Embedded Labworks

EMBEDDED LINUX CONFERENCE 2022

TOOLS AND TECHNIQUES TO DEBUG AN EMBEDDED LINUX SYSTEM

Sergio Prado, Embedded Labworks
https://www.linkedin.com/in/sprado

WHOAMI

- Designing and developing embedded software for 25+ years (Embedded Linux, Embedded Android, RTOS, etc).
- Consultant and trainer at Embedded Labworks for 10+ years.
  https://e-labworks.com/en
- Open source software contributor (Buildroot, Yocto Project, Linux kernel, etc).
- Blogger at EmbeddedBits.org.
  https://embeddedbits.org/

AGENDA

- Introduction to (software) debugging.
- Debugging tools and techniques (applied to embedded systems based on Linux).
  - Log/dump analysis.
  - Tracing.
  - Interactive debugging.
  - Debugging frameworks.
- Lot's of hands-on (hopefully)!

THE SIX STAGES OF DEBUGGING

- *Denial*: That can't happen.
- *Frustration*: That doesn't happen on my machine.
- *Disbelief*: That shouldn't happen.
- *Testing*: Why does that happen?
- *Confirmation*: Oh, I see.
- *Relief*: How did that ever work?

WHAT IS DEBUGGING?

- In the somewhat distant past, we started using "bug" in the engineering jargon to describe hardware and software errors.
- So debugging is the process of removing bugs from hardware and software designs.
- A wise old man once said: "In the software development process, we spend 50% debugging the software, and the other 50% bugging"!

DEBUGGING STEP-BY-STEP

- Debugging a software problem might involve the following steps:
  - Understanding the problem.
  - Reproducing the problem.
  - Identifying the root cause.
  - Fixing the problem.
  - Fixed? If so, celebrate! If not, go back to step 1.

THE 5 TYPES OF PROBLEMS

- We might classify software problems in 5 main categories:
  - Crash.
  - Lockup/Hang.
  - Logic/implementation.
  - Resource leakage.
  - (Lack of) performance.

TOOLS AND TECHNIQUES

- We might try to solve those problems using one or more of these 5 tools or techniques:
  - Our brain (aka knowledge).
  - Post mortem analysis (logging analysis, memory dump analysis, etc).
  - Tracing/profiling (specialized logging).
  - Interactive debugging (eg: GDB).
  - Debugging frameworks (eg: Valgrind).

POST MORTEM ANALYSIS

- Post mortem analysis can be done via information exported by the system, including logs and memory dumps.
  - *Logs*: any (text or binary) information related to the execution of the system, collected and stored by the operating system (application execution, kernel operation, system errors, etc).
  - *Memory dump*: When an application crashes, the kernel is able to generate a special file called *core*, that contains a snapshot of the memory of the offending process and can be used to debug and find the root cause of the crash.
- Post mortem analysis can be very helpful when analyzing crashes and logic problems.

EXAMPLE: KERNEL CRASH

```
[ 17.160336] Unable to handle kernel NULL pointer dereference at virtual address 00000000
[ 17.168531] pgd = 5df2196d
[ 17.171259] [00000000] *pgd=00000000
[ 17.174990] Internal error: Oops: 5 [#1] SMP ARM
[ 17.179622] Modules linked in:
[ 17.182686] CPU: 0 PID: 83 Comm: kworker/0:2 Not tainted 5.15.17-g85b8fc029a8d-dirty #2
[ 17.190700] Hardware name: Freescale i.MX6 Quad/DualLite (Device Tree)
[ 17.197232] Workqueue: usb_hub_wq hub_event
[ 17.201436] PC is at storage_probe+0x60/0x1a0
[ 17.205810] LR is at storage_probe+0x48/0x1a0
[ 17.210175] pc : [<c06a21cc>]    lr : [<c06a21b4>]    psr: 60000013
[ 17.216446] sp : c50239c0  ip : c50239c0  fp : c50239fc
[ 17.221674] r10: c53e2c00  r9 : c57c9a00  r8 : c0f60b4c
[ 17.226902] r7 : c53e2c80  r6 : c0a7d9fc  r5 : 00000001  r4 : c57c9a20
[ 17.233435] r3 : 00000000  r2 : 1ae1f000  r1 : c0a7d9fc  r0 : 00000000
[ 17.239968] Flags: nZCv  IRQs on  FIQs on  Mode SVC_32  ISA ARM  Segment none
...
[ 17.755646] Backtrace:
[ 17.758099] [<c06a216c>] (storage_probe) from [<c0682f2c>] (usb_probe_interface+0xe4/0x29c)
[ 17.766480] [<c0682e48>] (usb_probe_interface) from [<c05db4f8>] (really_probe.part.0+0xac/0x33c
[ 17.775384]  r10:c0f5ff48 r9:00000000 r8:00000008 r7:c57c9a20 r6:c0f60b4c r5:00000000
...
```

EXAMPLE: KERNEL CRASH (CONT.)

```
$ cd <linux_source_code>
$ ls
arch     Documentation  Kbuild       Makefile                 samples     tools
block    drivers        Kconfig      mm                       scripts     usr
certs    fs             kernel       modules.builtin          security    virt
COPYING  include        lib          modules.builtin.modinfo  sound       vmlinux
CREDITS  init           LICENSES     net                      System.map  vmlinux.o
crypto   ipc            MAINTAINERS  README                   tags        vmlinux.symvers

$ file vmlinux
vmlinux: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), statically linked, BuildID[sha1]
ca2de68ea4e39ca0f11e688a5e9ff0002a9b7733, with debug_info, not stripped
```

EXAMPLE: KERNEL CRASH (CONT.)

```
$ arm-linux-addr2line -f -p -e vmlinux 0xc06a21cc
storage_probe at /opt/labs/ex/linux/drivers/usb/storage/usb.c:1118

$ arm-linux-gdb vmlinux

(gdb) list *(storage_probe+0x60)
0xc06a21cc is in storage_probe (drivers/usb/storage/usb.c:1118).
1113             */
1114            if (usb_usual_ignore_device(intf))
1115                    return -ENXIO;
1116
1117            /* Print vendor and product name */
1118            v = (char *)unusual_dev->vendorName;
1119            p = (char *)unusual_dev->productName;
1120            if (v && p)
1121                    dev_dbg(&intf->dev, "vendor=%s product=%s\n", v, p);
```

