Additional Kernel Debug Strategies

Advanced Embedded Software Development

with **Dan Walkes**

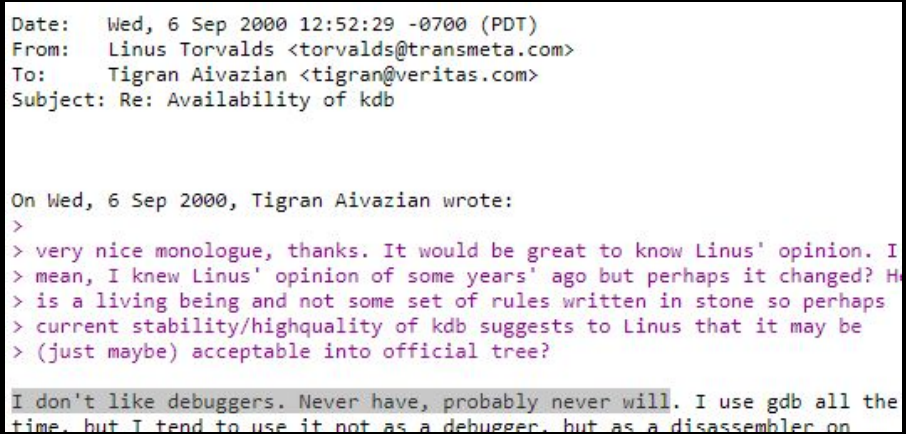
University of Colorado **Boulder**

Learning objectives:

Uinderstand kernel debug strategies for driver development

Using an interactive debugger

- Unfortunately not as easy as with user space programs
- Generally time consuming/should be avoided
  - Compile with CONFIG_DEBUG_INFO
- Linus is not a fan of interactive debuggers



```
Date:    Wed, 6 Sep 2000 12:52:29 -0700 (PDT)
From:    Linus Torvalds <torvalds@transmeta.com>
To:      Tigran Aivazian <tigran@veritas.com>
Subject: Re: Availability of kdb



On Wed, 6 Sep 2000, Tigran Aivazian wrote:
>
> very nice monologue, thanks. It would be great to know Linus' opinion. I
> mean, I knew Linus' opinion of some years' ago but perhaps it changed? H
> is a living being and not some set of rules written in stone so perhaps
> current stability/highquality of kdb suggests to Linus that it may be
> (just maybe) acceptable into official tree?

I don't like debuggers. Never have, probably never will. I use gdb all the
time, but I tend to use it not as a debugger, but as a disassembler on
```

Linux Device Drivers 3rd Edition Chapter 4
https://lwn.net/2000/0914/a/lt-debugger.php3

Using kdb and kgdb

- Both are now built into the kernel with CONFIG_KGDB_KDB and associated configuration (book is out of date on this).
- kdb is the shell you can interact with on the system
    - Typically uses the serial port for communication
    - Entered automatically after oops
- gdb on a development machine interacts with kdb/kgdb on the target, typically through a serial port
- Use SysRq->G to interrupt and enter kgdb mode

Other Debugging Tools

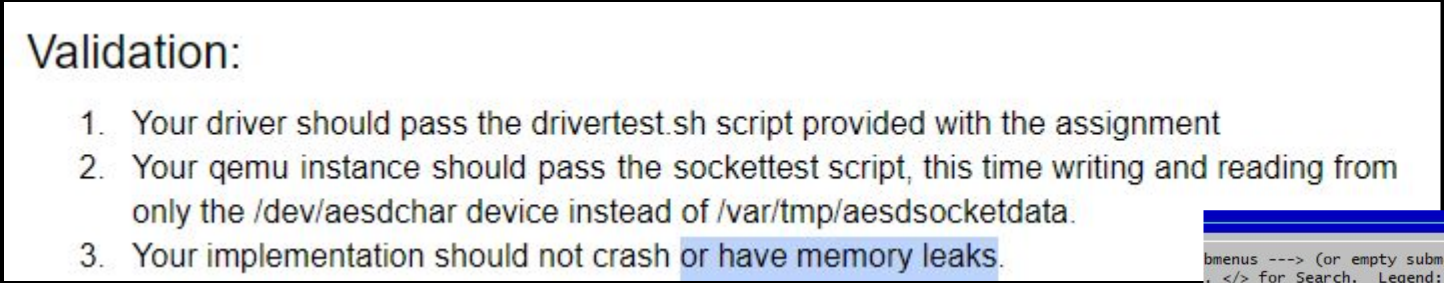
- User Mode Linux
  - No longer maintained/updated
- Linux Trace Toolkit
  - Traces events in the kernel for debugging and performance issues
- Dynamic Probing
  - replaced by SystemTap
  - Script framework to hook into Linux kernel

