

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

[PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-06/3042.html>

From: Kalin KOZHUHAROV (*kalin_at_ThinRope.net*)

Date: 06/14/04

Date: Mon, 14 Jun 2004 22:46:07 +0900

To: Norberto Bensa <norberto+linux-kernel@bensa.ath.cx>

Norberto Bensa wrote:

> *Willy Tarreau* wrote:

>

>> <http://developer.odsl.org/rddunlap/kmsgdump/>

>

>

> <http://developer.odsl.org/rddunlap/kmsgdump/>

>

> ; -)

>

OK, I reapplied the patch from the URL above to 2.6.6 (this is current for most of my machines) and I am posting it here.

I moved one line in kmsg-reboot inside the appropriate #ifdef as not to cause warnings on compile. Apart from that only applying with the code shifted a bit.

I have prepared my floppy for a dump and everything seems ok, but I have not got any oopses yet :-)

Not sure if I need the signed-off-by, as it is not for inclusion (yet), but here it is:

Signed-off-by: Kalin KOZHUHAROV <kalin@thinrope.net>

```
diff -Nuarp linux-2.6.6/Documentation/kmsgdump.txt linux-2.6.6_K1/Documentation/kmsgdump.txt
--- linux-2.6.6/Documentation/kmsgdump.txt 1970-01-01 09:00:00.000000000 +0900
+++ linux-2.6.6_K1/Documentation/kmsgdump.txt 2004-06-14 16:03:52.623741443 +0900
@@ -0,0 +1,370 @@
+ - Documentation for KMSGDUMP v0.4 -
+ [Sun Sep 19 19:30:32 CEST 1999] - Willy Tarreau
+ [updates by Randy Dunlap, Jan/Feb/Mar 2003]
+
+
+1. What is KMSGDUMP ?
```

[PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+~~~~~
+
+KMSGDUMP is an extension to the Linux kernel which allows the user on the
+console to dump the last kernel messages onto a floppy diskette, thus
+avoiding to take a pen and a paper to copy them when the system is stuck.
+Only 3"1/2, 1.44 MB diskettes are supported by default. Other capacities
+might work, provided you change the geometry in the file "kmsgdump.h".
+
+
+2. How does it work ?
+~~~~~
+
+There are two ways of getting a dump :
+
+ – by pressing SysRQ+D (RightAlt – PrintScr – D together) ;
+ – after a kernel panic has occurred, a dump may be automatically
+ generated.
+
+Before anything else, you MUST KNOW that in order to get maximal
+chances to complete the dump succesfully, the CPU is rebooted in
+real mode and disk accesses are made via the Bios. This ensures
+that even if kernel memory is really corrupted, the dump still
+has chances to work, but this also implies that after a dump has
+occurred, it is IMPOSSIBLE TO CONTINUE TO WORK WITH THE CURRENT
+KERNEL. You will have to REBOOT. So when your kernel still responds,
+you'd better get a similar dump by entering one of the following
+commands :
+
+# dmesg > /dev/fd0 ( for RAW mode )
+
+or
+
+# dmesg | mwrite a:messages.txt ( for FAT mode )
+
+
+Second, be sure that FLOPPY CONTENTS WILL BE LOST AFTER A DUMP.
+Even if there are cases in which you can dump at the end of a diskette without
+losing the beginning, consider that by default the beginning of the diskette
+will be ERASED and you won't be able to recover what's on it. You have been
+warned.
+
+
+3. Modes of operation
+~~~~~
+
+There are two modes of operation : manual and automated.
+
+Manual mode (or interactive mode) is always entered if you hit SysRQ+D.
+But it is also entered during a kernel panic if the current mode is set
+to "manual". This mode is recommended for a developer's workstation,
+or a kernel running under an emulator such as vmware. It's recommended
```

Linux–Kernel: [PATCH] kmsgdump–2.6.6.patch (was: Panics need better handling)

+to disable interactive mode on servers which may crash when nobody is
+near to reboot them.
+
+Automatic mode can only be entered during a kernel panic and if automatic
+mode was previously configured. Sometimes, the system is really weird and
+even kmsgdump can cause recursive crashes (this has been reported to me once).
+For this reason I've added a checkpoint mechanism to the code : every little
+part of code is checkpointed, and if a crash occurs again, the same part is
+not executed again, to prevent looping. So there are more chances to get
+to the reset routine which will, in the worst case, reboot the system, but
+not let it loop indefinitely.
+
+3.1. Manual mode
+~~~~~
+
+Under manual mode, the screen initialized to color 80x25 mode (bios mode 3)
+with a blue background.
+
+[Note: some people asked me to set other colors to avoid confusion
+with another OS' BSOD, but I couldn't find good associations.
+Even though I've received an interesting comment about the way
+to choose colors readable on any color or monochrome display,
+I'm waiting for suggestions, and for the moment we'll say that
+these are the colors of Midnight Commander and call this "BSOL"
+(blue screen of life) because this one is interactive.]
+
+The screen is divided in two portions. The upper one displays the current
+status (kernel version, drive unit, printer, format...), and the lower one
+the messages captured before switching to real mode. The internal speaker
+beeps if a key has not been hit within 3 seconds. This is simply to get
+someone's attention, mainly in cases where no monitor is connected to the
+PC.
+
+The interface is not case–sensitive about keys pressed. Keys used are :
+ Upper arrow : scroll messages to the beginning
+ Lower arrow : scroll messages to the end
+ B : immediately reBoot the system
+ D : Dump messages onto the selected floppy with selected format. Warning:
+ no check is done before, and the floppy will simply be overwritten by
+ the messages.
+ F : select Format, by switching between RAW and FAT12
+ H : immediately Halt the system.
+ I : display Information, little help about the keys.
+ P : Print messages on the currently selected printer. If you press this key
+ by accident, wait about one minute for the bios routine to timeout, and
+ you'll here the beeps again, stating that you can play again.
+ T : select next available prinTer. The system tests if a printer is
+ connected at the other end of the cable, and skips the empty ports.
+ U : change drive Unit. Although dump is possible on hard disks, they are
+ never proposed in the interface to avoid dramatic mistakes.
+

+Other keys are simply ignored.

+

+After a succesful dump or print, 3 quick beeps are played. In case of an error,
+only one beep is played. This is important if you act blindly with a keyboard
+and no monitor.

+

+3.2. Automatic mode

+~~~~~

+

+Automated operation is performed by the system only when a kernel panic
+occurs. In this case, the system waits for the "panic_timeout" delay
+to give you a few seconds if you want to try to play with SysRQ (sync,
+unmount filesystems, ...). This delay is configurable by entering a
+number of seconds in "/proc/sys/kernel/panic".

+

+After that, the system is rebooted to real mode, and depending on the
+mode of operation chosen, either the interactive mode is entered (see above)
+or it is the automatic mode, which we'll describe here.

+

+3.2.1. Start of operation

+~~~~~

+Some checks are performed. First, the system sees if the dump feature is
+enabled or not. If not, operation ends (see below). If dump is enabled,
+and if the "safe" flag is enabled, the diskette is verified to be a real
+"KMSGDUMP" diskette and not another one (read section 4 to know how to
+prepare a secure diskette for KMSGDUMP). If the diskette isn't a right one,
+operation ends. If the diskette is a right one, or if the check has been
+disabled, the dump is performed with the current parameters (unit, format...).

+

+3.2.2. End of operation

+~~~~~

+After completion of an automatic dump, or when a dump is aborted, the system
+can either halt or reboot. In case of redundant servers, you may prefer to halt
+a buggy system, because another one ensures the service continues to work.
+But in other cases, you may prefer rebooting to quickly restart services.
+This is also configurable (read section 4).

+

+

+4. How a crash can be prepared

+~~~~~

+4.1. Kernel options

+~~~~~

+

+First, choose the kernel compilation options which closely match your
+situation. This may seem obvious, but you can reduce the risks of crash
+by not enabling drivers designated for hardware you don't have. Specially
+on servers, use only a reduced feature set, because you know exactly what
+you need (eg: don't enable NTFS and QNX filesystems if you don't need them).

+

+Configure KMSGDUMP options to match your needs. Don't ask to auto–dump if
+you don't have a floppy drive. In this case, you might prefer to enable

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

+interactive mode to display messages on the screen and eventually print
+them.
+
+When you use SCSI hard disks, you can sometimes reduce the reset time to
+help the system recover faster. Eg: on my system, I have an AHA2940UW which
+waits 15 seconds by default. All peripherals still work well with 1 second,
+so 14 seconds are won.
+
+If you have changed your messages buffer size (which is 16 kB by default),
+you should modify the size in "include/asm/kmsgdump.h", parameter LOG_BUF_LEN.
+Some people required 32 kB. But you shouldn't exceed 60 kB since the dump is
+done in real mode (16 bits).
+For kernel versions 2.5.6x and later, the LOG_BUF_LEN parameter is part
+of the kernel .config file (LOG_BUF_SHIFT) so you don't need to modify
+it at all.

+
+4.2. Configure KMSGDUMP
+~~~~~

+
+If your kernel supports SYSCTL, you can adjust KMSGDUMP parameters by
+writing a string to /proc/sys/kernel/kmsgdump. This string consists of
+a concatenation of flags. Most of them are only booleans. For each boolean,
+a complementary flag exists to avoid any ambiguous interpretation.
+For the moment, the flags are :

+
+ Name Description Default Complement
+ F FAT mode Yes R
+ R Raw mode F
+ A Automatic mode Yes I
+ I Interactive mode A
+ B Boot after dump Yes H only used in automatic mode
+ H Halt after dump B only used in automatic mode
+ S Safe mode Yes O only used in automatic mode
+ O Overwrite disk S only used in automatic mode
+ E Enable dumping Yes D only used in automatic mode
+ D Disable dumping E only used in automatic mode
+ Txxx Track xxx 0 (N/A) first track is 0 per default
+ Uxxx Unit xxx 0 (N/A) bios drive is 0 (A:) per default

+
+Note: default means "default if none specified".
+
+Example: if you enter the following command, a kernel panic will generate
+ a dump in FAT mode after verifying that the disk has been prepared
+ for a dump, and then it will reboot :

+
+ # echo "FABSE" > /proc/sys/kernel/kmsgdump
+
+This one will ask to dump raw messages at the end of the diskette in drive B
+and halt :
+
+ # echo "RABOET79U1" > /proc/sys/kernel/kmsgdump

+
+And this one will ask for a quick reboot :
+
+ # echo "DB" > /proc/sys/kernel/kmsgdump
+
+
+4.3. Prepare a disk for kmsgdump
+~~~~~
+
+If safe mode is required, before an automatic dump, the system will read
+the beginning of the floppy in the drive and will look for the word "KMSGDUMP"
+at offset 3 of the first sector. This is the label of the diskette. The dump
+will only be performed if this word is found as-is. So if you enable safe mode
+don't forget to prepare your diskettes with the following command, provided
+your diskette is in drive A :
+
+ # echo "012KMSGDUMP" > /dev/fd0
+
+Please note that when the dump is performed in FAT mode, this word is written
+to the same place. This has two side effects :
+ - a diskette on which a dump has been done in FAT mode is re-usable without
+ intervention.
+ - you can prepare a diskette by entering kmsgdump (SysRQ+D) and doing a
+ FAT mode dump.
+
+On the other hand, when a RAW dump is done at the beginning of the disk, it
+cannot be used again as a "safe kmsgdump disk". Moreover, leaving it in the
+drive when rebooting will cause the system to hang if the bios tries to boot
+from the floppy first.
+
+4.4. Prepare the PC for a crash
+~~~~~
+
+Because you'll have to leave a diskette in a drive, you may have to setup
+your bios to boot from hard disk or anything but the floppy first because
+the bios will find anything but a bootable system on this floppy. The problem
+is with older systems on which the boot sequence cannot be changed. For this
+reason, when a diskette is formatted in FAT mode, a small code is inserted on
+the boot sector which tries to redirect the boot to the first hard disk seen
+by the bios. This is *generally* the bootable disk, but this may not be the
+right on specific systems, so you may have to do some tests before considering
+this option to be the right one for you.
+
+If your system is a server, you may reduce the time the bios tests the PC to
+ensure quick reboot. On some systems, you can turn on the option "Quick
+power-on self test", and disable testing of memory above 1MB.
+
+
+5. Reading the messages back
+~~~~~
+5.1. FAT-formatted disks

Linux–Kernel: [PATCH] kmsgdump–2.6.6.patch (was: Panics need better handling)

+~~~~~

+

+If the disk has been formatted as FAT12, you'll find on it a file
+named "MESSAGES.TXT" which contains all of the 'messages' buffer. If the
+buffer is not full, the end of the file is filled with zeroes, so
+it's better to delete them using "tr" under linux.

+

+ – under Linux, either mount the disk :