EXAMPLE: USER SPACE CRASH

```
# fping -c 3 192.168.0.1
Segmentation fault

# ulimit -c unlimited

# fping -c 3 192.168.0.1
Segmentation fault (core dumped)

# ls -la core
-rw-------    1 root     root        380928 May 25  2022 core

# file core
core: ELF 32-bit LSB core file, ARM, version 1 (SYSV), SVR4-style, from 'fping -c 3 192.168.0.1',
real uid: 0, effective uid: 0, real gid: 0, effective gid: 0, execfn: '/usr/sbin/fping',
platform: 'v7l'

# cat /proc/sys/kernel/core_pattern
/root/core
```

EXAMPLE: USER SPACE CRASH (CONT.)

```
$ cd <fping_source_code>
$ ls
aclocal.m4    config.guess  config.status  contrib  INSTALL      Makefile.in  stamp-h1
CHANGELOG.md  config.h      config.sub     COPYING  install-sh   missing
ci            config.h.in   configure      depcomp  Makefile     README.md
compile       config.log    configure.ac   doc      Makefile.am  src

$ file src/fping
src/fping: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 5.15.0, with debug_info, not stripped

$ file core
core: ELF 32-bit LSB core file, ARM, version 1 (SYSV), SVR4-style, from 'fping -c 3 192.168.0.1',
real uid: 0, effective uid: 0, real gid: 0, effective gid: 0, execfn: '/usr/sbin/fping',
platform: 'v7l'
```

EXAMPLE: USER SPACE CRASH (CONT.)

```
$ arm-linux-gdb src/fping -c core
...
Core was generated by `fping -c 3 192.168.0.1'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  optparse_long (options=0xbe8e8914, longopts=0xbe8e89f8, longindex=0x0) at optparse.c:217
217         char *option = options->argv[options->optind];

(gdb) list
212     int
213     optparse_long(struct optparse *options,
214                   const struct optparse_long *longopts,
215                   int *longindex)
216     {
217         char *option = options->argv[options->optind];
218         if (option == 0) {
219             return -1;
220         } else if (is_dashdash(option)) {
221             options->optind++; /* consume "--" */
```

EXAMPLE: USER SPACE CRASH (CONT.)

```
(gdb) p options
$1 = (struct optparse *) 0xbe8e8914

(gdb) p options->argv
$3 = (char **) 0x0

(gdb) up
#1  0x0042278c in main (argc=4, argv=0xbe8e8e54) at fping.c:509
509         while ((c = optparse_long(&optparse_state, longopts, NULL)) != EOF) {

(gdb) p optparse_state
$4 = {
  argv = 0x0,
  permute = 1,
  optind = 1,
  optopt = 0,
  optarg = 0x0,
  errmsg = '\000' <repeats 63 times>,
  subopt = 0
}
```

TRACING

- Tracing is a specialized form of logging, where data about the state and execution of a program (or the kernel) is collected and stored for runtime (or later) analysis.
- It's implemented via static and dynamic tracepoints (probes) injected in the code to instrument the software at runtime.
- Tracing can be used for debugging purposes and also for latency and performance analysis (profiling).
- Tracing tools can be especially helpful with lockup issues and performance analysis.

EXAMPLE: KERNEL TRACING

```
# time echo 1 > /sys/class/leds/ipe:red:ld1/brightness
real    0m 4.04s
user    0m 0.00s
sys     0m 0.00s

# zcat /proc/config.gz | grep TRACER=y
CONFIG_NOP_TRACER=y
CONFIG_HAVE_FUNCTION_TRACER=y
CONFIG_HAVE_FUNCTION_GRAPH_TRACER=y
CONFIG_CONTEXT_SWITCH_TRACER=y
CONFIG_GENERIC_TRACER=y
CONFIG_FUNCTION_TRACER=y
CONFIG_FUNCTION_GRAPH_TRACER=y
CONFIG_STACK_TRACER=y
CONFIG_IRQSOFF_TRACER=y
CONFIG_SCHED_TRACER=y
CONFIG_HWLAT_TRACER=y
CONFIG_OSNOISE_TRACER=y
CONFIG_TIMERLAT_TRACER=y
```

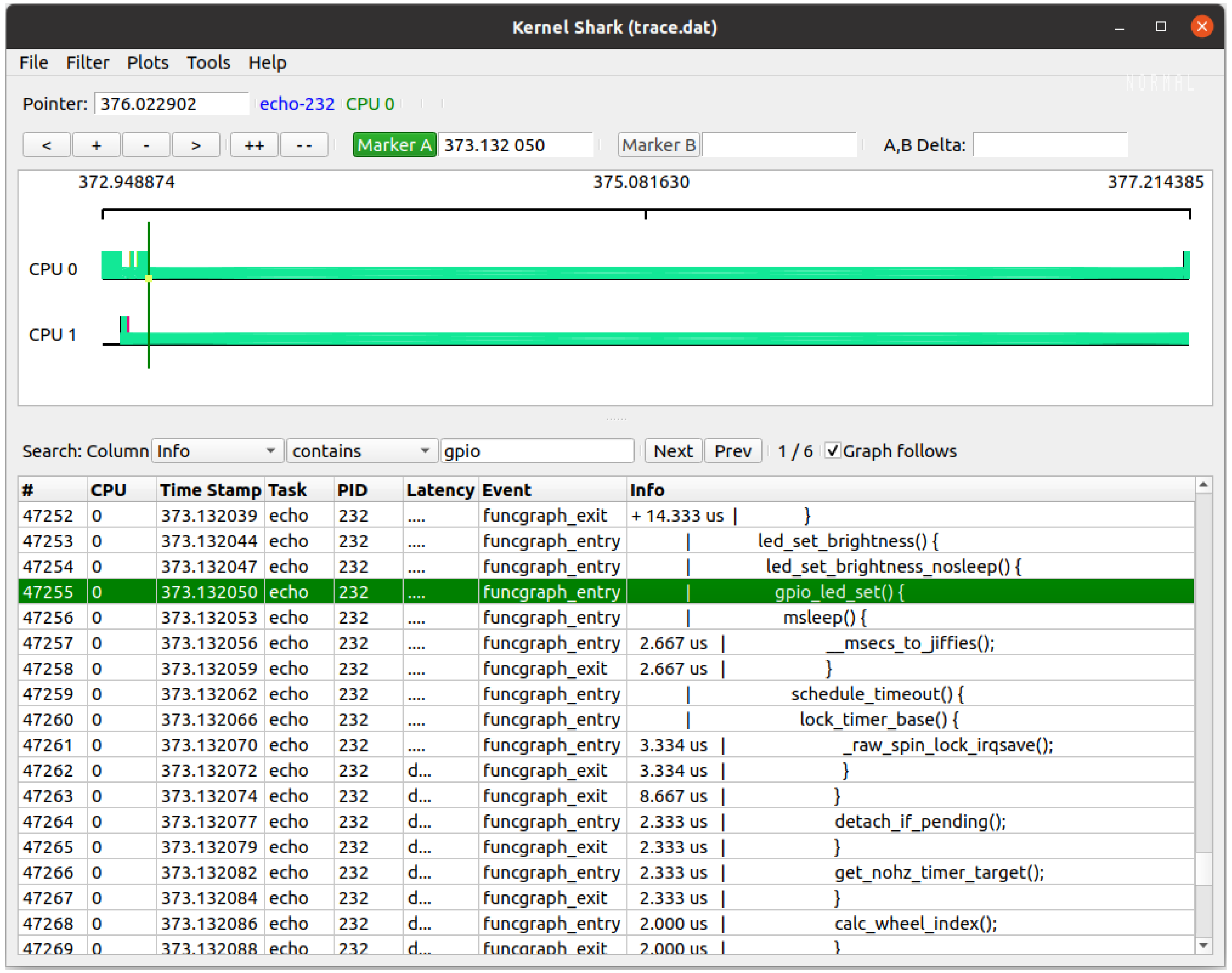
EXAMPLE: KERNEL TRACING (CONT.)

```
# mount -t tracefs tracefs /sys/kernel/tracing/

# trace-cmd record -p function_graph -F echo 1 > /sys/class/leds/ipe:red:ld1/brightness
  plugin 'function_graph'
CPU0 data recorded at offset=0x2f0000
    1421312 bytes in size
CPU1 data recorded at offset=0x44b000
    217088 bytes in size

# ls -l trace.dat
-rw-r--r--    1 root     root       4718592 May 26  2022 trace.dat
```

