# Kthreads, Mutexes, and Debugging

Sarah Diesburg
CS 3430
Operating Systems

# Story of Kernel Development

Some context…

# In the old days…

- There were no modules or virtual machines
- The kernel is a program
    - Has code, can compile, re-compile, make executable
    - When changes needed to be made, developers make changes in source and re-compile

# How is the kernel different from a regular program?

- Mostly in how it is executed
  - Boot loader loads the kernel image/executable during boot time
  - Sets kernel mode
  - Jumps to the entry point in the image/executable
- Remember the generic booting sequence?

# Quick Question

- How would you make changes to the kernel and run those changes?
  1. Make changes to the source
  2. Re-complie the kernel source
  3. Re-install the kernel source
  4. Make sure the bootloader sees the new kernel image (grub)
  5. Reboot and profit!

# Getting more modern..

- Modules were created as bits of code that can be loaded and unloaded by the kernel in kernel mode
- Made development easier
  - Instead of re-compiling, re-installing, and rebooting into the new kernel, one could just re-compile and load a module

# Quick Question

- How would you make changes to a module and run those changes?
  1. Make changes to module source code
  2. Re-compile the module
  3. Load the new module

# Present Day

- Reboots into new kernels and loading new modules often freezes machines
- Enter virtual machine software
  - **Process** that emulates the hardware necessary to run an OS in user-space
  - Guest OS is executed inside the virtual machine process!

# Kthreads

Run the main logic of your module in a kthread!

# Refresher: hello.c

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE(“Dual BSD/GPL”);

static int hello_init(void)
{
   printk(KERN_ALERT “Hello, world!\n”);
   return 0;
}

static void hello_exit(void)
{
   printk(KERN_ALERT “Goodbye, sleepy world.\n”);
}

module_init(hello_init);
module_exit(hello_exit);
```

# Kernel Modules

- Remember, kernel modules are very ***event-based***
- We need a way to start an independent thread of execution in response to an event
  - e.g. start_kitchen() for project 2…

# kthread_run

```
kthread_run(threadfn, data, namefmt, ...)
```

- Creates a new thread and tells it to run
  - `threadfn` – the name of the function the thread should run
  - `data` – data pointer for `threadfn` (can be NULL if the function does not take any args)
  - `namefmt` – name of the thread (displayed during “ps” command)
- Returns a task_struct

# kthread_run example

```
struct task_struct *t;

t = kthread_run(run, NULL, “my_elevator");
        if (IS_ERR(t)){
                ret=PTR_ERR(t);
        }
```

# kthread_stop

```
int kthread_stop(struct task_struct * k);
```

- Sets `kthread_should_stop` for k to return true, wakes the thread, and waits for the thread to exit
- Returns the result of the thread function

# kthread_stop_example

```
ret=kthread_stop(t);
if(ret != -EINTR)
  printk("Main logic tread stopped.\n“);
```

# Thread Function Example

```
static int run(void *arg)
{
        /* Lock here */

        while(!kthread_should_stop()) {

                /* Do stuff */
         }

        /* Unlock here */
        printk("%s: kernel thread exits.\n",
  __FUNCTION__);
        return 0;
}
```

# Sample kthread code

- Take a look at the sample module that spawns a kthread on load
  - You will want to move this kthread to start when user writes a “0” to /proc/kitchen
  - You will want to stop the kthread when user writes a “-1” to /proc/kitchen

# Concurrency Aspects of Project 2

- Synchronizing access to request queue(s)
  - Multiple producers may access request queue(s) at the same time
  - Multiple consumers may access request queue(s) at the same time
- Synchronizing access to other global data

# Kitchen Queue Concurrency

- Orders may appear on the queue at the same time the kitchen module checks the queue
- The status may be read at the same time that you're updating
  - Number of orders that you've serviced
  - Which order is currently being processed
  - Which slot in the queue kitchen is looking at
- How do you guarantee correctness?

# Global Data vs. Local Data

- ***Global data*** is declared at global scope, e.g. outside of any function body
  - Often necessary for kernel programming
- Particularly sensitive to concurrency issues
  - Be **extra** careful when handling globals

# Global Data vs. Local Data

- ***Local data*** is declared within a function
- Local data is sensitive to concurrency issues when it depends on global data or when parallel access is possible
  - Think carefully about whether it needs to be synchronized

# Synchronization Primitives

- Semaphores
    - User space
    - Kernel space
- Mutexes
    - User space
    - Kernel space
- Spin locks
    - Kernel space
- Atomic Functions

# Synchronization Primitives (We'll Only Cover These)

- Mutexes
  - User space
  - Kernel space

- Does anyone remember the differences between a mutex and semaphore?

# The Mutex

Caught up in the mutex?

# Mutexes

- Mutex – A construct to provide MUTual EXclusion
- Based on the concept of a semaphore
- Found at <source_dir>/include/linux/mutex.h
- Can be locked or unlocked
    - Only one thread of execution may hold the lock at a time

# Kernel-Space Mutex - Initialization

- `mutex_init(&mutex)`

- Declare and initialize a mutex
  - Only initialize it once

# Kernel-Space Mutex - Locking

```
void mutex_lock(struct mutex *);

int  mutex_lock_interruptible(struct mutex *);
```

- mutex_lock() can wait indefinitely
- mutex_lock_interruptible() locks a mutex as long as it is not interrupted
  - returns 0 if locking succeeded, < 0 if interrupted
- Use of the interruptible version is typically preferred

# Kernel-Space Mutex – Unlocking

```
void mutex_unlock(struct mutex *);
```

- Guarantees that the mutex is unlocked
  - Why is there no interruptible version of this function?

# Kernel-Space Mutex Example

```
/* Declare your mutex */
struct mutex my_mutex;

/* Initialize your mutex */
mutex_init(&my_mutex);

/* Lock */
if(mutex_lock_interruptible(&my_mutex))
  return -ERESTARTSYS;

  /* Do stuff to protected global variables */

/* Unlock */
mutex_unlock(&my_mutex);
```

# User-Space Mutex

- Also used with pthreads in regular user applications
  - pthreads operate very similar to kthreads
  - Might be useful if you are prototyping your elevator in user-space before porting to kernel

# User-Space Mutex - Initialization

```
int pthread_mutex_init(pthread_mutex_t *, NULL);

int pthread_mutex_destroy(pthread_mutex_t *);
```

- Pthread_mutex_init() dynamically allocates a mutex
- Pthread_mutex_destroy() frees a mutex

# User-Space Mutex - Locking

- `int pthread_mutex_lock(pthread_mutex_t *);`

- Returns 0 on locking, < 0 otherwise

# User-Space Mutex - Unlocking

