[ksoftirqd/1]
6	1	0	ffff81010f9cdd860	IN	0.0	0	0	[events/0]
7	1	1	ffff81010f9cdd100	IN	0.0	0	0	[events/1]
8	1	1	ffff81010f9cd7a0	IN	0.0	0	0	[khelper]
73	1	0	ffff81010f9db0040	IN	0.0	0	0	[kthread]
78	73	0	ffff81010f9d697a0	IN	0.0	0	0	[kblockd/0]
79	73	1	ffff81010f9d67100	IN	0.0	0	0	[kblockd/1]
80	73	1	ffff81010f9d67860	IN	0.0	0	0	[kacpid]
136	73	0	ffff81010f9d63080	IN	0.0	0	0	[cqueue/0]
137	73	1	ffff81010f9d637e0	IN	0.0	0	0	[cqueue/1]
140	73	1	ffff81010f9d5e100	IN	0.0	0	0	[khubd]
142	73	1	ffff81010f9d5c0c0	IN	0.0	0	0	[kseriod]
214	73	1	ffff81010f9b057e0	IN	0.0	0	0	[pdflush]
215	73	0	ffff81010f9b05080	IN	0.0	0	0	[pdflush]
216	73	1	ffff81010f9b04820	IN	0.0	0	0	[kswapd0]
217	73	0	ffff81010f9b040c0	IN	0.0	0	0	[aio/0]
218	73	1	ffff81010f9af9860	IN	0.0	0	0	[aio/1]
361	73	1	ffff81010f9effd7a0	IN	0.0	0	0	[kpsmoused]
407	73	1	ffff81010f9e87860	IN	0.0	0	0	[mpt_poll_0]
408	73	1	ffff81010f9efff7e0	IN	0.0	0	0	[scsi_eh_0]
412	73	0	ffff81010f9d7b0c0	IN	0.0	0	0	[ata/0]
413	73	1	ffff81010f9d7b820	IN	0.0	0	0	[ata/1]
414	73	1	ffff81010f9e87100	IN	0.0	0	0	[ata_aux]
424	73	1	ffff81010f970e860	IN	0.0	0	0	[kstriped]
437	73	0	ffff81010f9d5e860	IN	0.0	0	0	[kjournald]
463	73	1	ffff81010f9cd040	IN	0.0	0	0	[kauditfd]
496	1	1	ffff81010f9d247a0	IN	0.0	13032	1220	udevd
1329	73	0	ffff81010f7d7e7a0	IN	0.0	0	0	[kmpathd/0]
1330	73	1	ffff81010f80657a0	IN	0.0	0	0	[kmpathd/1]
1331	73	1	ffff81010f9efff080	IN	0.0	0	0	[kmpath handlerd]

-- MORE -- forward: <SPACE>, <ENTER> or j backward: b or k quit: q

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*

utility, which specifically requires `-w` flag to write to files<sup>17</sup>.

## 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:

---

<sup>17</sup>See Appendix (V) for more details.

```
crash <debuginfo> vmcore < inputfile > outputfile
```

Or perhaps:

```
crash <System map> <vmlinuz> 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

```
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"...(no debugging symbols found)...

crash: /boot/vmlinuz-2.6.18-164.10.1.el5: no debugging data available

[root@testhost2 2010-01-19-17:11]#
```

**Figure 35:** No debuginfo package on openSUSE

```

crash 4.1.0
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.

crash: vmlinuz-2.6.31.8-0.1-desktop.gz: no debugging data available
crash: vmlinuz-2.6.31.8-0.1-desktop.debug: debuginfo file not found

crash: either install the appropriate kernel debuginfo package, or
      copy vmlinuz-2.6.31.8-0.1-desktop.debug to this machine

```

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 correctly<sup>18</sup>.

**Figure 36:** Installing crash debug packages on CentOS 5.4

```

admin@testhost2:/home/admin
File Edit View Terminal Tabs Help
[root@testhost2 admin]# yum search kernel-debug
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * addons: anorien.csc.warwick.ac.uk
 * base: anorien.csc.warwick.ac.uk
 * extras: anorien.csc.warwick.ac.uk
 * updates: anorien.csc.warwick.ac.uk
=====
Matched: kernel-debug =====
kernel-debug.x86_64 : The Linux kernel compiled with extra debugging enabled.
kernel-debug-devel.x86_64 : Development package for building kernel modules to match
                           : the kernel.
[root@testhost2 admin]#

```

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

## 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

```
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"...

crash: /usr/lib/debug/lib/modules/2.6.18-164.10.1.el5debug/vmlinux and vmcore do not match!

Usage:
crash [-h [opt]][-v][-s][-i file][-d num] [-S] [mapfile] [namelist] [dumpfile]

Enter "crash -h" for details.
[root@testhost2 2010-01-19-17:11]# █
```

On SUSE, it may look like this:

**Figure 38:** CRC match error on openSUSE

```
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.

crash: /usr/lib/debug/boot/vmlinux-2.6.31.8-0.1-desktop.debug:
      CRC value does not match

crash: /usr/lib/debug/boot//vmlinux-2.6.31.8-0.1-desktop.debug: CRC does not match
```

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!

## 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

```
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
WARNING: panic task not 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.

## 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 investigation<sup>19</sup>. 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

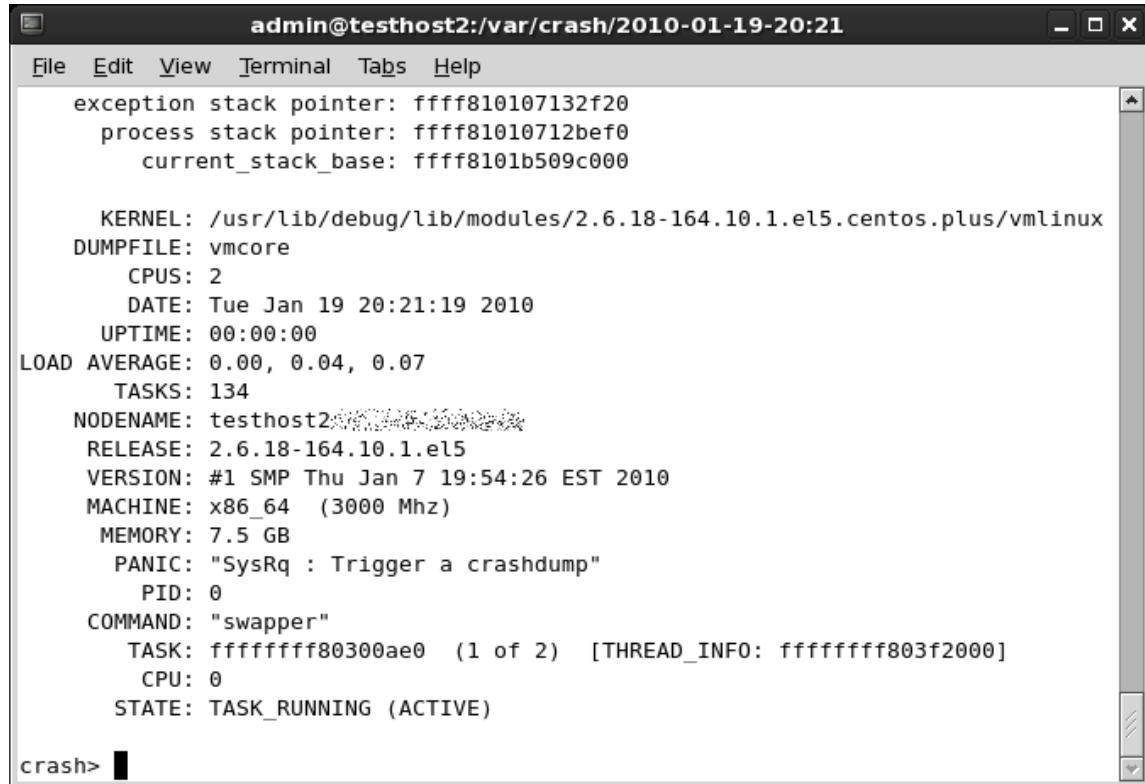
**Note:** Operating systems used to demonstrate the functionality are openSUSE 11.X and CentOS 5.X.

## 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.

---

<sup>19</sup>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.

**Figure 40:** Beginning crash analysis

The screenshot shows a terminal window titled "admin@testhost2:/var/crash/2010-01-19-20:21". The window contains the following text:

```
exception stack pointer: ffff810107132f20
process stack pointer: ffff81010712bef0
current_stack_base: ffff8101b509c000

KERNEL: /usr/lib/debug/lib/modules/2.6.18-164.10.1.el5.centos.plus/vmlinu
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
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)

crash>
```

```
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
```

```
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: ffffff80300ae0 (1 of 2)
      [THREAD_INFO: ffffff803f2000]
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 utility mailing list [thread](#) about changes in the crash utility 4.0-8.11 release, we learn the following information:

*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 x86\_64\_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:

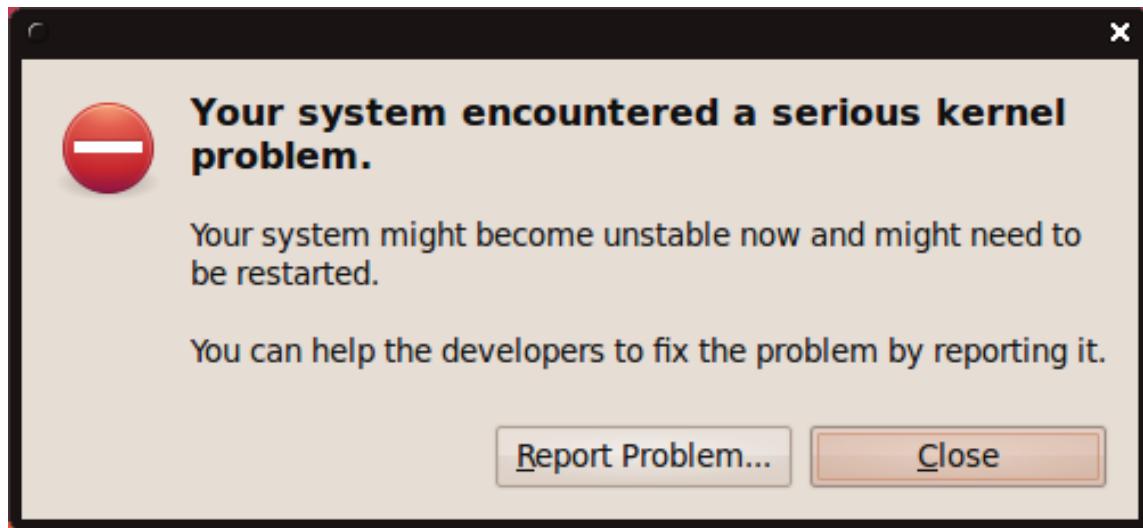
**PANIC:** specifies what kind of crash occurred on the machine. There are several types that you can see.

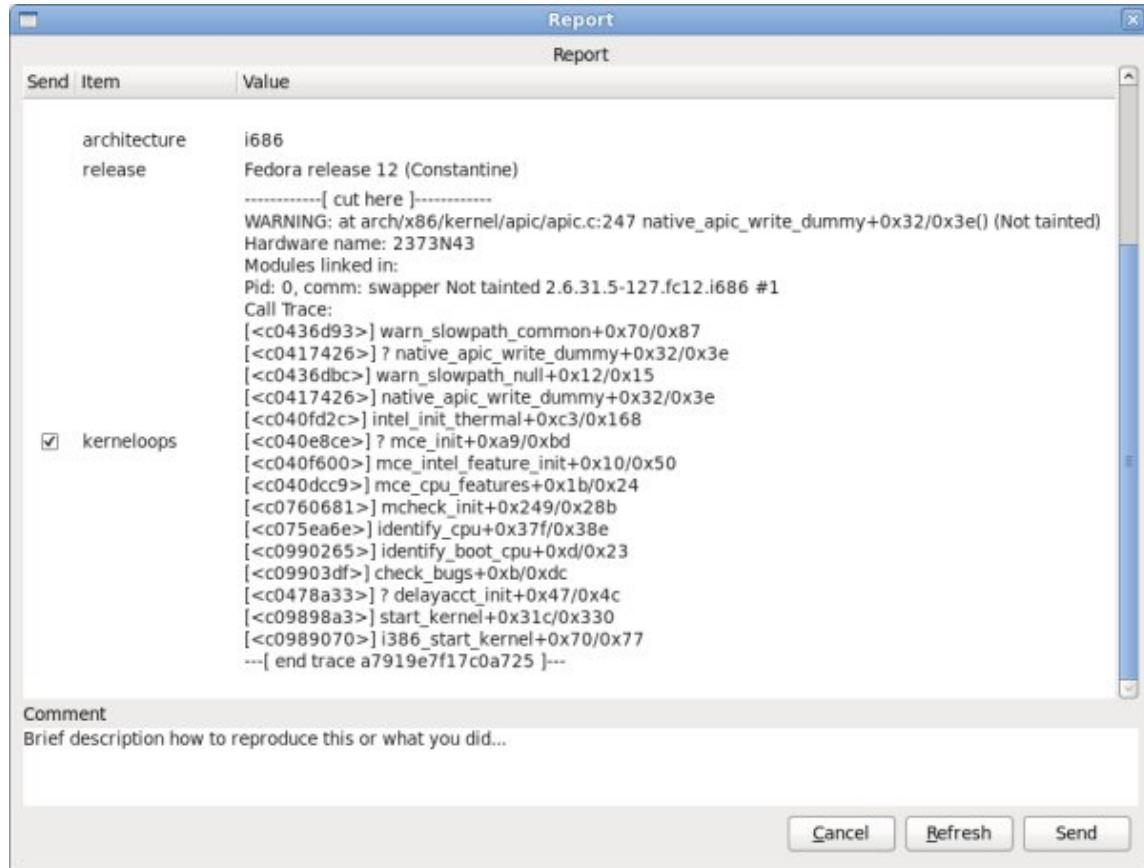
**SysRq** (System Request) refers to Magic Keys, which allow you to send instructions directly to the kernel. They can be invoked using a keyboard sequence or by echoing letter commands to `/proc/sysrq-trigger`, provided the functionality is enabled. We have discussed this in the Kdump part.

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](#) and Fedora [Constantine](#) 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



**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"

**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.

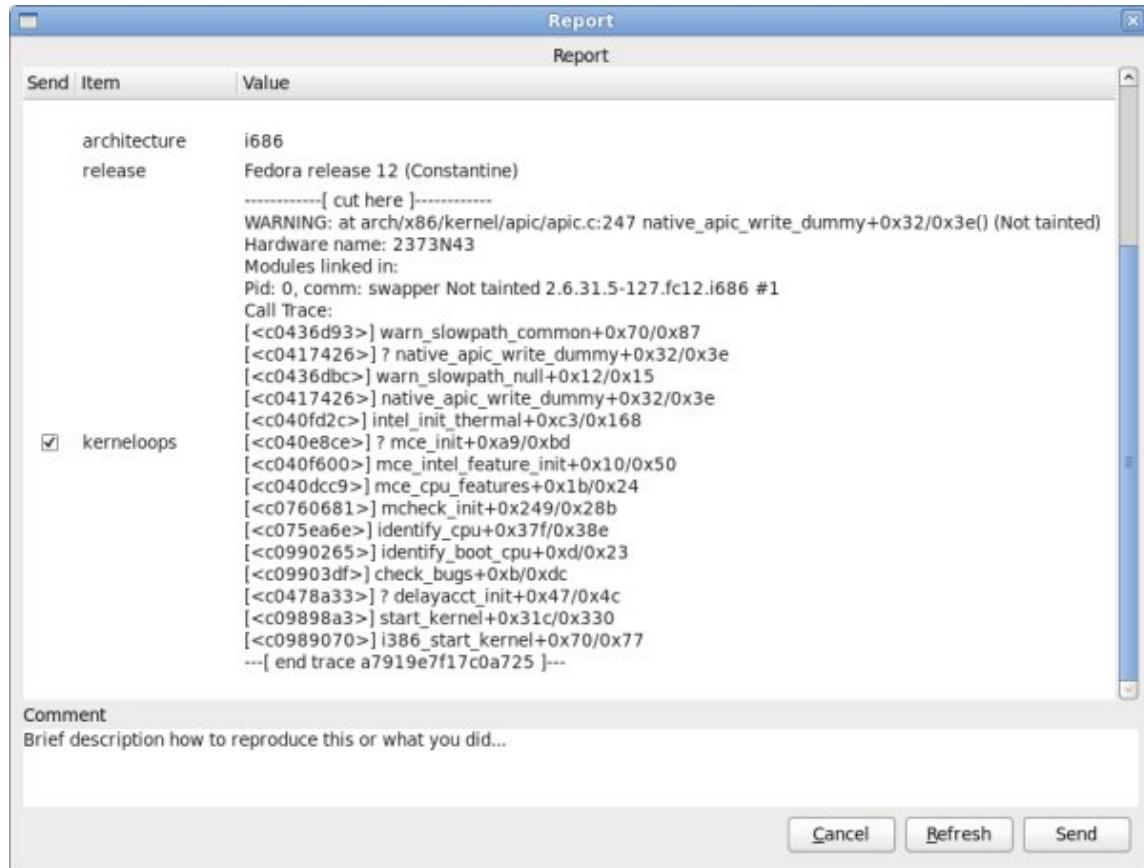
## 24 Getting warmer

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.

## 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.

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 Understanding Linux Kernel, Chapter 9: Process Address Space, Page Fault Exception Handler, pages 376-382, we learn the following information:

- 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 access <sup>20</sup>
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

<sup>20</sup>Sometimes, invalid access is also referred to as protection fault.

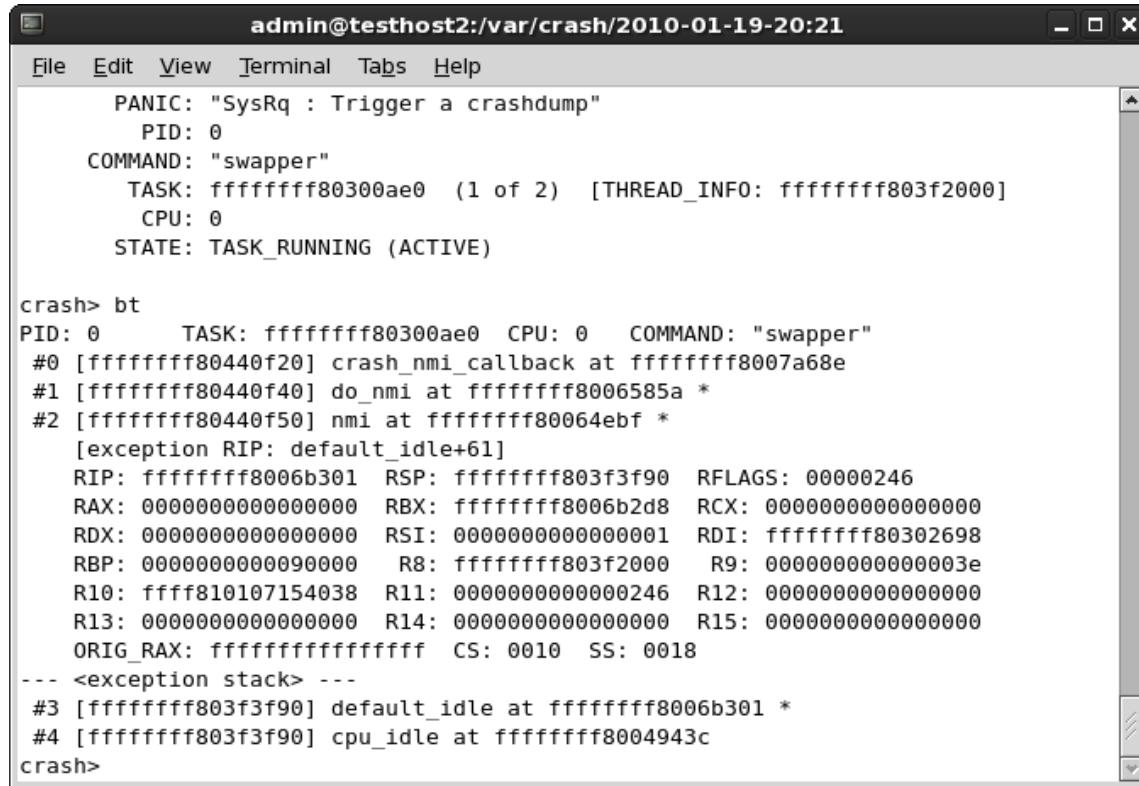
This is quite interesting. Seemingly incomprehensible information starts to feel very logical indeed. Therefore, we have a page not found during a write operation in Kernel mode; the fault was not an Instruction Fetch. Of course, it's a little more complicated than that, but still we're getting a very 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.

**Figure 44:** Backtrace of a crash dump

The screenshot shows a terminal window titled "admin@testhost2:/var/crash/2010-01-19-20:21". The window contains the following text:

```
PANIC: "SysRq : Trigger a crashdump"
PID: 0
COMMAND: "swapper"
TASK: ffffffff80300ae0 (1 of 2) [THREAD_INFO: ffffffff803f2000]
CPU: 0
STATE: TASK_RUNNING (ACTIVE)

crash> bt
PID: 0      TASK: ffffffff80300ae0 CPU: 0      COMMAND: "swapper"
#0 [fffffff80440f20] crash_nmi_callback at ffffffff8007a68e
#1 [fffffff80440f40] do_nmi at ffffffff8006585a *
#2 [fffffff80440f50] nmi at ffffffff80064ebf *
[exception RIP: default_idle+61]
RIP: ffffffff8006b301 RSP: ffffffff803f3f90 RFLAGS: 000000246
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: ffffffff8006b301 CS: 0010 SS: 0018
--- <exception stack> ---
#3 [fffffff803f3f90] default_idle at ffffffff8006b301 *
#4 [fffffff803f3f90] cpu_idle at ffffffff8004943c
crash>
```

```
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:  00000000000003e
    R10: ffff810107154038 R11:  0000000000000246
    R12: 0000000000000000
    R13: 0000000000000000 R14:  0000000000000000
    R15: 0000000000000000
    ORIG_RAX: ffffffffffffff CS: 0010 SS: 0018
--- <exception stack> ---
#3 [fffffff803f3f90] default_idle at ffffffff8006b301 *
#4 [fffffff803f3f90] 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.

```
#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: ffffffffffffff CS: 0010 SS: 0018
--- <exception stack> ---
#3 [fffffff803f3f90] default_idle at ffffffff8006b301 *
#4 [fffffff803f3f90] 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 pointer<sup>21</sup>. 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:

<sup>21</sup>On 32-bit architecture, the instruction pointer is called EIP.

