Linux Magic System Request Key Hacks
Documentation for sysrq.c

* What is the magic SysRq key?

It is a 'magical' key combo you can hit which the kernel will respond to regardless of whatever else it is doing, unless it is completely locked up.

* How do I enable the magic SysRq key?

You need to say "yes" to 'Magic SysRq key (CONFIG_MAGIC_SYSRQ)' when configuring the kernel. When running a kernel with SysRq compiled in, /proc/sys/kernel/sysrq controls the functions allowed to be invoked via the SysRq key. By default the file contains 1 which means that every possible SysRq request is allowed (in older versions SysRq was disabled by default, and you were required to specifically enable it at run-time but this is not the case any more). Here is the list of possible values in /proc/sys/kernel/sysrq:

```
   0 - disable sysrq completely
   1 - enable all functions of sysrq
  >1 - bitmask of allowed sysrq functions (see below for detailed function
       description):
         2 - enable control of console logging level
         4 - enable control of keyboard (SAK, unraw)
         8 - enable debugging dumps of processes etc.
        16 - enable sync command
        32 - enable remount read-only
        64 - enable signalling of processes (term, kill, oom-kill)
       128 - allow reboot/poweroff
       256 - allow nicing of all RT tasks
```

You can set the value in the file by the following command:

```
    echo "number" >/proc/sys/kernel/sysrq
```

Note that the value of /proc/sys/kernel/sysrq influences only the invocation via a keyboard. Invocation of any operation via /proc/sysrq-trigger is always allowed (by a user with admin privileges).

* How do I use the magic SysRq key?

On x86 - You press the key combo 'ALT-SysRq-<command key>'. Note - Some keyboards may not have a key labeled 'SysRq'. The 'SysRq' key is also known as the 'Print Screen' key. Also some keyboards cannot handle so many keys being pressed at the same time, so you might have better luck with "press Alt", "press SysRq", "release SysRq", "press <command key>", release everything.

On SPARC - You press 'ALT-STOP-<command key>', I believe.

On the serial console (PC style standard serial ports only) - You send a BREAK, then within 5 seconds a command key. Sending BREAK twice is interpreted as a normal BREAK.

On PowerPC - Press 'ALT - Print Screen (or F13) - <command key>, Print Screen (or F13) - <command key> may suffice.

On other - If you know of the key combos for other architectures, please
let me know so I can add them to this section.

On all - write a character to /proc/sysrq-trigger. e.g.:

```
echo t > /proc/sysrq-trigger
```

* What are the 'command' keys?

'b' - Will immediately reboot the system without syncing or unmounting your disks.

'c' - Will perform a system crash by a NULL pointer dereference. A crashdump will be taken if configured.

'd' - Shows all locks that are held.

'e' - Send a SIGTERM to all processes, except for init.

'f' - Will call oom_kill to kill a memory hog process.

'g' - Used by kgdb on ppc and sh platforms.

'h' - Will display help (actually any other key than those listed here will display help. but 'h' is easy to remember :-)

'i' - Send a SIGKILL to all processes, except for init.

'j' - Forcibly "Just thaw it" - filesystems frozen by the FIFREEZE ioctl.

'k' - Secure Access Key (SAK) Kills all programs on the current virtual console. NOTE: See important comments below in SAK section.

'l' - Shows a stack backtrace for all active CPUs.

'm' - Will dump current memory info to your console.

'n' - Used to make RT tasks nice-able

'o' - Will shut your system off (if configured and supported).

'p' - Will dump the current registers and flags to your console.

'q' - Will dump per CPU lists of all armed hrtimers (but NOT regular timer_list timers) and detailed information about all clockevent devices.

'r' - Turns off keyboard raw mode and sets it to XLATE.

's' - Will attempt to sync all mounted filesystems.

't' - Will dump a list of current tasks and their information to your console.

'u' - Will attempt to remount all mounted filesystems read-only.

'v'	- Dumps Voyager SMP processor info to your console.

'w'	- Dumps tasks that are in uninterruptable (blocked) state.

'x'	- Used by xmon interface on ppc/powerpc platforms.

'z'	- Dump the ftrace buffer

'0'-'9' - Sets the console log level, controlling which kernel messages
	will be printed to your console. ('0', for example would make
	it so that only emergency messages like PANICs or OOPSes would
	make it to your console.)

* Okay, so what can I use them for?

Well, un'R'aw is very handy when your X server or a svgalib program crashes.

sa'K' (Secure Access Key) is useful when you want to be sure there is no
trojan program running at console which could grab your password
when you would try to login. It will kill all programs on given console,
thus letting you make sure that the login prompt you see is actually
the one from init, not some trojan program.
IMPORTANT: In its true form it is not a true SAK like the one in a :IMPORTANT
IMPORTANT: c2 compliant system, and it should not be mistaken as :IMPORTANT
IMPORTANT: such. :IMPORTANT
	It seems others find it useful as (System Attention Key) which is
useful when you want to exit a program that will not let you switch consoles.
(For example, X or a svgalib program.)

re'B'oot is good when you're unable to shut down. But you should also 'S'ync
and 'U'mount first.