```
int pthread_mutex_unlock(pthread_mutex_t *);
```

- Returns 0 on unlocking, < 0 otherwise

# Kitchen Scheduling Advice

# General Advice

- Just make kitchen work first
  - Use a very simple algorithm
  - My kitchen search the queue in round-robin fashion and processes every Beef Wellington
- Optimize if there is time

# Round Robin

- Method:
    - Service requests in round-robin fashion (e.g. queue slot 0, 1, 2, 3, etc.)

- Pros/Cons?

# Shortest Job First

- Method:
  - Service fastest orders first
- Pros/Cons?

# SCAN

- Method:
  - Service requests in one direction, then go backwards
- Pros/Cons?

# Hybrid

- Combine methods, come up with something new
- Up to your creativity

# Project Demos and Deliverables

# Basic Information

- Project 2 is due on October 26th
  - Due at demo time
- Please sign up for a time to demo your elevator project
- If you wish to use slack days, make an appointment with me on the day you wish to turn in your project

# Project Deliverables

- Before final demo, please zip and upload the following to eLearning
  - README
  - Part 2 – source and Makefile
  - Part 3 – source and Makefile

- You may have to copy the files to vermin or your workstation to zip them (or else read the man page on ‘tar’ to zip on your virtual machine).

# Demo

- Will only have time to demo Part 3 – elevator
- Please look at grading sheet to understand what I will be looking for

# Getting Help

- Sign up for a halfway demo
- Regular office hours

# Other Hints

- This is not a simple project
  - Setup is different
  - You need to use different files and methods of compilation/running
  - Do NOT wait until 2 days before it is due to start
    - Too late
  - The Internet will likely NOT be very helpful

# Other Hints

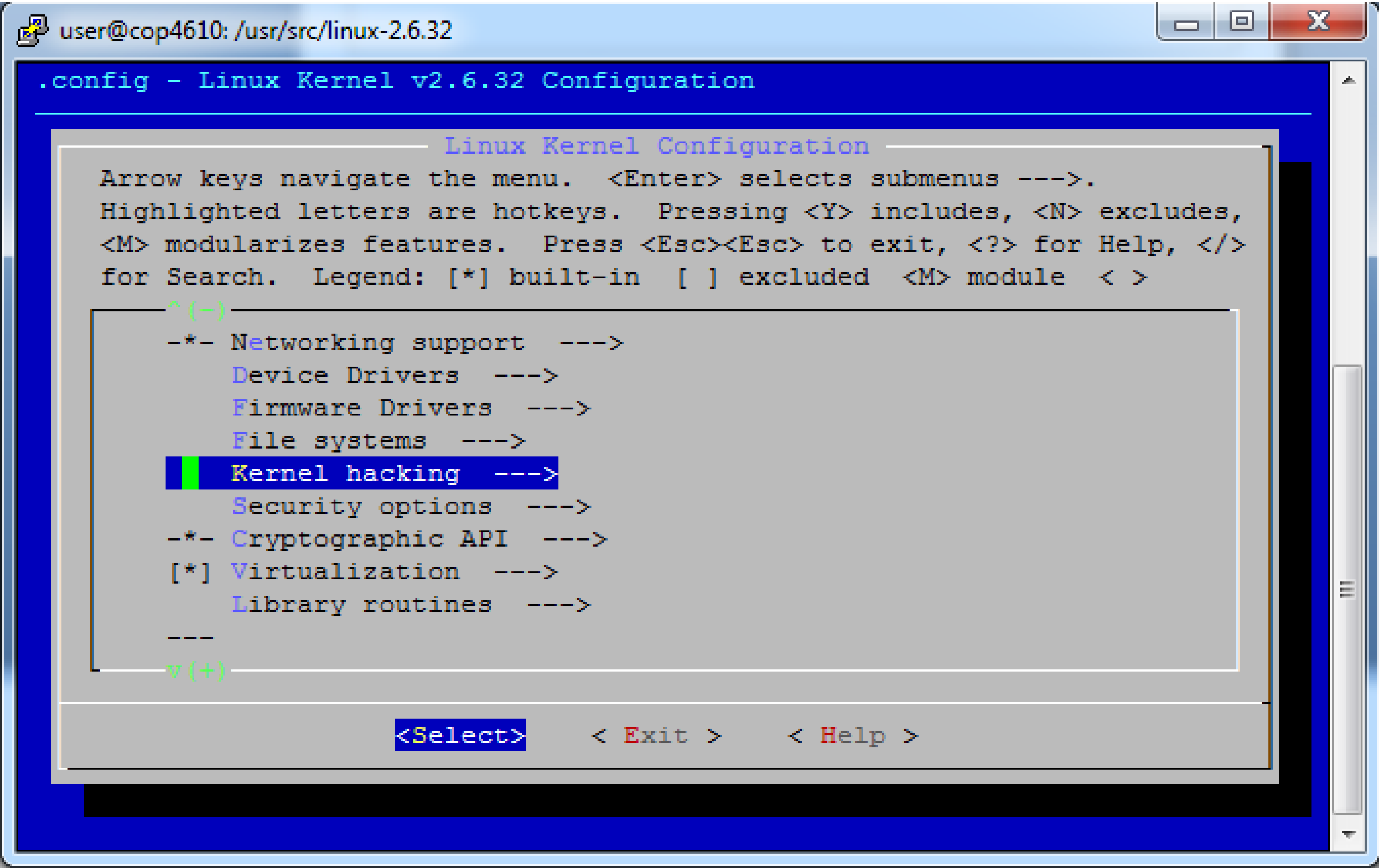
- Set milestones over the next few weeks
- Ask questions early
  - If it’s a good question, I’ll share it with the class

# Debugging

# Kernel Debugging Configurations

- Timing info on printks
- __depreciated logic
- Detection of hung tasks
- SLUB debugging
- Kernel memory leak detector
- Mutex/lock debugging
- Kmemcheck
- Check for stack overflow
- Linked list debugging

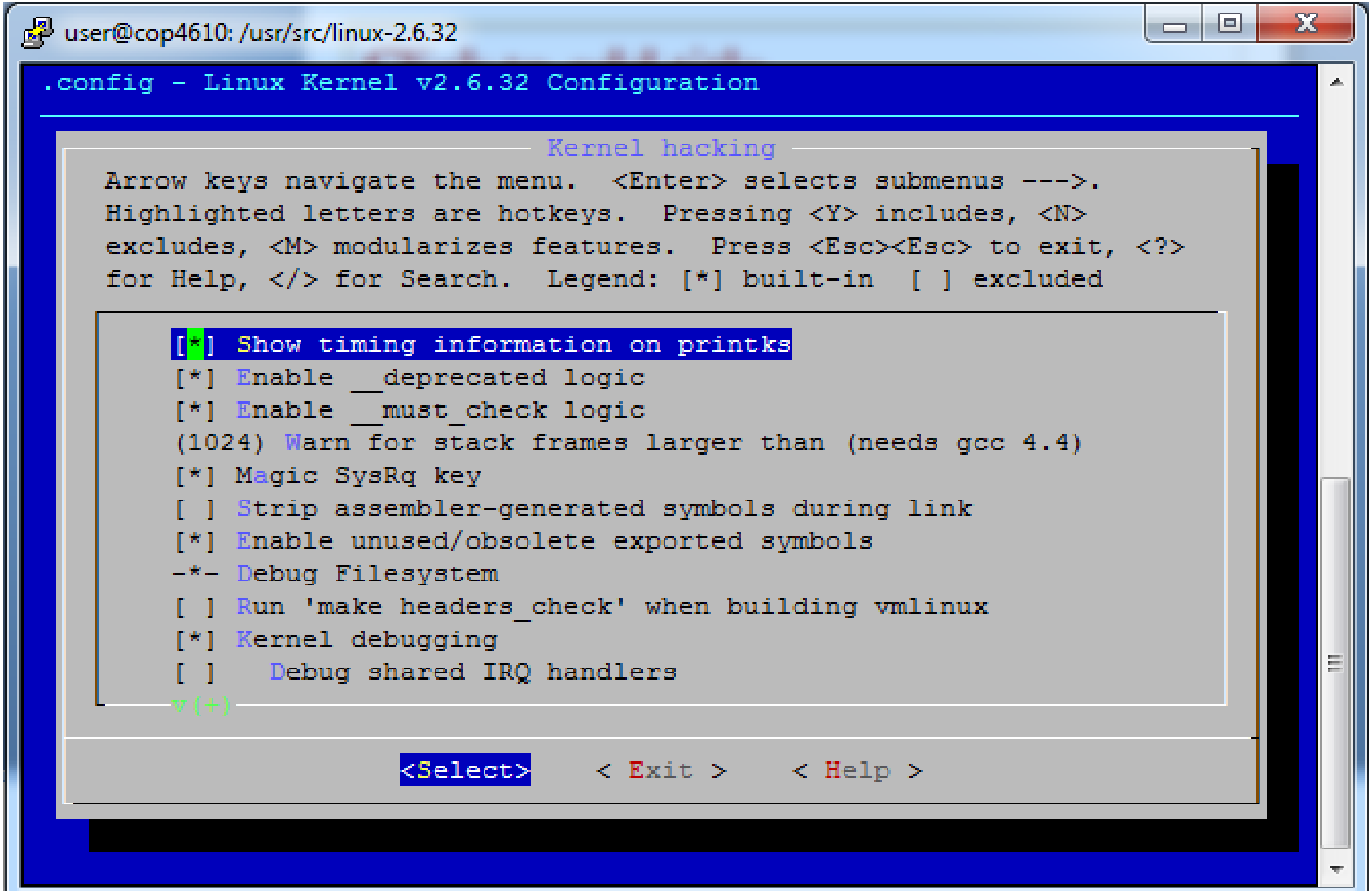
# Select Kernel Hacking



user@cop4610: /usr/src/linux-2.6.32