```
[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 instructions 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.

### 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. Mathematically, 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 Segment 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:

**Figure 45:** Example of a kernel crash with CPL 3

```
#5 [ffff8101d0de5f10] vfs_write at ffffffff80186656
#6 [ffff8101d0de5f40] sys_write at ffffffff80186c1f
#7 [ffff8101d0de5f80] system_call at ffffffff8010adba
    RIP: 00002aaaab127450  RSP: 00007ffffffffd350  RFLAGS: 00010202
    RAX: 0000000000000001  RBX: ffffffff8010adba  RCX: 00000000005bec01
    RDX: 0000000000000002  RSI: 00002aaaab15000  RDI: 0000000000000001
    RBP: 0000000000000002  R8: 00000000ffffffff  R9: 00002aaaab2b4ae0
    R10: 0000000000000000  R11: 0000000000000246  R12: 00002aaaab2ae7a0
    R13: 00002aaaab15000  R14: 0000000000000002  R15: 0000000000000000
    ORIG_RAX: 0000000000000001  CS: 0033  SS: 002b
crash> █
```

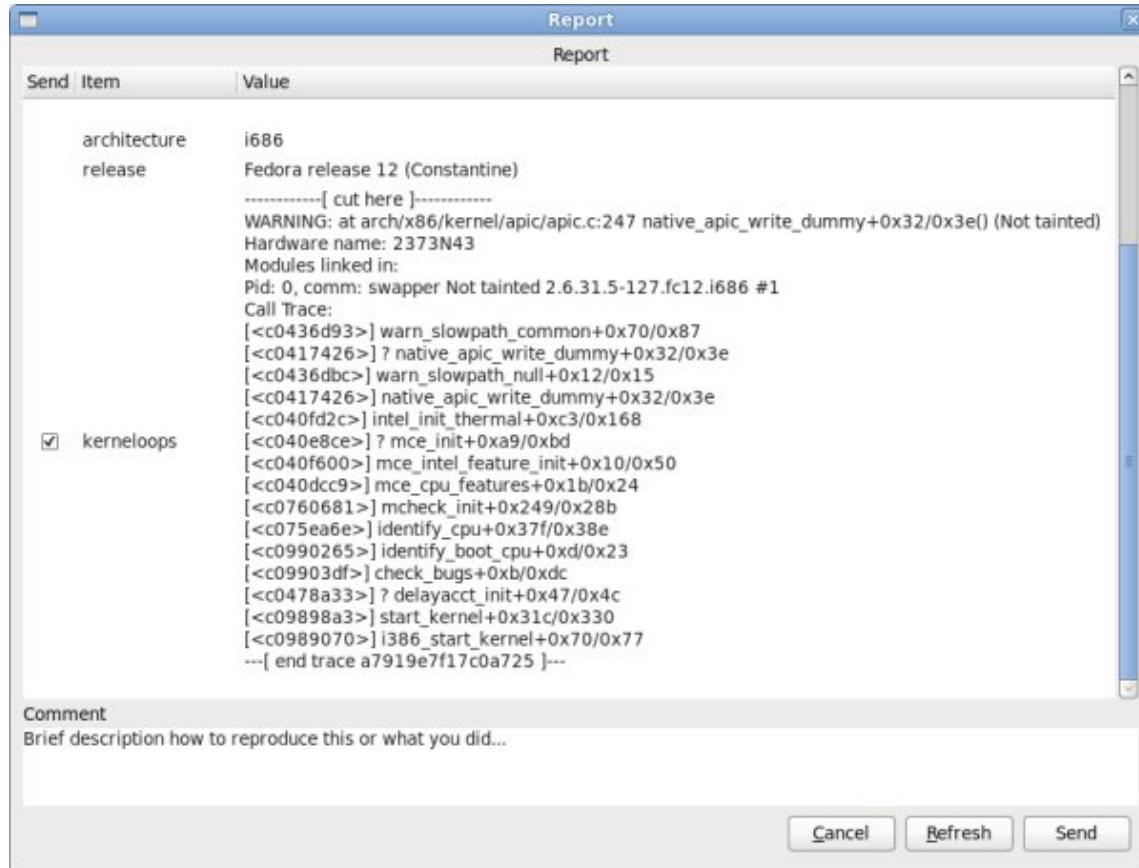
But that's just geeky talk. Back to our example, we have learned many useful, important details. We know the exact memory address where the instruction pointer was at the time of the crash. We know the privilege level.

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.

**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](#).

```
foreach bt
```

## 25.2 Dump system message buffer

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

**Figure 47:** Kernel crash log command output example

```

admin@testhost2:/var/crash/2010-01-19-17:11
File Edit View Terminal Tabs Help
Linux version 2.6.18-164.10.1.el5 (mockbuild@builder10.centos.org) (gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)) #1 SMP Thu
Jan 7 19:54:26 EST 2010
Command line: rr root=LABEL=/ rhgb quiet crashkernel=128M@16M
BIOS-provided physical RAM map:
BIOS-e820: 00000000000010000 - 0000000000009f800 (usable)
BIOS-e820: 0000000000009f800 - 000000000000a0000 (reserved)
BIOS-e820: 00000000000ca000 - 00000000000cc000 (reserved)
BIOS-e820: 00000000000dc000 - 00000000000100000 (reserved)
BIOS-e820: 00000000000100000 - 000000000efef0000 (usable)
BIOS-e820: 00000000000efef0000 - 000000000efeff0000 (ACPI data)
BIOS-e820: 00000000000efeff0000 - 000000000efff00000 (ACPI NVS)
BIOS-e820: 00000000000efff0000 - 000000000f0000000 (usable)
BIOS-e820: 000000000fec00000 - 000000000fec10000 (reserved)
BIOS-e820: 000000000fee00000 - 000000000fee010000 (reserved)
BIOS-e820: 000000000fffe0000 - 00000000100000000 (reserved)
BIOS-e820: 00000000100000000 - 000000001fa000000 (usable)
DMI present.
ACPI: RSDP (v000 PTLTD ) @ 0x000000000000f6c68
ACPI: RSDT (v001 PTLTD RSDT 0x06040000 LTP 0x00000000) @ 0x000000000efefab5a
ACPI: FADT (v001 INTEL 440BX 0x06040000 PTL 0x000f4240) @ 0x000000000efefef06
ACPI: MADT (v001 PTLTD APIC 0x06040000 LTP 0x00000000) @ 0x000000000efefef7a
ACPI: BOOT (v001 PTLTD $SBFTBLS 0x06040000 LTP 0x00000001) @ 0x000000000efefefd8
ACPI: DSDT (v001 PTLTD Custom 0x06040000 MSFT 0x100000d) @ 0x0000000000000000
No NUMA configuration found
Fakeing a node at 0000000000000000-000000001fa00000
Bootmem setup node 0 0000000000000000-000000001fa00000
On node 0 totalpages: 1977281
  DMA zone: 2633 pages, LIFO batch:0
  DMA32 zone: 964648 pages, LIFO batch:31
  Normal zone: 1010000 pages, LIFO batch:31
ACPI: PM-Timer IO Port: 0x1008
ACPI: Local APIC address 0xf0000000
ACPI: LAPIC (acpi_id[0x00] lpic_id[0x00] enabled)
Processor #0 6:15 APIC version 17
ACPI: LAPIC (acpi_id[0x01] lpic_id[0x01] enabled)
Processor #1 6:15 APIC version 17
ACPI: LAPIC NMI (acpi_id[0x00] high edge lint[0x1])
-- MORE -- forward: <SPACE>, <ENTER> or j backward: b or k quit: q

```

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:

```
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 >.

**Figure 48:** Kernel crash ps command output example

admin@testhost2:/var/crash/2010-01-19-20:21

PID	PPID	CPU	TASK	ST	%MEM	VSZ	RSS	COMM
> 0	0	0	ffffffff80300ae0	RU	0.0	0	0	[swapper]
0	1	1	fffff81010710b0c0	RU	0.0	0	0	[swapper]
1	0	1	fffff8101070eb7a0	IN	0.0	10348	708	init
2	1	0	fffff8101070eb040	IN	0.0	0	0	[migration/0]
3	1	0	fffff8101070ed7e0	IN	0.0	0	0	[ksoftirqd/0]
4	1	1	fffff8101070ed080	IN	0.0	0	0	[migration/1]
5	1	1	fffff81010710b820	IN	0.0	0	0	[ksoftirqd/1]
6	1	0	fffff8101f9cdd860	IN	0.0	0	0	[events/0]
7	1	1	fffff8101f9cdd100	IN	0.0	0	0	[events/1]
8	1	1	fffff8101f9cde7a0	IN	0.0	0	0	[khelper]
73	1	0	fffff8101f9db0040	IN	0.0	0	0	[kthread]
78	73	0	fffff8101f9e27860	IN	0.0	0	0	[kblockd/0]
79	73	1	fffff8101f9e27100	IN	0.0	0	0	[kblockd/1]
80	73	0	fffff8101f9e2e7a0	IN	0.0	0	0	[kacpid]
136	73	0	fffff8101f9d5c820	IN	0.0	0	0	[cqueue/0]
137	73	1	fffff8101f9d650c0	IN	0.0	0	0	[cqueue/1]
140	73	0	fffff8101f9d67860	IN	0.0	0	0	[khabd]
142	73	1	fffff8101f9d67100	IN	0.0	0	0	[kseriod]
214	73	0	fffff8101f9d697a0	IN	0.0	0	0	[pdflush]
215	73	0	fffff8101f9d69040	IN	0.0	0	0	[pdflush]
216	73	0	fffff8101f9d6c7e0	IN	0.0	0	0	[kswapd0]
217	73	0	fffff8101f9d6c080	IN	0.0	0	0	[aio/0]

-- MORE -- forward: <SPACE>, <ENTER> or j backward: b or k quit: q

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

```
PANIC: "SysRq : Trigger a crashdump"
PID: 0
COMMAND: "swapper"
TASK: ffffffff80300ae0  (1 of 2)  [THREAD_INFO: ffffffff803f2000]
CPU: 0
STATE: TASK_RUNNING
WARNING: panic task not found

crash>
```

**Figure 50:** bt command for wrong process

```
admin@testhost2:/var/crash/2010-01-19-17:11
File Edit View Terminal Tabs Help
crash> bt
PID: 0      TASK: ffffffff80300ae0  CPU: 0    COMMAND: "swapper"
#0 [ffffffffff803f3eb8] schedule at ffffff80062f66
#1 [ffffffffff803f3f90] cpu_idle at ffffff8004945d
crash> █
```

In this case, the pointer in the ps output marks the "wrong" process:

**Figure 51:** ps command output pointing at wrong process

PID	PPID	CPU	TASK	ST	%MEM	VSZ	RSS	COMM
0	0	0	ffffffffff80300ae0	RU	0.0	0	0	[swapper]
> 0	1	1	ffff81010710b0c0	RU	0.0	0	0	[swapper]
1	0	1	ffff8101070eb7a0	IN	0.0	10348	636	init
2	1	0	ffff8101070eb040	IN	0.0	0	0	[migration/0]
3	1	0	ffff8101070ed7e0	IN	0.0	0	0	[ksoftirqd/0]
4	1	1	ffff8101070ed080	IN	0.0	0	0	[migration/1]
5	1	1	ffff81010710b0820	IN	0.0	0	0	[ksoftirqd/1]
6	1	0	ffff8101f9cdd860	IN	0.0	0	0	[events/0]
7	1	1	ffff8101f9cdd100	IN	0.0	0	0	[events/1]
8	1	1	ffff8101f9cd7a0	IN	0.0	0	0	[khelper]
73	1	0	ffff8101f9db0040	IN	0.0	0	0	[kthread]
78	73	0	ffff8101f9d697a0	IN	0.0	0	0	[kblockd/0]
79	73	1	ffff8101f9d67100	IN	0.0	0	0	[kblockd/1]
80	73	1	ffff8101f9d67860	IN	0.0	0	0	[kacpid]
136	73	0	ffff8101f9d63080	IN	0.0	0	0	[cqqueue/0]

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

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!

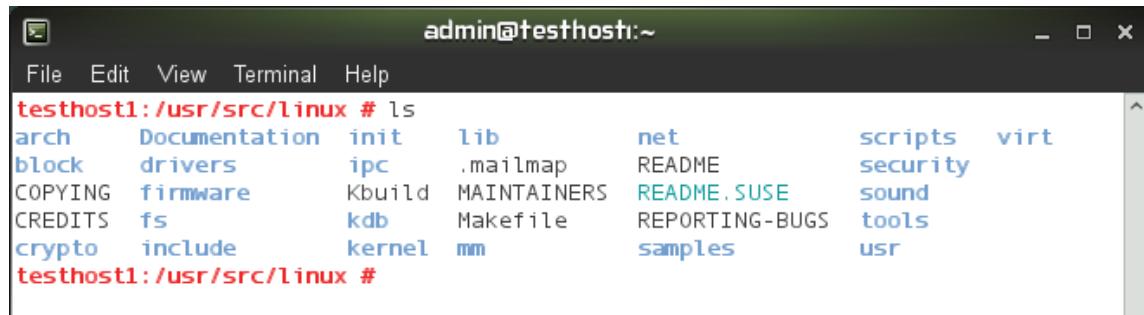
Now, time to get ultra-serious. Let's say you may even want to analyze the C code for the offending function. Needless to say, you should have the C sources available and be able to read them. This is not something everyone should do, but it's an interesting mental exercise. Source code. All right, you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



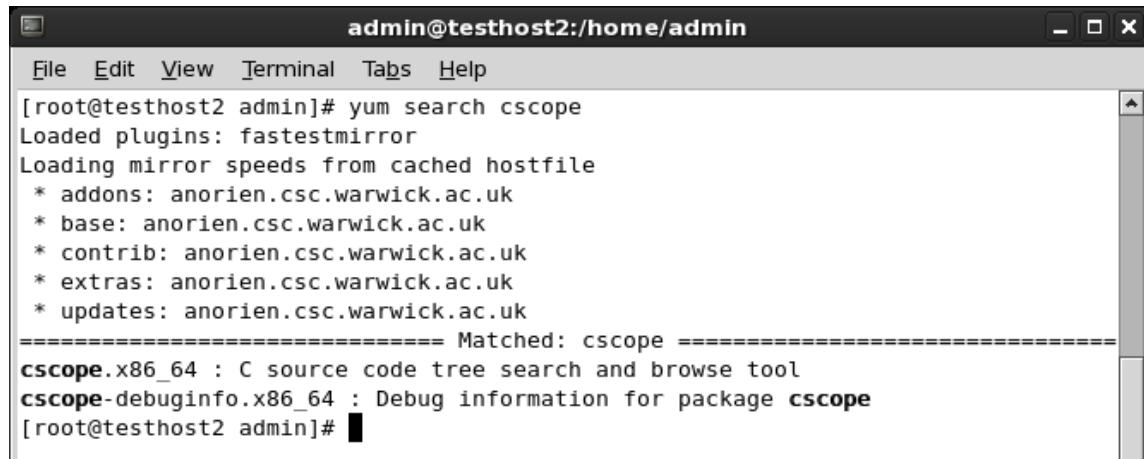
```
testhost1: /usr/src/linux # ls
arch      Documentation  init     lib      net      scripts  virt
block     drivers        ipc     .mailmap  README   security
COPYING   firmware       Kbuild  MAINTAINERS README.SUSE
CREDITS   fs            kdb     Makefile   REPORTING-BUGS
crypto    include        kernel  mm       samples
testhost1: /usr/src/linux #
```

## 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](#). The tool runs from the command line

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



```
admin@testhost2:~/home/admin
File Edit View Terminal Tabs Help
[root@testhost2 admin]# yum search cscope
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
* addons: anorien.csc.warwick.ac.uk
* base: anorien.csc.warwick.ac.uk
* contrib: anorien.csc.warwick.ac.uk
* extras: anorien.csc.warwick.ac.uk
* updates: anorien.csc.warwick.ac.uk
=====
===== Matched: cscope =====
cscope.x86_64 : C source code tree search and browse tool
cscope-debuginfo.x86_64 : Debug information for package cscope
[root@testhost2 admin]#
```

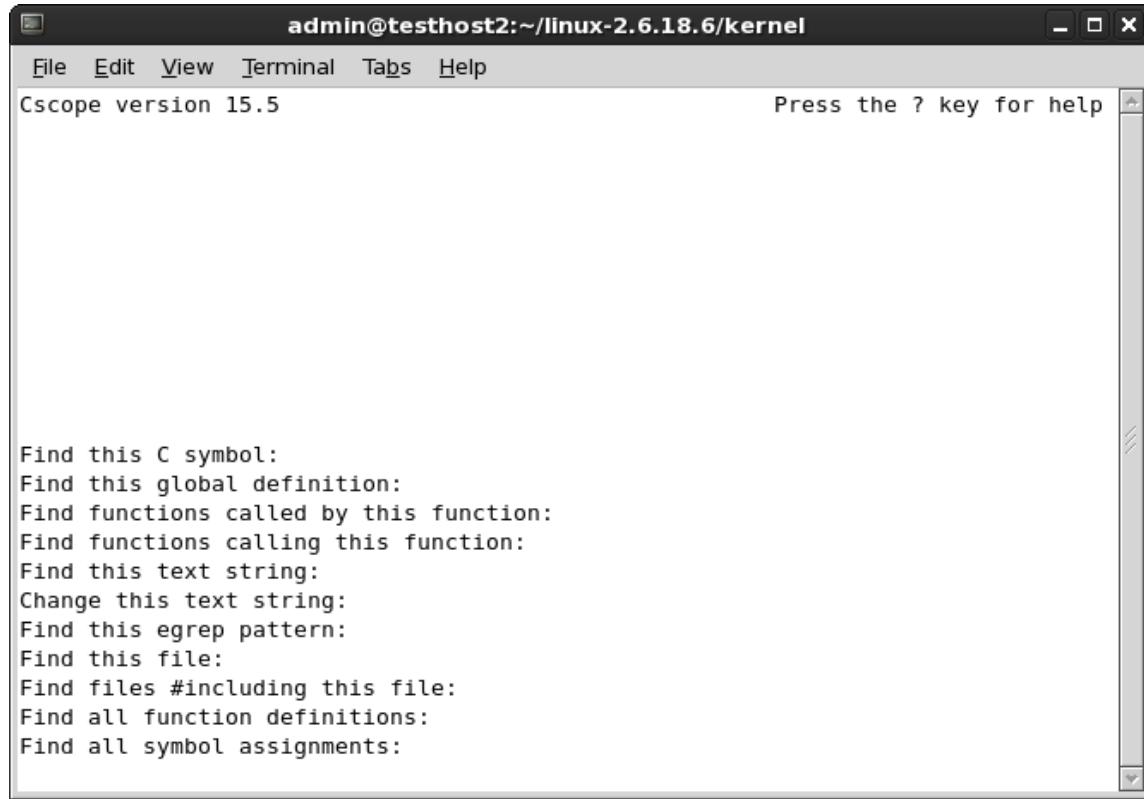
Now, in the directory containing sources<sup>22</sup>, run cscope:

```
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.

---

<sup>22</sup>By default, the sources are located under */usr/src/linux*.

**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:

**Figure 55:** Find C symbol using cscope

The screenshot shows a terminal window titled "admin@testhost2:~/linux-2.6.18.6". The window contains the output of a cscope search for the symbol "default\_idle". The results are as follows:

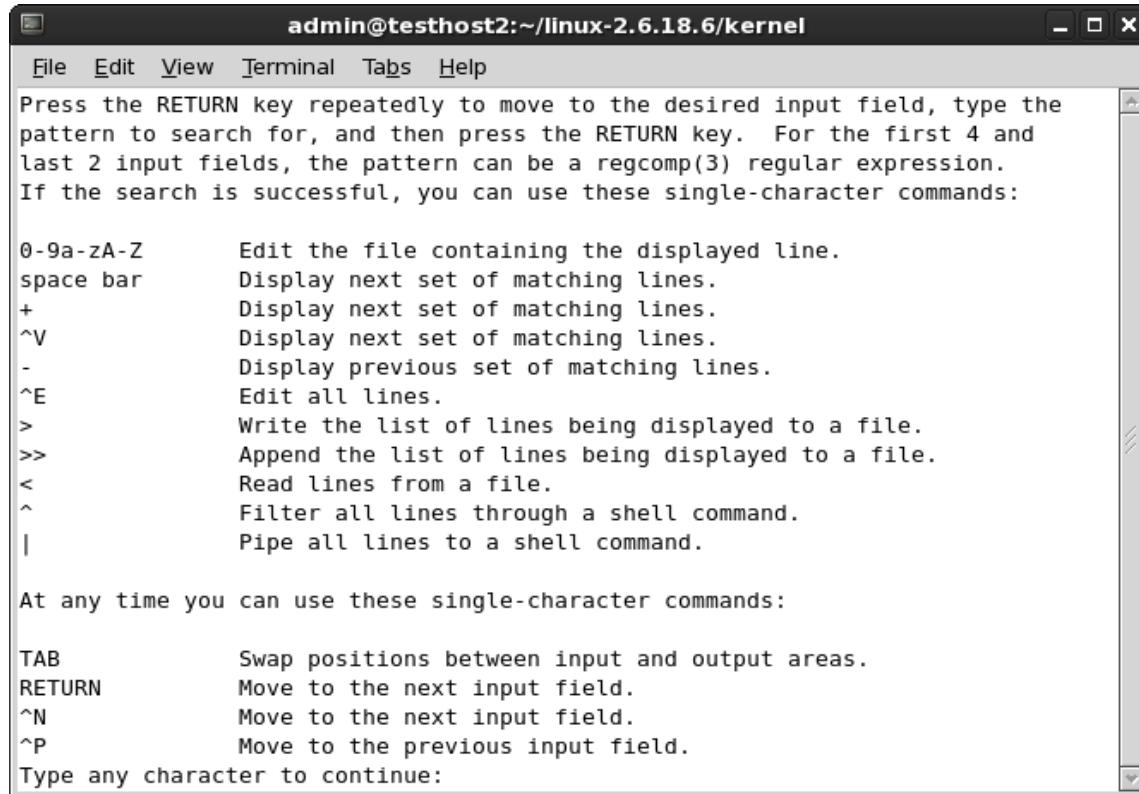
File	Function	Line
0 process.c	<global>	72 void (*idle)(void ) = default_idle;
1 process.c	<global>	120 EXPORT_SYMBOL(default_idle);
2 process.c	<global>	90 void (*idle)(void ) = default_idle;
3 process.c	<global>	64 void (*idle)(void ) = default_idle;
4 kern_util.h	<global>	67 extern void default_idle(void );
5 process.c	<global>	45 void (*idle)(void ) = default_idle;
6 system.h	<global>	72 void default_idle(void );

\* 32 more lines - press the space bar to display more \*

Find this C symbol:  
Find this global definition:  
Find functions called by this function:  
Find functions calling this function:  
Find this text string:  
Change this text string:  
Find this egrep pattern:  
Find this file:  
Find files #including this file:  
Find all function definitions:  
Find all symbol assignments:

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:

**Figure 56:** cscope help menu

The screenshot shows a terminal window titled "admin@testhost2:~/linux-2.6.18.6/kernel". The window contains the cscope help menu. It starts with instructions on how to use the search function, followed by a table of single-character commands and their descriptions. Below this, it lists additional single-character commands for navigating input areas. At the bottom, it prompts the user to type any character to continue.