EXAMPLE: KERNEL TRACING (CONT.)

```
# trace-cmd report > trace.log

# cat trace.log
...
echo-232   [000]   373.132044: funcgraph_entry:               |  led_set_brightness() {
echo-232   [000]   373.132047: funcgraph_entry:               |    led_set_brightness_nosleep() {
echo-232   [000]   373.132050: funcgraph_entry:               |      gpio_led_set() {
echo-232   [000]   373.132053: funcgraph_entry:               |        msleep() {
echo-232   [000]   373.132056: funcgraph_entry:    2.667 us   |          __msecs_to_jiffies();
echo-232   [000]   373.132062: funcgraph_entry:               |          schedule_timeout() {
echo-232   [000]   373.132066: funcgraph_entry:               |            lock_timer_base() {
echo-232   [000]   373.132070: funcgraph_entry:    3.334 us   |              _raw_spin_lock_irqsa
echo-232   [000]   373.132074: funcgraph_exit:     8.667 us   |            }
echo-232   [000]   373.132077: funcgraph_entry:    2.333 us   |            detach_if_pending();
echo-232   [000]   373.132082: funcgraph_entry:    2.333 us   |            get_nohz_timer_target(
echo-232   [000]   373.132086: funcgraph_entry:    2.000 us   |            calc_wheel_index();
echo-232   [000]   373.132090: funcgraph_entry:    2.334 us   |            enqueue_timer();
...
echo-232   [000]   377.194984: funcgraph_entry:    2.666 us   |              _raw_spin_unlock_irq
echo-232   [000]   377.194990: funcgraph_exit:   + 23.000 us  |            }
echo-232   [000]   377.194993: funcgraph_exit:   $ 4062931 us |          }
echo-232   [000]   377.194996: funcgraph_exit:   $ 4062943 us |        }
```

EXAMPLE: KERNEL TRACING (CONT.)



Pointer: 376.022902 echo-232 CPU 0

Marker A 373.132 050 Marker B A,B Delta:

Search: Column Info contains gpio 1 / 6 ☑ Graph follows

```
#      CPU  Time Stamp  Task  PID  Latency  Event            Info
47252  0    373.132039  echo  232  ....     funcgraph_exit   + 14.333 us |          }
47253  0    373.132044  echo  232  ....     funcgraph_entry              |        led_set_brightness() {
47254  0    373.132047  echo  232  ....     funcgraph_entry              |         led_set_brightness_nosleep() {
47255  0    373.132050  echo  232  ....     funcgraph_entry              |          gpio_led_set() {
47256  0    373.132053  echo  232  ....     funcgraph_entry              |           msleep() {
47257  0    373.132056  echo  232  ....     funcgraph_entry    2.667 us  |             __msecs_to_jiffies();
47258  0    373.132059  echo  232  ....     funcgraph_exit     2.667 us  |             }
47259  0    373.132062  echo  232  ....     funcgraph_entry              |            schedule_timeout() {
47260  0    373.132066  echo  232  ....     funcgraph_entry              |             lock_timer_base() {
47261  0    373.132070  echo  232  ....     funcgraph_entry    3.334 us  |               _raw_spin_lock_irqsave();
47262  0    373.132072  echo  232  d...     funcgraph_exit     3.334 us  |               }
47263  0    373.132074  echo  232  d...     funcgraph_exit     8.667 us  |              }
47264  0    373.132077  echo  232  d...     funcgraph_entry    2.333 us  |              detach_if_pending();
47265  0    373.132079  echo  232  d...     funcgraph_exit     2.333 us  |              }
47266  0    373.132082  echo  232  d...     funcgraph_entry    2.333 us  |              get_nohz_timer_target();
47267  0    373.132084  echo  232  d...     funcgraph_exit     2.333 us  |              }
47268  0    373.132086  echo  232  d...     funcgraph_entry    2.000 us  |              calc_wheel_index();
47269  0    373.132088  echo  232  d...     funcgraph_exit     2.000 us  |              }
```

EXAMPLE: USER SPACE TRACING

```
# netcat -l -p 1234
Error: Couldn't setup listening socket (err=-3)

# strace netcat -l -p 1234
...
read(3, "# /etc/services:\n# $Id: services"..., 4096) = 4096
read(3, "tcp\t\t\t\t# UNIX Listserv\nulistserv"..., 4096) = 4096
read(3, "inding Protocol\necho\t\t4/ddp\t\t\t#"..., 4096) = 2681
read(3, "", 4096)                       = 0
close(3)                                = 0
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
setsockopt(3, SOL_SOCKET, SO_LINGER, {l_onoff=1, l_linger=0}, 8) = 0
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
bind(3, NULL, 16)                       = -1 EFAULT (Bad address)
close(3)                                = 0
write(2, "Error: Couldn't setup listening "..., 48Error: Couldn't setup listening socket (err=-3)
) = 48
exit_group(1)                           = ?
+++ exited with 1 +++
```

EXAMPLE: USER SPACE TRACING (CONT.)

```
# ethtool eth0
Settings for eth0:
<hanging>

# zcat /proc/config.gz | grep CONFIG_UPROBE
CONFIG_UPROBES=y
CONFIG_UPROBE_EVENTS=y

# file /usr/sbin/ethtool
/usr/sbin/ethtool: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically
linked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 5.15.0, with debug_info, not
stripped
```