```
.config - Linux Kernel v2.6.32 Configuration

                    Linux Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >
       ^(-)
       -*- Networking support  --->
           Device Drivers  --->
           Firmware Drivers  --->
           File systems  --->
           Kernel hacking  --->
           Security options  --->
       -*- Cryptographic API  --->
       [*] Virtualization  --->
           Library routines  --->
       ---
       v(+)

              <Select>    < Exit >    < Help >
```

# Enable Debugging Options



user@cop4610: /usr/src/linux-2.6.32

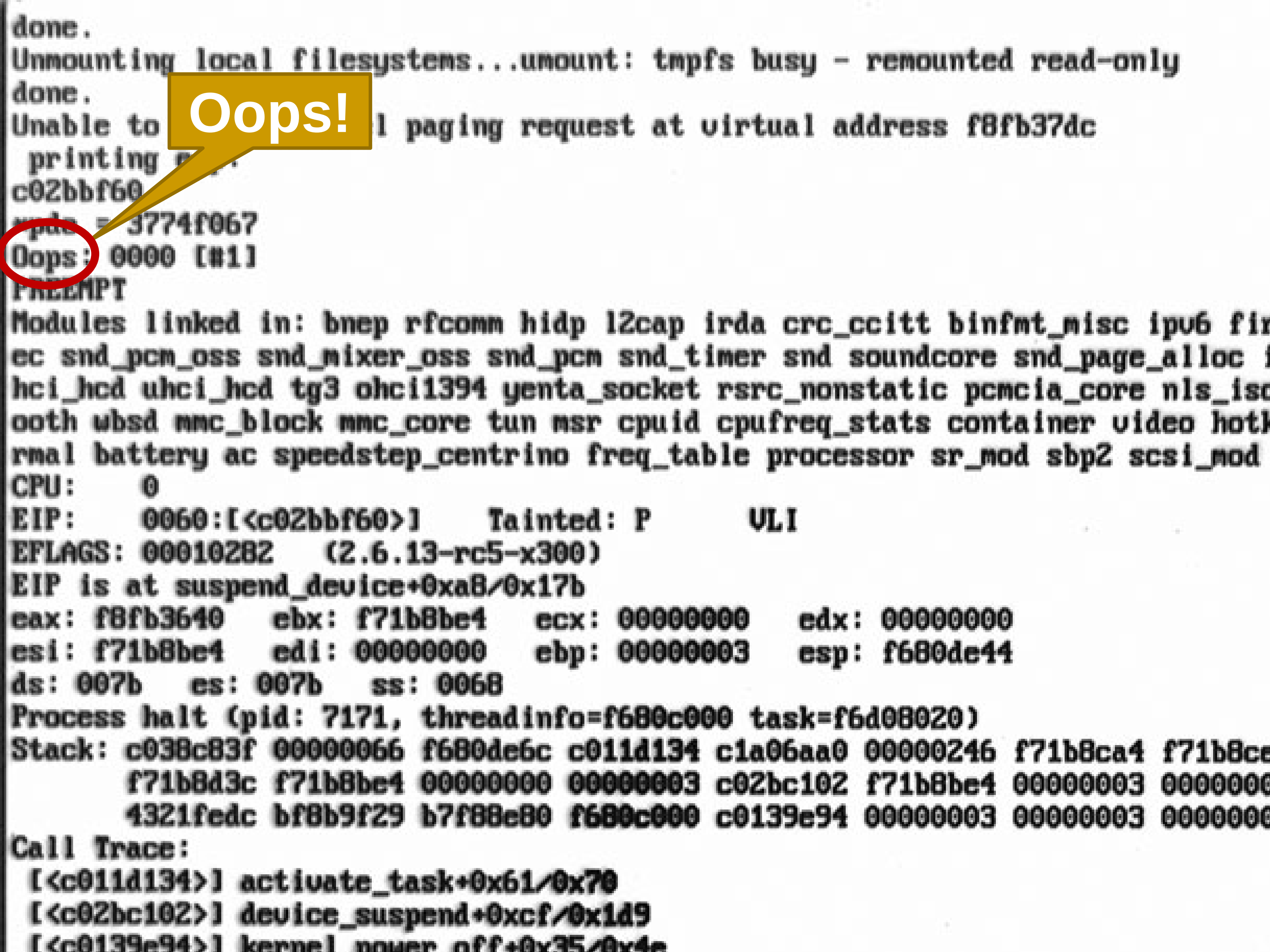
```
.config - Linux Kernel v2.6.32 Configuration