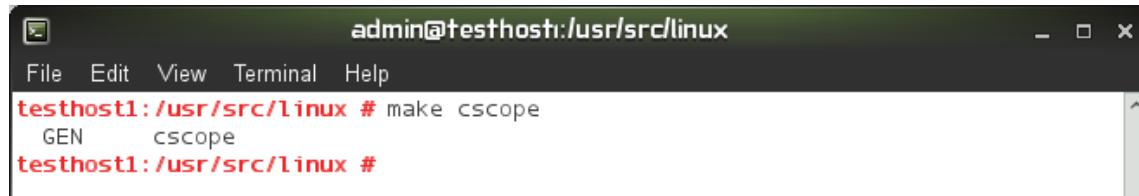
0-9a-zA-Z	Edit the file containing the displayed line.
space bar	Display next set of matching lines.
+	Display next set of matching lines.
^V	Display next set of matching lines.
-	Display previous set of matching lines.
^E	Edit all lines.
>	Write the list of lines being displayed to a file.
>>	Append the list of lines being displayed to a file.
<	Read lines from a file.
^	Filter all lines through a shell command.
	Pipe all lines to a shell command.

At any time you can use these single-character commands:

TAB	Swap positions between input and output areas.
RETURN	Move to the next input field.
^N	Move to the next input field.
^P	Move to the previous input field.

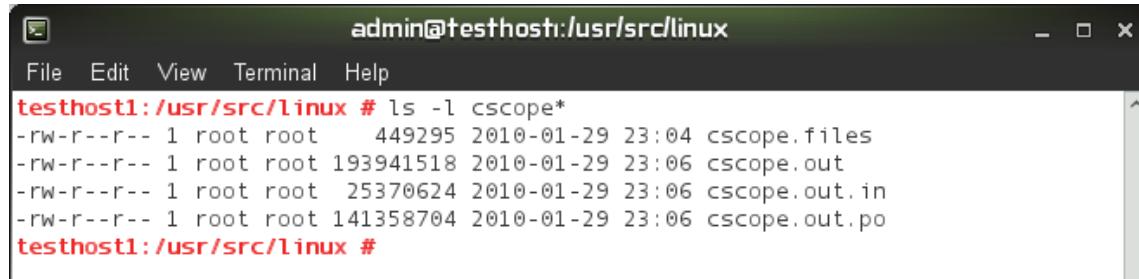
Type any character to continue:

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

The screenshot shows a terminal window titled "admin@testhost1:/usr/src/linux". The window displays the command "make cscope" being run in the "/usr/src/linux" directory. The output shows the command "GEN cscope" and then a prompt "#".

```
testhost1:/usr/src/linux # make cscope
GEN      cscope
testhost1:/usr/src/linux #
```

**Figure 58:** cscope files

```
admin@testhost1:/usr/src/linux
File Edit View Terminal Help
testhost1:/usr/src/linux # ls -l cscope*
-rw-r--r-- 1 root root 449295 2010-01-29 23:04 cscope.files
-rw-r--r-- 1 root root 193941518 2010-01-29 23:06 cscope.out
-rw-r--r-- 1 root root 25370624 2010-01-29 23:06 cscope.out.in
-rw-r--r-- 1 root root 141358704 2010-01-29 23:06 cscope.out.po
testhost1:/usr/src/linux #
```

## 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.

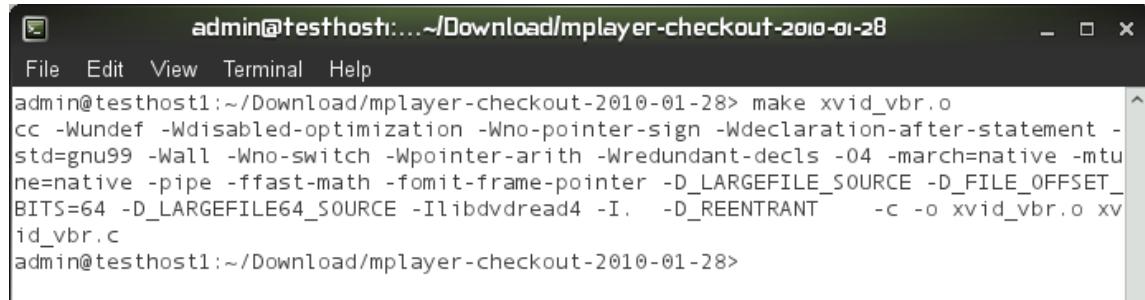
This is not a simple or a trivial task. First, if you use a compiler that is different than the one used to compile the original, your object will be different from the one in the crash report, rendering your efforts difficult if not impossible.

## 26.4 Trivial example

I call this example trivial because it has nothing to do with the kernel. It merely demonstrates how to compile objects and then disassemble them. Any source will do. In our case, we'll use [MPlayer](#), 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.

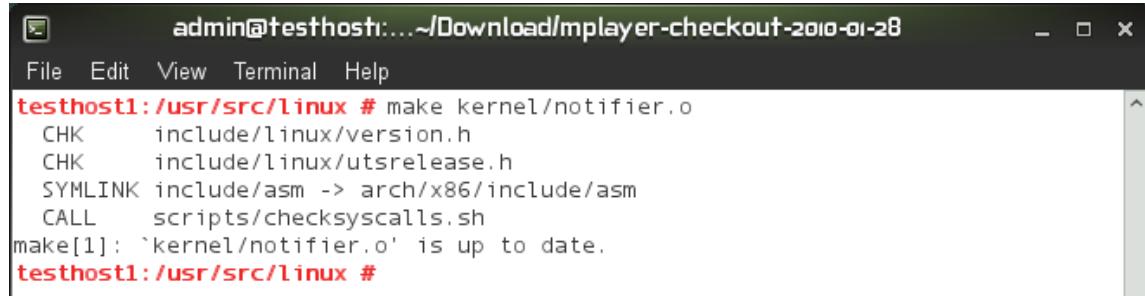
Run `make <object name>`, for instance:

```
make xvid_bvr.o
```

**Figure 59:** Compiling from sources with make

```
admin@testhost1:~/Download/mplayer-checkout-2010-01-28> make xvid_vbr.o
cc -Wundef -Wdisabled-optimization -Wno-pointer-sign -Wdeclaration-after-statement -std=gnu99 -Wall -Wno-switch -Wpointer-arith -Wredundant-decls -O4 -march=native -mtune=native -pipe -ffast-math -fomit-frame-pointer -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -D_LARGEFILE64_SOURCE -Ilibdvdread4 -I. -D_REENTRANT -c -o xvid_vbr.o xvid_vbr.c
admin@testhost1:~/Download/mplayer-checkout-2010-01-28>
```

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

```
admin@testhost1:~/Download/mplayer-checkout-2010-01-28>
File Edit View Terminal Help
testhost1:/usr/src/linux # make kernel/notifier.o
CHK     include/linux/version.h
CHK     include/linux/utsrelease.h
SYMLINK include/asm -> arch/x86/include/asm
CALL    scripts/checksyscalls.sh
make[1]: `kernel/notifier.o' is up to date.
testhost1:/usr/src/linux #
```

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:

**Figure 61:** Object compiled with debug symbols

```
admin@testhost1:~> gcc memhog.c -o memhog
admin@testhost1:~> ls -ld memhog
-rwxr-xr-x 1 admin users 10297 2010-01-29 23:22 memhog
admin@testhost1:~>
admin@testhost1:~> rm memhog
admin@testhost1:~>
admin@testhost1:~> gcc -g memhog.c -o memhog
admin@testhost1:~> ls -ld memhog
-rwxr-xr-x 1 admin users 12125 2010-01-29 23:22 memhog
admin@testhost1:~>
admin@testhost1:~>
```

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.

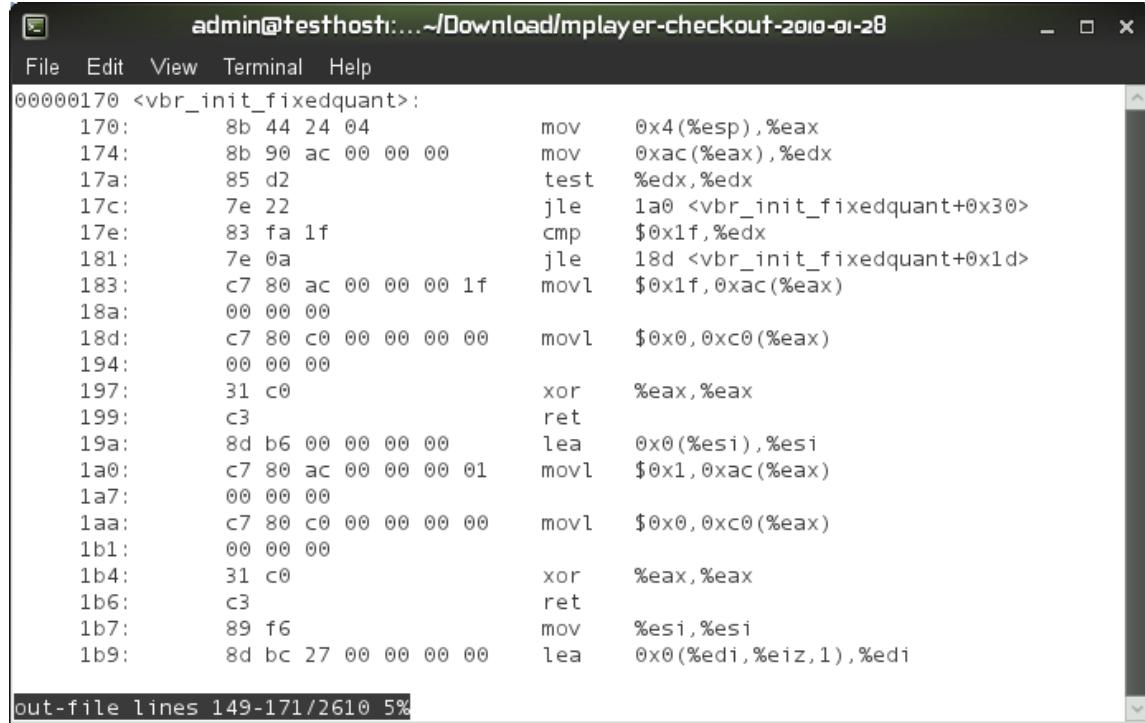
## 26.5 objdump

A utility you want to use for disassembly is `objdump`. You will probably want to use the utility with `-S` flag, which means display source code intermixed with assembly instructions. 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:

**Figure 62:** Disassembled object example

The screenshot shows a terminal window titled "admin@testhost:...~/Download/mplayer-checkout-2010-01-28". The window contains assembly code for a function named <vbr\_init\_fixedquant>. The code consists of several instructions, mostly involving moves (mov), tests (test), jumps (jle), and comparisons (cmp). The assembly code is as follows:

```
000000170 <vbr_init_fixedquant>:
170: 8b 44 24 04      mov    0x4(%esp),%eax
174: 8b 90 ac 00 00 00  mov    0xac(%eax),%edx
17a: 85 d2            test   %edx,%edx
17c: 7e 22            jle    1a0 <vbr_init_fixedquant+0x30>
17e: 83 fa 1f          cmp    $0x1f,%edx
181: 7e 0a            jle    18d <vbr_init_fixedquant+0x1d>
183: c7 80 ac 00 00 00 1f  movl   $0x1f,0xac(%eax)
18a: 00 00 00
18d: c7 80 c0 00 00 00 00  movl   $0x0,0xc0(%eax)
194: 00 00 00
197: 31 c0            xor    %eax,%eax
199: c3                ret
19a: 8d b6 00 00 00 00  lea    0x0(%esi),%esi
1a0: c7 80 ac 00 00 00 01  movl   $0x1,0xac(%eax)
1a7: 00 00 00
1aa: c7 80 c0 00 00 00 00  movl   $0x0,0xc0(%eax)
1b1: 00 00 00
1b4: 31 c0            xor    %eax,%eax
1b6: c3                ret
1b7: 89 f6            mov    %esi,%esi
1b9: 8d bc 27 00 00 00 00  lea    0x0(%edi,%eiz,1),%edi

out-file lines 149-171/2610 5%
```

And an even better example, the memhog dump:

**Figure 63:** Memhog binary dumped with objdump

```

admin@testhost:~ 
File Edit View Terminal Help
08048458 <__libc_start_main@plt>:
8048458: ff 25 08 a0 04 08      jmp    *0x804a008
804845e: 68 10 00 00 00          push   $0x10
8048463: e9 c0 ff ff ff      jmp    8048428 <_init+0x30>

08048468 <fflush@plt>:
8048468: ff 25 0c a0 04 08      jmp    *0x804a00c
804846e: 68 18 00 00 00          push   $0x18
8048473: e9 b0 ff ff ff      jmp    8048428 <_init+0x30>

08048478 <printf@plt>:
8048478: ff 25 10 a0 04 08      jmp    *0x804a010
804847e: 68 20 00 00 00          push   $0x20
8048483: e9 a0 ff ff ff      jmp    8048428 <_init+0x30>

08048488 <atoi@plt>:
8048488: ff 25 14 a0 04 08      jmp    *0x804a014
804848e: 68 28 00 00 00          push   $0x28
8048493: e9 90 ff ff ff      jmp    8048428 <_init+0x30>

out lines 376-395/2770 13%

```

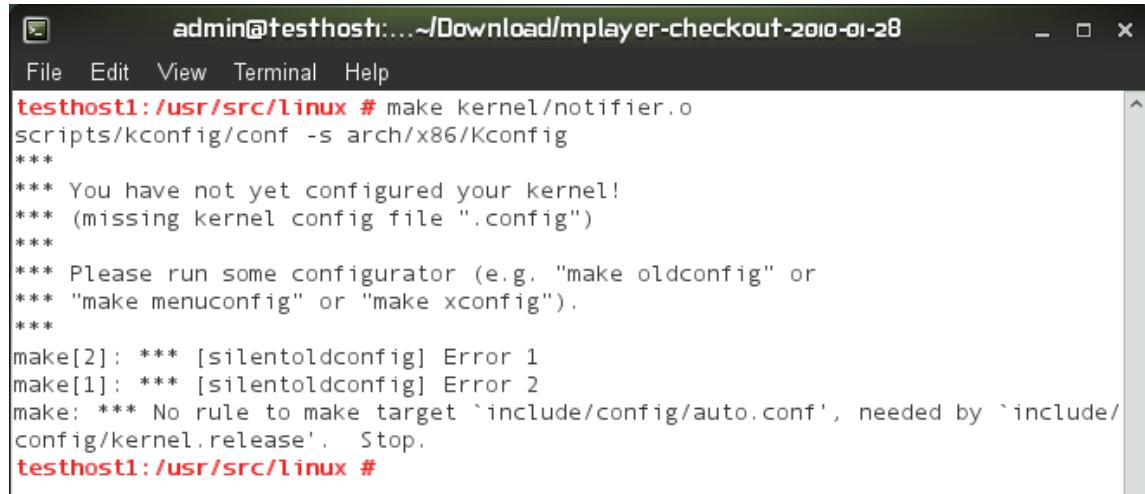
## 26.6 Moving on to kernel sources

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:

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



The screenshot shows a terminal window titled "admin@testhost1:...~/Download/mplayer-checkout-2010-01-28". The command "make kernel/notifier.o" is run, but fails because it cannot find a kernel configuration file. The error message indicates that ".config" is missing and suggests running "make oldconfig", "make menuconfig", or "make xconfig". The terminal prompt "testhost1:/usr/src/linux # " is visible at the bottom.

```
admin@testhost1:...~/Download/mplayer-checkout-2010-01-28
File Edit View Terminal Help
testhost1:/usr/src/linux # make kernel/notifier.o
scripts/kconfig/conf -s arch/x86/Kconfig
***
*** You have not yet configured your kernel!
*** (missing kernel config file ".config")
***
*** Please run some configurator (e.g. "make oldconfig" or
*** "make menuconfig" or "make xconfig").
***
make[2]: *** [silentoldconfig] Error 1
make[1]: *** [silentoldconfig] Error 2
make: *** No rule to make target `include/config/auto.conf', needed by `include/
config/kernel.release'. Stop.
testhost1:/usr/src/linux #
```

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
```

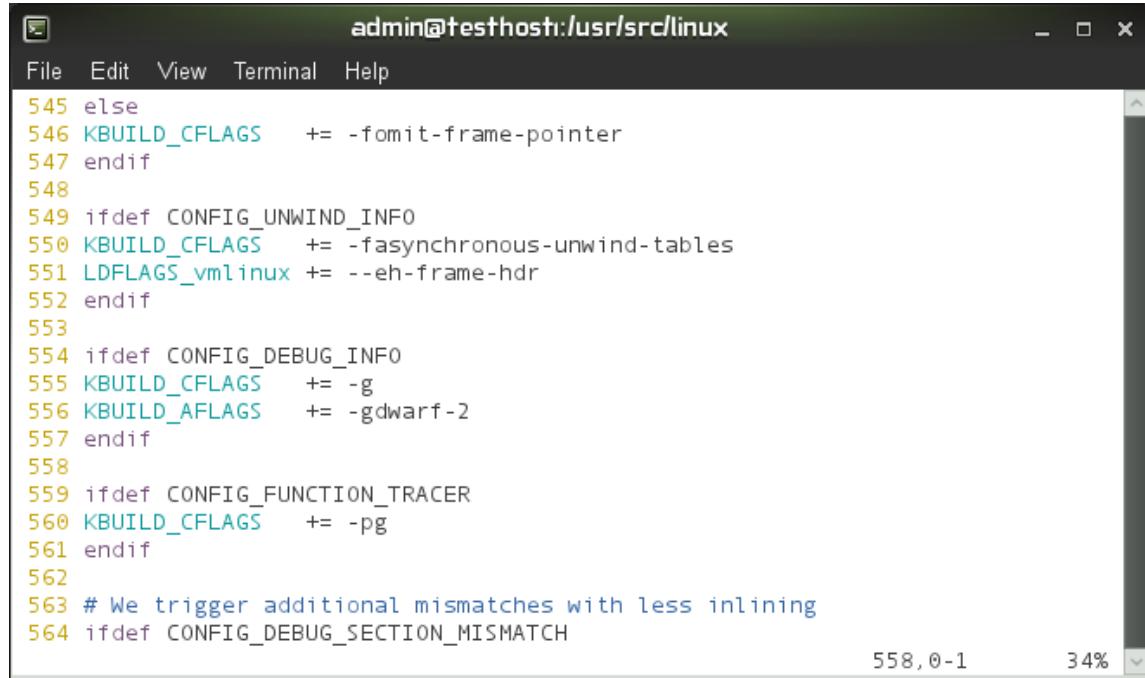
As a long term solution, you could also create symbolic links under */usr/src/linux* from the would-be bad architecture to the right one. This is not strictly related to the analysis of kernel crashes, but if and when you compile kernel sources, you may encounter this issue. Now, regarding the *CONFIG\_DEBUG\_INFO* 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
```

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



```

545 else
546 KBUILD_CFLAGS += -fomit-frame-pointer
547 endif
548
549 ifdef CONFIG_UNWIND_INFO
550 KBUILD_CFLAGS += -fasynchronous-unwind-tables
551 LDFLAGS_vmlinux += --eh-frame-hdr
552 endif
553
554 ifdef CONFIG_DEBUG_INFO
555 KBUILD_CFLAGS += -g
556 KBUILD_AFLAGS += -gdwarf-2
557 endif
558
559 ifdef CONFIG_FUNCTION_TRACER
560 KBUILD_CFLAGS += -pg
561 endif
562
563 # We trigger additional mismatches with less inlining
564 ifdef CONFIG_DEBUG_SECTION_MISMATCH

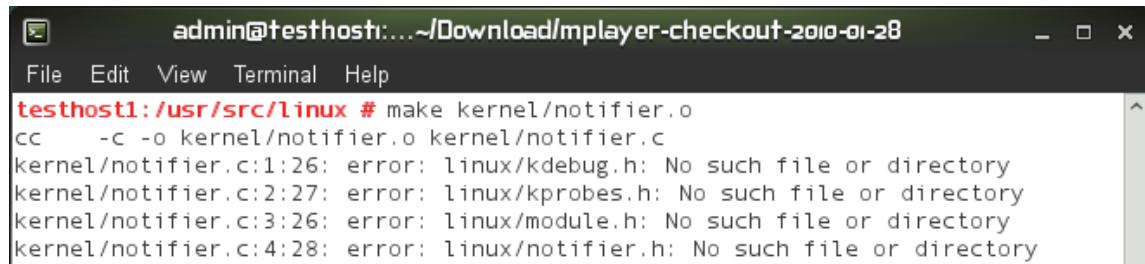
```