+

```
+ # mount -rt msdos /dev/fd0 /mnt  
+ # cat /mnt/messages.txt | tr -d '\000'  
+ # umount /mnt
```

+

+ or read it using mtools :

+

```
+ # mtype a:messages.txt | tr -d '\000'
```

+

+ – under DOS, you can simply run EDIT :

+

```
+ C:\> edit a:messages.txt
```

+

+ – under Windows, you can open the file with Wordpad. Avoid using
+ Notepad since it doesn't understand linefeeds only.

+

+5.2. RAW disks

+~~~~~

+

+Raw disks will be readable under linux by using the utility DD. By
+default, the dump will be performed from the first sector of the disk.
+Example with 16 kB messages buffer:

+

```
+ # dd if=/dev/fd0 bs=512 count=32 | tr -d '\000'
```

+

+If you specified "T79" in the parameters to dump on track 79 of the disk,
+you have to do some calculations :

+

+A 1.44 MB disk has 18 sectors/track, 2 heads and 512 bytes/sector so
+18*2*512 equals 18432 bytes/track. You'll have to skip 18432 bytes for
+each unwanted track. But you can also count only with kilobytes : if you
+consider that a track is exactly 18 kilobytes, then skip the number of
+tracks times 18 kilobytes :

+

```
+ # expr 79 \* 18  
+ 1422  
+ # dd if=/dev/fd0 bs=1024 skip=1422 count=16 | tr -d '\000'
```

+

+The default dd utility reads all data from the start of the disk so this can
+be quite long. There are other implementations on the net which do an "lseek"
+before the first read.

+

+

+6. Other speed improvements

+~~~~~

+Here is some advice to make a system reboot faster, especially if
+you don't use filesystem journalling.

+

+When a file server crashes, it may FSCK for a long time. There are good docs
+about how to dramatically reduce FSCK time, but at least consider these
+methods :

+

+ - in /etc/fstab, set the sixth field (fs_passno) to 1 for the root fs,
+ and 2 for every other fs. FSCK will know it what it can parallelize
+ depending on hardware dependencies. In the better case, you can divide
+ the total time by the number of physical disks.

+(man fstab and man fsck for more info).

+

+ - when possible, mount filesystems read-only. On an anonymous FTP server,
+ for example, it's not always necessary to mount everything RW. So before
+ copying files onto an fs, remount it RW :

+

+ # mount -wo remount /mount/point

+

+ At the end, remount it RO :

+

+ # mount -ro remount /mount/point

+

+ - change the number of bytes by inode and the block size when formatting
+ your FS. I personally use 16384 bytes/inode, a block size of 4096 bytes,
+ the sparse flag set (reduces the number of superblocks). This makes me
+ waste about 1% space, but total mount time is about 1 second for a total
+ of 8 FS's, 11 gigs on 5 separate disks and the total FSCK time after a
+ loopy power-off is less than 3 minutes.

+

+ - use a Linux journaling file system, such as ext3fs, IBM JFS, or
+ reiserfs in Linux 2.4, or any any of those or SGI XFS in Linux 2.5

+

+And of course, don't start services you don't need ! Sendmail itself can take
+a long time if it cannot resolve the domain name.

+

+

+7. For more information and/or suggestions

+~~~~~

+

+For more informations, you can email me at :

+

+ willy AT meta-x.org

+

+(be patient, I read my mail when I can, and can't always reply. I'm
+ used to "tail -1000 \$MAIL|less" or "less +G \$MAIL")

+

+For suggestions, you can either email them to me, or share them with
+the Linux Kernel Mailing List :

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+
+ linux-kernel@vger.kernel.org
+
+Enjoy using it,
+
+Willy
+
diff -Nuarp linux-2.6.6/arch/i386/Kconfig linux-2.6.6_K1/arch/i386/Kconfig
--- linux-2.6.6/arch/i386/Kconfig 2004-05-10 11:32:01.000000000 +0900
+++ linux-2.6.6_K1/arch/i386/Kconfig 2004-06-14 16:03:52.624741294 +0900
@@ -1294,6 +1294,94 @@ config X86_MPPARSE
     depends on X86_LOCAL_APIC && !X86_VISWS
     default y

+config KMSGDUMP
+ bool "Kernel messages dumper"
+ depends on EXPERIMENTAL
+ default n
+ help
+ If you say Y here, you will get maximal chances to save your kernel
+ messages under weird conditions : you'll be able to display them on
+ the screen, print them on a parallel printer, and dump them onto a
+ floppy diskette. You will have to press SysRQ + D to access this
+ feature, or wait for a kernel panic. In the last case, it is also
+ possible to ask the kernel not to wait for any human operation, and
+ automatically dump its messages onto a diskette (see the option
+ CONFIG_KMSGDUMP_AUTO below).
+
+ Warning: when using this feature, the CPU is rebooted in real mode
+ and the kernel won't recover. If messages are dumped onto a floppy,
+ this floppy will be erased, unless you enable the option
+ CONFIG_KMSGDUMP_SAFE below. More info can be found in the file
+ Documentation/kmsgdump.txt.
+
+ All options below are used for default behaviour, but can be set
+ by writing to /proc/sys/kernel/kmsgdump if CONFIG_SYSCTL is enabled.
+
+ Don't say Y unless you really want to hack your kernel and/or help
+ developers to debug it. This isn't a toy, you have been warned !
+
+config KMSGDUMP_FAT
+ bool "floppy as FAT by default"
+ depends on KMSGDUMP
+ help
+ By default, when CONFIG_KMSGDUMP is enabled, all messages dumped on
+ a floppy are in raw format, beginning at the first sector of the
+ diskette. By enabling this option, you'll change the default format
+ to FAT12. Although this is selectable under interactive mode, this is
+ not the case after a kernel panic, where this option applies most.
+
+ When FAT is selected, the messages are saved into a file named
```