                         Kernel hacking
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N>
  excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?>
  for Help, </> for Search.  Legend: [*] built-in  [ ] excluded

      [*] Show timing information on printks
      [*] Enable __deprecated logic
      [*] Enable __must_check logic
      (1024) Warn for stack frames larger than (needs gcc 4.4)
      [*] Magic SysRq key
      [ ] Strip assembler-generated symbols during link
      [*] Enable unused/obsolete exported symbols
      -*- Debug Filesystem
      [ ] Run 'make headers_check' when building vmlinux
      [*] Kernel debugging
      [ ]   Debug shared IRQ handlers
      v(+)

              <Select>    < Exit >    < Help >
```

# Debugging through reads to /proc/kitchen

- Necessary to help you “see” what’s going on!
- General process
  - Identify data to monitor in your module
  - Run your module
  - Query /proc/kitchen for that information at any time

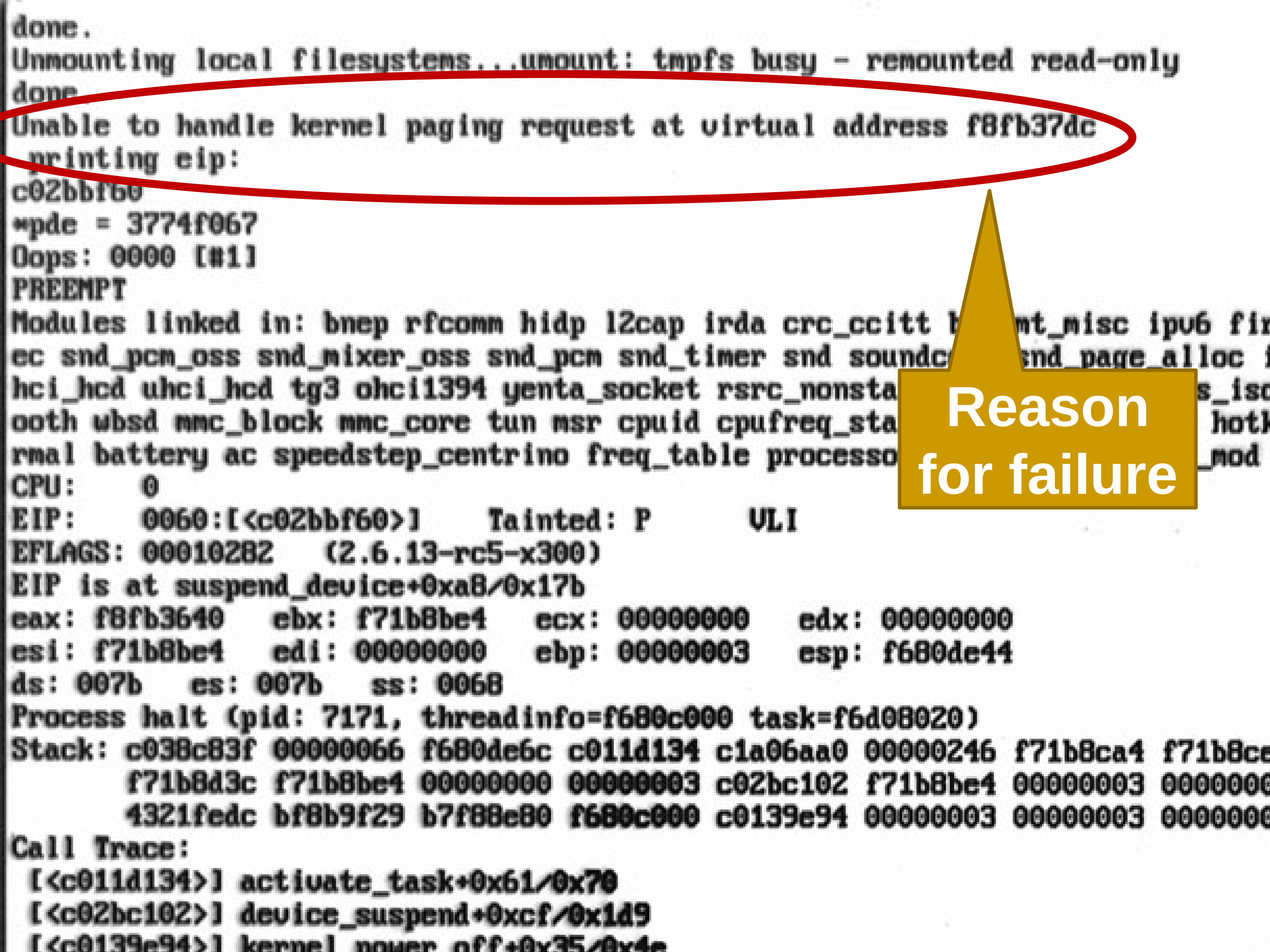
# Kernel Oops and Other Errors

- Kernel errors often only appear on first tty (terminal interface)
  - Why?



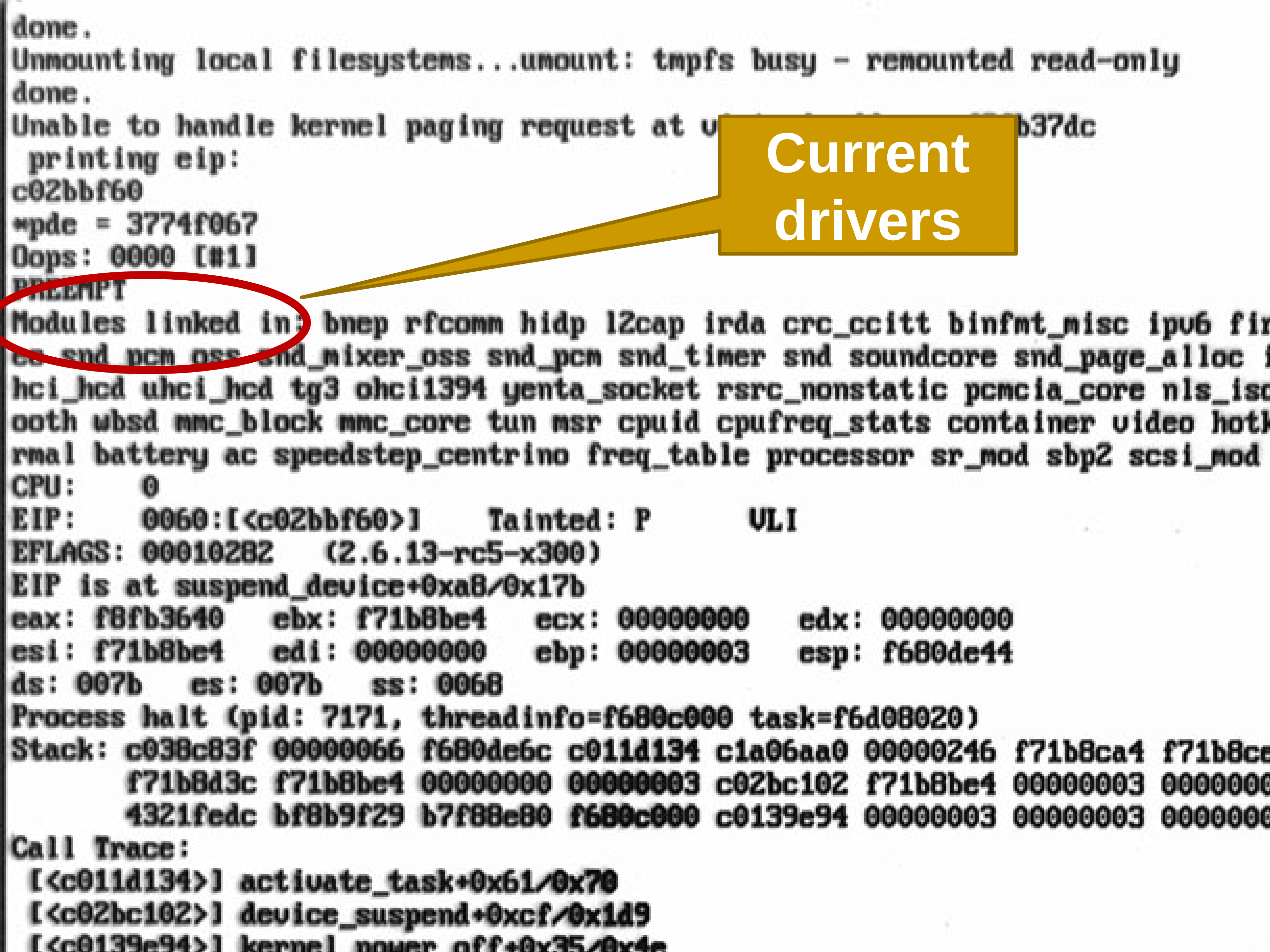
Oops!