558, 0-1

34%

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



```

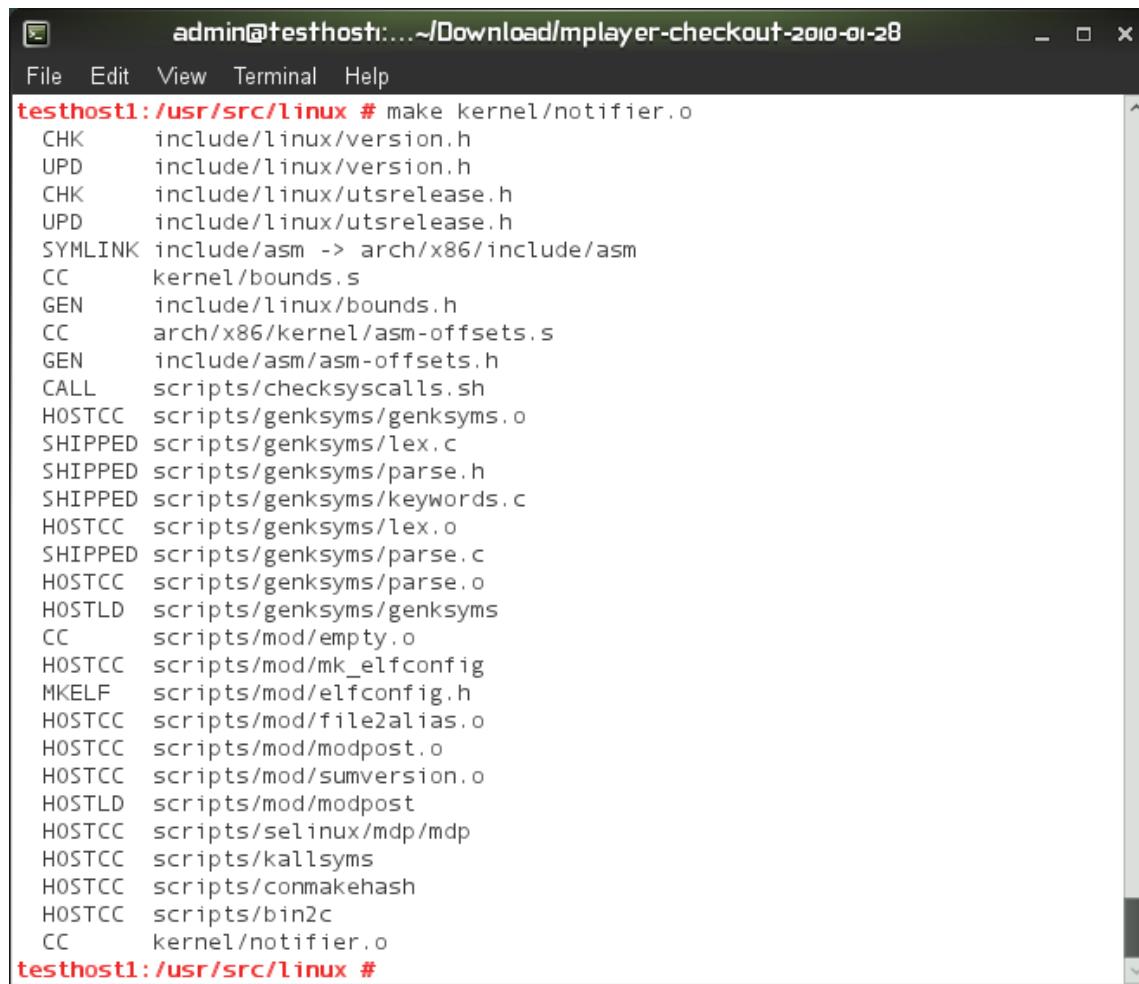
testhost1:/usr/src/linux # make kernel/notifier.o
cc -c -o kernel/notifier.o kernel/notifier.c
kernel/notifier.c:1:26: error: linux/kdebug.h: No such file or directory
kernel/notifier.c:2:27: error: linux/kprobes.h: No such file or directory
kernel/notifier.c:3:26: error: linux/module.h: No such file or directory
kernel/notifier.c:4:28: error: linux/notifier.h: No such file or directory

```

558, 0-1

34%

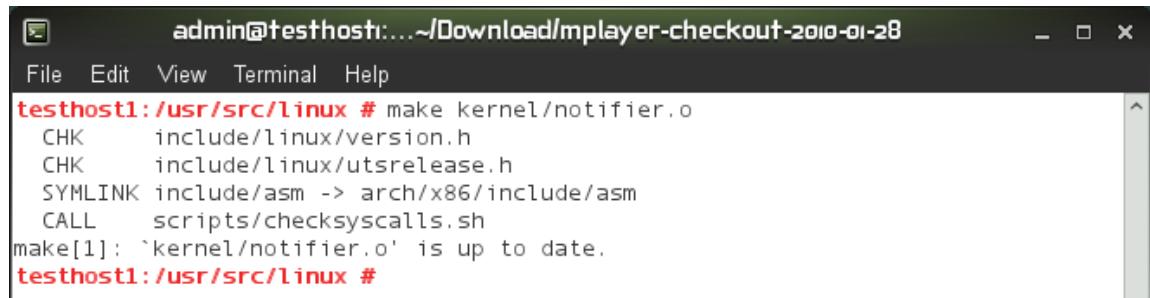
But with the Makefile in place, it should all work smoothly.

**Figure 67:** Successfully compiling kernel object

The screenshot shows a terminal window titled "admin@testhost1:...~/Download/mplayer-checkout-2010-01-28". The window contains the following text:

```
testhost1:/usr/src/linux # make kernel/notifier.o
CHK    include/linux/version.h
UPD    include/linux/version.h
CHK    include/linux/utsrelease.h
UPD    include/linux/utsrelease.h
SYMLINK include/asm -> arch/x86/include/asm
CC     kernel/bounds.s
GEN    include/linux/bounds.h
CC     arch/x86/kernel/asm-offsets.s
GEN    include/asm/asm-offsets.h
CALL   scripts/checksyscalls.sh
HOSTCC scripts/genksyms/genksyms.o
SHIPPED scripts/genksyms/lex.c
SHIPPED scripts/genksyms/parse.h
SHIPPED scripts/genksyms/keywords.c
HOSTCC scripts/genksyms/lex.o
SHIPPED scripts/genksyms/parse.c
HOSTCC scripts/genksyms/parse.o
HOSTLD scripts/genksyms/genksyms
CC     scripts/mod/empty.o
HOSTCC scripts/mod/mk_elfconfig
MKELF  scripts/mod/elfconfig.h
HOSTCC scripts/mod/file2alias.o
HOSTCC scripts/mod/modpost.o
HOSTCC scripts/mod/sumversion.o
HOSTLD scripts/mod/modpost
HOSTCC scripts/selinux/mdp/mdp
HOSTCC scripts/kallsyms
HOSTCC scripts/conmakehash
HOSTCC scripts/bin2c
CC     kernel/notifier.o
testhost1:/usr/src/linux #
```

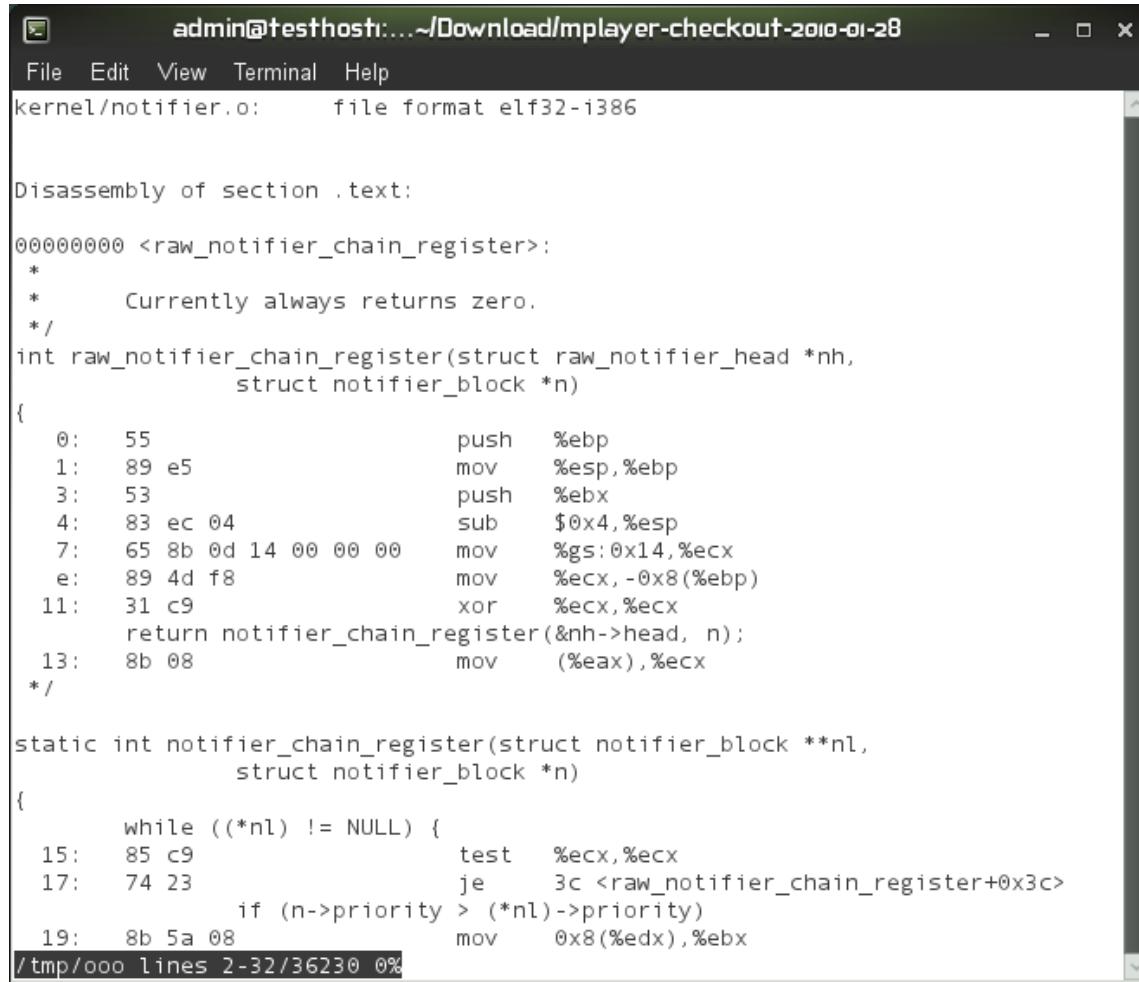
And then, there's the object up to date example again. If you do not remove an existing one, you won't be able to compile a new one, especially if you need debug symbols for later disassembly.

**Figure 68:** Kernel object is up to date

The screenshot shows a terminal window titled "admin@testhost1:...~/Download/mplayer-checkout-2010-01-28". The window contains the following text:

```
File Edit View Terminal Help
testhost1:/usr/src/linux # make kernel/notifier.o
CHK    include/linux/version.h
CHK    include/linux/utsrelease.h
SYMLINK include/asm -> arch/x86/include/asm
CALL   scripts/checksyscalls.sh
make[1]: `kernel/notifier.o' is up to date.
testhost1:/usr/src/linux #
```

Finally, the disassembled object:

**Figure 69:** Disassembled kernel object


The screenshot shows a terminal window titled "admin@testhost:...~/Download/mplayer-checkout-2010-01-28". The window displays the assembly code for the "raw\_notifier\_chain\_register" function from the "kernel/notifier.o" file, which is in ELF32-i386 format. The assembly code is shown in Intel syntax, with addresses, opcodes, and comments. The code implements a linked list insertion logic.

```

File Edit View Terminal Help
kernel/notifier.o:      file format elf32-i386

Disassembly of section .text:
00000000 <raw_notifier_chain_register>:
* *     Currently always returns zero.
*/
int raw_notifier_chain_register(struct raw_notifier_head *nh,
                                struct notifier_block *n)
{
    0: 55                      push  %ebp
    1: 89 e5                   mov   %esp,%ebp
    3: 53                      push  %ebx
    4: 83 ec 04                sub   $0x4,%esp
    7: 65 8b 0d 14 00 00 00    mov   %gs:0x14,%ecx
    e: 89 4d f8                mov   %ecx,-0x8(%ebp)
   11: 31 c9                  xor   %ecx,%ecx
        return notifier_chain_register(&nh->head, n);
   13: 8b 08                  mov   (%eax),%ecx
*/
static int notifier_chain_register(struct notifier_block **nl,
                                  struct notifier_block *n)
{
    while ((*nl) != NULL) {
   15: 85 c9                  test  %ecx,%ecx
   17: 74 23                  je    3c <raw_notifier_chain_register+0x3c>
        if (n->priority > (*nl)->priority)
   19: 8b 5a 08                mov   0x8(%edx),%ebx

```

/tmp/ooo lines 2-32/36230 0%

### 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

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.

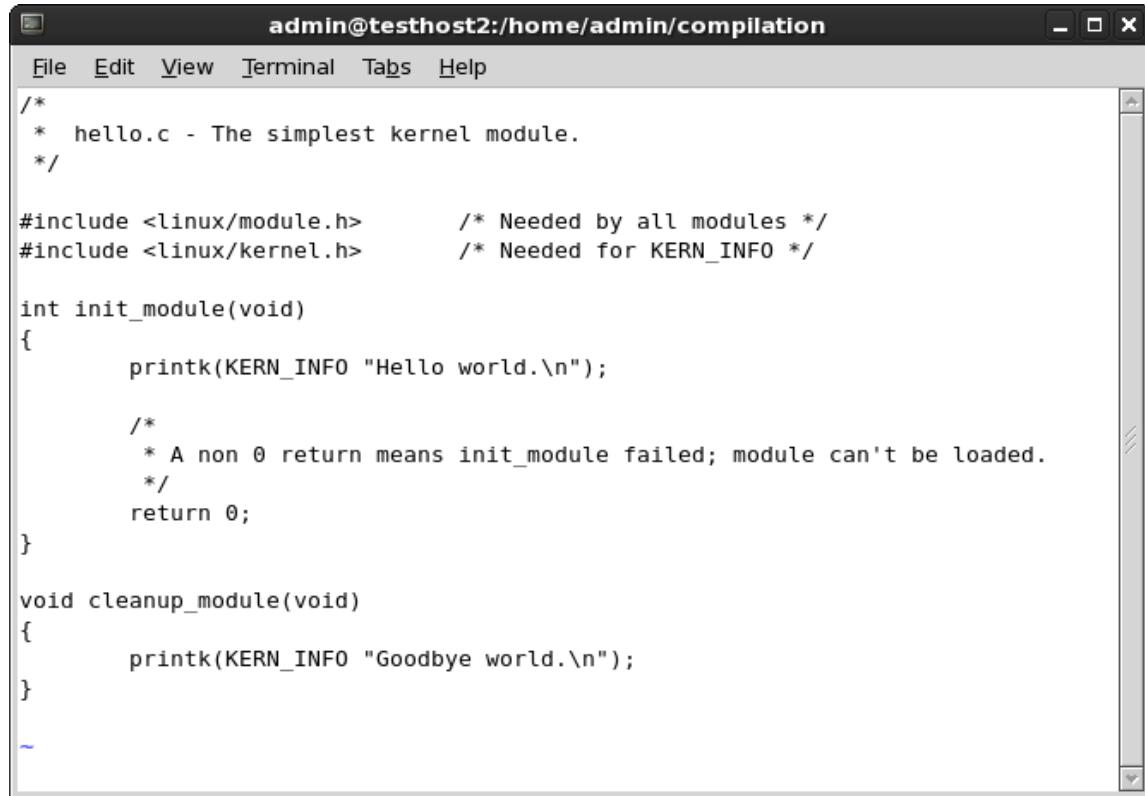
### 26.7.1 Create problematic kernel module

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.

**Figure 70:** Basic kernel module

The screenshot shows a terminal window titled "admin@testhost2:/home/admin/compilation". The window contains the source code for a Linux kernel module named "hello.c". The code includes comments explaining the purpose of the module and its functions, which print "Hello world." and "Goodbye world." to the kernel log.

```
/*  
 * hello.c - The simplest kernel module.  
 */  
  
#include <linux/module.h>          /* Needed by all modules */  
#include <linux/kernel.h>           /* Needed for KERN_INFO */  
  
int init_module(void)  
{  
    printk(KERN_INFO "Hello world.\n");  
  
    /*  
     * A non 0 return means init_module failed; module can't be loaded.  
     */  
    return 0;  
}  
  
void cleanup_module(void)  
{  
    printk(KERN_INFO "Goodbye world.\n");  
}
```

```
1  /*
2   *  hello.c - The simplest kernel module.
3   */
4
5 #include <linux/module.h>      /* Needed by all modules */
6 #include <linux/kernel.h>      /* Needed for KERN_INFO */
7
8 int init_module(void)
9 {
10    printk(KERN_INFO "Hello\u2022world.\n");
11
12    /*
13     * A non 0 return means init_module failed; module can't be
14     * loaded.
15     */
16    return 0;
17}
18 void cleanup_module(void)
19 {
20    printk(KERN_INFO "Goodbye\u2022world.\n");
21 }
```

We need to compile this module, so we need a Makefile:

**Figure 71:** Basic example Makefile



The screenshot shows a terminal window titled "admin@testhost2:/home/admin/compilation". The window contains a Makefile with the following content:

```
admin@testhost2:/home/admin/compilation
File Edit View Terminal Tabs Help
obj-m += hello.o

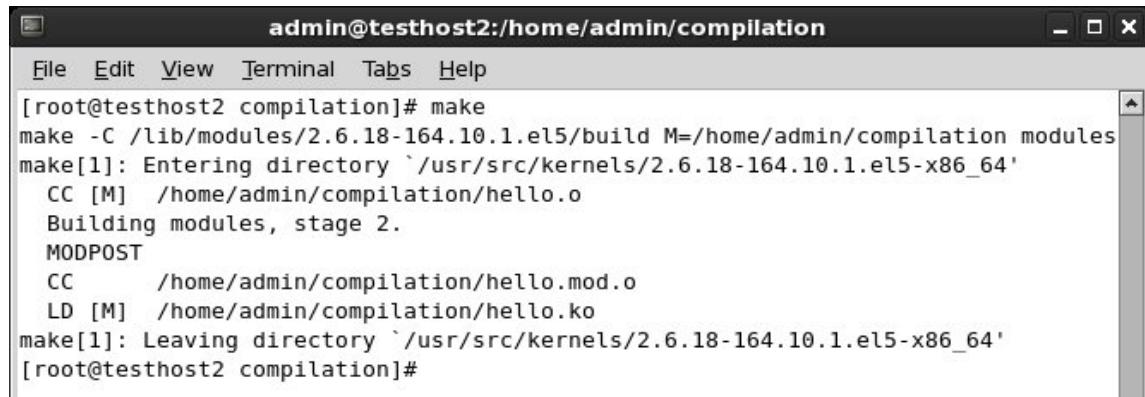
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

```
1 obj-m += hello.o
2
3 all:
4     make -C /lib/modules/$(shell uname -r)/build M=$(PWD)
      modules
5
6 clean:
7     make -C /lib/modules/$(shell uname -r)/build M=$(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

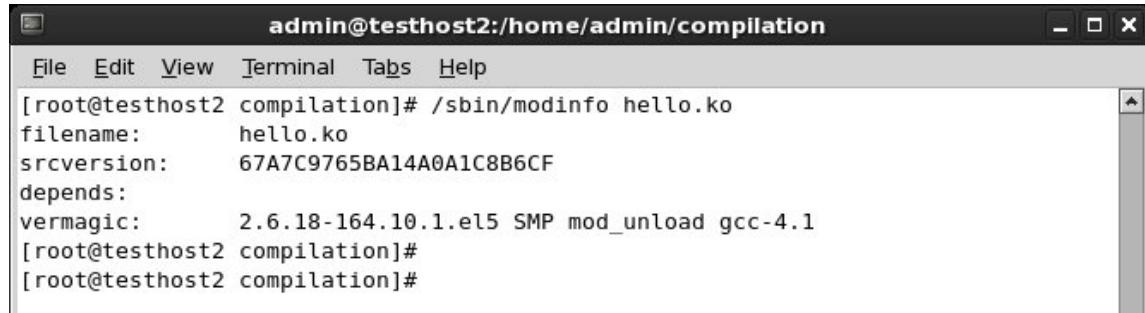


The screenshot shows a terminal window titled "admin@testhost2:/home/admin/compilation". The window has a standard Linux-style title bar with icons for minimize, maximize, and close. Below the title bar is a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main area of the terminal displays the following text:

```
[root@testhost2 compilation]# make
make -C /lib/modules/2.6.18-164.10.1.el5/build M=/home/admin/compilation modules
make[1]: Entering directory `/usr/src/kernels/2.6.18-164.10.1.el5-x86_64'
  CC [M] /home/admin/compilation/hello.o
  Building modules, stage 2.
  MODPOST
  CC      /home/admin/compilation/hello.mod.o
  LD [M]  /home/admin/compilation/hello.ko
make[1]: Leaving directory `/usr/src/kernels/2.6.18-164.10.1.el5-x86_64'
[root@testhost2 compilation]#
```

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
```

**Figure 73:** modinfo example

The screenshot shows a terminal window titled "admin@testhost2:/home/admin/compilation". The window contains the following text:

```
[root@testhost2 compilation]# /sbin/modinfo hello.ko
filename:      hello.ko
srcversion:     67A7C9765BA14A0A1C8B6CF
depends:
vermagic:      2.6.18-164.10.1.el5 SMP mod_unload gcc-4.1
[root@testhost2 compilation]#
[root@testhost2 compilation]#
```

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

The screenshot shows a terminal window titled "admin@testhost2:/home/admin/compilation". The window contains the following text:

```
[root@testhost2 compilation]# /sbin/lsmod | grep hello
hello           34432  0
[root@testhost2 compilation]#
[root@testhost2 compilation]#
```

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

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

```
/sbin/rmmmod hello
```

If you take 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

A screenshot of a terminal window titled "admin@testhost2:/home/admin". The window has a standard Linux-style title bar with icons for minimize, maximize, and close. Below the title bar is a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main area of the terminal shows two lines of text: "Mar 25 17:28:31 testhost2 kernel: Hello world." and "Mar 25 17:28:39 testhost2 kernel: Goodbye world.". The terminal window has scroll bars on the right side.

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

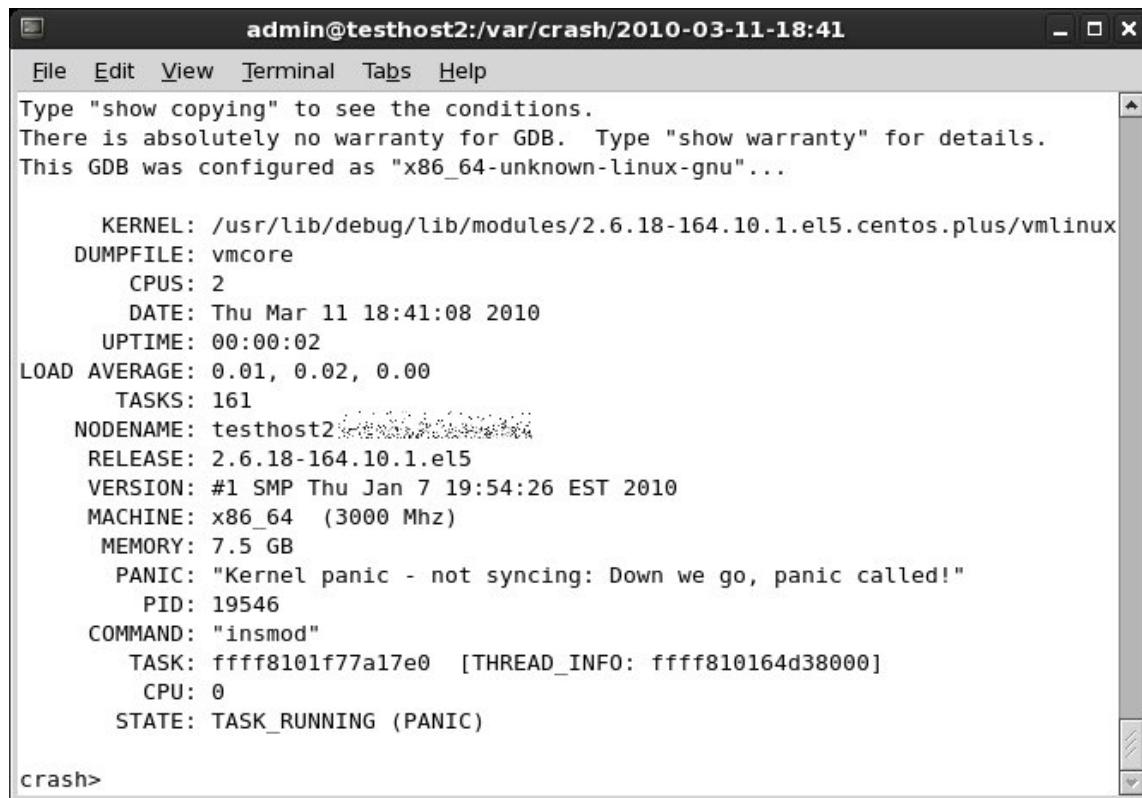
We'll now create a new C program that uses the panic system call on initialization. Not very useful, but good enough for demonstrating the power of crash analysis. Here's the code, we call it *kill-kernel.c*.