debugfs

- A way for developers to make information available to userspace.
  - /proc - fallen out of favor for new development - intended for process information
  - /sysfs - highly organized/restricted content
- “Not intended to serve as a stable ABI to user space”

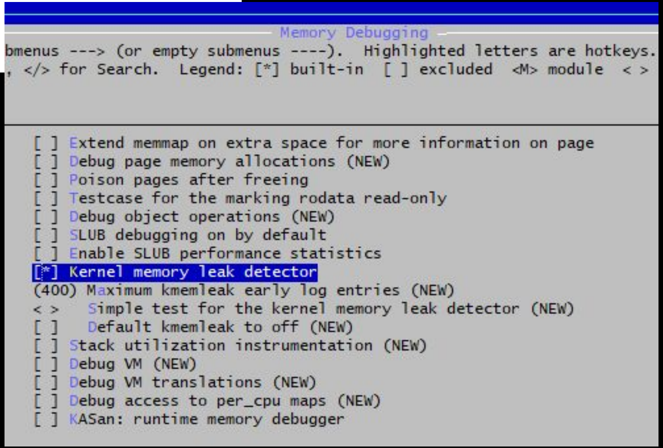
https://www.kernel.org/doc/Documentation/filesystems/debugfs.txt

Kernel Memory Leaks



Validation:

1. Your driver should pass the drivertest.sh script provided with the assignment
2. Your qemu instance should pass the sockettest script, this time writing and reading from only the /dev/aesdchar device instead of /var/tmp/aesdsocketdata.
3. Your implementation should not crash or have memory leaks.



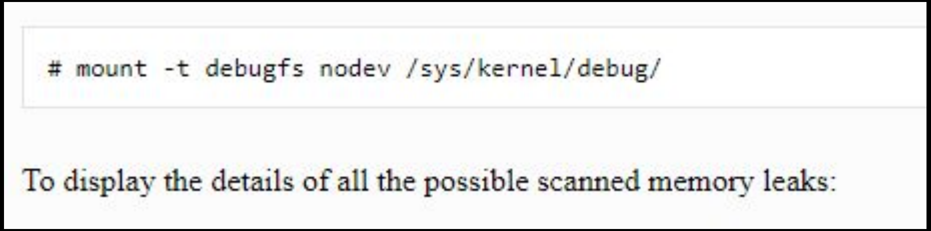
```
                               Memory Debugging
bmenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.
, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

   [ ] Extend memmap on extra space for more information on page
   [ ] Debug page memory allocations (NEW)
   [ ] Poison pages after freeing
   [ ] Testcase for the marking rodata read-only
   [ ] Debug object operations (NEW)
   [ ] SLUB debugging on by default
   [ ] Enable SLUB performance statistics
   [*] Kernel memory leak detector
   (400) Maximum kmemleak early log entries (NEW)
   < >   Simple test for the kernel memory leak detector (NEW)
   [ ]   Default kmemleak to off (NEW)
   [ ] Stack utilization instrumentation (NEW)
   [ ] Debug VM (NEW)
   [ ] Debug VM translations (NEW)
   [ ] Debug access to per_cpu maps (NEW)
   [ ] KASan: runtime memory debugger
```