Linux–Kernel: [PATCH] kmsgdump–2.6.6.patch (was: Panics need better handling)

+ MESSAGES.TXT. This practice is discouraged if you share your computer
+ with anyone else, since other people may believe that if FAT is
+ supported, the contents of the diskette would not be lost, which is
+ false. It's better if people remember that the diskette is unusable
+ after a dump. On the other hand, accessing the file after a crash is
+ easier when the diskette is formatted as FAT. One other advantage of
+ FAT is that the boot sector of the diskette is filled with a boot
+ redirector which makes the system boot from the first hard disk even
+ if the bios tried to boot from the floppy. If unsure, say Y.
+
+
+
+
+config KMSGDUMP_AUTO
+ bool "Automatic dump then reboot on kernel panic"
+ depends on KMSGDUMP
+ help
+ When the kernel panics, it normally hangs or reboots. With
+ CONFIG_KMSGDUMP enabled, it will either enter the interactive mode
+ you can see by pressing SysRQ + D (default), or, if you say Y here,
+ automatically dump its messages buffer onto a diskette and then
+ reboot.
+
+ Enabling this is dangerous because if a diskette is left in the
+ floppy drive, it may be destroyed (unless you enable the option
+ CONFIG_KMSGDUMP_SAFE below), but in case of a server, this might
+ be interesting since this one will automatically reboot after a
+ panic. Don't say Y here if this kernel is to be installed on a
+ floppy ! You would say Y in case of a server, N in case of a
+ workstation.
+
+config KMSGDUMP_SAFE
+ bool "Limit dump to pre–initialized floppies only"
+ depends on KMSGDUMP
+ help
+ When CONFIG_KMSGDUMP_AUTO is enabled, the kernel can automatically
+ dump its messages onto the floppy present in the drive during a
+ crash. This can be dangerous because if a floppy has been forgotten
+ in the drive it may be overwritten. By enabling this option, the
+ kernel will accept to write to the floppy only if it finds the word
+ "KMSGDUMP" at offset 3 on the floppy.
+
+ So to prepare a disk to accept dumps, you just have to do this :
+
+ echo "012KMSGDUMP" > /dev/fd0
+
+ Please note that in case of interactive mode dump, this check is
+ never made because we consider that the user knows what stands on
+ his diskettes. If unsure, just say "Y".
+
+

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

endmenu

```
source "security/Kconfig"
diff -Nuarp linux-2.6.6/arch/i386/kernel/Makefile linux-2.6.6_K1/arch/i386/kernel/Makefile
--- linux-2.6.6/arch/i386/kernel/Makefile 2004-05-10 11:32:02.000000000 +0900
+++ linux-2.6.6_K1/arch/i386/kernel/Makefile 2004-06-14 16:03:52.625741145 +0900
@@ -30,6 +30,7 @@ obj-y += sysenter.o vsyscall.o
obj-$(CONFIG_ACPI_SRAT) += srato.o
obj-$(CONFIG_HPET_TIMER) += time_hpet.o
obj-$(CONFIG_EFI) += efi.o efi_stub.o
+obj-$(CONFIG_KMSGDUMP) += kmsg_reboot.o kmsgdump.o
obj-$(CONFIG_EARLY_PRINTK) += early_printk.o
obj-$(CONFIG_X86_STD_RESOURCES) += std_resources.o

diff -Nuarp linux-2.6.6/arch/i386/kernel/kmsg_reboot.c linux-2.6.6_K1/arch/i386/kernel/kmsg_reboot.c
--- linux-2.6.6/arch/i386/kernel/kmsg_reboot.c 1970-01-01 09:00:00.000000000 +0900
+++ linux-2.6.6_K1/arch/i386/kernel/kmsg_reboot.c 2004-06-14 16:21:55.749954051 +0900
@@ -0,0 +1,315 @@
+/*
+ * kmsg_reboot.c
+ * reboots to real mode for kmsgdump
+ * author: Willy Tarreau <willy AT meta-x.org>
+ */
+
+#include <linux/config.h>
+#include <linux/kernel.h>
+#include <linux/mm.h>
+#include <linux/interrupt.h>
+#include <linux/delay.h>
+#include <linux/mc146818rtc.h>
+
+static struct
+{
+ unsigned short size __attribute__((packed));
+ unsigned long long * base __attribute__((packed));
+ }
+no_idt = { 0, 0 };
+
+#include "mach_reboot.h"
+#include <asm/kmsgdump.h>
+#include <asm/apic.h>
+
+/* use KDEBUG to control whether kdebug() is defined */
+#undef KDEBUG
+
+#ifdef KDEBUG
+#define kdebug(fmt,arg...) \
+ printk(KERN_DEBUG fmt,##arg)
+#else
+#define kdebug(fmt,arg...) \
+ do { } while (0)

```

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+#endif
+
+
+extern void dump_syslog(char * buf);
+
+/* checkpoint counter used to try to recover from recursive system crashes */
+static int chkpnt = 0;
+
+/* boot-time default flags. These are single character indicators except for T (track number)
+ and U (unit number) which must be followed by a number. To avoid confusion with default
+ values, each flag has its complement and you are strongly encouraged to use one of each.
+
+ flags available are :
+ Name Description Default Complement
+ F FAT mode Yes R
+ R Raw mode F
+ A Automatic mode Yes I
+ I Interactive mode A
+ B Boot after dump Yes H only used in automatic mode
+ H Halt after dump B only used in automatic mode
+ S Safe mode Yes O only used in automatic mode
+ O Overwrite disk S only used in automatic mode
+ E Enable dumping Yes D only used in automatic mode
+ D Disable dumping E only used in automatic mode
+ Txxx Track xxx 0 (N/A) first track is 0 per default
+ Uxxx Unit xxx 0 (N/A) bios drive is 0 (A:) per default
+*/
+char kmsgdump_flags[16]=
+#ifdef CONFIG_KMSGDUMP_FAT
+ "F" /* Fat */
+#else
+ "R" /* Raw */
+#endif
+#ifdef CONFIG_KMSGDUMP_AUTO
+ "AEB" /* Auto, Dump, Boot */
+#else
+ "IE" /* Interactive, Dump */
+#endif
+#ifdef CONFIG_KMSGDUMP_SAFE
+ "S"
+#else
+ "O"
+#endif
+ "TOU0"; /* Safe,Track0,Unit0 */
+
+/*
+ * machine_dump() : Added on 19990628 by Willy Tarreau <willy AT meta-x.org>
+ *
+ * This is used to dump the kernel messages buffer onto a floppy diskette.
+ * The system is rebooted to real mode so it will be unuseable after a call
+ * to this function which never returns.
```

Linux–Kernel: [PATCH] kmsgdump–2.6.6.patch (was: Panics need better handling)