EXAMPLE: USER SPACE TRACING (CONT.)

```
# for f in `perf probe -F -x /usr/sbin/ethtool`; \
    do perf probe -q -x /usr/sbin/ethtool $f; done

# perf probe -l | tee
  probe_ethtool:altera_tse_dump_regs (on altera_tse_dump_regs@build/ethtool-5.12/tse.c in /usr/sb
  probe_ethtool:amd8111e_dump_regs (on amd8111e_dump_regs@build/ethtool-5.12/amd8111e.c in /usr/s
  probe_ethtool:at76c50x_usb_dump_regs (on at76c50x_usb_dump_regs@ethtool-5.12/at76c50x-usb.c in
  ...

# perf record -e probe_ethtool:* -aR -- /usr/sbin/ethtool eth0
Couldn't synthesize bpf events.
Settings for eth0:
^C[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.084 MB perf.data (185 samples) ]

# ls -l perf.data
-rw-------    1 root     root        308153 May 26  2022 perf.data
```

EXAMPLE: USER SPACE TRACING (CONT.)

```
# perf script | tee
...
ethtool   812 [000]  4908.289466:  probe_ethtool:ethtool_link_mode_set_bit: (4a4bc0)
ethtool   812 [000]  4908.289493:  probe_ethtool:ethtool_link_mode_set_bit: (4a4bc0)
ethtool   812 [000]  4908.289520:  probe_ethtool:ethtool_link_mode_set_bit: (4a4bc0)
ethtool   812 [000]  4908.289546:  probe_ethtool:ethtool_link_mode_set_bit: (4a4bc0)
ethtool   812 [000]  4908.289573:  probe_ethtool:ethtool_link_mode_set_bit: (4a4bc0)
ethtool   812 [000]  4908.289600:  probe_ethtool:ethtool_link_mode_set_bit: (4a4bc0)
ethtool   812 [000]  4908.289626:  probe_ethtool:ethtool_link_mode_set_bit: (4a4bc0)
ethtool   812 [000]  4908.289660:                probe_ethtool:find_option: (4b5014)
ethtool   812 [000]  4908.289719:        probe_ethtool:netlink_run_handler: (4a4c3c)
ethtool   812 [000]  4908.289750:                 probe_ethtool:ioctl_init: (4b5e50)
ethtool   812 [000]  4908.289849:                    probe_ethtool:do_gset: (4ac63c)
ethtool   812 [000]  4908.290452:     probe_ethtool:do_ioctl_glinksettings: (4abd68)
ethtool   812 [000]  4908.290492:                 probe_ethtool:send_ioctl: (4b4cec)
ethtool   812 [000]  4908.290544:                 probe_ethtool:send_ioctl: (4b4cec)
ethtool   812 [000]  4908.290596:        probe_ethtool:dump_link_usettings: (4a6520)
ethtool   812 [000]  4908.290628:             probe_ethtool:dump_supported: (4a5f3c)
```

INTERACTIVE DEBUGGING

- An interactive debugging tool allows us to interact with the application at runtime.
- This kind of tool makes it possible to execute the code step-by-step, set breakpoints, display information (variables, stack, etc), list function call history (backtrace), etc.
- On Linux systems, the most used interactive debugging tool is GDB. https://www.sourceware.org/gdb/
- An interactive debug tool can especially help with crashes, lockups and logic problems.

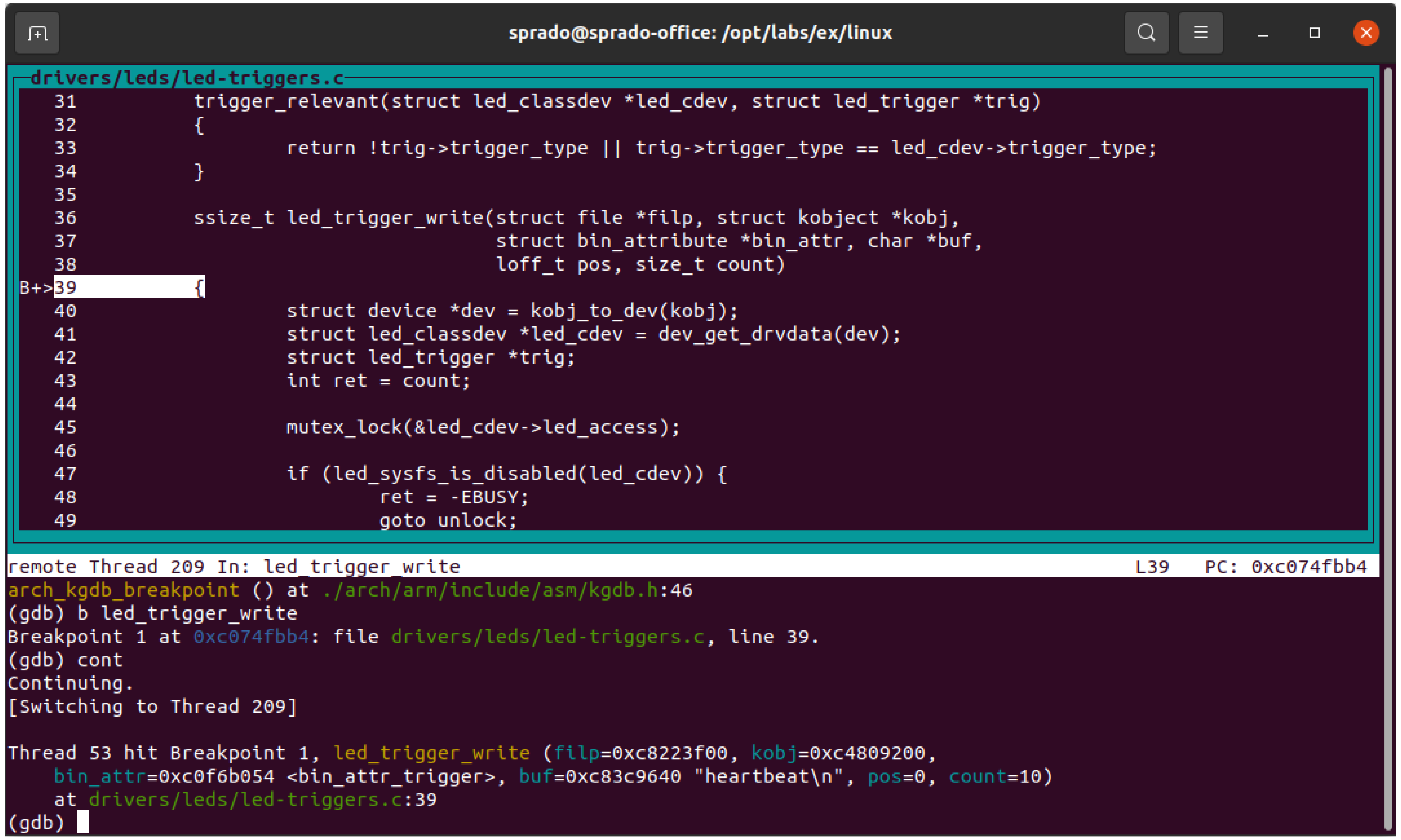
EXAMPLE: KERNEL DEBUGGING WITH GDB

```
# echo heartbeat > /sys/class/leds/ipe:red:ld1/trigger

# zcat /proc/config.gz | grep ^CONFIG_KGDB
CONFIG_KGDB=y
CONFIG_KGDB_HONOUR_BLOCKLIST=y
CONFIG_KGDB_SERIAL_CONSOLE=y

# echo ttymxc0 > /sys/module/kgdboc/parameters/kgdboc
[ 6794.040785] KGDB: Registered I/O driver kgdboc

# echo g > /proc/sysrq-trigger
[ 6797.741657] sysrq: DEBUG
[ 6797.744216] KGDB: Entering KGDB
```