```
1  /*
2   * kill-kernel.c - The simplest kernel module to crash kernel.
3   */
4
5 #include <linux/module.h>      /* Needed by all modules */
6 #include <linux/kernel.h>      /* Needed for KERN_INFO */
7
8 int init_module(void)
9 {
10    printk(KERN_INFO "Hello\u00a0world.\u00a0Now\u00a0we\u00a0crash.\n");
11    panic("Down\u00a0we\u00a0go,\u00a0panic\u00a0called!");
12
13    return 0;
14 }
15
16 void cleanup_module(void)
17 {
18    printk(KERN_INFO "Goodbye\u00a0world.\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.

**Figure 76:** Intermediate example crash summary

The screenshot shows a terminal window titled "admin@testhost2:/var/crash/2010-03-11-18:41". The window contains the output of the "crash" command, which provides a summary of a kernel crash. The output includes system configuration details like the kernel version (2.6.18-164.10.1.el5), CPU count (2), and memory size (7.5 GB). It also details the panic event, showing the command "insmod" was run, the task involved (fffff8101f77a17e0), and the state (TASK\_RUNNING). The terminal prompt "crash>" is visible at the bottom.

```
admin@testhost2:/var/crash/2010-03-11-18:41
File Edit View Terminal Tabs Help
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"...

KERNEL: /usr/lib/debug/lib/modules/2.6.18-164.10.1.el5.centos.plus/vmlinu
DUMPFILE: vmcore
CPUS: 2
DATE: Thu Mar 11 18:41:08 2010
UPTIME: 00:00:02
LOAD AVERAGE: 0.01, 0.02, 0.00
TASKS: 161
NODENAME: testhost2
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: "Kernel panic - not syncing: Down we go, panic called!"
PID: 19546
COMMAND: "insmod"
TASK: fffff8101f77a17e0 [THREAD_INFO: fffff810164d38000]
CPU: 0
STATE: TASK_RUNNING (PANIC)

crash>
```

And the backtrace:

**Figure 77:** Intermediate example backtrace

```

VERSION: #1 SMP Thu Jan 7 19:54:26 EST 2010
MACHINE: x86_64 (3000 Mhz)
MEMORY: 7.5 GB
PANIC: "Kernel panic - not syncing: Down we go, panic called!"
PID: 19546
COMMAND: "insmod"
TASK: fffff8101f77a17e0 [THREAD_INFO: fffff810164d38000]
CPU: 0
STATE: TASK_RUNNING (PANIC)

crash> bt
PID: 19546 TASK: fffff8101f77a17e0 CPU: 0 COMMAND: "insmod"
#0 [ffff810164d39e50] panic at ffffffff8009103e
#1 [ffff810164d39f40] init_module at ffffffff8848e02e
#2 [ffff810164d39f50] sys_init_module at ffffffff800a61e5
#3 [ffff810164d39f80] tracesys at ffffffff8005d28d (via system_call)
    RIP: 0000003b912d408a RSP: 00007ffff8f32cd8 RFLAGS: 00000206
    RAX: ffffffffffffffd8 RBX: ffffffff8005d28d RCX: ffffffffffffffff
    RDX: 0000000008c95030 RSI: 000000000001ed58 RDI: 00000000008c95050
    RBP: 000000000001ed58 R8: 0000000000020010 R9: 0000000000000003
    R10: ffffffffffffffd8 R11: 0000000000000206 R12: 00007ffff8f339fb
    R13: 0000000000000003 R14: 0000000008c95050 R15: 00000000000020000
    ORIG_RAX: 00000000000000af CS: 0033 SS: 002b
crash>

```

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 informative, 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.

For more information about writing kernel modules, including benevolent purposes, please consult the [Linux Kernel Module Programming Guide](#).

## 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 - A not so simple kernel module to crash
3   *  kernel.
4   */
5  #include <linux/module.h>      /* Needed by all modules */
6  #include <linux/kernel.h>       /* Needed for KERN_INFO */
7
8  char *p=NULL;
9
10 int init_module(void)
11 {
12     printk(KERN_INFO "We\u2019re\u2019 gonna\u2019 KABOOM\u2019 now!\n");
13
14     *p = 1;
15     return 0;
16 }
17
18 void cleanup_module(void)
19 {
20     printk(KERN_INFO "Goodbye\u2019 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.

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

```
admin@testhost2:/var/crash/2010-03-25-18:30
File Edit View Terminal Tabs Help
process stack pointer: ffff8101dda79c78
current stack base: ffff8101f8a02000

KERNEL: /usr/lib/debug/lib/modules/2.6.18-164.10.1.el5.centos.plus/vmlinux
DUMPFILE: vmcore
CPUS: 2
DATE: Thu Mar 25 18:30:14 2010
UPTIME: 00:00:00
LOAD AVERAGE: 0.72, 0.41, 0.16
TASKS: 136
NODENAME: testhost2
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: "Oops: 0002 [1] SMP " (check log for details)
PID: 0
COMMAND: "swapper"
TASK: ffffff80300ae0 (1 of 2) [THREAD_INFO: ffffff803f2000]
CPU: 0
STATE: TASK_RUNNING
WARNING: panic task not found

crash>
```

Let's digest the stuff:

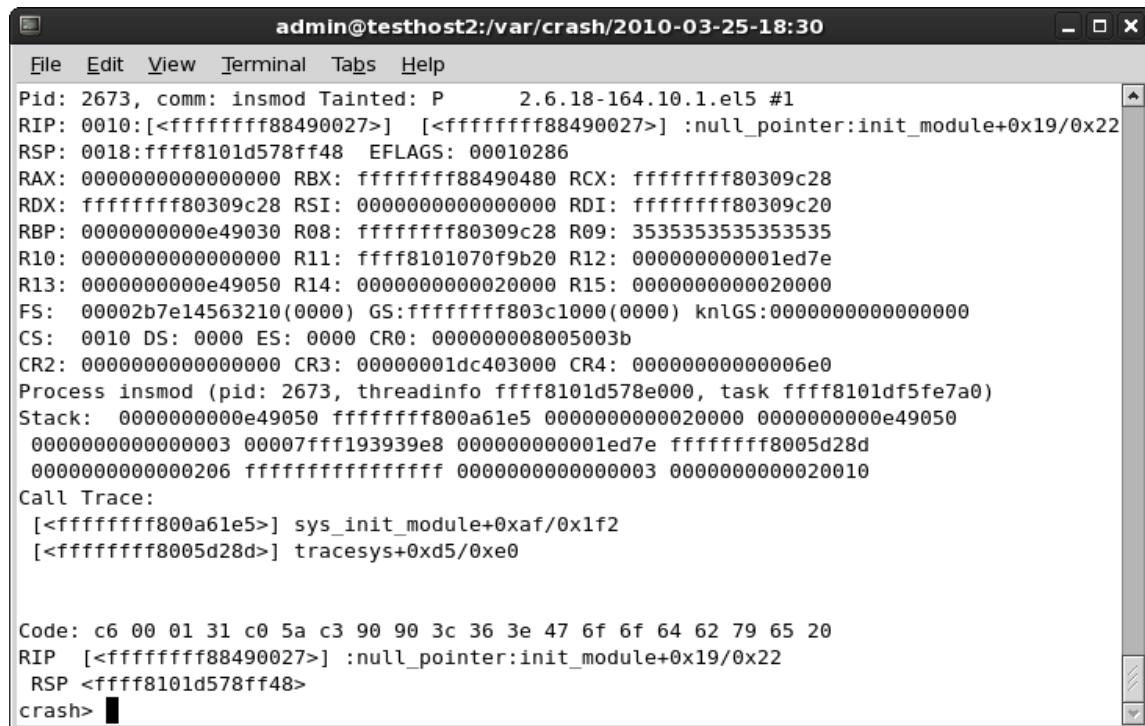
```
PANIC: "Oops: 0002 [1] SMP " (check log for details)
```

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



```
admin@testhost2:/var/crash/2010-03-25-18:30
File Edit View Terminal Tabs Help
Pid: 2673, comm: insmod Tainted: P      2.6.18-164.10.1.el5 #1
RIP: 0010:[<ffffffff88490027>]  [<ffffffff88490027>] :null_pointer:init_module+0x19/0x22
RSP: 0018:ffff8101d578ff48  EFLAGS: 00010286
RAX: 0000000000000000 RBX: ffffffff88490480 RCX: ffffffff80309c28
RDX: ffffffff80309c28 RSI: 0000000000000000 RDI: ffffffff80309c20
RBP: 00000000e49030 R08: ffffffff80309c28 R09: 3535353535353535
R10: 0000000000000000 R11: fffff8101070f9b20 R12: 000000000001ed7e
R13: 00000000e49050 R14: 0000000000020000 R15: 0000000000020000
FS: 00002b7e14563210(0000) GS:ffffffff803c1000(0000) knlGS:0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 000000008005003b
CR2: 0000000000000000 CR3: 00000001dc403000 CR4: 00000000000006e0
Process insmod (pid: 2673, threadinfo fffff8101d578e000, task fffff8101df5fe7a0)
Stack: 00000000e49050 ffffffff800a61e5 0000000000020000 0000000000e49050
00000000000003 00007fff193939e8 000000000001ed7e ffffffff8005d28d
0000000000000206 ffffffff8005d28d 0000000000000003 00000000000020010
Call Trace:
[<ffffffff800a61e5>] sys_init_module+0xaf/0x1f2
[<ffffffff8005d28d>] tracesys+0xd5/0xe0

Code: c6 00 01 31 c0 5a c3 90 90 3c 36 3e 47 6f 6f 64 62 79 65 20
RIP  [<ffffffff88490027>] :null_pointer:init_module+0x19/0x22
RSP <ffff8101d578ff48>
crash>
```

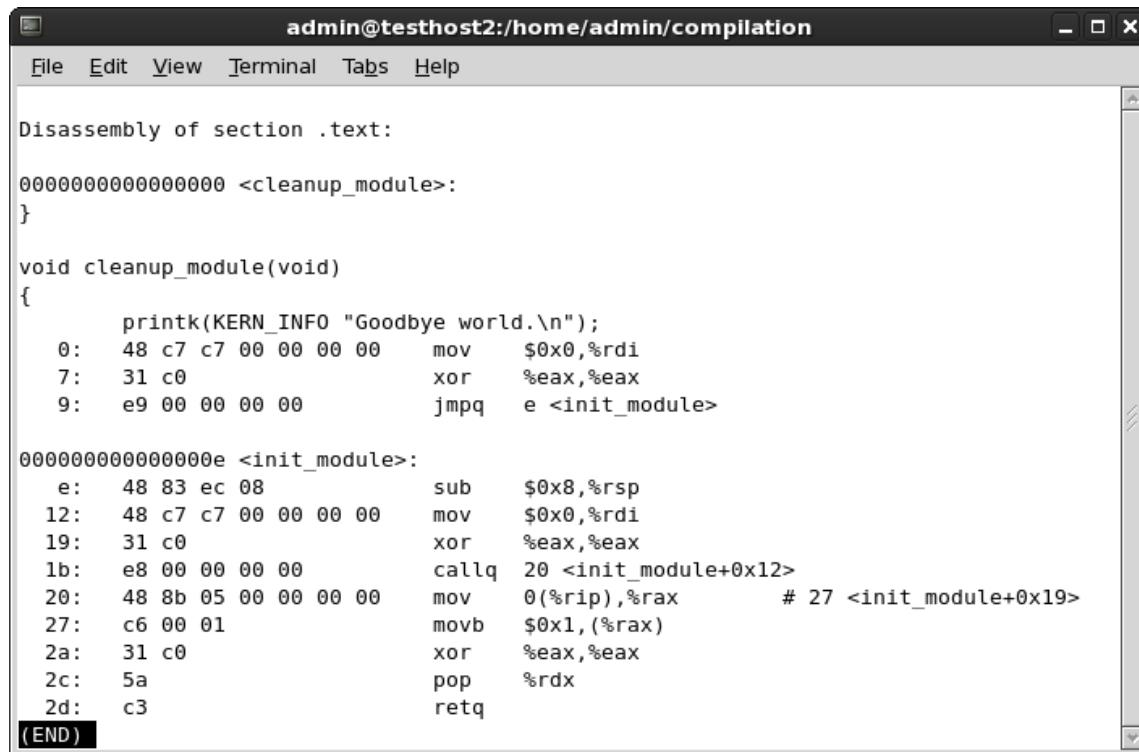
The RIP says **null\_pointer:init\_module+0x19/0x22**. We're making progress here. We know there was a problem with NULL pointer in the init\_module function. Time

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 screenshot shows a terminal window titled "admin@testhost2:/home/admin/compilation". The window contains the assembly code for a kernel module. The code starts with a cleanup function, followed by an init function. The init function contains several instructions, including a call to another function at offset 0x12, which is marked with a comment "# 27 <init\_module+0x19>".

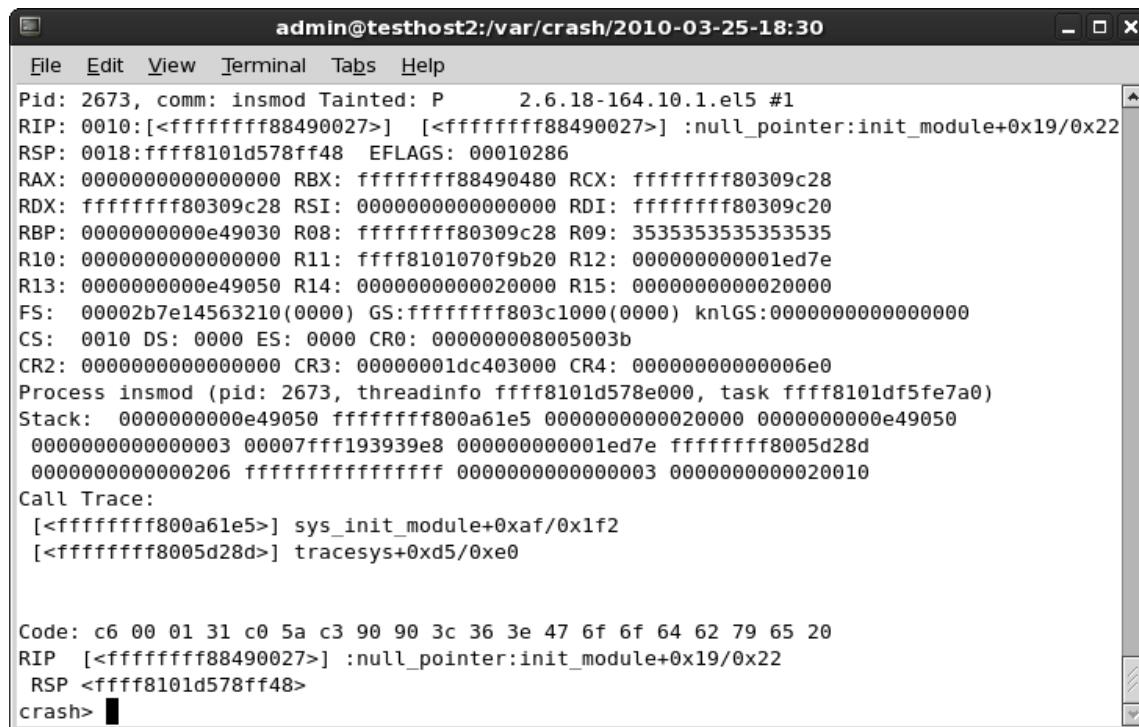
```
Disassembly of section .text:  
0000000000000000 <cleanup_module>:  
}  
  
void cleanup_module(void)  
{  
    printk(KERN_INFO "Goodbye world.\n");  
    0: 48 c7 c7 00 00 00 00    mov    $0x0,%rdi  
    7: 31 c0                  xor    %eax,%eax  
    9: e9 00 00 00 00          jmpq   e <init_module>  
  
000000000000000e <init_module>:  
    e: 48 83 ec 08            sub    $0x8,%rsp  
   12: 48 c7 c7 00 00 00 00  mov    $0x0,%rdi  
   19: 31 c0                  xor    %eax,%eax  
   1b: e8 00 00 00 00          callq  20 <init_module+0x12>  
   20: 48 8b 05 00 00 00 00  mov    0(%rip),%rax      # 27 <init_module+0x19>  
   27: c6 00 01                movb   $0x1,(%rax)  
   2a: 31 c0                  xor    %eax,%eax  
   2c: 5a                      pop    %rdx  
   2d: c3                      retq  
(END)
```

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>.

```
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



The screenshot shows a terminal window titled "admin@testhost2:/var/crash/2010-03-25-18:30". The window contains the following text:

```

File Edit View Terminal Tabs Help
Pid: 2673, comm: insmod Tainted: P      2.6.18-164.10.1.el5 #1
RIP: 0010:[<ffffffff88490027>]  [<ffffffff88490027>] :null_pointer:init_module+0x19/0x22
RSP: 0018:ffff8101d578ff48 EFLAGS: 00010286
RAX: 0000000000000000 RBX: ffffffff88490480 RCX: ffffffff80309c28
RDX: ffffffff80309c28 RSI: 0000000000000000 RDI: ffffffff80309c20
RBP: 0000000000e49030 R08: ffffffff80309c28 R09: 3535353535353535
R10: 0000000000000000 R11: fffff8101070f9b20 R12: 000000000001ed7e
R13: 0000000000e49050 R14: 0000000000020000 R15: 0000000000020000
FS: 00002b7e14563210(0000) GS:ffffffffff803c1000(0000) knlGS:0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 00000008005003b
CR2: 0000000000000000 CR3: 00000001dc403000 CR4: 0000000000006e0
Process insmod (pid: 2673, threadinfo ffff8101d578e000, task ffff8101df5fe7a0)
Stack: 0000000000e49050 ffffffff800a61e5 0000000000020000 0000000000e49050
0000000000000003 00007fff193939e8 000000000001ed7e ffffffff8005d28d
0000000000000206 ffffffff8005d28d 0000000000000003 00000000000020010
Call Trace:
[<ffffffff800a61e5>] sys_init_module+0xaf/0x1f2
[<ffffffff8005d28d>] tracesys+0xd5/0xe0

