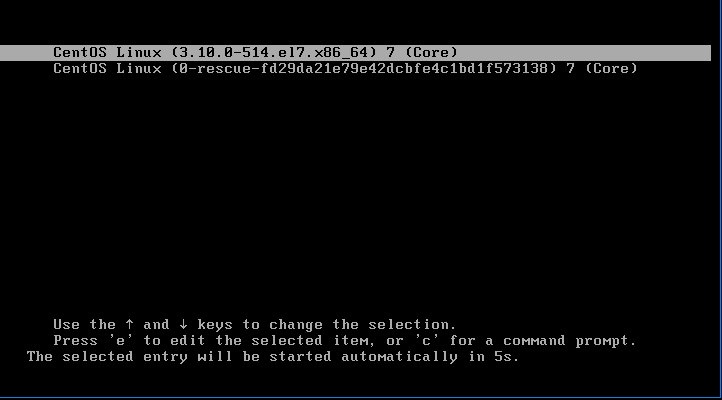


**Debugging**

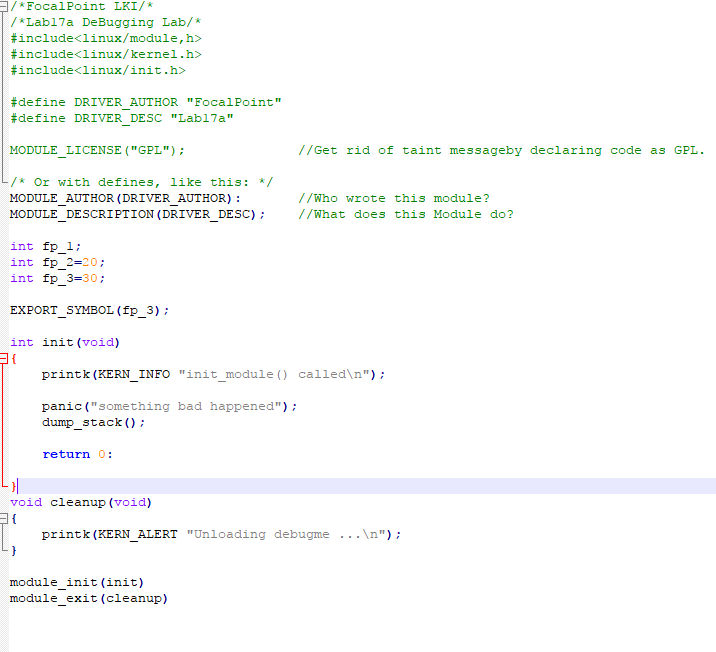
***Objective: In this lab you will debug a kernel module using gdb post mortem (not live since this would require kgdb and multiple machines).***

1. Reboot the computer and choose the debug kernel.

**File(s) for this lab:**



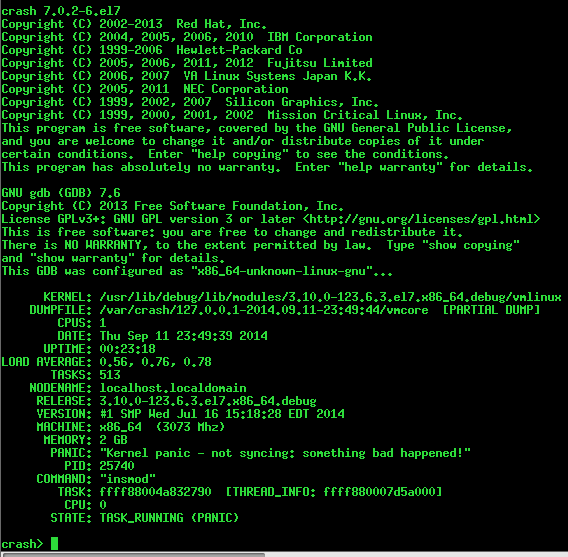
1. Open a terminal and verify that the kdump service is operational using the command sudo service kdump status.
2. Navigate to the “~/LKI/labs/Lab17” folder and view the debugme.c file.



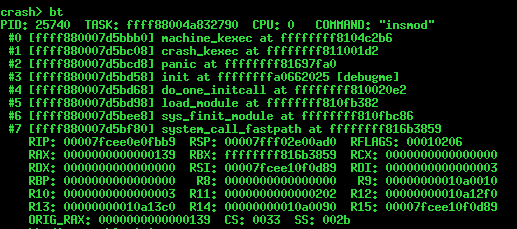
1. The program debugme simply causes a kernel panic during LKM initialization.
2. Load the module via “sudo /sbin/insmod debugme.ko”. Observe what happens. (Make sure you do not interfere with the system until it boots for the second time and choose the debug kernel again).
3. When the system resumes open a terminal and view the crash dump file using the “crash” command.



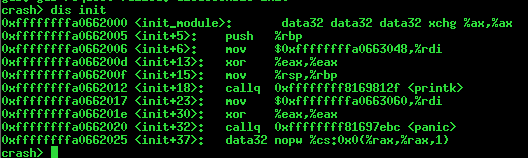
1. The crash program gives us a wealth of information about what happened.



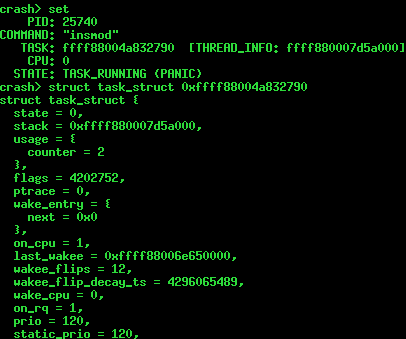
1. Where is the result of our panic() call?
2. Finally you can issue normal gdb commands to get to the bottom of what may have gone wrong. Example “bt” or backtrace.



1. If you are familiar with gdb, disassemble (dis) the init function and locate the call to panic.



1. Other helpful commands include: set (provides information on current process context, can be changed to other process, ex. *set 1*, change to init), dmesg (system log, page down to see any important information), ps (view process information, note insmod was last active process), task (dump active task\_struct, likely the culprit which brought down the system, this works because the crash utility operates in the current context), kmem –i (was crash memory consumption related?), files (provide information on active file handles), sig (signal information), net (network device info), vm (memory map status of active task), use struct (to dereference memory as structures, ex. struct task\_struct 0xdeadc0de)



1. Type quit to exit crash. Continue to Lab 17b.