EXAMPLE: KERNEL DEBUGGING WITH GDB (CONT.)

```
$ cd <linux_source_code>

$ file vmlinux
vmlinux: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), statically linked, BuildID[sha1]
ca2de68ea4e39ca0f11e688a5e9ff0002a9b7733, with debug_info, not stripped

$ arm-linux-gdb vmlinux -tui

(gdb) target remote localhost:5551
Remote debugging using localhost:5551
[Switching to Thread 4294967294]
arch_kgdb_breakpoint () at ./arch/arm/include/asm/kgdb.h:46

(gdb) b led_trigger_write
Breakpoint 1 at 0xc074fbb4: file drivers/leds/led-triggers.c, line 39.

(gdb) cont
```

EXAMPLE: KERNEL DEBUGGING WITH GDB (CONT.)



sprado@sprado-office: /opt/labs/ex/linux

```
drivers/leds/led-triggers.c
   31          trigger_relevant(struct led_classdev *led_cdev, struct led_trigger *trig)
   32          {
   33                  return !trig->trigger_type || trig->trigger_type == led_cdev->trigger_type;
   34          }
   35
   36          ssize_t led_trigger_write(struct file *filp, struct kobject *kobj,
   37                                    struct bin_attribute *bin_attr, char *buf,
   38                                    loff_t pos, size_t count)
B+>39          {
   40                  struct device *dev = kobj_to_dev(kobj);
   41                  struct led_classdev *led_cdev = dev_get_drvdata(dev);
   42                  struct led_trigger *trig;
   43                  int ret = count;
   44
   45                  mutex_lock(&led_cdev->led_access);
   46
   47                  if (led_sysfs_is_disabled(led_cdev)) {
   48                          ret = -EBUSY;
   49                          goto unlock;
remote Thread 209 In: led_trigger_write                                           L39   PC: 0xc074fbb4
arch_kgdb_breakpoint () at ./arch/arm/include/asm/kgdb.h:46
(gdb) b led_trigger_write
Breakpoint 1 at 0xc074fbb4: file drivers/leds/led-triggers.c, line 39.
(gdb) cont
Continuing.
[Switching to Thread 209]

Thread 53 hit Breakpoint 1, led_trigger_write (filp=0xc8223f00, kobj=0xc4809200,
    bin_attr=0xc0f6b054 <bin_attr_trigger>, buf=0xc83c9640 "heartbeat\n", pos=0, count=10)
    at drivers/leds/led-triggers.c:39
(gdb) 
```

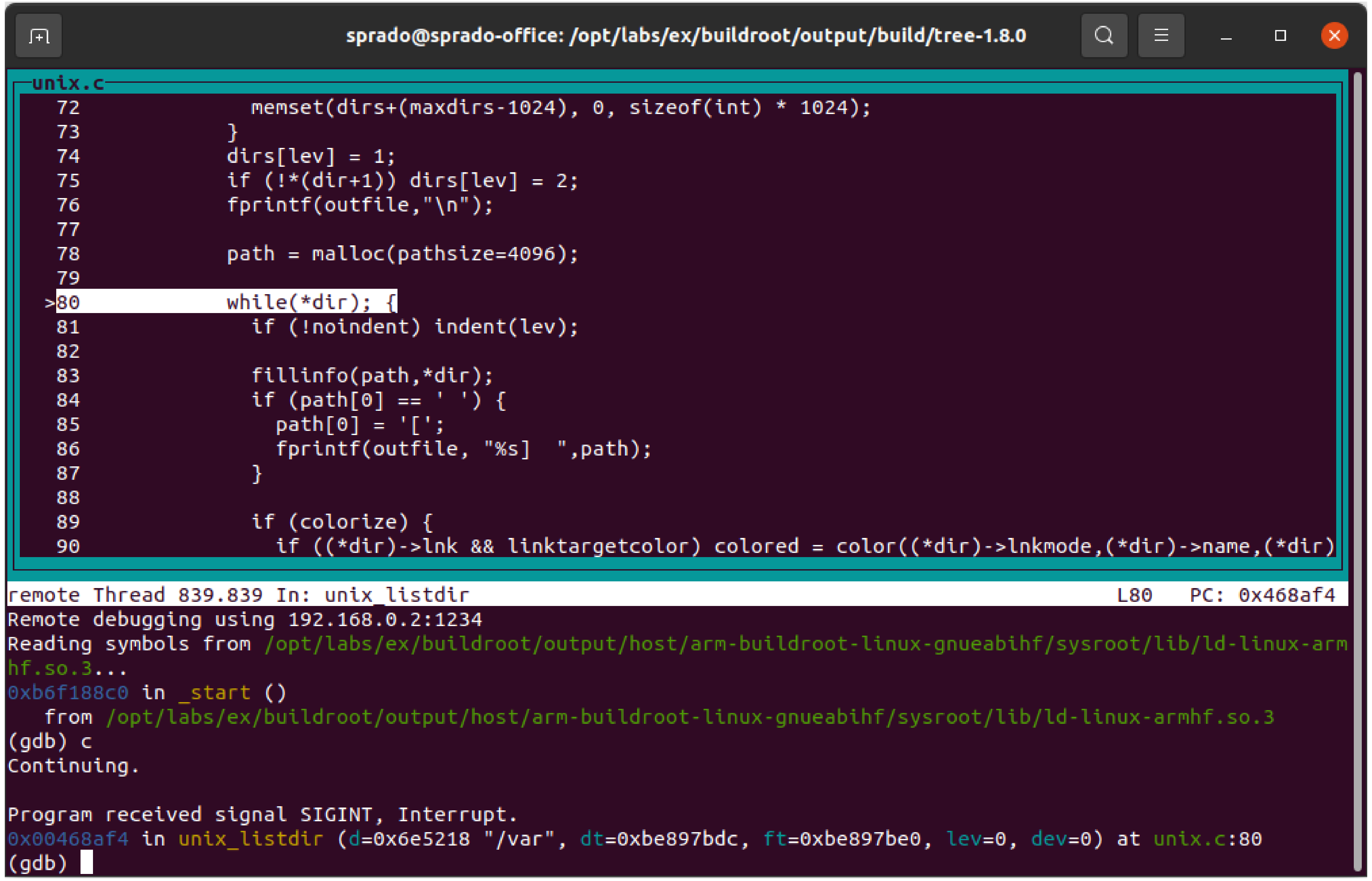
EXAMPLE: USER SPACE DEBUGGING WITH GDB

```
# tree /var
/var
<hanging>

# gdbserver :1234 tree /var
Process tree created; pid = 834
Listening on port 1234
```