Code: c6 00 01 31 c0 5a c3 90 90 3c 36 3e 47 6f 6f 64 62 79 65 20
RIP  [<ffffffff88490027>] :null_pointer:init_module+0x19/0x22
RSP <ffff8101d578ff48>
crash>
```

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!

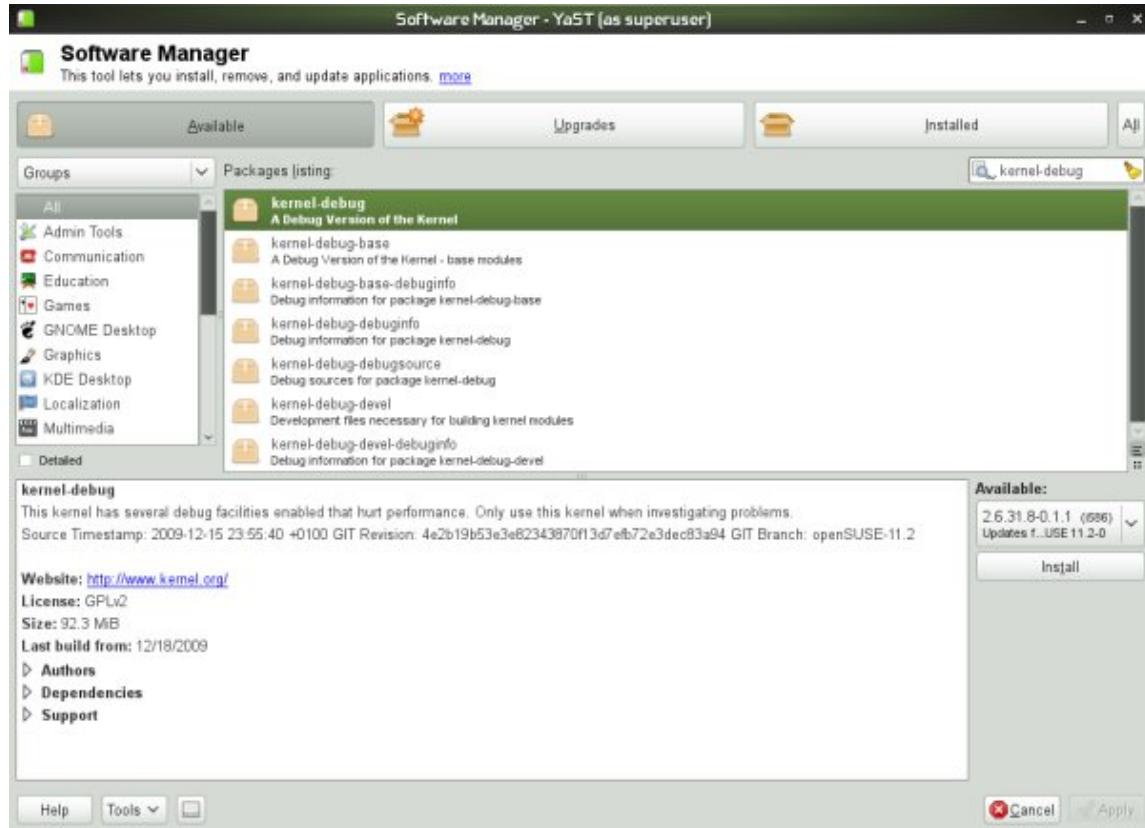
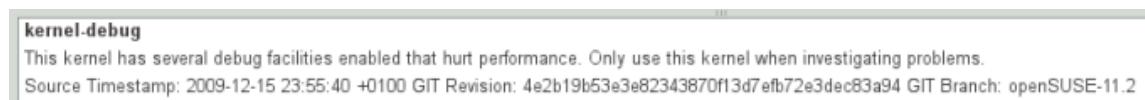
In some cases, you will see a problem that is not immediately apparent from looking at the sources. This means you will need to work your way through the *vmcore*, carefully

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](#) 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.

**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.

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 simplify 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's 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:** kerneloops.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.

**Figure 85:** kerneloops.org example

## About kerneloops.org

kerneloops.org is a website that tries to help the developers of the Linux kernel by collecting so-called *oopses*, which are the crash signatures of the Linux kernel. The collected oopses are processed statistically to present information for the kernel developers, such as

- Which crash signatures occur the most? (and thus need to be fixed most urgently)
- When did a certain crash signature show up first?
- Which API functions are the most error prone?

Ooops are collected from the linux-kernel mailing list (and a few related lists), the bugzilla.kernel.org bugzilla and from the client application that you can download from this page.

Kernel oops count		Current top 10 issues		
2.6.34-rc	0	1. dquot_claim_space	18471	Interaction between EXT4 and the quota code
2.6.33-release	11	2. rcu_exit_nohz	7273	
2.6.33-rc	1765	3. rcu_enter_nohz	6436	
2.6.32-release	4972	4. nv_post_event	2497	
2.6.32-rc	1093	5. azx_send_cmd	956	
2.6.31-release	729416	6. radeon_object_create	655	
2.6.31-rc	7697	7. usbcam_videobuf_queue	597	
2.6.30-release	152412	8. native_apic_write_dummy	395	
2.6.30-rc	2564	9. memcpy_tolovecend	361	
2.6.29-release	217830			
2.6.29-rc	20149			

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.

**Figure 86:** kerneloops.org example - continued

History for native_apic_write_dummy		Detailed oops backtraces	
2.6.32-rc8-git5	4 x	Oops number:	#1291780 (3 times)
2.6.32-rc8-git4	1 x		#1302565 (9 times)
2.6.32-rc8-git1	4 x	Versions	2x 2.6.31.12-174.2.3.fc12.i686 1x 2.6.31.12-174.2.3.fc12.i686.PAE
2.6.32-rc8	8 x		6x 2.6.31.12-174.2.3.fc12.i686 2x 2.6.31.12-174.2.3.fc12.i686.PAE 1x 2.6.31.6-166.fc12.i686
2.6.32-rc7-git2	4 x	Function	native_apic_write_dummy
2.6.32-rc7	2 x	Process	native_apic_write_dummy 3x swapper
2.6.32-rc6	4 x		native_apic_write_dummy 9x swapper
2.6.32-rc5	6 x		
2.6.32-rc3	2 x		
2.6.32-rc2	1 x		
2.6.32.1	27 x		
2.6.32	20 x		
2.6.31-rc8	1 x		
2.6.31-rc7	1 x		
2.6.31-rc5-git3	32 x		
2.6.31-rc5-git2	36 x		
2.6.31-rc5	10 x		
2.6.31-rc4-git3	18 x	Backtrace	warn_slowpath_common ? native_apic_write_dummy warn_slowpath_null *native_apic_write_dummy intel_init_thermal ? mce_init mce_intel_feature_init mce_cpu_features
2.6.31-rc4-git2	14 x		warn_slowpath_common ? native_apic_write_dummy warn_slowpath_null *native_apic_write_dummy intel_init_thermal ? mce_init mce_intel_feature_init mce_cpu_features
2.6.31-rc3-git5	14 x		
2.6.31-rc3-git4	1 x		
2.6.31-rc3	16 x		

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!

**Figure 87:** kerneloops.org example code

```
251 {
252     WARN_ON_ONCE((cpu_has_apic && !disable_apic));
253     return 0;
254 }
255
256 /*
257  * right after this call apic->write/read doesn't do anything
258  * note that there is no restore operation it works one way
259 */
260 void apic_disable(void)
261 {
262     apic->read = native_apic_read_dummy;
263     apic->write = native_apic_write_dummy;
264 }
265
266 void native_apic_wait_icr_idle(void)
267 {
268     while (apic_read(APIC_ICR) & APIC_ICR_BUSY)
269         cpu_relax();
270 }
271
272 u32 native_safe_apic_wait_icr_idle(void)
273 {
274     u32 send_status;
275     int timeout;
276
277     timeout = 0;
278     do {
279         send_status = apic_read(APIC_ICR) & APIC_ICR_BUSY;
280         if (!send_status)
281             break;
282         udelay(100);
283     } while (timeout++ < 1000);
284
285     return send_status;
286 }
287
```

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

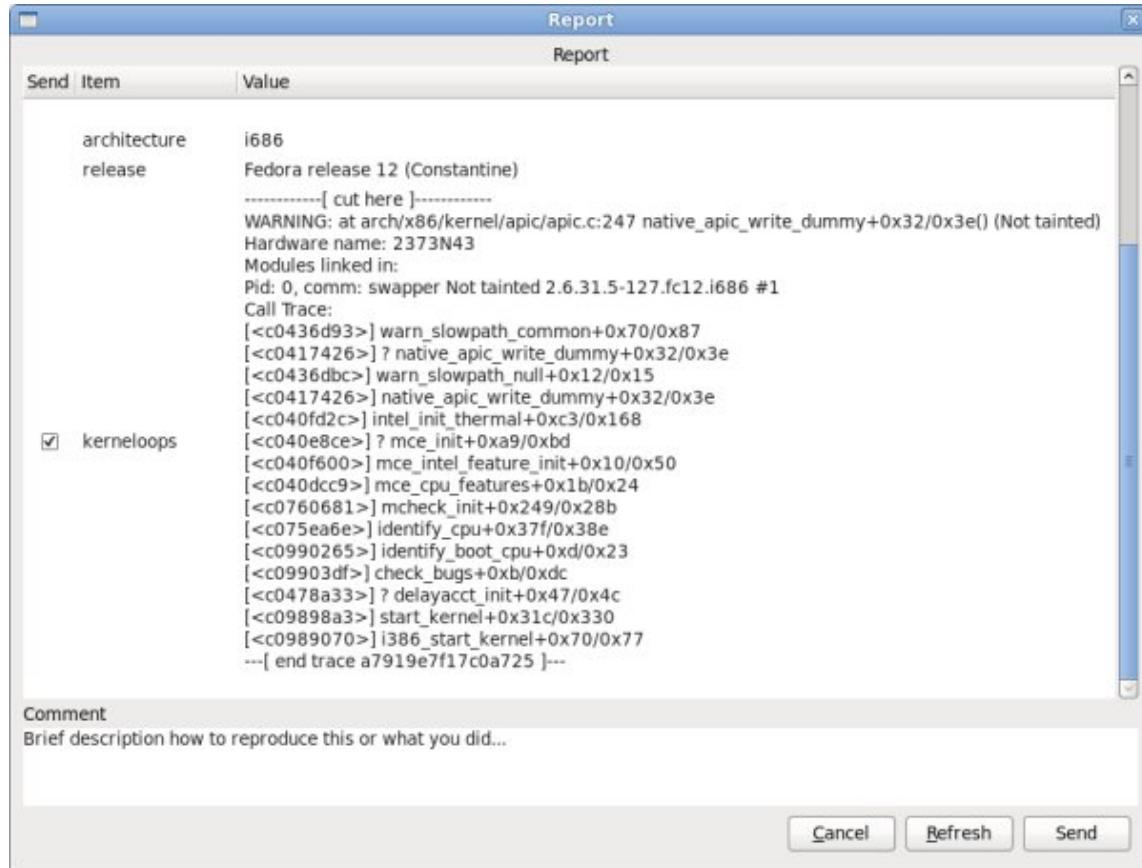
For example, Fedora 12 uses the Automatic Bug Reporting Tool (ABRT), which collects crash data, runs a report and then sends it for analysis with the developers. For more details, you may want to read the [Wiki](#). Beforehand, Fedora 11 used kerneloops utility,

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](#):

**Figure 88:** Kernel crash report in Fedora 11

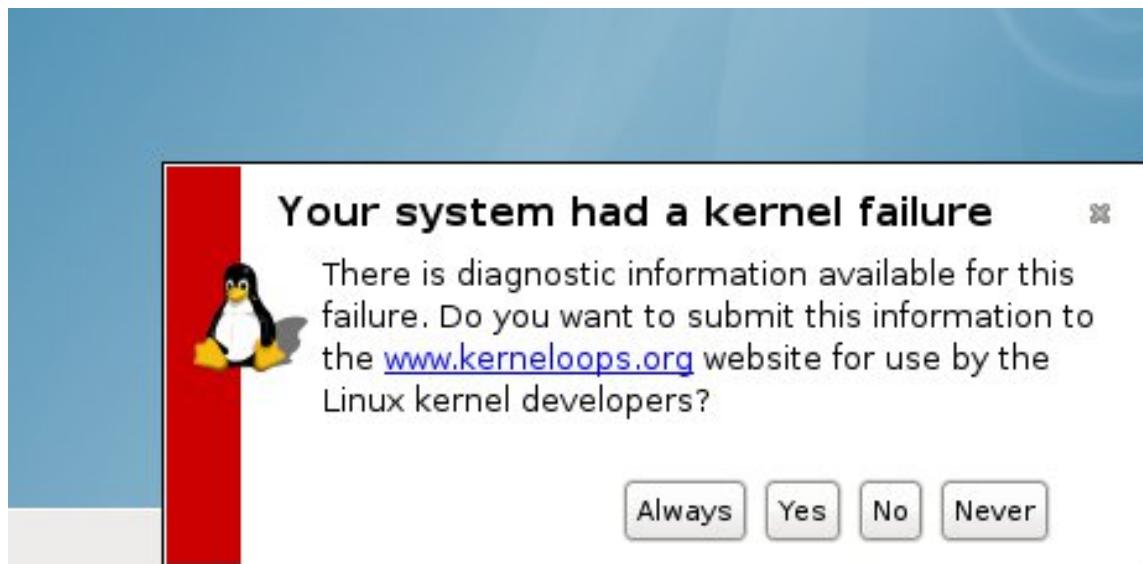


And more recently in [Fedora 12](#):

**Figure 89:** Kernel crash report in Fedora 12

And here's [Debian 5.03 Lenny](#):

**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 someone 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 found 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 as an example:

**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:

### 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 consider 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

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.

## Part V

# Appendix

## 29 Kdump

This section contains a few more details about Kdump. Namely, it provides instructions 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 documentation 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:

### 29.3.1 Download the package

You can look for the package either on the [official site](#) or download a version from [kernel.org](#), 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

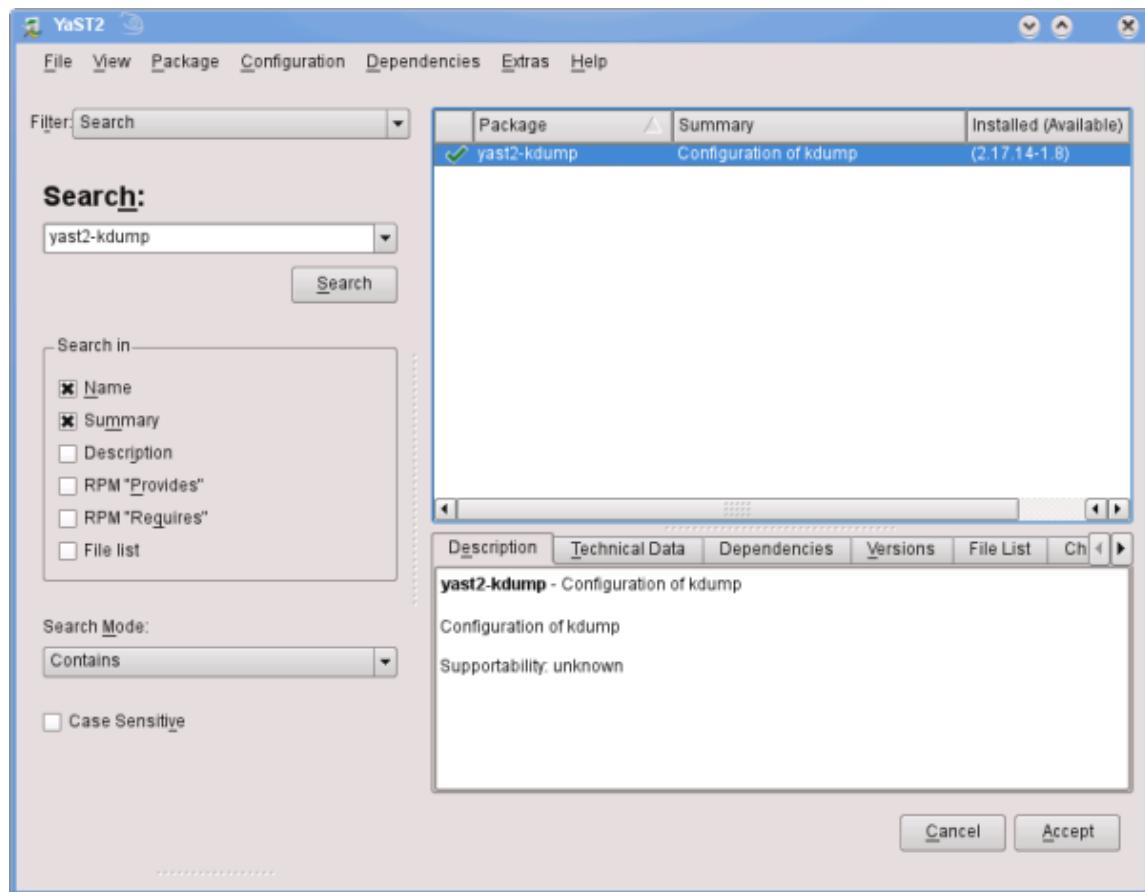
Please make sure you download the right package that matches your Kdump version. Otherwise, when you try to run Kexec, you are likely to see strange errors, similar to Possible errors (14.2.1) we have seen earlier during the Kdump testing.

## 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.

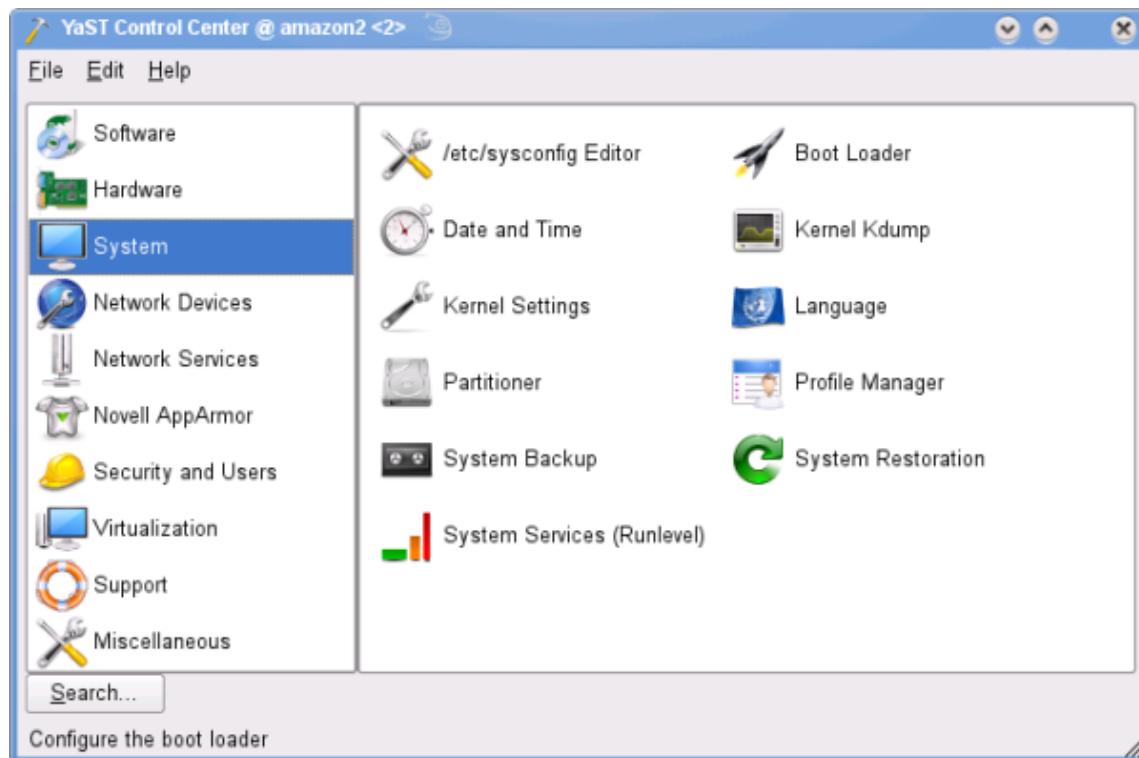
Nevertheless, I thought it would be useful to mention this. Indeed, you can see a number of screenshots taken on an openSUSE 11.1 machine, demonstrating the installation and the use of the *yast-kdump* package.

**Figure 92:** *yast2-kdump* package installation

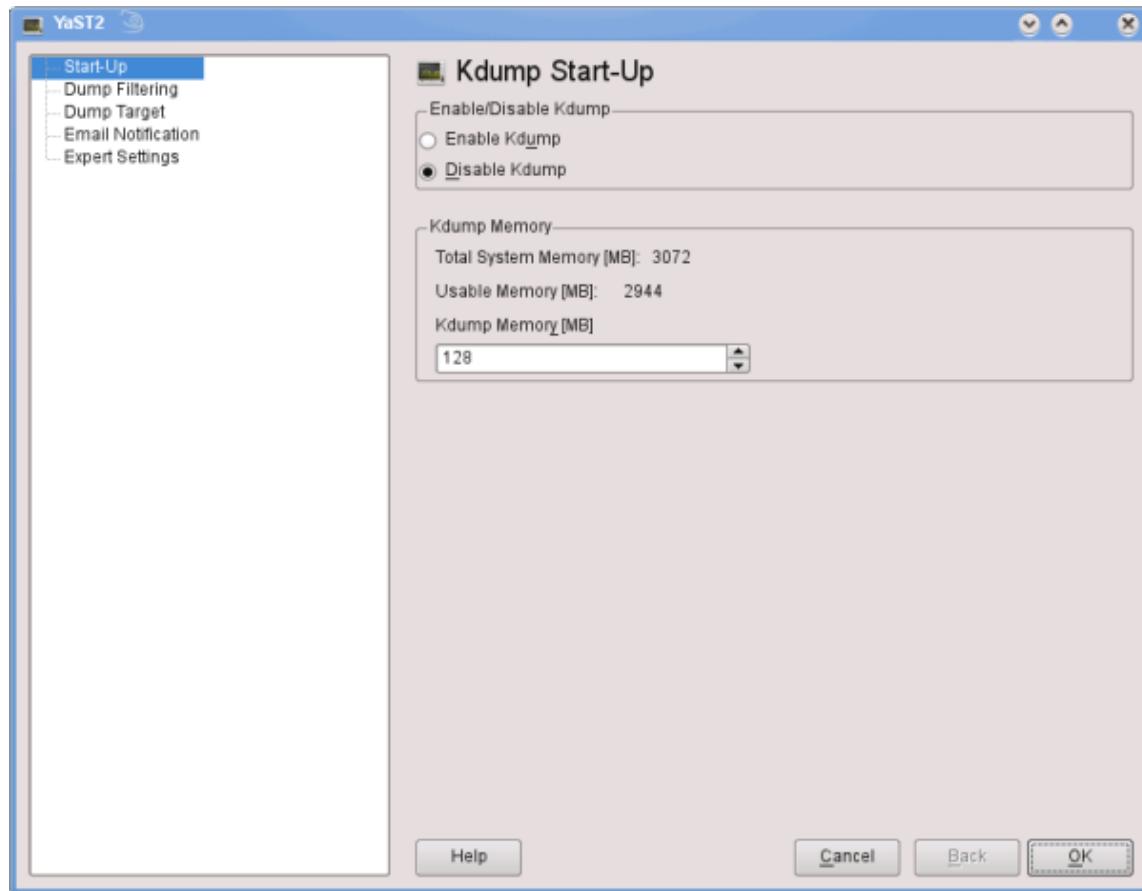


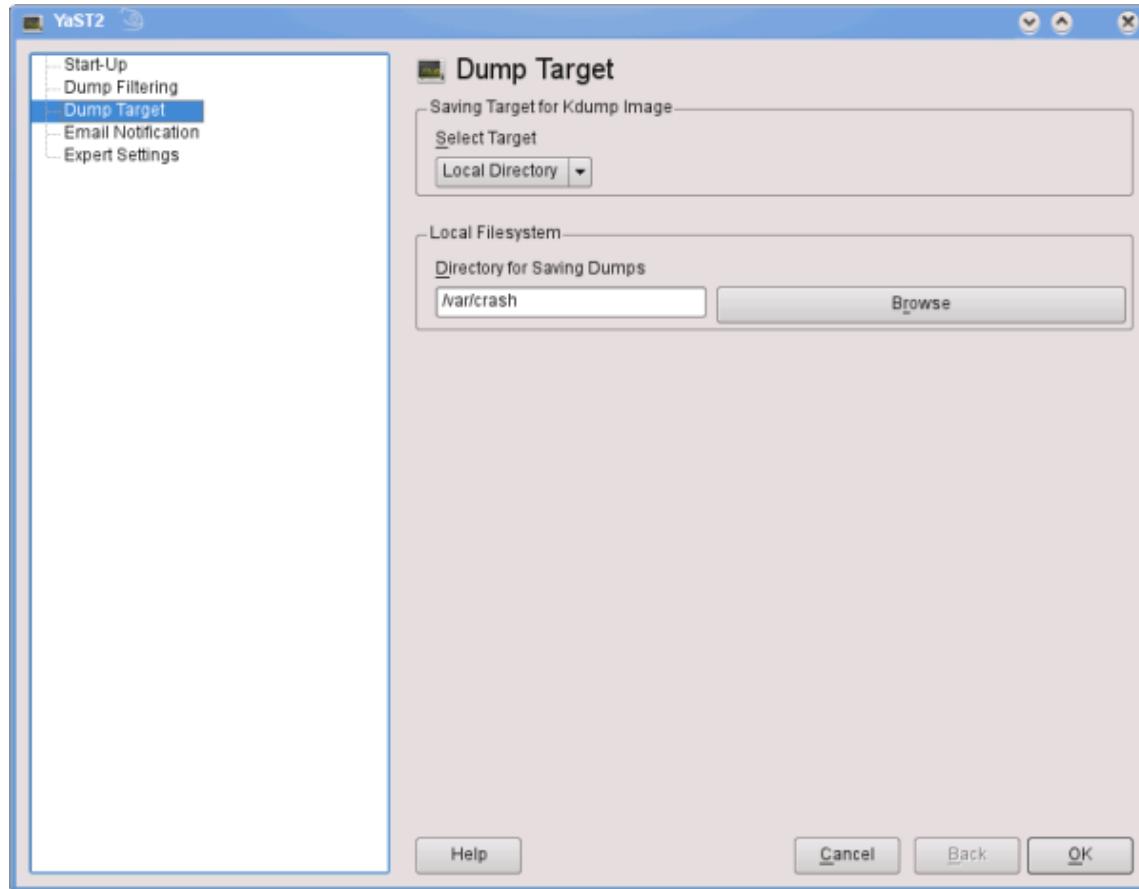
After the installation, you can find the module in the System sub-menu. It's called Kernel Kdump.

**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.

**Figure 94:** Kdump configuration via YaST module

**Figure 95:** Kdump configuration via YaST module - continued

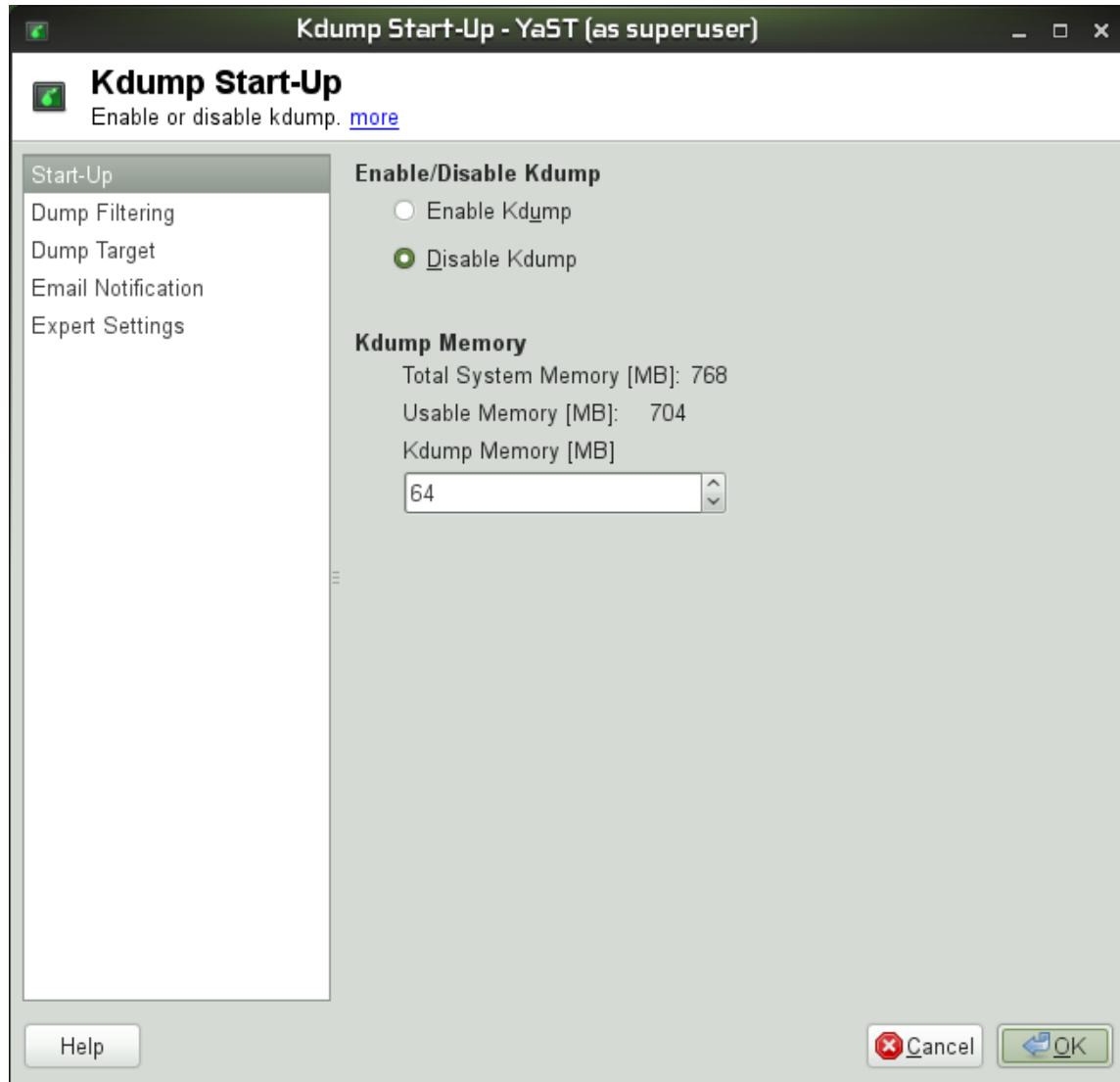
## 29.5 SUSE (and openSUSE) 11.X setup

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.

### 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.

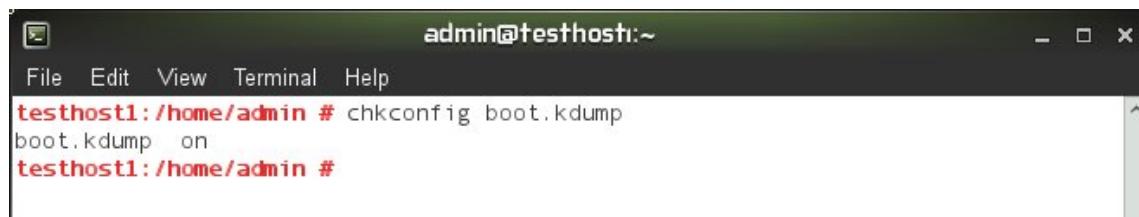
**Figure 96:** Kdump startup configuration via YaST



## Kdump service

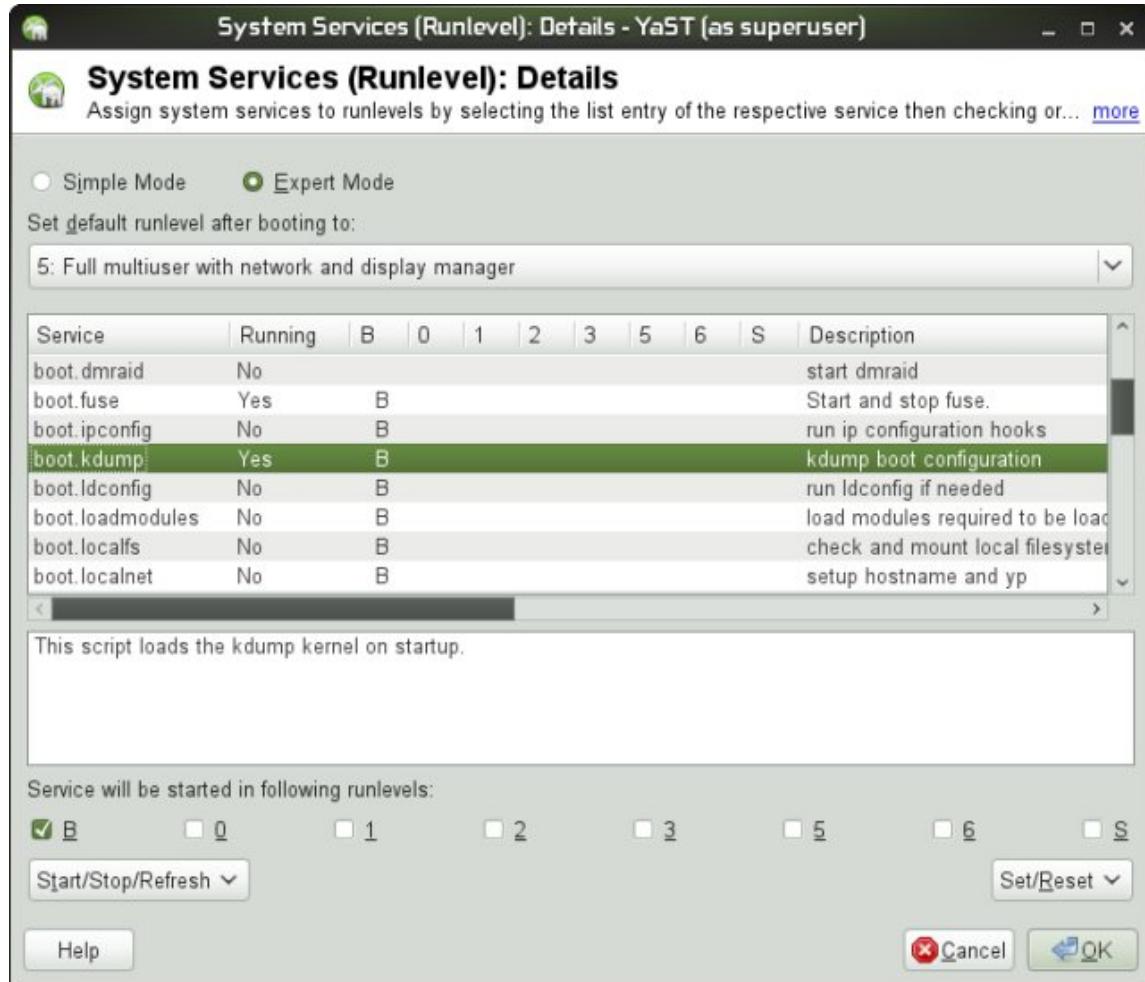
Kdump service has changed. It is no longer called *kdump*, it's called *boot.kdump* and is invoked during the boot. This means that you will have to adjust the usage of *chkconfig* for enabling/disabling Kdump. The new Kdump startup scripts make it more similar to LKCD.

**Figure 97:** boot.kdump chkconfig command



The screenshot shows a terminal window titled "admin@testhost1:~". The window contains the following text:  
File Edit View Terminal Help  
testhost1:/home/admin # chkconfig boot.kdump  
boot.kdump on  
testhost1:/home/admin #

You can also use the System Services module in YaST:

**Figure 98:** Runlevel configuration via YaST

## Memory allocation syntax

The memory allocation syntax has also changed. Although you can use the old `crashkernel=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 than just the starting point for the allocation. It's nothing cardinal, but worth paying attention to.

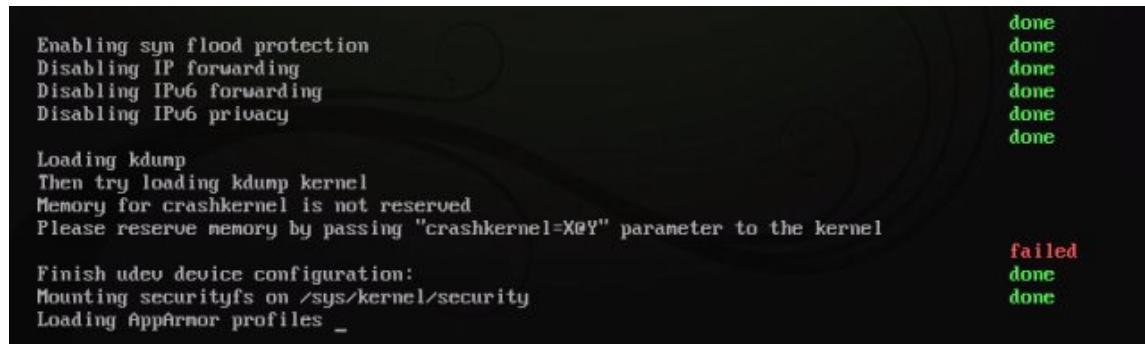
**Figure 99:** Kdump GRUB syntax change