```
+ * The argument to this function is the caller type (DUMP_FROM_PANIC=0 or
+ * DUMP_FROM_SYSRQ=1).
+ *
+ * Note 1 : Part of this code has been shamelessly stolen from machine_restart().
+ * Perhaps it should be better to implement a generic rebooting function.
+ * Note 2 : I couldn't manage to get the <reboot_thru_bios> method to work
+ * here. The bios never calls my code. After hundreds of reboots,
+ * I finally accepted the fact that the PC is stronger than the human
+ * and only kept the reset method. But be careful, Linus' comment
+ * about it not working properly on some hardware still applies here.
+ * Note 3 : I don't know what means we have on other architectures to dump
+ * messages, but it will be interesting to add this feature to all archs.
+ */
+
+#if LOG_BUF_LEN >= 0x10000
+#error LOG_BUF_LEN is too large!!!
+#error The log buffer must be < 64 KB for kmsgdump.
+#endif
+
+void machine_dump(int callertype)
+{
+ int modeflags=MASK_OUTPUTFAT | MASK_PANICDUMP | MASK_PANICBOOT |
+ MASK_SAFEDUMP;
+ int track=0; /* by default, track 0 */
+ int unit=0; /* and drive A */
+
+ if (chkpnt<100) {
+   chkpnt=100;
+   kdebug("KMD: cp 100\n");
+   local_irq_disable();
+   chkpnt++;
+   #ifdef CONFIG_SMP
+   /*
+    * Stop all CPUs and turn off local APICs and the IO–APIC, so
+    * other OSs see a clean IRQ state.
+    */
+   smp_send_stop();
+   disable_IO_APIC();
+   chkpnt++;
+   #endif
+
+   #ifdef CONFIG_X86_LOCAL_APIC
+   /* if this CPU is using local APIC, we must disable it */
+   if (test_bit(X86_FEATURE_APIC, boot_cpu_data.x86_capability)) {
+     unsigned long l, r;
+     disable_local_APIC();
+     disconnect_bsp_APIC();
+     rdmsr(MSR_IA32_APICBASE, l, r);
+     l &= ~MSR_IA32_APICBASE_ENABLE;
+     wrmsr(MSR_IA32_APICBASE, l, r);
+   }
+ }
```

```

+ #endif
+ }
+ /* Write 0x09 to CMOS register number 0x0f, which the BIOS POST
+ routine will recognize as telling it to :
+ - re-initialize a few parts of hardware ;
+ - load registers from a structure pointed to at [40:67] ;
+ - branch to the program at offset 0x14 in this structure.
+
+ [Note: funciton 0x0A is easier to use, but doesn't re-initialize
+ anything and might not work on specific harware. ]
+ */
+
+ if (chkpnt<200) {
+ unsigned long flags;
+ chkpnt=200;
+ kdebug("KMD: cp 200\n");
+ spin_lock_irqsave(&rtc_lock, flags);
+ outb_p (0x8f, 0x70);
+ chkpnt++;
+ outb_p (0x09, 0x71);
+ chkpnt++;
+ spin_unlock_irqrestore(&rtc_lock, flags);
+ }
+
+ /* Remap the kernel at virtual address zero, as well as offset zero
+ from the kernel segment. This assumes the kernel segment starts at
+ virtual address PAGE_OFFSET. */
+
+ if (chkpnt<300) {
+ chkpnt=300;
+ kdebug("KMD: cp 300\n");
+ memcpy (swapper_pg_dir, swapper_pg_dir + USER_PGD_PTRS,
+ sizeof (swapper_pg_dir [0]) * KERNEL_PGD_PTRS);
+ chkpnt++;
+ }
+
+ /* Make sure the first page is mapped to the start of physical memory.
+ It is normally not mapped, to trap kernel NULL pointer dereferences.
+ */
+
+ if (chkpnt<400) {
+ int page;
+
+ chkpnt=400;
+ kdebug("KMD: cp 400\n");
+ for (page=0; page < ((FREEMEMORY+PAGE_SIZE-1) >> PAGE_SHIFT); page++)
+ pg0[page] = _PAGE_RW | _PAGE_PRESENT;
+ chkpnt++;
+ }
+
+ /*

```

Linux–Kernel: [PATCH] kmsgdump–2.6.6.patch (was: Panics need better handling)

```
+ * Use `swapper_pg_dir' as our page directory.
+ */
+
+ if (chkpnt<500) {
+   chkpnt=500;
+   kdebug("KMD: cp 500\n");
+   asm volatile("movl %0,%%cr3": : "r" (__pa(swapper_pg_dir)));
+   chkpnt++;
+ }
+
+ /* to avoid that the bios thinks we pressed Ctrl–Alt–Del : */
+ if (chkpnt<600) {
+   chkpnt=600;
+   kdebug("KMD: cp 600\n");
+   #ifdef REALLY_USE_REBOOT_SETUP
+   reboot_setup("c", NULL);
+   chkpnt++;
+   *((unsigned short *)0x472) = reboot_mode;
+ #else
+   *((unsigned short *)0x472) = 0x0;
+ #endif
+   chkpnt++;
+ }
+ /* For the switch to real mode, copy some code to low memory, out of
+ the way of BIOS variables. */
+
+ if (chkpnt<700) {
+   chkpnt=700;
+   kdebug("KMD: cp 700\n");
+   memset ((char *)CODEORIGIN, 0, 0x1000–CODEORIGIN); /* get a clean first page */
+   chkpnt++;
+ }
+ /* the main real mode code */
+ memcpy ((char *)CODEORIGIN, kmsgdump, kmsgdump_end–kmsgdump);
+   chkpnt++;
+ }
+
+ /* configure parameters from kmsgdump_flags */
+ if (chkpnt<800) {
+   char *c;
+   int *var = NULL;
+   int val = 0;
+   chkpnt = 800;
+   kdebug("KMD: cp 800\n");
+
+   c = kmsgdump_flags;
+   while (*c) {
+     switch(*c) {
+     case 'F' : modeflags |= MASK_OUTPUTFAT; break;
+     case 'R' : modeflags &= ~MASK_OUTPUTFAT; break;
+     case 'I' : modeflags |= MASK_PANICMAN; break;
```

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+ case 'A' : modeflags &= ~MASK_PANICMAN; break;
+ case 'B' : modeflags |= MASK_PANICBOOT; break;
+ case 'H' : modeflags &= ~MASK_PANICBOOT; break;
+ case 'S' : modeflags |= MASK_SAFEDUMP; break;
+ case 'O' : modeflags &= ~MASK_SAFEDUMP; break;
+ case 'E' : modeflags |= MASK_PANICDUMP; break;
+ case 'D' : modeflags &= ~MASK_PANICDUMP; break;
+ case 'T' : var = &track; val = 0; break;
+ case 'U' : var = &unit; val = 0; break;
+ }
+ if (*c>='0' && *c<='9')
+ val=(val*10)+(*c-'0');
+ c++;
+ /* when the number ends, assign it to the variable */
+ if ((*c<'0' || *c>'9') && (var != NULL)) {
+ *var = val;
+ }
+ }
+ chkpnt++;
+
+ *(char *)DRIVENUM = unit;
+ chkpnt++;
+ *(char *)TRACKNUM = track;
+ chkpnt++;
+
+ /* this will be configurable one day too :-) */
+ *(char *)MODEFLAGS = modeflags;
+ chkpnt++;
+ }
+
+
+ if (chkpnt<900) {
+ chkpnt=900;
+ kdebug("KMD: cp 900\n");
+ if (callertype == DUMP_FROM_SYSRQ)
+ *(char *)MODEFLAGS |= MASK_SYSRQMODE;
+ chkpnt++;
+ }
+
+
+ if (chkpnt<1000) {
+ chkpnt=1000;
+ kdebug("KMD: cp 1000\n");
+ memset (phys_to_virt(BEGINOFMESSAGES), 0, LOG_BUF_LEN); /* pre-initialize the messages buffer
*/
+ chkpnt++;
+ dump_syslog(phys_to_virt(BEGINOFMESSAGES)); /* get last messages */
+ chkpnt++;
+ }
+
+
+ /* We only have to inform the BIOS to where we need to jump,
```