EXAMPLE: USER SPACE DEBUGGING WITH GDB

```
$ cd <tree_source_code>
$ ls
CHANGES  doc      hash.c  html.o   json.o    README        tree    tree.o  xml.c
color.c  file.c   hash.o  INSTALL  LICENSE   strverscmp.c  tree.c  unix.c  xml.o
color.o  file.o   html.c  json.c   Makefile  TODO          tree.h  unix.o

$ file tree
tree: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 5.15.0, with debug_info, not stripped

$ arm-linux-gdb tree -tui

(gdb) target remote 192.168.0.2:1234
Remote debugging using 192.168.0.2:1234
Reading symbols from /opt/labs/ex/buildroot/output/host/arm-buildroot-linux-gnueabihf/sysroot/lib
0xb6f388c0 in _start () from /opt/labs/ex/buildroot/output/host/arm-buildroot-linux-gnueabihf/sys

(gdb) cont
Continuing
<CTRL-C>
```

EXAMPLE: USER SPACE DEBUGGING WITH GDB



sprado@sprado-office: /opt/labs/ex/buildroot/output/build/tree-1.8.0

```
unix.c
 72              memset(dirs+(maxdirs-1024), 0, sizeof(int) * 1024);
 73            }
 74            dirs[lev] = 1;
 75            if (!*(dir+1)) dirs[lev] = 2;
 76            fprintf(outfile,"\n");
 77
 78            path = malloc(pathsize=4096);
 79
>80            while(*dir); {
 81              if (!noindent) indent(lev);
 82
 83              fillinfo(path,*dir);
 84              if (path[0] == ' ') {
 85                path[0] = '[';
 86                fprintf(outfile, "%s]  ",path);
 87              }
 88
 89              if (colorize) {
 90                if ((*dir)->lnk && linktargetcolor) colored = color((*dir)->lnkmode,(*dir)->name,(*dir)
remote Thread 839.839 In: unix_listdir                                     L80   PC: 0x468af4
Remote debugging using 192.168.0.2:1234
Reading symbols from /opt/labs/ex/buildroot/output/host/arm-buildroot-linux-gnueabihf/sysroot/lib/ld-linux-arm
hf.so.3...
0xb6f188c0 in _start ()
   from /opt/labs/ex/buildroot/output/host/arm-buildroot-linux-gnueabihf/sysroot/lib/ld-linux-armhf.so.3
(gdb) c
Continuing.

Program received signal SIGINT, Interrupt.
0x00468af4 in unix_listdir (d=0x6e5218 "/var", dt=0xbe897bdc, ft=0xbe897be0, lev=0, dev=0) at unix.c:80
(gdb) 
```

DEBUGGING FRAMEWORKS

- There are a number of support tools and frameworks that can help with debugging Linux systems.
- A classic example is Valgrind, which provides a framework for creating memory debugging tools (memory leak, race condition, profiling, etc).
https://valgrind.org/
- Debugging frameworks can be very useful when analysing resource leaks and lockups.

EXAMPLE: DEBUGGING KERNEL HANGS

```
# cat /proc/uptime
<hanging>

# zcat /proc/config.gz | grep "CONFIG_SOFTLOCKUP_DETECTOR\|CONFIG_DETECT_HUNG_TASK"
CONFIG_SOFTLOCKUP_DETECTOR=y
CONFIG_DETECT_HUNG_TASK=y

# cat /proc/uptime
<wait for a few seconds>
```