```
done.
Unmounting local filesystems...umount: tmpfs busy - remounted read-only
done.
Unable to           l paging request at virtual address f8fb37dc
 printing e
c02bbf60
*pde = 3774f067
Oops: 0000 [#1]
PREEMPT
Modules linked in: bnep rfcomm hidp l2cap irda crc_ccitt binfmt_misc ipv6 fir
ec snd_pcm_oss snd_mixer_oss snd_pcm snd_timer snd soundcore snd_page_alloc f
hci_hcd uhci_hcd tg3 ohci1394 yenta_socket rsrc_nonstatic pcmcia_core nls_iso
ooth wbsd mmc_block mmc_core tun msr cpuid cpufreq_stats container video hotk
rmal battery ac speedstep_centrino freq_table processor sr_mod sbp2 scsi_mod
CPU:    0
EIP:    0060:[<c02bbf60>]    Tainted: P      VLI
EFLAGS: 00010282   (2.6.13-rc5-x300)
EIP is at suspend_device+0xa8/0x17b
eax: f8fb3640   ebx: f71b8be4   ecx: 00000000   edx: 00000000
esi: f71b8be4   edi: 00000000   ebp: 00000003   esp: f680de44
ds: 007b   es: 007b   ss: 0068
Process halt (pid: 7171, threadinfo=f680c000 task=f6d08020)
Stack: c038c83f 00000066 f680de6c c011d134 c1a06aa0 00000246 f71b8ca4 f71b8ce
       f71b8d3c f71b8be4 00000000 00000003 c02bc102 f71b8be4 00000003 0000000
       4321fedc bf8b9f29 b7f88e80 f680c000 c0139e94 00000003 00000003 0000000
Call Trace:
 [<c011d134>] activate_task+0x61/0x70
 [<c02bc102>] device_suspend+0xcf/0x1d9
 [<c0139e94>] kernel_power_off+0x35/0x4e
```



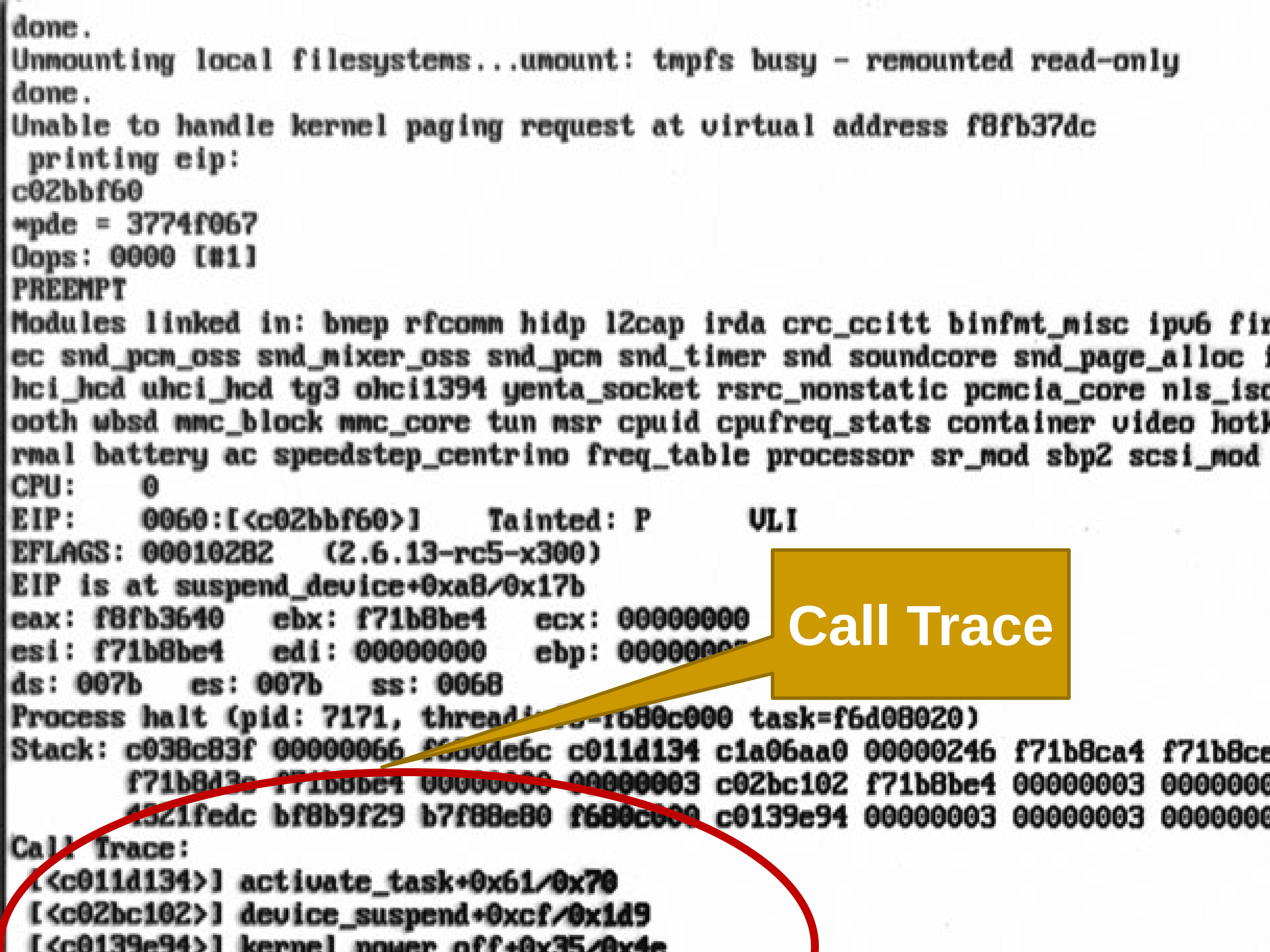
```
done.
Unmounting local filesystems...umount: tmpfs busy - remounted read-only
done.
Unable to handle kernel paging request at virtual address f8fb37dc
 printing eip:
c02bbf60
*pde = 3774f067
Oops: 0000 [#1]
PREEMPT
Modules linked in: bnep rfcomm hidp l2cap irda crc_ccitt b     mt_misc ipv6 fi
ec snd_pcm_oss snd_mixer_oss snd_pcm snd_timer snd soundc     snd_page_alloc i
hci_hcd uhci_hcd tg3 ohci1394 yenta_socket rsrc_nonsta                 s_isc
ooth wbsd mmc_block mmc_core tun msr cpuid cpufreq_sta                 hotk
rmal battery ac speedstep_centrino freq_table processo                 _mod
CPU:    0
EIP:    0060:[<c02bbf60>]    Tainted: P      VLI
EFLAGS: 00010282   (2.6.13-rc5-x300)
EIP is at suspend_device+0xa8/0x17b
eax: f8fb3640   ebx: f71b8be4   ecx: 00000000   edx: 00000000
esi: f71b8be4   edi: 00000000   ebp: 00000003   esp: f680de44
ds: 007b   es: 007b   ss: 0068
Process halt (pid: 7171, threadinfo=f680c000 task=f6d08020)
Stack: c038c83f 00000066 f680de6c c011d134 c1a06aa0 00000246 f71b8ca4 f71b8ce
       f71b8d3c f71b8be4 00000000 00000003 c02bc102 f71b8be4 00000003 0000000
       4321fedc bf8b9f29 b7f88e80 f680c000 c0139e94 00000003 00000003 0000000
Call Trace:
 [<c011d134>] activate_task+0x61/0x70
 [<c02bc102>] device_suspend+0xcf/0x1d9
 [<c0139e94>] kernel_power_off+0x35/0x4e
```

Reason for failure