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+ and initialize an image of the register bank in memory. */
+ if (chkpnt<1100) {
+   chkpnt=1100;
+   kdebug("KMD: cp 1100\n");
+   *((unsigned short*)(BOOTSTACKFRAME+0x16)) = 0; /* CS */
+   chkpnt++;
+   *((unsigned short*)(BOOTSTACKFRAME+0x14)) = CODEORIGIN; /* IP */
+   chkpnt++;
+
+   *((unsigned short*)0x467) = ((int)BOOTSTACKFRAME) & 0xFFFF;
+   chkpnt++;
+   *((unsigned short*)0x469) = ((int)BOOTSTACKFRAME)>>4 & 0xF000;
+   chkpnt++;
+ }
+
+ /* now we have to hard-reset the CPU. */
+ if (chkpnt<1200) {
+   chkpnt=1200;
+   for (;;) {
+     kdebug("KMD: cp %d\n", chkpnt);
+     mach_reboot();
+
+     /* That didn't work - force a triple fault.. */
+     __asm__ __volatile__("lidt %0": : "m" (no_idt));
+     __asm__ __volatile__("int3");
+   }
+   chkpnt++;
+ }
+ }
diff -Nuarp linux-2.6.6/arch/i386/kernel/kmsgdump.S linux-2.6.6_K1/arch/i386/kernel/kmsgdump.S
--- linux-2.6.6/arch/i386/kernel/kmsgdump.S 1970-01-01 09:00:00.000000000 +0900
+++ linux-2.6.6_K1/arch/i386/kernel/kmsgdump.S 2004-06-14 16:03:52.628740699 +0900
@@ -0,0 +1,898 @@
+/*
+ * file : kmsgdump.S
+ * version : 0.4.4/2002042801
+ * author : Willy Tarreau <willy AT meta-x.org>
+ *
+ * This code is called from the Bios when the processor has been
+ * externally reset and the NVRAM has been set to tell the bios
+ * to quickly branch to the pointer at [0x40:0x67].
+ * It executes in real mode, and starts from 0:0x700.
+ *
+ * Its purpose is to dump the last kernel messages onto a floppy disk
+ * or to print them on a parallel printer.
+ *
+ * Useful information are stored at the following hexadecimal adresses :
+ * 0:0700 - 0FAF = this code followed by the stack
+ * 0:0FB0 - 0FEF = bios stack frame at reboot time
+ * 0:0FF0 ( 8 bits ) = destination drive number (bios number)
+ * 0:0FF1 ( 8 bits ) = track number (<256 !)
```

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+ * 0:0FF2, bit 7 = caller mode (0=panic, 1=SysRq)
+ * bit 6 = output format (0=RAW, 1=FAT)
+ * bits 5->3 = behaviour when called from panic() :
+ * 000 = no dump, just halt
+ * 001 = no dump, just reboot
+ * 010 = dump, then halt
+ * 011 = dump, then reboot
+ * 100 = go to manual operation, just like SysRq.
+ * others undefined, so don't use please. Thanks.
+ * 0:0FF3 (13 bytes) = not used yet
+ *
+ * 0:1000 - 4FFF = kernel messages buffer contents (16 kB).
+ *
+ * Warning: This code is 16-bits, while the rest of the kernel is 32-bits.
+ * I couldn't find a way to link this file together with 32-bits
+ * files, just because of relative references which caused
+ * relocation issues. As a last resort, every variable is referenced
+ * "variable-base+CODEORIGIN" which is absolute and links well :-/
+ * If someone finds a better way to do this, I'd like to learn from
+ * him.
+ */
+#include <linux/version.h>
+#include <linux/config.h>
+#include <asm/kmsgdump.h>
+
+#define _pause jmp .+2
+.text
+.code16
+
+.global kmsgdump
+.global kmsgdump_end
+
+/*
+ * I use this only for development, it's a bios bootstrap.
+ * You only have to define UNDER_DEVEL and the binary object
+ * can be copied to a raw diskette and will boot from it.
+ * Moreover, the code contains the KMSGDUMP identifier so that
+ * the diskette is useable as-is.
+ */
+#ifdef UNDER_DEVEL
+bootstrap:
+ jmp 0f
+ .byte 0x90
+ .ascii "KMSGDUMP"
+0:
+ cli
+ xorw %ax, %ax
+ movw %ax, %ss
+ movw $0x7C00, %sp
+ pushw %ss
+ popw %es
```

Linux–Kernel: [PATCH] kmsgdump–2.6.6.patch (was: Panics need better handling)

```
+ movw %cs, %ax
+ movw %ax, %ds
+ movw $1000,%cx
+ movw $(kmsgdump–bootstrap+0x7c00), %si
+ movw $CODEORIGIN, %di
+ cld
+ rep
+ movsb
+/* movw $MASK_SYSRQMODE, MODEFLAGS*/
+ movw $(MASK_PANICDUMP|MASK_PANICBOOT|MASK_OUTPUTFAT|MASK_SAFEDUMP),
MODEFLAGS
+
+# read remaining sectors
+
+ movw $0x0203, %ax
+ movw $(bootstrap–kmsgdump+CODEORIGIN+0x200), %bx
+ movw $0x0002, %cx
+ movw $0x0000, %dx
+ int $0x13
+
+# pre–read messages
+ mov $0x220, %ax /* ah=2 (read), al=32 (32 sectors) */
+ mov $(BEGINOFMESSAGES), %bx
+ mov $0x0005, %cx
+ mov $0, %dx /* head always begins with 0 */
+ int $0x13
+
+ ljmp $0, $CODEORIGIN
+#endif /* UNDER_DEVEL */
+
+kmsgdump:
+ cli
+ xorw %ax, %ax
+ movw %ax, %ds
+ movw %ax, %es
+ movw %ax, %ss
+ movw $(TOPOFSTACK), %sp
+ call hardware_reset
+ sti
+ testb $(MASK_SYSRQMODE | MASK_PANICMAN), MODEFLAGS
+ jz automatic
+
+/* Here is the manual mode loop (either SysRq or Panic with MANUAL flag set).
+ * Each second, a short beep is sent so that the user knows the computer
+ * is waiting for him/her to press a key.
+ * When 'B' is hit, the system immediately reboots. When 'H' is hit, the
+ * system halts, and when another key is hit, the system attempts a dump
+ * and beeps two more times in case of success. It then goes on waiting
+ * for a key in case the user wants another dump.
+ */
+manual_dump:
```