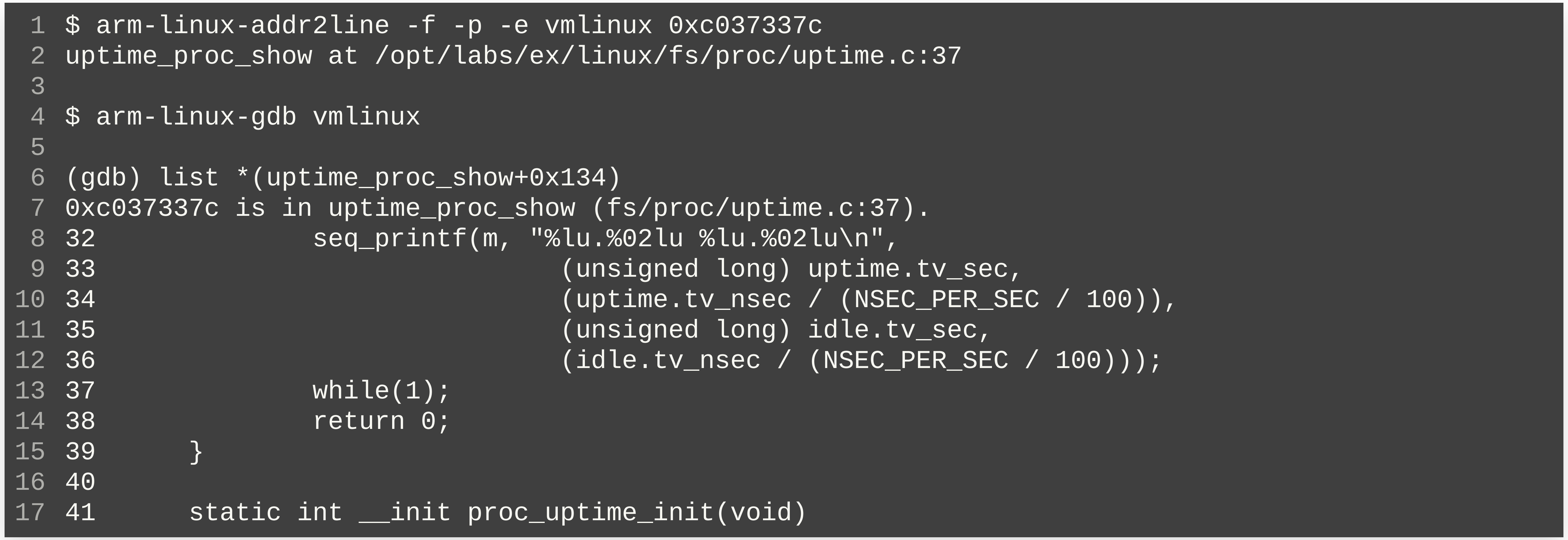
EXAMPLE: DEBUGGING KERNEL HANGS (CONT.)

```
[ 2604.004290] watchdog: BUG: soft lockup - CPU#1 stuck for 45s! [cat:209]
[ 2604.010927] Modules linked in:
[ 2604.013991] CPU: 1 PID: 209 Comm: cat Not tainted 5.15.17-g85b8fc029a8d-dirty #2
[ 2604.021399] Hardware name: Freescale i.MX6 Quad/DualLite (Device Tree)
[ 2604.027931] PC is at uptime_proc_show+0x134/0x15c
[ 2604.032651] LR is at vsnprintf+0x28c/0x42c
[ 2604.036760] pc : [<c037337c>]    lr : [<c0528660>]    psr: 600f0013
[ 2604.043031] sp : c5103c90  ip : c5103c08  fp : c5103d34
[ 2604.048260] r10: f87aa400  r9 : 89705f41  r8 : 36b4a597
[ 2604.053488] r7 : 0000027d  r6 : 4b14b59a  r5 : 000004a3  r4 : 00000000
[ 2604.060019] r3 : 82889af3  r2 : 82889af3  r1 : 00000010  r0 : 00000010
[ 2604.066552] Flags: nZCv  IRQs on  FIQs on  Mode SVC_32  ISA ARM  Segment none
[ 2604.073696] Control: 10c5387d  Table: 158ac04a  DAC: 00000051
[ 2604.079446] CPU: 1 PID: 209 Comm: cat Not tainted 5.15.17-g85b8fc029a8d-dirty #2
[ 2604.086851] Hardware name: Freescale i.MX6 Quad/DualLite (Device Tree)
[ 2604.093382] Backtrace:
...
[ 2604.285229] [<c0373248>] (uptime_proc_show) from [<c0305400>] (seq_read_iter+0x1bc/0x560)
[ 2604.293433]  r10:00400cc0 r9:00000001 r8:c5103db8 r7:c5910018 r6:00000000 r5:00000000
[ 2604.301268]  r4:c5910000
[ 2604.303803] [<c0305244>] (seq_read_iter) from [<c03671b4>] (proc_reg_read_iter+0x9c/0xe4)
...
```

EXAMPLE: DEBUGGING KERNEL HANGS (CONT.)

```
$ cd <linux_source_code>
$ ls
arch     Documentation  Kbuild       Makefile                 samples     tools
block    drivers        Kconfig      mm                       scripts     usr
certs    fs             kernel       modules.builtin          security    virt
COPYING  include        lib          modules.builtin.modinfo  sound       vmlinux
CREDITS  init           LICENSES     net                      System.map  vmlinux.o
crypto   ipc            MAINTAINERS  README                   tags        vmlinux.symvers

$ file vmlinux
vmlinux: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), statically linked, BuildID[sha1]
ca2de68ea4e39ca0f11e688a5e9ff0002a9b7733, with debug_info, not stripped
```

EXAMPLE: DEBUGGING KERNEL HANGS (CONT.)



```
$ arm-linux-addr2line -f -p -e vmlinux 0xc037337c
uptime_proc_show at /opt/labs/ex/linux/fs/proc/uptime.c:37

$ arm-linux-gdb vmlinux

(gdb) list *(uptime_proc_show+0x134)
0xc037337c is in uptime_proc_show (fs/proc/uptime.c:37).
32              seq_printf(m, "%lu.%02lu %lu.%02lu\n",
33                              (unsigned long) uptime.tv_sec,
34                              (uptime.tv_nsec / (NSEC_PER_SEC / 100)),
35                              (unsigned long) idle.tv_sec,
36                              (idle.tv_nsec / (NSEC_PER_SEC / 100)));
37              while(1);
38              return 0;
39      }
40
41      static int __init proc_uptime_init(void)
```

EXAMPLE: MEMORY LEAKS IN USER SPACE

```
# cpuload
Time  CPU  total  nice  user  system  irq  softirq  iowait  steal  guest
0     CPU  5.9    0.0   0.2   5.2     0.0  0.5      0.3     0.0    0.0
1     CPU  0.0    0.0   0.0   0.0     0.0  0.0      0.0     0.0    0.0
2     CPU  0.0    0.0   0.0   0.0     0.0  0.0      0.0     0.0    0.0
3     CPU  0.0    0.0   0.0   0.0     0.0  0.0      0.0     0.0    0.0
...
<memory is leaking>

# ls -l /usr/bin/valgrind
-rwxr-xr-x    1 root     root         25900 May 24  2022 /usr/bin/valgrind

# file /usr/bin/cpuload
/usr/bin/cpuload: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 5.15.0, with debug_info, not stripped
```

EXAMPLE: MEMORY LEAKS IN USER SPACE (CONT.)

```
# valgrind --leak-check=full /usr/bin/cpuload
==212== Memcheck, a memory error detector
==212== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==212== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==212== Command: /usr/bin/cpuload
==212==
Time  CPU  total  nice  user  system  irq  softirq  iowait  steal  guest
0     CPU  5.9    0.0   0.2   5.2     0.0  0.5      0.3     0.0    0.0
1     CPU  0.0    0.0   0.0   0.0     0.0  0.0      0.0     0.0    0.0
2     CPU  0.0    0.0   0.0   0.0     0.0  0.0      0.0     0.0    0.0
3     CPU  0.0    0.0   0.0   0.0     0.0  0.0      0.0     0.0    0.0
4     CPU  0.0    0.0   0.0   0.0     0.0  0.0      0.0     0.0    0.0
5     CPU  0.0    0.0   0.0   0.0     0.0  0.0      0.0     0.0    0.0
6     CPU  0.0    0.0   0.0   0.0     0.0  0.0      0.0     0.0    0.0
7     CPU  0.0    0.0   0.0   0.0     0.0  0.0      0.0     0.0    0.0
<CTRL-C>
```