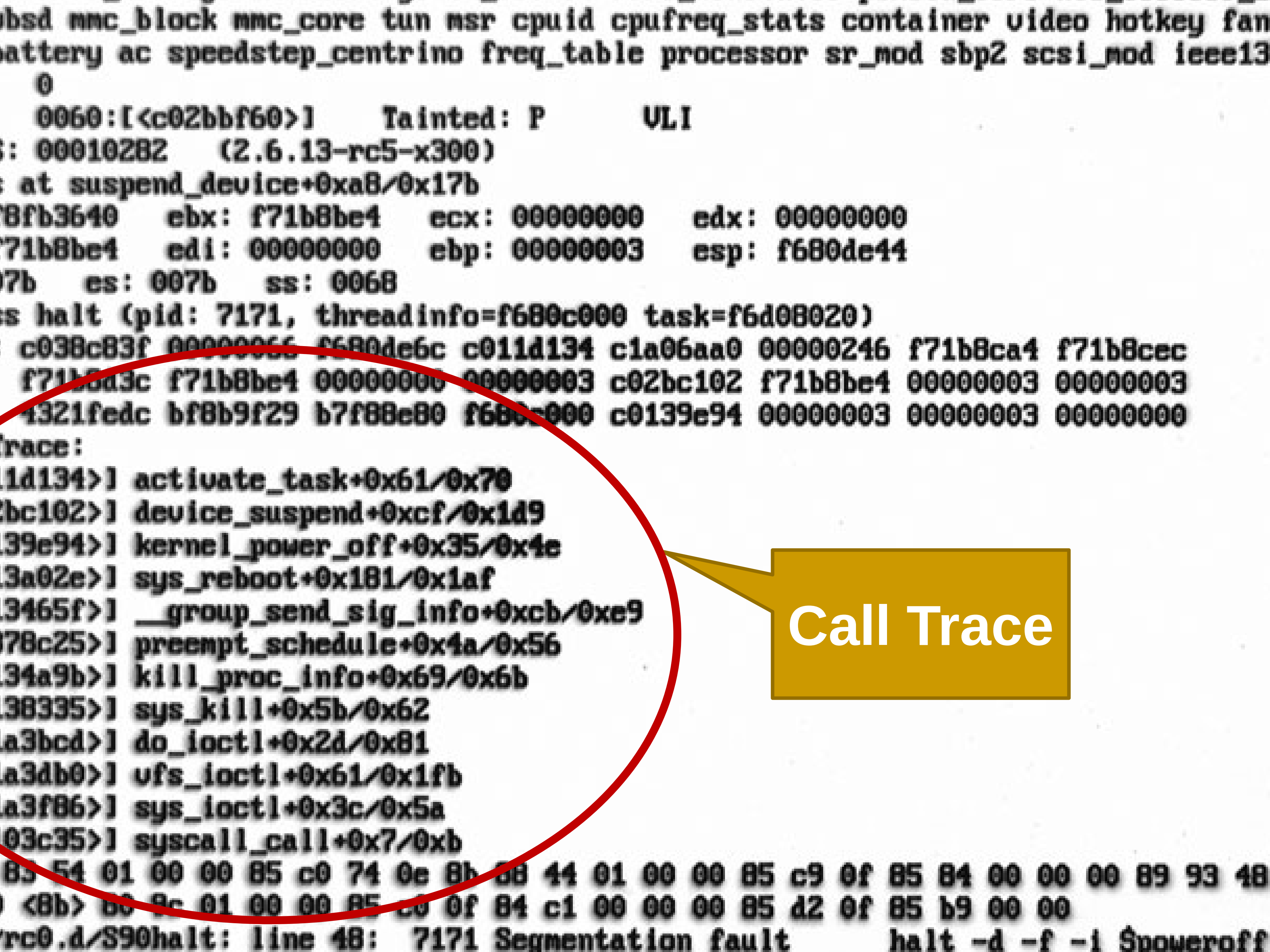
```
done.
Unmounting local filesystems...umount: tmpfs busy - remounted read-only
done.
Unable to handle kernel paging request at v[illegible]b37dc
 printing eip:
c02bbf60
*pde = 3774f067
Oops: 0000 [#1]
PREEMPT
Modules linked in: bnep rfcomm hidp l2cap irda crc_ccitt binfmt_misc ipv6 fir
ec snd_pcm_oss snd_mixer_oss snd_pcm snd_timer snd soundcore snd_page_alloc f
hci_hcd uhci_hcd tg3 ohci1394 yenta_socket rsrc_nonstatic pcmcia_core nls_isc
ooth wbsd mmc_block mmc_core tun msr cpuid cpufreq_stats container video hotk
rmal battery ac speedstep_centrino freq_table processor sr_mod sbp2 scsi_mod
CPU:    0
EIP:    0060:[<c02bbf60>]    Tainted: P      VLI
EFLAGS: 00010282   (2.6.13-rc5-x300)
EIP is at suspend_device+0xa8/0x17b
eax: f8fb3640   ebx: f71b8be4   ecx: 00000000   edx: 00000000
esi: f71b8be4   edi: 00000000   ebp: 00000003   esp: f680de44
ds: 007b   es: 007b   ss: 0068
Process halt (pid: 7171, threadinfo=f680c000 task=f6d08020)
Stack: c038c83f 00000066 f680de6c c011d134 c1a06aa0 00000246 f71b8ca4 f71b8ce
       f71b8d3c f71b8be4 00000000 00000003 c02bc102 f71b8be4 00000003 0000000
       4321fedc bf8b9f29 b7f88e80 f680c000 c0139e94 00000003 00000003 0000000
Call Trace:
 [<c011d134>] activate_task+0x61/0x70
 [<c02bc102>] device_suspend+0xcf/0x1d9
 [<c0139e94>] kernel_power_off+0x35/0x4e
```

Current drivers



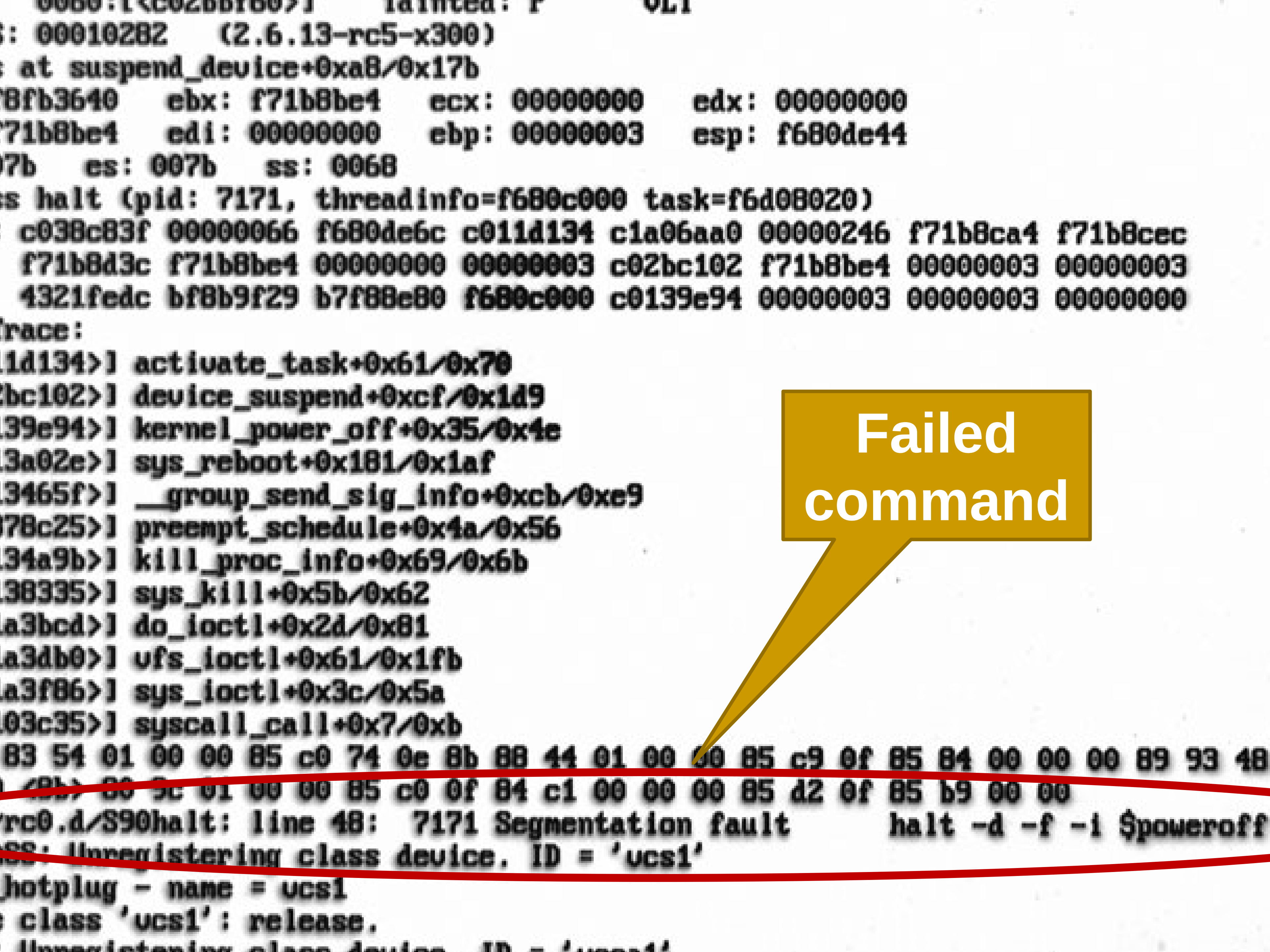
```
done.
Unmounting local filesystems...umount: tmpfs busy - remounted read-only
done.
Unable to handle kernel paging request at virtual address f8fb37dc
 printing eip:
c02bbf60
*pde = 3774f067
Oops: 0000 [#1]
PREEMPT
Modules linked in: bnep rfcomm hidp l2cap irda crc_ccitt binfmt_misc ipv6 fir
ec snd_pcm_oss snd_mixer_oss snd_pcm snd_timer snd soundcore snd_page_alloc f
hci_hcd uhci_hcd tg3 ohci1394 yenta_socket rsrc_nonstatic pcmcia_core nls_iso
ooth wbsd mmc_block mmc_core tun msr cpuid cpufreq_stats container video hotk
rmal battery ac speedstep_centrino freq_table processor sr_mod sbp2 scsi_mod
CPU:    0
EIP:    0060:[<c02bbf60>]    Tainted: P      VLI
EFLAGS: 00010282   (2.6.13-rc5-x300)
EIP is at suspend_device+0xa8/0x17b
eax: f8fb3640   ebx: f71b8be4   ecx: 00000000
esi: f71b8be4   edi: 00000000   ebp: 00000000
ds: 007b   es: 007b   ss: 0068
Process halt (pid: 7171, threadinfo=f680c000 task=f6d08020)
Stack: c038c83f 00000066 f680de6c c011d134 c1a06aa0 00000246 f71b8ca4 f71b8ce
       f71b8d3c f71b8be4 00000000 00000003 c02bc102 f71b8be4 00000003 0000000
       4321fedc bf8b9f29 b7f88e80 f680c000 c0139e94 00000003 00000003 0000000
Call Trace:
 [<c011d134>] activate_task+0x61/0x70
 [<c02bc102>] device_suspend+0xcf/0x1d9
 [<c0139e94>] kernel_power_off+0x35/0x4e
```

Call Trace