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+/* reinitialize display to color_80x25 */
+ movw $0xe, %ax /* force a switch to graph mode to help in reinitialization */
+ int $0x10
+ movw $VIDEOMODE, %ax
+ int $0x10
+/* hide the cursor */
+ movb $1, %ah
+ movw $0xFFFF,%cx
+ int $0x10
+/* redraw the screen */
+ xorw %cx, %cx
+ call clearwindow
+/* display title */
+ movw $MKWORD(0,22), %dx
+ movw $(title-kmsgdump+CODEORIGIN), %si
+ call displayline
+/* display separation and help line */
+ movw $MKWORD(HELPLINE, 0), %dx
+ call movecursor
+ movw $MKWORD(0x0A, 177), %ax /* write a line of hashes */
+ movb $0, %bh
+ movw $(RIGHTCOL+1), %cx /* a full line */
+ int $0x10
+ movw $MKWORD(HELPLINE, 12), %dx /* display help at column 12 */
+ movw $(helpline-kmsgdump+CODEORIGIN), %si
+ call displayline
+
+# now we'll have to print the messages buffer in the lower window.
+redrawmessages:
+ movw $MKWORD(HELPLINE+2,0), %cx /* first line of the messages window */
+ pushw %cx /* save this, it will serve us later */
+ call clearwindow
+ popw %dx /* restore top of window */
+ mov $(BEGINOFMESSAGES), %si /* SI points to the beginning of the messages */
+ xorw %di,%di /* first line */
+ jmp findline
+1: lodsb
+ or %al,%al /* end of data ? */
+ jnz 3f
+ movw %di, currentline-kmsgdump+CODEORIGIN /* we fix the end at this point. */
+ jmp waitevent /* and don't show anything */
+3: cmp $10, %al
+ jnz 1b /* while not LF, look for an end of line */
+ inc %di /* count one more line */
+findline:
+ cmpw currentline-kmsgdump+CODEORIGIN, %di
+ jb 1b
+showmessages:
+ cmpw $(LOG_BUF_LEN+BEGINOFMESSAGES), %si /* end of buffer reached */
+ jnb waitevent
+ call displayline
```

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+ cmp $BOTTOMLINE, %dh /* stop displaying when screen is full */
+ ja waitevent
+ or %al,%al /* or when end of messages is reached */
+ jnz showmessages
+# redisplay current parameters after each pressed key
+waitevent:
+ cmpb $0, mustredraw-kmsgdump+CODEORIGIN /* if we must redraw the messages window, let's do it */
+ movb $0, mustredraw-kmsgdump+CODEORIGIN
+ jnz redrawmessages
+
+/* now, we'll display current settings (drive, format, status ...) */
+ movw $MKWORD(2, 8), %dx /* row 2, col 8 */
+ movw $(msgprinter-kmsgdump+CODEORIGIN), %si
+ call displayline
+
+ movb DRIVENUM,%al
+ addb '$A', %al /* A and B if < 0x80 */
+ test $0x80, %al
+ jz 1f
+ subb $(0x80-'C'+'A'), %al /* C and D if >= 0x80 */
+1: movb %al, unitname-kmsgdump+CODEORIGIN /* write the drive letter into the string */
+ movw $MKWORD(2, 40), %dx /* row 2, col 40*/
+ movw $(msgunit-kmsgdump+CODEORIGIN), %si
+ call displayline
+/* SI is already positionned at msgFAT, so we avoid "movw $msgFAT,%si" */
+ testb $(MASK_OUTPUTFAT), MODEFLAGS
+ jnz 2f
+ movw $(msgRAW-kmsgdump+CODEORIGIN), %si
+2: call displayline
+ movw $MKWORD(3, 8), %dx /* row 3, col 8 */
+ movw $(kversion-kmsgdump+CODEORIGIN), %si
+ call displayline
+
+1: movb $STATUS_WAITING, %al
+ call showstatus
+8: call waitkey
+ jnz 1f /* a key has been hit */
+ call beep /* no key hit: let's beep once a second */
+ jmp waitevent /* loop while no key is pressed */
+1: cmpb $0x50, %ah /* down arrow */
+ jnz 5f
+7: incw currentline-kmsgdump+CODEORIGIN /* go one line below */
+1: movb $1, mustredraw-kmsgdump+CODEORIGIN
+ mov $1, %ah /* we check if another key has been hit to speed up scrolling */
+ int $0x16
+ jnz 8b /* handle this new key */
+ jmp waitevent
+5: cmpb $0x48, %ah /* up arrow */
+ jnz 6f
+ subw $1,currentline-kmsgdump+CODEORIGIN /* one line back */
+ jc 7b /* aie, negative line. Let's correct this */
```

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+ jmp 1b
+6: cmpb '$b', %al /* 'B' stands for 'boot' */
+ jz do_reboot
+ cmpb '$h', %al /* 'H' stands for 'halt' */
+ jz do_halt
+ cmpb '$f', %al /* 'F' stands for 'change Format' */
+ jnz 3f
+ xorw $(MASK_OUTPUTFAT), MODEFLAGS
+ jmp waitevent
+3: cmpb '$u', %al /* 'U' stands for 'change Unit' */
+ jnz 4f
+ xorb $1, DRIVENUM /* altern first/second drive */
+ jmp waitevent
+4: cmpb '$t', %al /* 'T' stands for 'prinTer' */
+ jnz 5f
+ call probepriinter
+ jmp waitevent
+5: cmpb '$i', %al /* 'I' stands for 'Info' (help/about) */
+ jnz 6f
+ movw $MKWORD(HELPLINE+2,0), %cx /* first line of the messages window */
+ pushw %cx /* save this, it will serve us later */
+ call clearwindow
+ popw %dx /* restore top of window */
+ movw $(aboutmsg-kmsgdump+CODEORIGIN), %si
+5: call displayline
+ or %al, %al
+ jnz 5b
+ jmp waitevent
+6: cmpb '$p', %al /* 'P' stands for 'Print' */
+ jnz 7f
+ call printbuffer
+ jmp 8f /* display same return codes as for diskette */
+7: cmpb '$d', %al /* 'D' stands for 'dump' */
+ jnz waitevent
+ call do_dump
+8: movb $STATUS_WERR, %al /* assume a dump error for now */
+ jc 9f /* on error, we return immediately after only one beep */
+ movb $STATUS_DUMPOK, %al /* else we say that's good */
+ call beep /* make 3 audible beeps once successfully dumped */
+ call beep
+9: call showstatus
+ call beep
+ movw $16,%cx /* wait 1 second after the message is displayed */
+0: call sleep54
+ loop 0b
+ jmp waitevent
+
+/* clears the screen from the upper left corner defined by CX to the bottom right of the screen */
+clearwindow:
+ movw $MKWORD(BOTTOMLINE,RIGHTCOL), %dx /* bottom of screen */
+ movw $0x0600, %ax /* clear the messages window */
```

Linux–Kernel: [PATCH] kmsgdump–2.6.6.patch (was: Panics need better handling)

```
+ movb $ATTRIB, %bh
+ int $0x10
+ ret
+
+/* probes the next printer, and if not found, the next one, ... until
+ we find one which is OK, or we fall back to the current one.
+*/
+probeprinter:
+ movb $STATUS_PROBING, %al
+ call showstatus
+ movw $3, %cx /* we allow to automatically probe 3 printers */
+0: movb currentlpt–kmsgdump+CODEORIGIN, %al
+ incw %ax
+ cmpb $'3', %al
+ jbe 1f
+ mov $'1', %al
+1: movb %al, currentlpt–kmsgdump+CODEORIGIN
+ subb $'1', %al
+ movzbw %al, %dx /* printer number */
+ movb $1, %ah /* initialize printer */
+ int $0x17
+ testb $0x2F, %ah /* status = OK ? */
+ loopnz 0b /* no, probe next printer */
+2: ret
+
+/* prints all the buffer on the current printer. On error, CF is set. */
+printbuffer:
+ mov $(BEGINOFMESSAGES), %si /* SI points to the beginning of the messages */
+ movb $STATUS_PRINTING, %al
+ call showstatus
+ movb currentlpt–kmsgdump+CODEORIGIN, %al
+ subb $'1', %al
+ movzbw %al, %dx /* printer number */
+0: cmpw $(BEGINOFMESSAGES+LOG_BUF_LEN), %si
+ jae 9f /* end of buffer */
+ lodsb
+ or %al, %al /* end of data */
+ jz 9f
+ cmp $10, %al /* line feed : we'll insert a carriage return */
+ jnz 1f
+ mov $MKWORD(0, 13), %ax /* print carriage return */
+ int $0x17
+ mov $10, %al /* line feed */
+1: movb $0, %ah /* print character */
+ int $0x17
+ testb $0x2F, %ah /* status = OK ? */
+ jz 0b /* fetch next char */
+ stc
+ ret
+9: movw $MKWORD(0, 12), %ax /* form feed*/
+ int $0x17
```

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+ clc
+ ret
+
+/* Automatic dump mode, without user operation. The function was called
+ * from panic(). Mode flags also allow an immediate reboot without a dump,
+ * which is the same than playing with "panic_timeout" in case of a kernel
+ * panic.
+ */
+automatic:
+ testb $(MASK_PANICDUMP), MODEFLAGS
+ jz 1f
+ testb $(MASK_SAFEDUMP), MODEFLAGS
+ jz 0f
+ /* a safe dump is requested. we must ensure the diskette contains a "KMSGDUMP"
+ label before scratching it. */
+ xorw %ax,%ax /* reinitialize drive */
+ movzbw DRIVENUM, %dx
+ int $0x13
+
+ movw $0x201, %ax /* read 1 sector */
+ movw $(FREEMEMORY), %bx /* pointer to the buffer */
+ movw $0x1, %cx /* from sector 1 */
+ call int13retry
+ jc 1f /* read error, don't dump */
+ cmpl $0x47534d4b, FREEMEMORY+3 /* "KMSG" */
+ jnz 1f
+ cmpl $0x504d5544, FREEMEMORY+7 /* "DUMP" */
+ jnz 1f
+0: call do_dump
+1: testb $(MASK_PANICBOOT), MODEFLAGS
+ jnz do_reboot
+ jmp do_halt
+
+/* beep():
+ * Make a 54 ms beep, and silently pause during 54 ms
+ */
+beep: call sound_on
+ call sleep54
+ call sound_off
+ call sleep54
+ ret
+
+sleep54:
+ pushw %ax
+ movw CLOCKTICKS, %ax
+ incw %ax
+0: cmpw CLOCKTICKS, %ax
+ jnb 0b
+ popw %ax
+ ret
+
```