- How do you test for memory leaks?
- You can use kmemleak for this
  - make linux-menuconfig from buildroot
    - kernel hacking->kernel debugging
    - kernel hacking->memory debugging->kernel memory leak detector

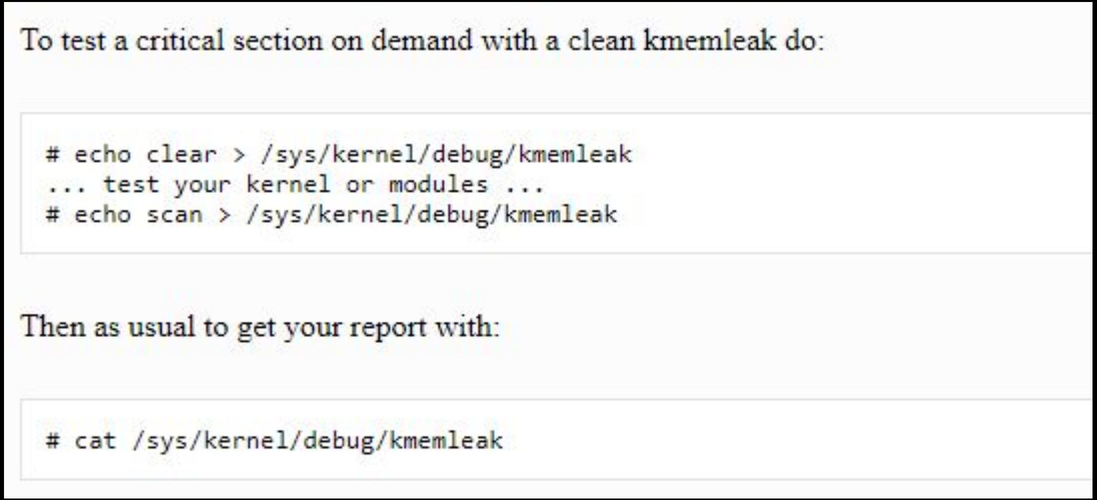
https://www.kernel.org/doc/html/latest/dev-tools/kmemleak.html

Kernel Memory Leaks



```
# mount -t debugfs nodev /sys/kernel/debug/
```

To display the details of all the possible scanned memory leaks:



To test a critical section on demand with a clean kmemleak do:

```
# echo clear > /sys/kernel/debug/kmemleak
... test your kernel or modules ...
# echo scan > /sys/kernel/debug/kmemleak
```

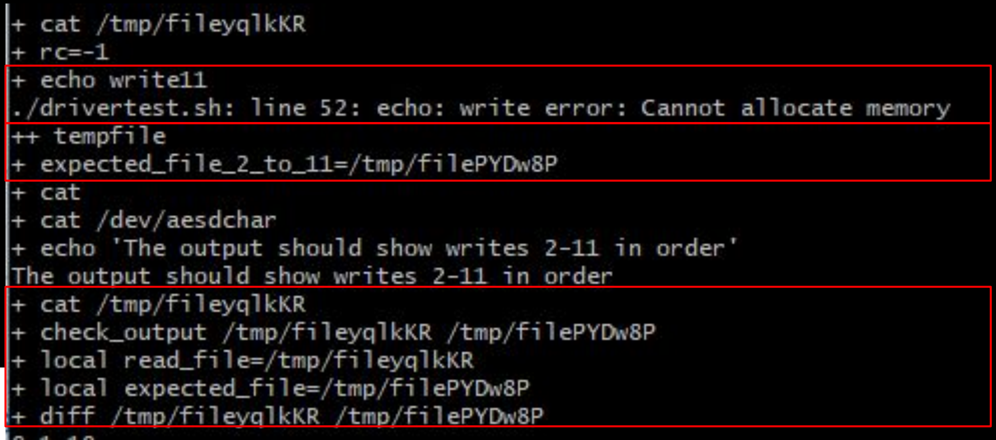
Then as usual to get your report with:

```
# cat /sys/kernel/debug/kmemleak
```