'C'rash can be used to manually trigger a crashdump when the system is hung.
Note that this just triggers a crash if there is no dump mechanism available.

'S'ync is great when your system is locked up, it allows you to sync your
disks and will certainly lessen the chance of data loss and fscking. Note
that the sync hasn't taken place until you see the "OK" and "Done" appear
on the screen. (If the kernel is really in strife, you may not ever get the
OK or Done message...)

'U'mount is basically useful in the same ways as 'S'ync. I generally 'S'ync,
'U'mount, then re'B'oot when my system locks. It's saved me many a fsck.
Again, the unmount (remount read-only) hasn't taken place until you see the
"OK" and "Done" message appear on the screen.

The loglevels '0'-'9' are useful when your console is being flooded with
kernel messages you do not want to see. Selecting '0' will prevent all but
the most urgent kernel messages from reaching your console. (They will
still be logged if syslogd/klogd are alive, though.)

t'E'rm and k'I'll are useful if you have some sort of runaway process you
are unable to kill any other way, especially if it's spawning other
processes.

"'J'ust thaw it" is useful if your system becomes unresponsive due to a frozen

(probably root) filesystem via the FIFREEZE ioctl.

* Sometimes SysRq seems to get 'stuck' after using it, what can I do?

That happens to me, also. I've found that tapping shift, alt, and control on both sides of the keyboard, and hitting an invalid sysrq sequence again will fix the problem. (i.e., something like alt-sysrq-z). Switching to another virtual console (ALT+Fn) and then back again should also help.

* I hit SysRq, but nothing seems to happen, what's wrong?

There are some keyboards that produce a different keycode for SysRq than the pre-defined value of 99 (see KEY_SYSRQ in include/linux/input.h), or which don't have a SysRq key at all. In these cases, run 'showkey -s' to find an appropriate scancode sequence, and use 'setkeycodes <sequence> 99' to map this sequence to the usual SysRq code (e.g., 'setkeycodes e05b 99'). It's probably best to put this command in a boot script. Oh, and by the way, you exit 'showkey' by not typing anything for ten seconds.

* I want to add SysRQ key events to a module, how does it work?

In order to register a basic function with the table, you must first include the header 'include/linux/sysrq.h', this will define everything else you need. Next, you must create a sysrq_key_op struct, and populate it with A) the key handler function you will use, B) a help_msg string, that will print when SysRQ prints help, and C) an action_msg string, that will print right before your handler is called. Your handler must conform to the prototype in 'sysrq.h'.

After the sysrq_key_op is created, you can call the kernel function register_sysrq_key(int key, struct sysrq_key_op *op_p); this will register the operation pointed to by 'op_p' at table key 'key', if that slot in the table is blank. At module unload time, you must call the function unregister_sysrq_key(int key, struct sysrq_key_op *op_p), which will remove the key op pointed to by 'op_p' from the key 'key', if and only if it is currently registered in that slot. This is in case the slot has been overwritten since you registered it.

The Magic SysRQ system works by registering key operations against a key op lookup table, which is defined in 'drivers/char/sysrq.c'. This key table has a number of operations registered into it at compile time, but is mutable, and 2 functions are exported for interface to it:

        register_sysrq_key and unregister_sysrq_key.

Of course, never ever leave an invalid pointer in the table. I.e., when your module that called register_sysrq_key() exits, it must call unregister_sysrq_key() to clean up the sysrq key table entry that it used. Null pointers in the table are always safe. :)

If for some reason you feel the need to call the handle_sysrq function from within a function called by handle_sysrq, you must be aware that you are in a lock (you are also in an interrupt handler, which means don't sleep!), so you must call __handle_sysrq_nolock instead.

* When I hit a SysRq key combination only the header appears on the console?

Sysrq output is subject to the same console loglevel control as all other console output. This means that if the kernel was booted 'quiet'

as is common on distro kernels the output may not appear on the actual
console, even though it will appear in the dmesg buffer, and be accessible
via the dmesg command and to the consumers of /proc/kmsg. As a specific
exception the header line from the sysrq command is passed to all console
consumers as if the current loglevel was maximum. If only the header
is emitted it is almost certain that the kernel loglevel is too low.
Should you require the output on the console channel then you will need
to temporarily up the console loglevel using alt-sysrq-8 or:

    echo 8 > /proc/sysrq-trigger

Remember to return the loglevel to normal after triggering the sysrq
command you are interested in.

* I have more questions, who can I ask?

And I'll answer any questions about the registration system you got, also
responding as soon as possible.
 -Crutcher

* Credits

Written by Mydraal <vulpyne@vulpyne.net>
Updated by Adam Sulmicki <adam@cfar.umd.edu>
Updated by Jeremy M. Dolan <jmd@turbogeek.org> 2001/01/28 10:15:59
Added to by Crutcher Dunnavant <crutcher+kernel@datastacks.com>