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+/* waitkey():
+ * Wait at most 3 second for a key press.
+ * If no key is pressed during that time, 0 is returned in AX and ZF is set.
+ * If a key is pressed, its lower case ascii code is returned in AL, its scan
+ * code in AH and the ZF is cleared.
+ */
+waitkey:
+ pushw %cx
+ movw CLOCKTICKS, %cx
+ addw $55, %cx /* 55/18.2 = 3 seconds */
+0: movb $1, %ah /* function="tell if a key was pressed" */
+ int $0x16
+ jnz 1f
+ cmpw CLOCKTICKS, %cx
+ jnb 0b
+ xorw %ax,%ax /* timeout, no key */
+ jmp 2f
+1: movb $0, %ah
+ int $0x16
+ cmpb $'A',%al
+ jb 2f
+ cmpb $'Z',%al
+ ja 2f
+ orb $0x20,%al
+2: popw %cx
+ orw %ax,%ax /* ZF=(AX==0) */
+ ret
+
+
+/* hardware_reset():
+ Reconfigure what can be wrong now in hardware :
+ - interrupt controller
+ - timers
+ - keyboard
+ - disk units
+ */
+hardware_reset:
+/* PIC reconfiguration consists in mapping PIC#1 IRQ 0 to software INT 8.
+ * Although not needed, PIC#2 is initialized in case we meet buggy chipsets.
+ */
+
+ movb $0x11, %al /* reconfiguration */
+ out %al, $0x20
+# out %al, $0xA0
+ _pause
+ movb $8, %al /* map irq 0 to int 0x08 */
+ out %al, $0x21
+# movb $0x70, %al /* map irq 8 to int 0x70 */
+# out %al, $0xA1
+ _pause
+ movb $4, %al /* master mode for PIC#1 */
```

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+ out %al, $0x21
+# movb $2, %al /* slave mode for PIC#2 */
+# out %al, $0xa1
+ _pause
+ movb $1, %al /* 8086 mode */
+ out %al, $0x21
+# out %al, $0xA1
+ _pause
+ movb $0xFF, %al /* all IRQs are masked at the moment */
+ out %al, $0x21
+# out %al, $0xA1
+ _pause
+
+/* Disable internal speaker output if it was enabled */
+
+ call sound_off
+
+/* PIT reconfiguration : set Timer0 clock to 18.2 Hz */
+
+ movb $0x34, %al /* timer 0 in interrupt mode */
+ out %al, $0x43
+ _pause
+ movb $0, %al /* divide by 65536 (1193182 Hz -> 18.2 Hz) */
+ out %al, $0x40 /* LSB first */
+ _pause
+ out %al, $0x40 /* MSB next */
+ _pause
+
+/* PIT reconfiguration (cont'd) : set Timer2 clock to 880 Hz for the buzzer */
+
+ movb $0xb6, %al /* timer 2 in square wave mode */
+ out %al, $0x43
+ _pause
+ movw $1355, %ax /* divider set to 1193182/1355 = 880 Hz */
+ out %al, $0x42 /* LSB first */
+ _pause
+ movb %ah, %al
+ out %al, $0x42 /* MSB next */
+ _pause
+
+ movb $0xBC, %al /* only irq 0,1,6 unmasked */
+ out %al, $0x21
+ _pause
+
+/* Keyboard reset: The keyboard may be misconfigured. We will send it a
+ reconfiguration (max speed) so that the Bios will at least reconfigure it
+ and its controller (i8042).
+*/
+
+ mov $0x305, %ax /* set speed */
+ xorw %bx, %bx /* full speed */
```

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+ int $0x16
+
+/* Reset the minimum so that int 13 works */
+ movw $0x3F2, %dx
+ movb $0x0, %al
+ outb %al, %dx
+ _pause
+ _pause
+ movb $0xC, %al
+ outb %al, %dx
+ _pause
+ _pause
+
+ xorw %ax, %ax
+ xorw %dx, %dx
+ int $0x13
+ ret
+
+/* sound_off(): disable internal speaker output. */
+sound_off:
+ push %ax
+ in $0x61, %al
+ andb $0xFC, %al
+ jmp 1f /* same code below */
+
+/* sound_on(): enable internal speaker output. */
+sound_on:
+ push %ax
+ in $0x61, %al
+ orb $3, %al
+1: _pause
+ out %al, $0x61
+ pop %ax
+ ret
+
+/* HALT the CPU */
+do_halt:
+ movb $STATUS_HALTING, %al
+ call showstatus
+0: cli
+ hlt
+ jmp 0b /* just in case of NMIs */
+
+
+/* REBOOT the CPU */
+do_reboot:
+ movb $STATUS_BOOTING, %al
+ call showstatus
+ cli
+/* Make sure [40:72] doesn't contain 0x1234, which the Bios will interpret as
+ Ctrl-Alt-Del. */
```

Linux–Kernel: [PATCH] kmsgdump–2.6.6.patch (was: Panics need better handling)

```
+ movw $0, 0x472
+ ljmp $0xFFFF,$0 /* branch to bios boot strap */
+
+/*
+ * do_dump():
+ * This function dumps the message buffer onto a disk. CF is set if an error
+ * occurred.
+ *
+ */
+do_dump:
+ movb DRIVENUM, %dl
+ xorw %ax, %ax
+ int $0x13 /* reinitialize drive */
+ xorw %si, %si /* assume first data sector = 0 */
+ testb $(MASK_OUTPUTFAT), MODEFLAGS
+ jz 1f
+ movw $(FATSIZE*2+2), %si /* when FAT is used, data go further */
+1: call makefat
+ jc 2f
+ call dumpdata
+2: ret
+
+/*
+ * Shows, in the status line, the message matching the status code in AL.
+ * AX, BX, DX and FLAGS contents are lost. CX and SI are kept.
+ */
+showstatus:
+ cbw
+ pushw %si
+ pushw %cx
+ testb $(MASK_SYSRQMODE | MASK_PANICMAN), MODEFLAGS
+ jz 2f /* is not in interactive mode, don't show anything */
+ pushw %ax
+ movw $MKWORD(3, 40), %dx
+ movw $(msgstatus–kmsgdump+CODEORIGIN), %si
+ call displayline
+ popw %si
+ cmpw $STATUS_MAXMSG, %si
+ ja 2f
+ shlw $4, %si
+ addw $(statwait–kmsgdump+CODEORIGIN), %si
+1: call displayline
+2: popw %cx
+ popw %si
+ ret
+/*
+ * sets the cursor to the begin of the line in DH, col in DL, and displays the message from SI
+ * which ends with \n, \r or \0. On return, AL contains the byte that ended the line, and SI
+ * points to the next byte.
+ * if LF is encountered, the cursor is positioned at the beginning of the next line and
+ * LF is returned in AL, the cursor position in DX.
```

Linux–Kernel: [PATCH] kmsgdump–2.6.6.patch (was: Panics need better handling)