```
bsd mmc_block mmc_core tun msr cpuid cpufreq_stats container video hotkey fan
attery ac speedstep_centrino freq_table processor sr_mod sbp2 scsi_mod ieee13
  0
  0060:[<c02bbf60>]    Tainted: P      VLI
: 00010282   (2.6.13-rc5-x300)
 at suspend_device+0xa8/0x17b
8fb3640   ebx: f71b8be4   ecx: 00000000   edx: 00000000
71b8be4   edi: 00000000   ebp: 00000003   esp: f680de44
7b   es: 007b   ss: 0068
s halt (pid: 7171, threadinfo=f680c000 task=f6d08020)
 c038c83f 00000066 f680de6c c011d134 c1a06aa0 00000246 f71b8ca4 f71b8cec
 f71b8d3c f71b8be4 00000000 00000003 c02bc102 f71b8be4 00000003 00000003
 4321fedc bf8b9f29 b7f88e80 f680e000 c0139e94 00000003 00000003 00000000
race:
1d134>] activate_task+0x61/0x70
bc102>] device_suspend+0xcf/0x1d9
39e94>] kernel_power_off+0x35/0x4e
3a02e>] sys_reboot+0x181/0x1af
3465f>] __group_send_sig_info+0xcb/0xe9
78c25>] preempt_schedule+0x4a/0x56
34a9b>] kill_proc_info+0x69/0x6b
38335>] sys_kill+0x5b/0x62
a3bcd>] do_ioctl+0x2d/0x81
a3db0>] vfs_ioctl+0x61/0x1fb
a3f86>] sys_ioctl+0x3c/0x5a
03c35>] syscall_call+0x7/0xb
83 54 01 00 00 85 c0 74 0e 8b 88 44 01 00 00 85 c9 0f 85 84 00 00 00 89 93 48
 <8b> 80 8c 01 00 00 85 c0 0f 84 c1 00 00 00 85 d2 0f 85 b9 00 00
rc0.d/S90halt: line 48:  7171 Segmentation fault     halt -d -f -i $poweroff
```

Call Trace