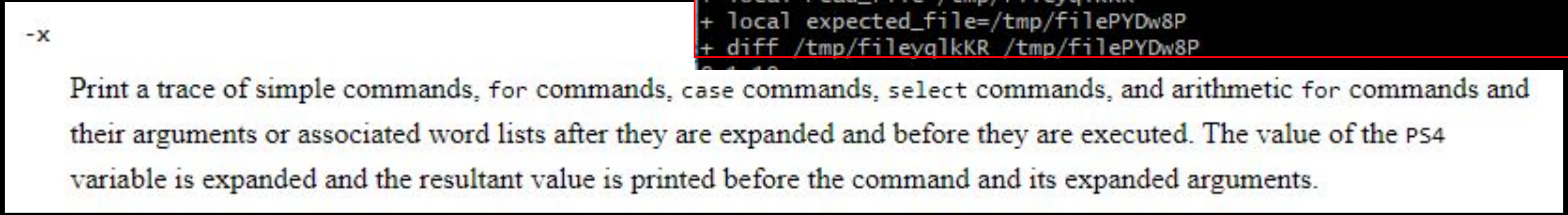
https://www.kernel.org/doc/html/latest/dev-tools/kmemleak.html

bash -x (or sh -x)

- example bash -x ./drivertest.sh > /tmp/bash_result.txt 2>& on unimplemented driver



```
+ cat /tmp/fileyqlkKR
+ rc=-1
+ echo write11
./drivertest.sh: line 52: echo: write error: Cannot allocate memory
++ tempfile
+ expected_file_2_to_11=/tmp/filePYDw8P
+ cat
+ cat /dev/aesdchar
+ echo 'The output should show writes 2-11 in order'
The output should show writes 2-11 in order
+ cat /tmp/fileyqlkKR
+ check_output /tmp/fileyqlkKR /tmp/filePYDw8P
+ local read_file=/tmp/fileyqlkKR
+ local expected_file=/tmp/filePYDw8P
+ diff /tmp/fileyqlkKR /tmp/filePYDw8P
```



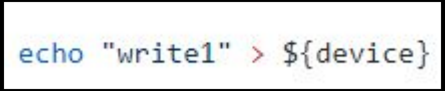
-x

Print a trace of simple commands, `for` commands, `case` commands, `select` commands, and arithmetic `for` commands and their arguments or associated word lists after they are expanded and before they are executed. The value of the `PS4` variable is expanded and the resultant value is printed before the command and its expanded arguments.

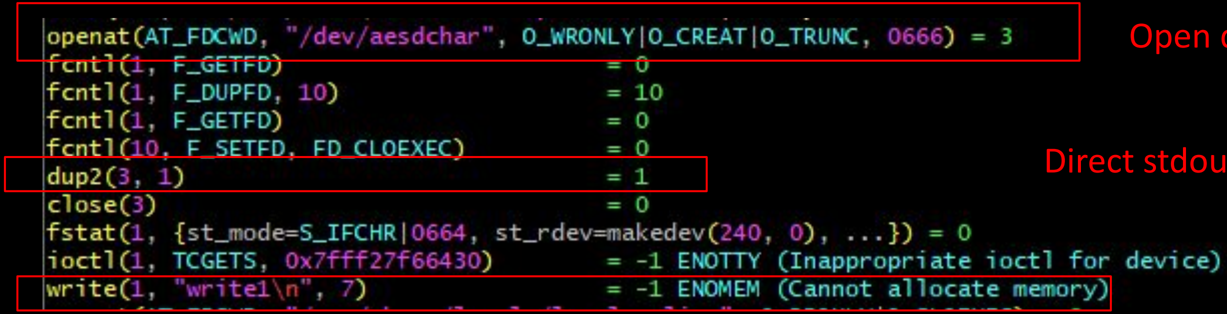
https://www.gnu.org/software/bash/manual/html_node/The-Set-Builtin.html