```
+ * If DH is set to -1, then the cursor isn't moved. The new cursor position is returned into
+ * DX so it's possible to recall displayline immediately.
+ */
+displayline:
+ cmpb $-1,%dh
+ jz 0f
+ call movecursor
+0: cld
+1: movb $ATTRIB, %bl /* use standard text attribute by default */
+ lodsb
+ cmp $0377, %al /* a 0377 char means next one will be highlight */
+ jnz 3f
+ movb $HIGHLIGHT, %bl
+ lodsb
+3: cmp $13, %al /* CR is ignored */
+ jz 1b
+ or %al, %al /* end of messages */
+ jz 2f
+ cmp $10, %al /* LF: position onto next line and return from function */
+ jnz 5f
+ pushw %ax
+ call findcursor /* DX gets the current cursor position */
+ movb $255, %dl /* if we pass the end of the line, we'll begin on a new row */
+ call movecursor
+ popw %ax /* restore LF in AX */
+ jmp 2f /* and we return */
+5: movb $0, %bh /* current page = 0 */
+ movw $1, %cx /* 1 char */
+ movb $9, %ah /* write char with attribute */
+ int $0x10
+ call findcursor /* DX gets the current cursor position */
+ inc %dx /* next position */
+ call movecursor
+ jmp 1b
+movecursor:
+ cmp $RIGHTCOL, %dl /* did we override the end of the line ? */
+ jbe 1f /* no */
+ inc %dh /* yes, so position onto the next line */
+ movb $0, %dl
+1: movb $2, %ah
+4: movb $0, %bh
+ int $0x10
+2: ret
+findcursor:
+ mov $3, %ah
+ jmp 4b
+
+#if defined(UNDER_DEVEL) & defined(DEBUG)
+showhex:
+ pushfw
+ pushaw
```

Linux-Kernel: [PATCH] kmsgdump-2.6.6.patch (was: Panics need better handling)

```
+ movw %ax, %si
+ movw $4, %cx
+0: pushw %cx
+ rolw $4, %si
+ movw %si, %ax
+ andb $0xF, %al
+ addb $'0', %al
+ cmpb $'9', %al
+ jbe 1f
+ addb $7, %al
+1: movb $0xE, %ah
+ int $0x10
+ popw %cx
+ loop 0b
+#ifdef STEP_BY_STEP
+ movb $0, %ah
+ int $0x16
+#endif
+ popaw
+ popfw
+ ret
+#endif
+
+
+/*
+ * makefat():
+ * writes a 18-sectors/track FAT on the diskette, at logical sector 0.
+ * CF is set if an error occurs.
+ * SI is preserved.
+ *
+ */
+makefat:
+ cld
+ pushw %si
+ movb $STATUS_FORMATING, %al
+ call showstatus
+/* make DI point to the first free memory */
+ movw $(FREEMEMORY), %di
+ pushw %di /* will be used later */
+/* initialize the data area : 1 boot, 1 fat, 1 root */
+ movw $((FATSIZE+2)<<8), %cx
+ cld
+ xorw %ax, %ax
+ rep
+ stosw
+ popw %di
+ pushw %di
+/* make boot sector */
+ movw $(bootsect-kmsgdump+CODEORIGIN), %si
+ movw $(rootsect-bootsect), %cx
+ rep
```

Linux–Kernel: [PATCH] kmsgdump–2.6.6.patch (was: Panics need better handling)

```
+ movsb
+
+/* make fat1 (which is also fat2) */
+ movw $(FREEMEMORY+0x200), %di
+ movw $2, %bx /* first usable cluster is always 2 */
+ movw $(LOG_BUF_LEN >> 9), %cx /* #of sectors per FAT */
+
+ addw %bx, %cx /* decide now of the "end of chain" */
+ incw %bx
+ movb $0xF0, %al /* FAT ID */
+ stosb
+ movw $-1, %ax /* beginning of FAT */
+ stosw
+0: movw %bx, %ax
+ stosb
+ incw %bx /* next cluster */
+ cmpw %cx, %bx /* did we reach the last cluster ? */
+ jb 1f
+ movw $0xFFFF, %bx /* yes: mark "end of chain" */
+1: shlb $4, %ah
+ movb $0, %al
+ orw %bx, %ax
+ rolw $4, %ax
+ stosw /* write 2 bytes at once */
+ incw %bx /* next cluster */
+ testb $0x10, %bh /* are we above the "end of chain" magic number ? */
+ jz 0b /* not yet, let's go on for 2 more clusters */
+
+/* make root sector */
+ movw $(FREEMEMORY+0x200+FATSIZE<<9), %di
+ movw $(datasect-rootsect), %cx
+ rep
+ movsb
+ movw $(FIRSTCLUSTER), 14(%di) /* file begins at cluster 2 */
+ movw $(LOG_BUF_LEN), 16(%di) /* file length in bytes */
+
+/* in memory, we now have :
+ * BOOT | FAT | ROOT.
+ * We'll use a trick to write the two FATs :
+ * we first write (BOOT+FAT), and then append (FAT+ROOT)
+ */
+ popw %di /* beginning of the area to write */
+ movw $(1+FATSIZE), %cx /* one boot sector + N FAT sectors */
+ xorw %si, %si /* write onto the first sector */
+ pushw %cx /* fortunately, it will be the same size after... */
+ call wrsect /* let's write the boot sector onto the disk */
+ popw %cx /* 1+FATSIZE, let's save one byte of code */
+ jc 0f
+/* now write fat2 + root */
+ movw $(FREEMEMORY+0x200), %di
+ call wrsect /* let's write the boot sector onto the disk */
```

```

+0: popw %si
+ ret
+
+/*
+ * dumpdata():
+ * writes <LOG_BUF_LEN> bytes of data on drive DRIVENUM, from logical sector %si.
+ * CF is set if an error occurred.
+ *
+ */
+dumpdata:
+ clc
+ pushw %si
+ movb $STATUS_DUMPING, %al
+ call showstatus
+ movw $(LOG_BUF_LEN >> 9), %cx /* #of sectors that will be written */
+ movw $(BEGINOFMESSAGES), %di
+ call wrsect
+ popw %si
+ ret
+
+/* this function calls int 13 and if any error occurs, it will retry twice.
+ * all registers are kept except flags.
+ */
+int13retry:
+ pushaw
+ clc
+ int $0x13
+ jnc 0f
+ popaw
+ pushaw
+ clc
+ int $0x13
+0: popaw
+ ret
+
+/*
+ * wrsect():
+ * starts writing CX sectors of data from ES:DI to the diskette at logical sector SI.
+ * first logical sector is numbered 0. This function can write more than one track at
+ * a time, and is limited to the buffer's address in ES segment.
+ * all registers are modified, particularly :
+ * - SI which contains the first sector that should be written next time
+ * - DI which points to next data to be written
+ * - CX which tells how many sectors should be still written in case of CF=1
+ */
+wrsect:
+0: jcxz 9f /* when there are no more sectors to write, we return */
+ pushw %cx /* save this number of remaining sectors to write */
+ movw %si, %ax
+
+ movw $(NBHEADS*NBSECT), %cx /* we'll compute the starting track number */

```

Linux–Kernel: [PATCH] kmsgdump–2.6.6.patch (was: Panics need better handling)

```
+ divb %cl
+ xchgb %al, %ch /* al = 0, ch = track number */
+ xchgb %ah, %al /* ah = 0, al = head*nbsect + sect */
+ movb $(NBSECT), %cl
+ divb %cl
+ movb %al, %dh /* dh = head number */
+ xchgb %cl, %ah /* ah = NBSECT, cl = sector number (0–based) */
+ subb %cl, %ah /* ah = number of sectors before end of track */
+ popw %bx /* number of sectors to write */
+ movzbw %ah, %ax /* max #of sectors left on this track */
+ cmpw %ax, %bx /* does it fit ? */
+ ja 1f /* no, let's use AX */
+ movw %bx, %ax /* take all the remainder */
+1:
+ subw %ax, %bx /* subtract these sectors from the rest */
+ addw %ax, %si /* increment the start sector for the next round */
+ pushw %bx /* save the actual number of sectors to be written */
+ movb $3, %ah
```