```
0060:[<c02bbf60>]    Tainted: P      VLI
: 00010282   (2.6.13-rc5-x300)
 at suspend_device+0xa8/0x17b
8fb3640   ebx: f71b8be4   ecx: 00000000   edx: 00000000
71b8be4   edi: 00000000   ebp: 00000003   esp: f680de44
7b   es: 007b   ss: 0068
s halt (pid: 7171, threadinfo=f680c000 task=f6d08020)
 c038c83f 00000066 f680de6c c011d134 c1a06aa0 00000246 f71b8ca4 f71b8cec
 f71b8d3c f71b8be4 00000000 00000003 c02bc102 f71b8be4 00000003 00000003
 4321fedc bf8b9f29 b7f88e80 f680c000 c0139e94 00000003 00000003 00000000
race:
1d134>] activate_task+0x61/0x70
bc102>] device_suspend+0xcf/0x1d9
39e94>] kernel_power_off+0x35/0x4e
3a02e>] sys_reboot+0x181/0x1af
3465f>] __group_send_sig_info+0xcb/0xe9
78c25>] preempt_schedule+0x4a/0x56
34a9b>] kill_proc_info+0x69/0x6b
38335>] sys_kill+0x5b/0x62
a3bcd>] do_ioctl+0x2d/0x81
a3db0>] vfs_ioctl+0x61/0x1fb
a3f86>] sys_ioctl+0x3c/0x5a
03c35>] syscall_call+0x7/0xb
83 54 01 00 00 85 c0 74 0e 8b 88 44 01 00 00 85 c9 0f 85 84 00 00 00 89 93 48
 <8b> 80 9c 01 00 00 85 c0 0f 84 c1 00 00 00 85 d2 0f 85 b9 00 00
rc0.d/S90halt: line 48:  7171 Segmentation fault      halt -d -f -i $poweroff
SS: Unregistering class device. ID = 'vcs1'
hotplug - name = vcs1
 class 'vcs1': release.
 Unregistering class device. ID = 'vcsa1'
```

Failed command