strace

- example strace -o /tmp/strace.txt ./drivertest.sh on unimplemented driver



```
echo "write1" > ${device}
```



```
openat(AT_FDCWD, "/dev/aesdchar", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
fcntl(1, F_GETFD)                        = 0
fcntl(1, F_DUPFD, 10)                    = 10
fcntl(1, F_GETFD)                        = 0
fcntl(10, F_SETFD, FD_CLOEXEC)           = 0
dup2(3, 1)                               = 1
close(3)                                 = 0
fstat(1, {st_mode=S_IFCHR|0664, st_rdev=makedev(240, 0), ...}) = 0
ioctl(1, TCGETS, 0x7fff27f66430)         = -1 ENOTTY (Inappropriate ioctl for device)
write(1, "write1\n", 7)                  = -1 ENOMEM (Cannot allocate memory)
```

Open device as fd 3

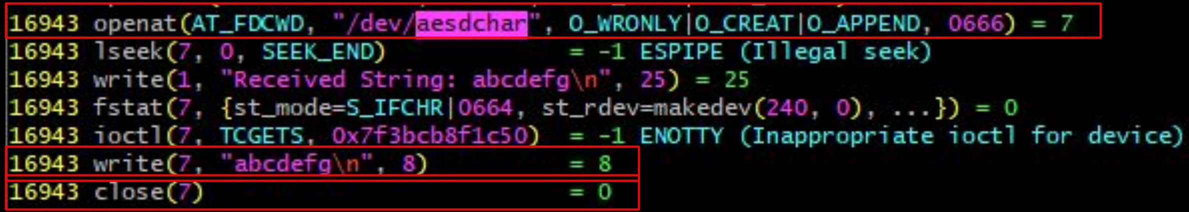
Direct stdout to fd 3 (our driver device endpoint)

Write to stdout (directed to our device driver)
Why did we get ENOMEM?

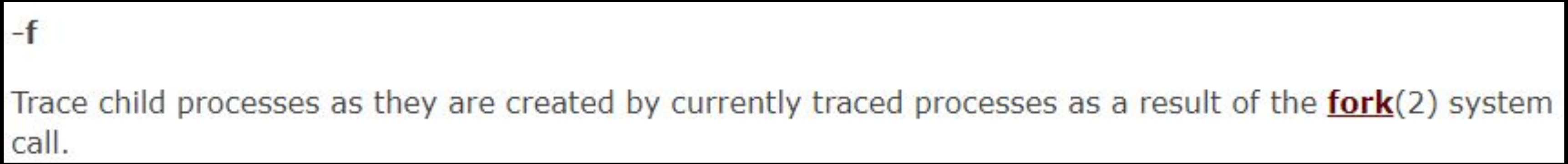
https://linux.die.net/man/1/strace
http://man7.org/linux/man-pages/man2/dup.2.html

strace

- example strace -o /tmp/strace-aesdsocket.txt -f ./aesdsocket on working driver



```
16943 openat(AT_FDCWD, "/dev/aesdchar", O_WRONLY|O_CREAT|O_APPEND, 0666) = 7
16943 lseek(7, 0, SEEK_END)              = -1 ESPIPE (Illegal seek)
16943 write(1, "Received String: abcdefg\n", 25) = 25
16943 fstat(7, {st_mode=S_IFCHR|0664, st_rdev=makedev(240, 0), ...}) = 0
16943 ioctl(7, TCGETS, 0x7f3bcb8f1c50)  = -1 ENOTTY (Inappropriate ioctl for device)
16943 write(7, "abcdefg\n", 8)          = 8
16943 close(7)                           = 0
```



-f

Trace child processes as they are created by currently traced processes as a result of the **fork**(2) system call.

https://linux.die.net/man/1/strace
https://stackoverflow.com/questions/14694582/stracing-to-attach-to-a-multi-threaded-process