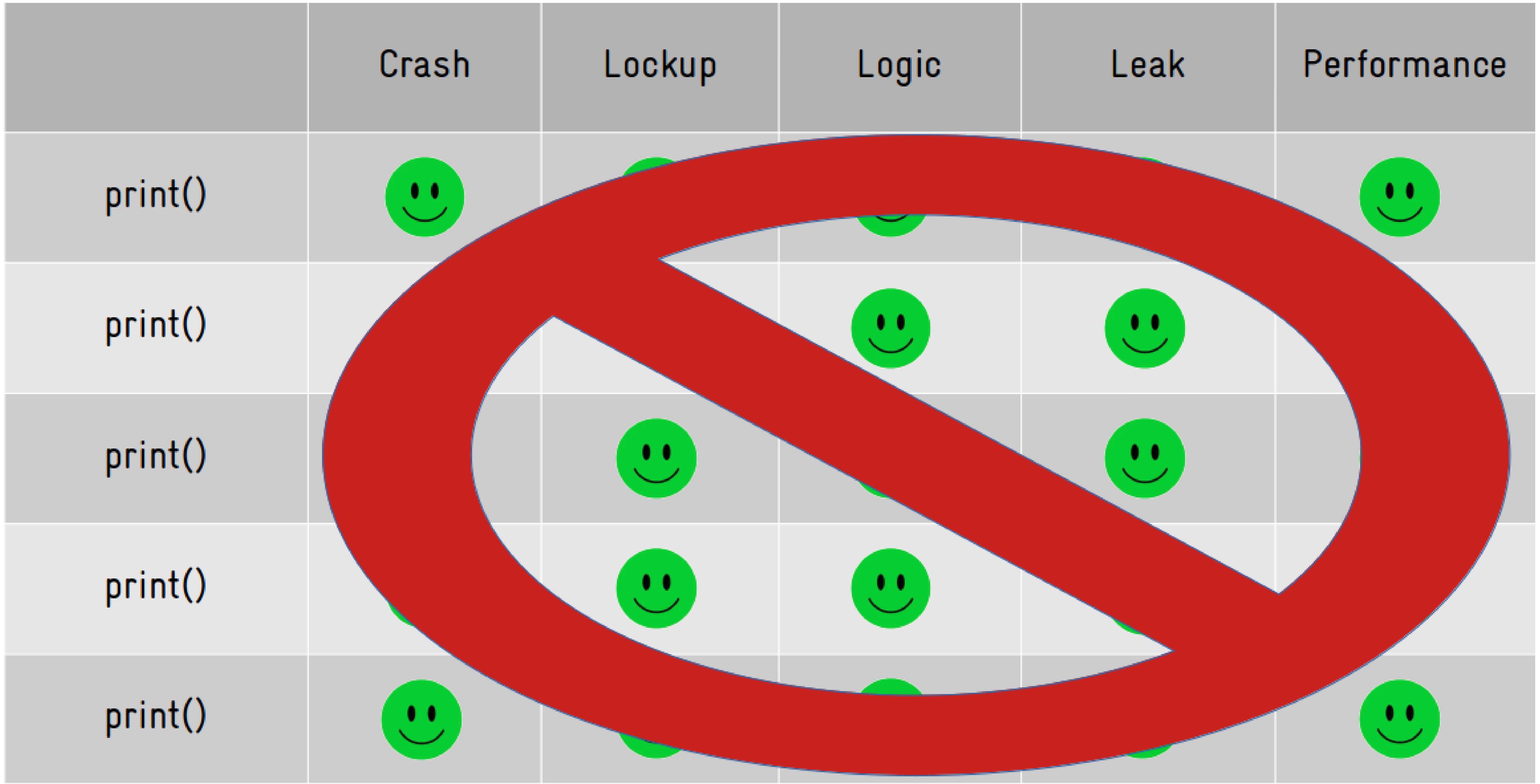
EXAMPLE: MEMORY LEAKS IN USER SPACE (CONT.)

```
==212== Process terminating with default action of signal 2 (SIGINT)
==212==    at 0x492491C: pause (in /lib/libc.so.6)
==212==    by 0x10ACFB: main (cpu_load.c:193)
==212==
==212== HEAP SUMMARY:
==212==     in use at exit: 52,964 bytes in 14 blocks
==212==   total heap usage: 34 allocs, 20 frees, 66,324 bytes allocated
==212==
==212== 36,864 bytes in 9 blocks are definitely lost in loss record 6 of 6
==212==    at 0x484EF68: malloc (vg_replace_malloc.c:381)
==212==    by 0x10A727: print_cpu_load (cpu_load.c:79)
==212==    by 0x10B177: do_stat (cpu_load.c:244)
==212==    by 0x48A888F: ??? (in /lib/libc.so.6)
==212==
==212== LEAK SUMMARY:
==212==    definitely lost: 36,864 bytes in 9 blocks
==212==    indirectly lost: 0 bytes in 0 blocks
==212==      possibly lost: 0 bytes in 0 blocks
==212==    still reachable: 16,100 bytes in 5 blocks
==212==         suppressed: 0 bytes in 0 blocks
==212== Reachable blocks (those to which a pointer was found) are not shown.
==212== To see them, rerun with: --leak-check=full --show-leak-kinds=all
```

PROBLEMS VS TECHNIQUES (1)

	Crash	Lockup	Logic	Leak	Performance
print()	🙂	🙂	🙂	🙂	🙂
print()	🙂	🙂	🙂	🙂	🙂
print()	🙂	🙂	🙂	🙂	🙂
print()	🙂	🙂	🙂	🙂	🙂
print()	🙂	🙂	🙂	🙂	🙂

PROBLEMS VS TECHNIQUES (2)



	Crash	Lockup	Logic	Leak	Performance
print()	☺	☺	☺	☺	☺
print()			☺	☺	
print()		☺	☺	☺	
print()	☺	☺	☺	☺	
print()	☺	☺	☺	☺	☺

PROBLEMS VS TECHNIQUES (3)

	Crash	Lockup	Logic	Leak	Performance
Knowledge	🙂	🙂	🙂	🙂	🙂
Post mortem	🙂	😐	🙂	😐	😐
Tracing	😐	🙂	😐	😐	🙂
Interactive debugging	🙂	🙂	🙂	😐	☹️
Debugging frameworks	☹️	🙂	☹️	🙂	😐

EMBEDDED LINUX CONFERENCE 2022

THANK YOU! QUESTIONS?

Sergio Prado, Embedded Labworks
sergio.prado@e-labworks.com
https://www.linkedin.com/in/sprado
https://twitter.com/sergioprado