```
splash=silent quiet showopts crashkernel=128M-:64M@16M vga=0x332  
2.6.31.8-0.1-desktop
```

### 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

The screenshot shows a terminal window with a black background and white text. It displays the following log output:

```
Enabling syn flood protection done
Disabling IP forwarding done
Disabling IPv6 forwarding done
Disabling IPv6 privacy done
Loading kdump
done
Then try loading kdump kernel
done
Memory for crashkernel is not reserved
done
Please reserve memory by passing "crashkernel=XeY" parameter to the kernel
failed
Finish udev device configuration:
done
Mounting securityfs on /sys/kernel/security
done
Loading AppArmor profiles _
```

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

```
kexec -p
```

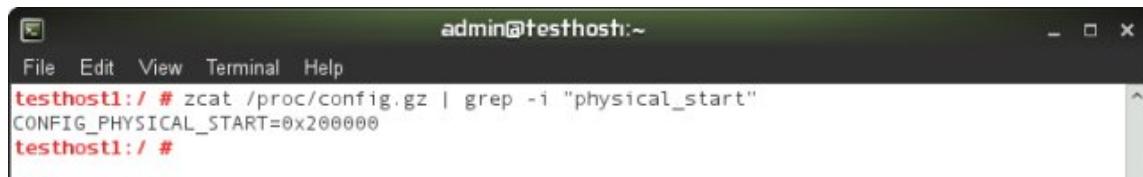
**Figure 101:** Kexec command line error

The screenshot shows a terminal window titled "admin@testhost1:~". The window has a standard title bar with icons for minimize, maximize, and close. The menu bar includes "File", "Edit", "View", "Terminal", and "Help". The terminal area displays the following text:  
testhost1:/home/admin # kexec -p  
Memory for crashkernel is not reserved  
Please reserve memory by passing "crashkernel=X@Y" parameter to the kernel  
Then try loading kdump kernel  
testhost1:/home/admin #

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 architecture. 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

The screenshot shows a terminal window titled "admin@testhost1:~". The window has a standard title bar with icons for minimize, maximize, and close. The menu bar includes "File", "Edit", "View", "Terminal", and "Help". The terminal area displays the following text:  
testhost1:/ # zcat /proc/config.gz | grep -i "physical\_start"  
CONFIG\_PHYSICAL\_START=0x200000  
testhost1:/ #

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.

**Figure 103:** Kdump working after reconfigured physical start value

```
00001-part1 /root
Nothing to delete in /var/crash.
Saving dump          #####-----| 35%
```

### 29.5.3 Other changes

#### Uncompressed kernel images

The new Kdump can work 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 ways<sup>23</sup>.

#### 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.

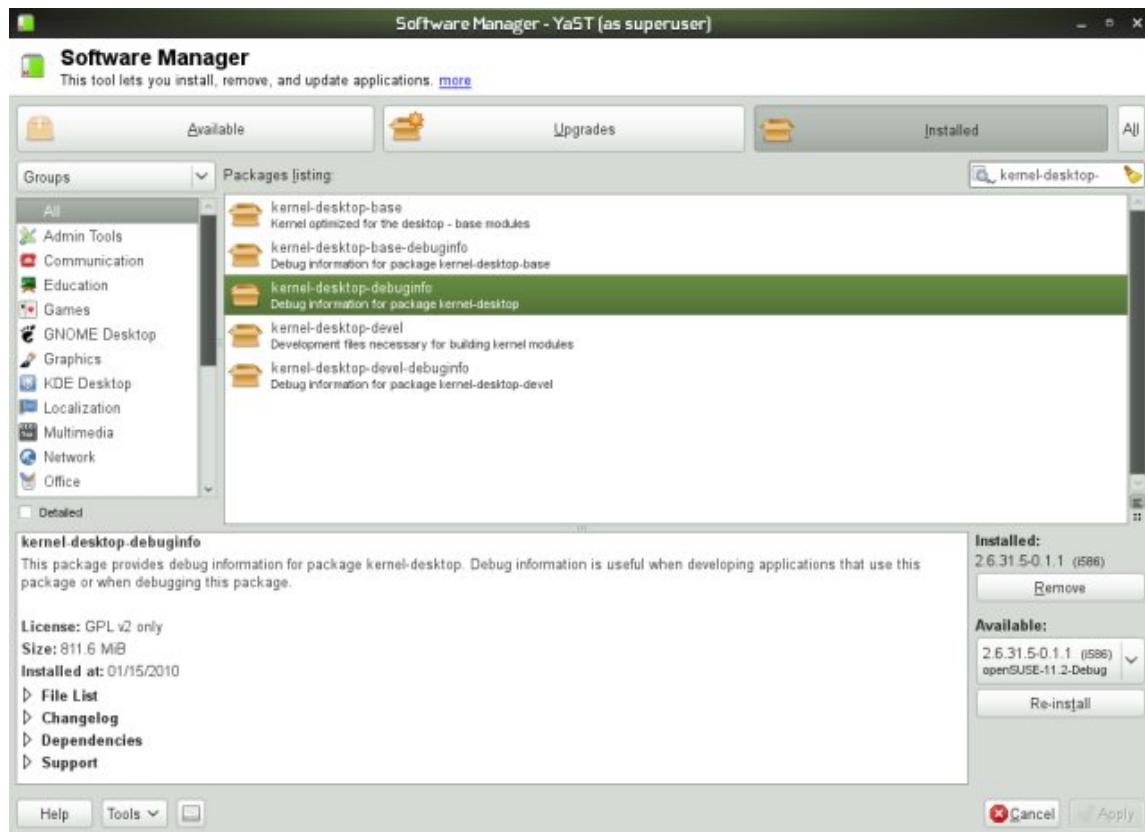
**Figure 104:** debuginfo package missing

```
Nothing to delete in /home/crash.
Saving dump          Finished.
Generating README    Finished.
Copying System.map   Finished.
Copying kernel       Finished.
INFO: Cannot find debug information: Unable to find debuginfo file.
[ 27.554598] Restarting system.
```

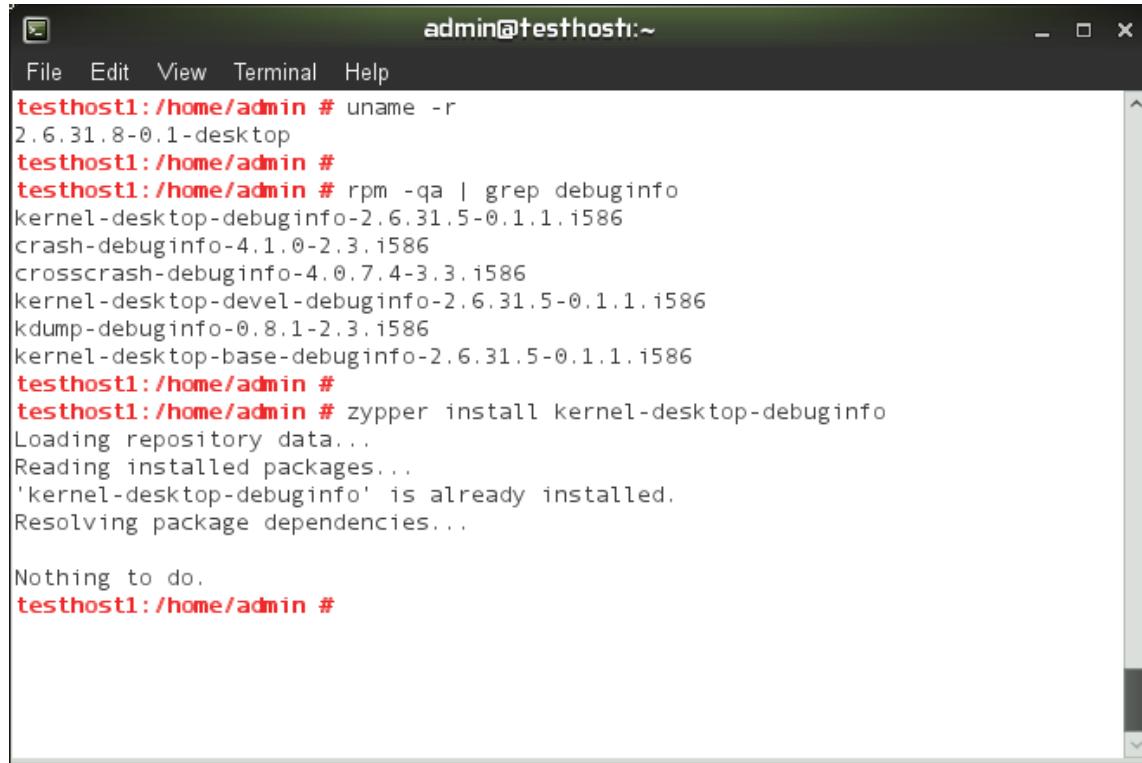
Available package in the repositories:

---

<sup>23</sup>See Crash Collection (III).

**Figure 105:** Available kernel debuginfo package in the repository

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

**Figure 106:** Kernel debuginfo package installation status

The screenshot shows a terminal window titled "admin@testhost1:~". The terminal displays the following command-line session:

```
File Edit View Terminal Help
testhost1:/home/admin # uname -r
2.6.31.8-0.1-desktop
testhost1:/home/admin #
testhost1:/home/admin # rpm -qa | grep debuginfo
kernel-desktop-debuginfo-2.6.31.5-0.1.1.i586
crash-debuginfo-4.1.0-2.3.i586
crosscrash-debuginfo-4.0.7.4-3.3.i586
kernel-desktop-devel-debuginfo-2.6.31.5-0.1.1.i586
kdump-debuginfo-0.8.1-2.3.i586
kernel-desktop-base-debuginfo-2.6.31.5-0.1.1.i586
testhost1:/home/admin #
testhost1:/home/admin # zypper install kernel-desktop-debuginfo
Loading repository data...
Reading installed packages...
'kernel-desktop-debuginfo' is already installed.
Resolving package dependencies...

Nothing to do.
testhost1:/home/admin #
```

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 soon<sup>24</sup>.

## 30 Crash

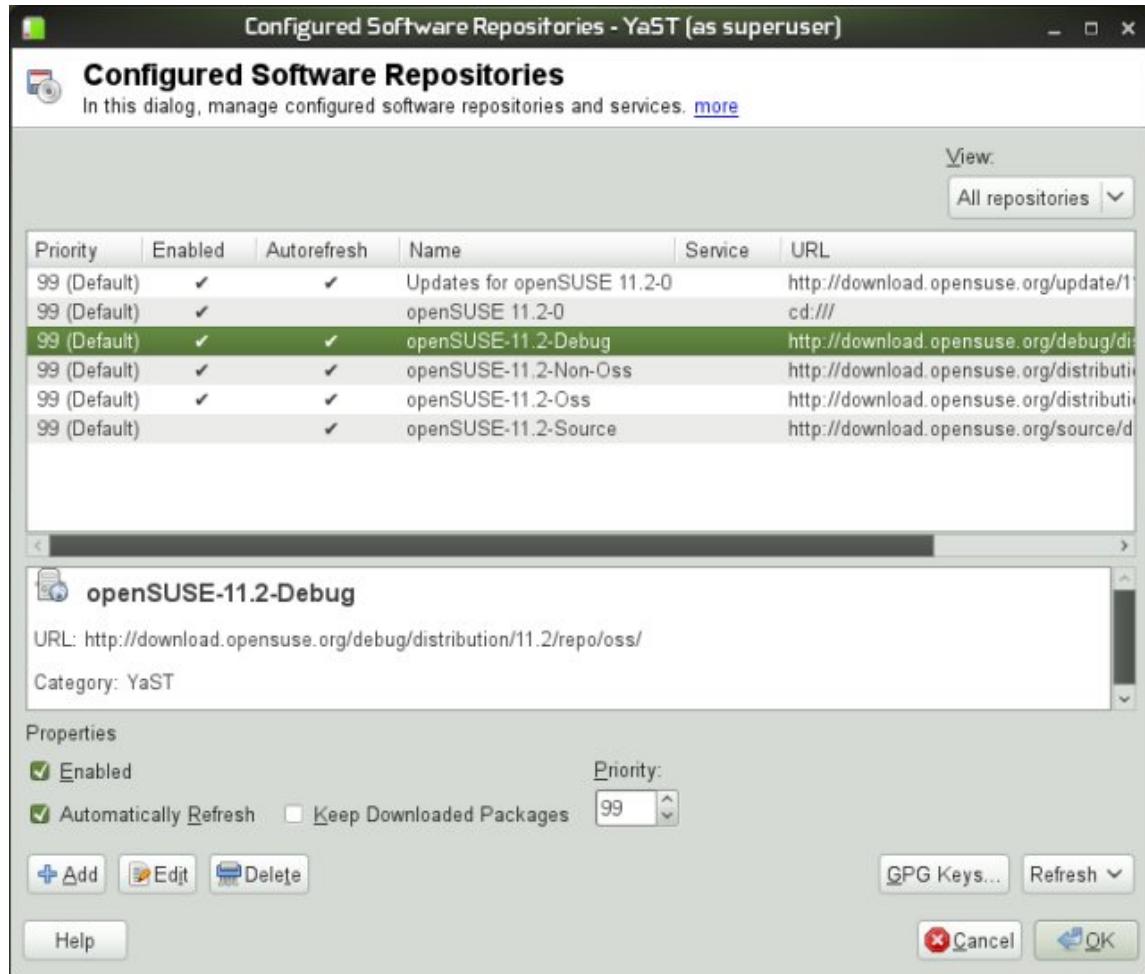
### 30.1 Enable debug repositories

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

<sup>24</sup>No change at the time this book was written.

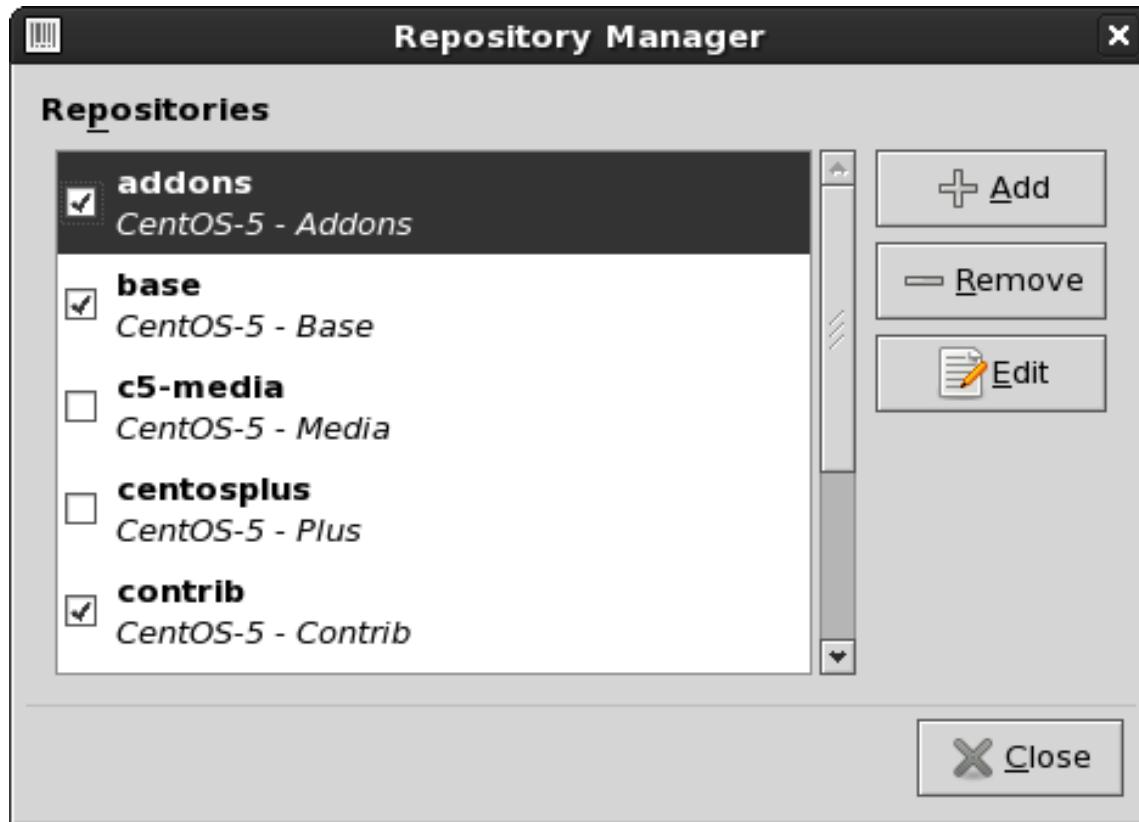
of SUSE and RedHat, debug repositories are enabled by default. However, in openSUSE, the repositories are available, but disabled, whereas in CentOS, they are missing entirely; you will have to add them manually.

**Figure 107:** Enabling Debug repository in openSUSE 11.2

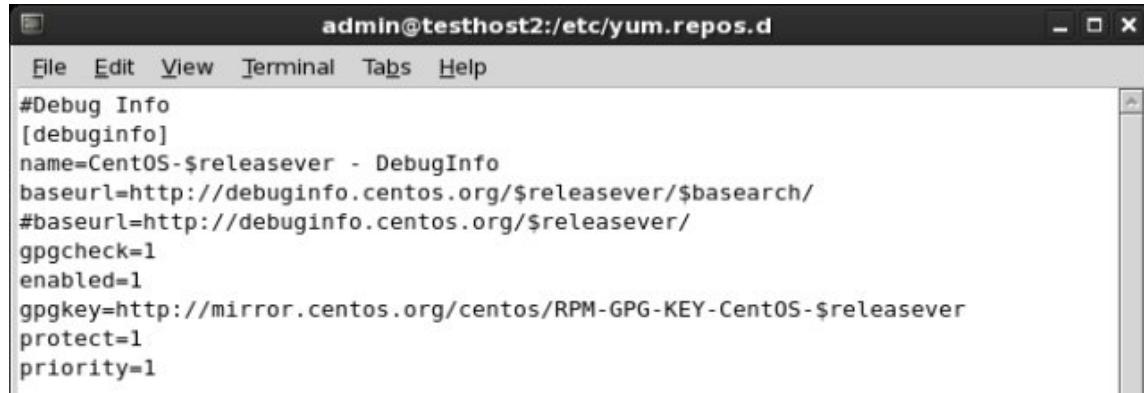


### 30.1.1 Enable repositories in CentOS

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

**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](#) 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**.

**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=1
gpgkey=http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-$releasever
protect=1
priority=1
```

```
#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-
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 work<sup>25</sup>. 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:

<sup>25</sup>This may change at any time.

```
#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)<sup>26</sup>, 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](#), 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 using LKCD. In general, you will need not use the tool manually, because the *lkcd save* 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.

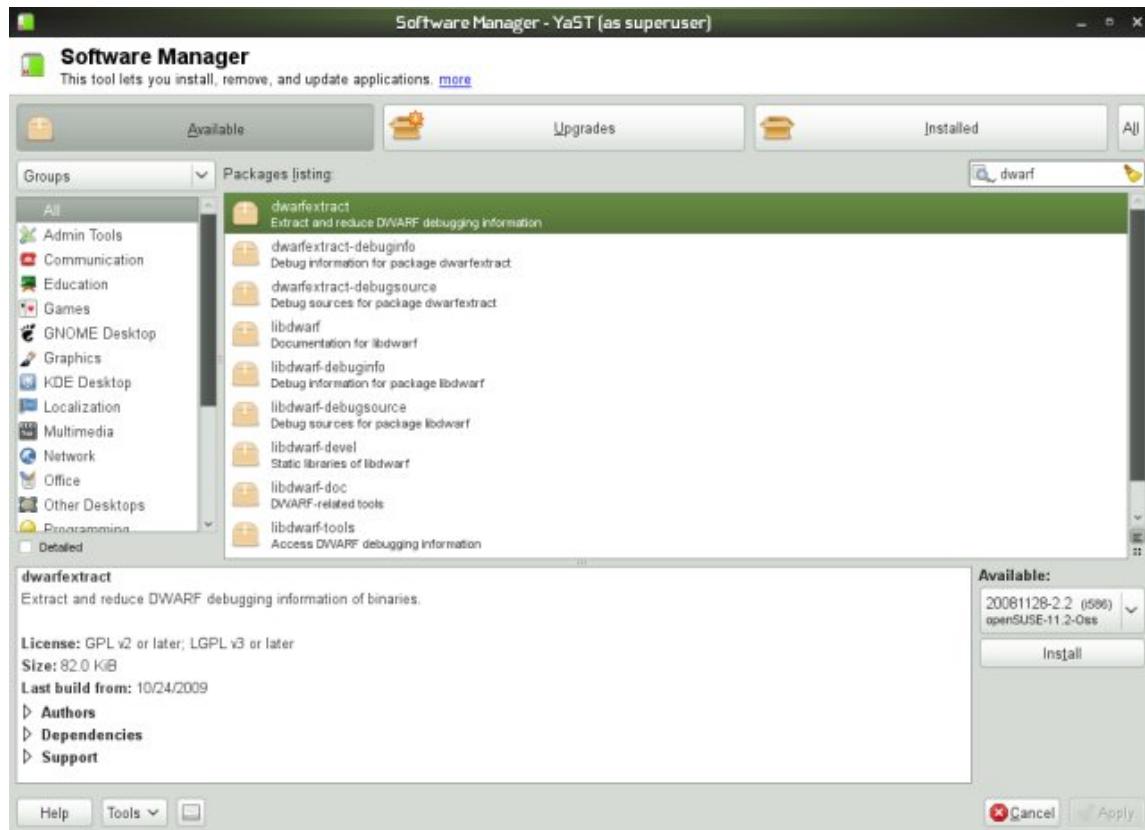
<sup>26</sup>You will need to toggle *enabled=0* to *enabled=1* in the file or run the Repository Manager and select the Debug repository before you can start searching and installing packages.

### 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](#) in the future.

**Figure 110:** dwarfextract installation via YaST



The usage is as follows:

```
dwarfextract vmlinuz <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](#) documentation. You may also want to read the somewhat [older](#) howto on [faqs.org](#). Furthermore, there's a very detailed guide in PDF format is available (direct link): [lcrash HOWTO](#). And that's all. If your system is setup correctly, lcrash should load:

**Figure 111:** lcrash example

```
Loading type info (Kernotypes) ... Done.
Loading kernel symbol information ... Done.
Initialize virttop address translator... Done.
Initialize dump specific data ... Done.
Version of map,dump and types:
    harvey_Thu_Mar__6_20_38_33.UTC_2008
Loading ksyms from dump ..... Done.

DUMP INFORMATION:

    architecture: x86_64
        byte order: little
        pointer size: 64
    bytes per word: 8

    kernel release: 2.6.27.1
        memory size: 9261023232 (8G 640M 0K 0Byte)
    num phys pages: 2260992
    number of cpus: 2

>> bt
=====
STACK TRACE FOR TASK: 0x10014bc5610(bash)

0 sysrq_handle_crashdump+140 [0xfffffffffa00e8c3c]
1 __handle_sysrq_nolock+147 [0xfffffffff80273ea3]
2 handle_sysrq+56 [0xfffffffff80273fa8]
3 write_sysrq_trigger+53 [0xfffffffff801ca405]
4 vfs_write+244 [0xfffffffff8018dfe4]
5 sys_write+157 [0xfffffffff8018e23d]
6 system_call+124 [0xfffffffff801106c4]
=====
>> █
```

## 31 Other tools

### 31.1 gdb-kdump

gdb-kdump is a helper script that you can use instead of crash, although you will experience 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 *vmlinu*x file.

```
gdb-kdump vmcore
```

Here's a sample output:

**Figure 112:** gdb-kdump sample run

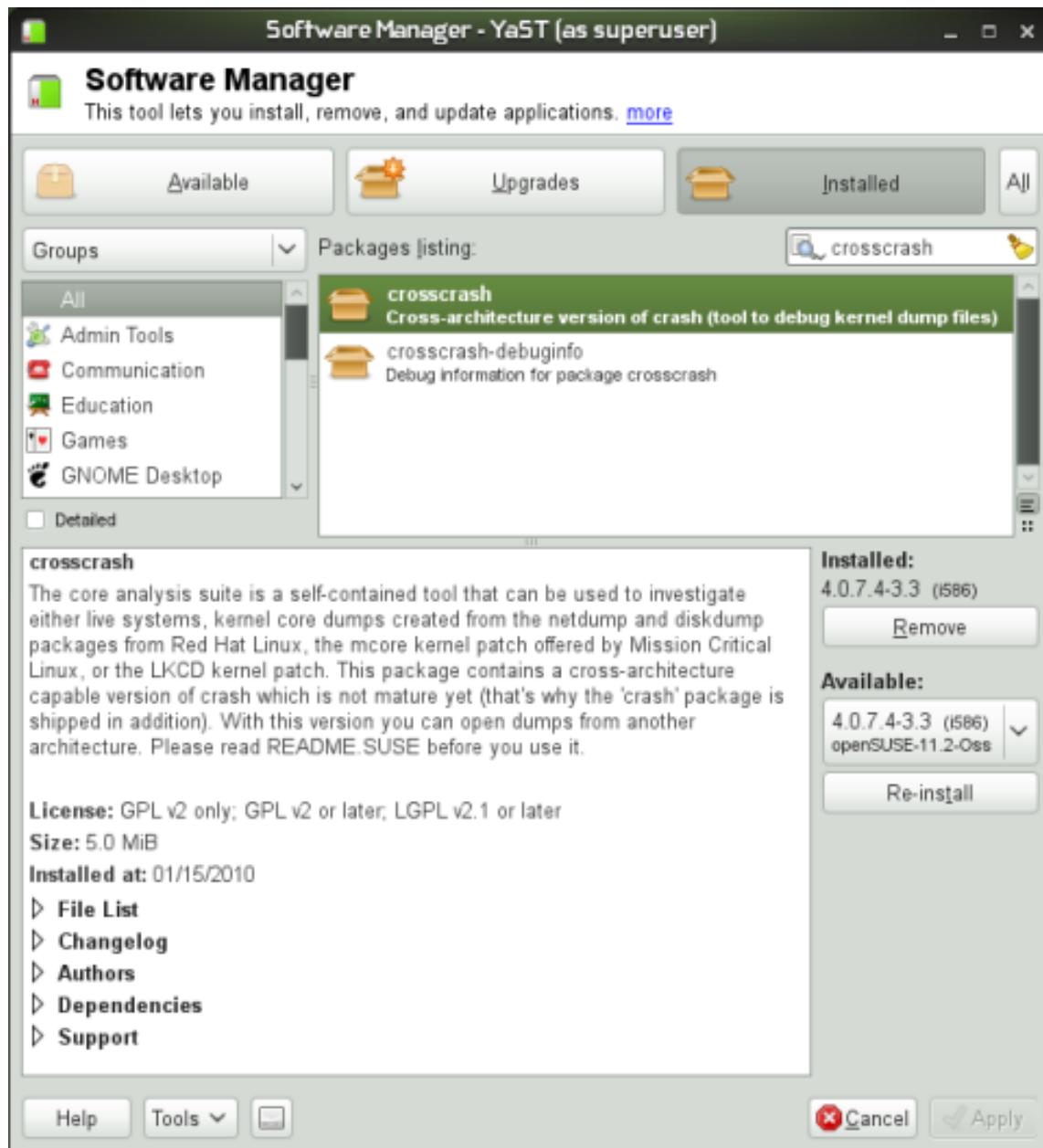
```
root@dedoimedo:~# gdb-kdump vmcore
Using /boot/vmlinuz-2.6.32-573.1.1.el6.x86_64 as vmlinux
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain 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-suse-linux"...
Using host libthread_db library "/lib64/libthread_db.so.1".

warning: shared library handler failed to enable breakpoint
#0  do_syslog (type=<value optimized out>, buf=0x2b97e2ad6010 <Address 0x2b97e2ad6010 out of bounds>,
   len=<value optimized out>) at kernel/printk.c:259
259          cond_resched();
(gdb) bt
#0  do_syslog (type=<value optimized out>, buf=0x2b97e2ad6010 <Address 0x2b97e2ad6010 out of bounds>,
   len=<value optimized out>) at kernel/printk.c:259
#1  0xffffffff801c01b0 in kmsg_read (file=<value optimized out>,
   buf=0x2b97e2ad6010 <Address 0x2b97e2ad6010 out of bounds>, count=262143, ppos=0x4e40)
   at fs/proc/kmsg.c:38
#2  0xffffffff801867d0 in vfs_read (file=0xfffff8102177da980,
   buf=0x2b97e2ad6010 <Address 0x2b97e2ad6010 out of bounds>, count=60, pos=0xfffff810215059f50)
   at fs/read_write.c:264
#3  0xffffffff80186bb0 in sys_read (fd=<value optimized out>,
   buf=0x2b97e2ad6010 <Address 0x2b97e2ad6010 out of bounds>, count=262143) at fs/read_write.c:351
#4  0xffffffff8010adba in system_call ()
#5  0x0000000000000246 in ?? ()
#6  0x00005555565b560 in ?? ()
#7  0x00002b97e2bbac00 in ?? ()
#8  0x000000000000400000 in ?? ()
#9  0x0000000000000000 in ?? ()
(gdb) 
```

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.

**Figure 113:** crosscrash installation via YaST

## 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>
2. 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)

## 33 Kdump references

1. Kdump official site  
<http://lse.sourceforge.net/kdump/>
2. Kdump official documentation  
<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)

## 34 Crash references

1. System.map on Wikipedia  
<http://en.wikipedia.org/wiki/System.map>

2. The Linux Kernel HOWTO - 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)
4. Kdump - openSUSE  
<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>
7. Linux Crash HOWTO  
<http://www.faqs.org/docs/Linux-HOWTO/Linux-Crash-HOWTO.html>
8. lcrash HOWTO  
<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>
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/>
13. cscope  
<http://cscope.sourceforge.net/>
14. MPlayer source download  
<http://www.mplayerhq.hu/design7/dload.html>
15. objdump man page  
<http://linux.die.net/man/1/objdump>
16. Linux Kernel Module Programming Guide  
<http://tldp.org/LDP/lkmpg/2.6/html/>

17. Crash Whitepaper case study  
[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>
19. CentOS Debug 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>
2. Collecting and analyzing Linux kernel crashes - Kdump  
<http://www.dedoimedo.com/computers/kdump.html>
3. How to enable debug repositories in CentOS Linux  
<http://www.dedoimedo.com/computers/centos-debug.html>
4. Kdump on openSUSE 11.2  
<http://www.dedoimedo.com/computers/kdump-opensuse.html>
5. Kdump on CentOS 5.4  
<http://www.dedoimedo.com/computers/kdump-centos.html>
6. Collecting and analyzing Linux kernel crashes - crash  
<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>
8. openSUSE 11.2 KDE  
<http://www.dedoimedo.com/computers/opensuse-11sp2.html>
9. openSUSE 11.2 Gnome  
<http://www.dedoimedo.com/computers/opensuse-11sp2-gnome.html>
10. CentOS 5.3  
<http://www.dedoimedo.com/computers/centos.html>

11. Ubuntu 9.10 Karmic Koala  
<http://www.dedoimedo.com/computers/ubuntu-9-10.html>
12. Fedora 11 Leonidas  
<http://www.dedoimedo.com/computers/fedora-11.html>
13. Fedora 12 Constantine  
<http://www.dedoimedo.com/computers/fedora-12.html>
14. Debian 5.03 Lenny  
<http://www.dedoimedo.com/computers/debian.html>
15. RedHat Linux Enterprise 6 Beta  
<http://www.dedoimedo.com/computers/rh6-beta.html>
16. Linux system utilities  
<http://www.dedoimedo.com/computers/linux-system-utilities.html>
17. GRUB bootloader - Full tutorial  
<http://www.dedoimedo.com/computers/grub.html>
18. GRUB 2 bootloader - Full tutorial  
<http://www.dedoimedo.com/computers/grub-2.html>