Using GDB to Help You Read Linux Kernel Code without Running It

Synopsis

1. Build kernel with -g3
    a. This will include C macro information in the DWARF debug info.
2. Build GDB index to speed up loading
3. Learn some GDB features:
    a. Macro support
    b. Source listing & editing
4. Use GDB as a source browser
    a. Expand macros, jump to function definitions, disassemble functions
    b. Explore types
    c. Invoke make and external commands directly in GDB

Build Kernel with -g3

Hack the toplevel Makefile:

```
diff --git a/Makefile b/Makefile
index 320663d..aa2cdf3 100644
--- a/Makefile
+++ b/Makefile
@@ -593,8 +593,8 @@ endif
 endif

 ifdef CONFIG_DEBUG_INFO
-KBUILD_CFLAGS  += -g
-KBUILD_AFLAGS  += -gdwarf-2
+KBUILD_CFLAGS  += -g3
+#KBUILD_AFLAGS += -gdwarf-2
 endif

 ifdef CONFIG_DEBUG_INFO_REDUCED
```

Explanation

“-gdwarf-2” limits the information generated by GCC. Not needed when only consuming debug info with GDB.

Make sure you have **`CONFIG_DEBUG_INFO=y`** in your kernel config, then build the kernel until you get its ELF image, “`vmlinux`”.

Build GDB Index to Speed Up Loading

```
$ time gdb -q ./vmlinux -ex 'quit'
Reading symbols from ./vmlinux...done.

real    0m0.587s
user    0m0.556s
sys     0m0.032s

$ gdb -q ./vmlinux
Reading symbols from ./vmlinux...done.
(gdb) !mkdir debuginfo
(gdb) save gdb-index debuginfo
(gdb) !ls debuginfo/
vmlinux.gdb-index
(gdb) quit
$ objcopy --add-section .gdb_index=debuginfo/vmlinux.gdb-index
--set-section-flags .gdb_index=readonly ./vmlinux ./vmlinux
$ time gdb -q ./vmlinux -ex 'quit'
Reading symbols from ./vmlinux...done.

real    0m0.143s
user    0m0.119s
sys     0m0.027s
```

Note how the load time was reduced from 0.58s to 0.14s.
Reference: [Index Files Speed Up gdb].

Learn GDB Features for Macros and Source Code Listing

Read parts of the GDB Manual

Read:

1. GDB Manual: C Preprocessor Macros
2. GDB Manual: Printing Source Lines

Use GDB as a Source Browser

Let's use the implementation of the open(2) syscall, **`sys_open`** as an example.

$ **`gdb -q ./vmlinux`**
(gdb) **`list sys_open`**
file: "fs/open.c", line number: 999
file: "fs/open.c", line number: 1005
(gdb) **`list fs/open.c:999`**
(gdb) **`macro expand current`**
expands to: get_current()
(gdb) **`edit get_current`**
#scottt: vim opens at correct location
(gdb) **`make cscope`**
  GEN   cscope
(gdb) **`disassemble get_current`**
No symbol "get_current" in current context.

#scottt: this implies get_current is inlined. Let's find a call site of get_current

(gdb) **`edit fs/open.c:999`**

#scottt: reading the code starting from **sys_open**(), we see that **get_current**() is called in **alloc_fd**()

Finding “current” in the Disassembly

(gdb) **`disassemble /s alloc_fd`**
Dump of assembler code for function alloc_fd:
fs/file.c:
431 {
0xffffffff81148240 <+0>: push %rbp

/home/scottt/work/aple/linux-stable/arch/x86/include/asm/current.h:
14 return percpu_read_stable(current_task);
0xffffffff81148241 <+1>: mov %gs:0xb7c0,%rax

#scottt: per-cpu variables on x86-64 linux is referenced through the GS segment register

fs/file.c:
431 {
0xffffffff8114824a <+10>: mov %rsp,%rbp
0xffffffff8114824d <+13>: push %r15
0xffffffff8114824f <+15>: push %r14
0xffffffff81148251 <+17>: push %r13
0xffffffff81148253 <+19>: push %r12
0xffffffff81148255 <+21>: mov %edi,%r12d
0xffffffff81148258 <+24>: push %rbx
0xffffffff81148259 <+25>: sub $0x18,%rsp

432 struct files_struct *files = current->files;
0xffffffff8114825d <+29>: mov 0x508(%rax),%r13

431 {
0xffffffff81148264 <+36>: mov %esi,-0x3c(%rbp)

include/linux/spinlock.h:
285 raw_spin_lock(&lock->rlock);
0xffffffff81148267 <+39>: lea 0x80(%r13),%rax
0xffffffff8114826e <+46>: mov %rax,%rdi
0xffffffff81148271 <+49>: mov %rax,-0x38(%rbp)
0xffffffff81148275 <+53>: callq 0xffffffff814b27a0 <_raw_spin_lock>
0xffffffff8114827a <+58>: jmp 0xffffffff81148297 <alloc_fd+87>
0xffffffff8114827c <+60>: nopl 0x0(%rax)

fs/file.c:

```
447             error = expand_files(files, fd);
  0xffffffff81148280 <+64>:    mov    %ebx,%esi
  0xffffffff81148282 <+66>:    mov    %r13,%rdi
  0xffffffff81148285 <+69>:    callq  0xffffffff81147da0 <expand_files>

448             if (error < 0)
  0xffffffff8114828a <+74>:    test   %eax,%eax

447             error = expand_files(files, fd);
  0xffffffff8114828c <+76>:    mov    %eax,%r15d

448             if (error < 0)
  0xffffffff8114828f <+79>:    js     0xffffffff81148324 <alloc_fd+228>

449                     goto out;
450
451             /*
452              * If we needed to expand the fs array we
453              * might have blocked - try again.
454              */
455             if (error)
  0xffffffff81148295 <+85>:    je     0xffffffff811482c8 <alloc_fd+136>

439             fdt = files_fdtable(files);
  0xffffffff81148297 <+87>:    mov    0x8(%r13),%r14
  0xffffffff8114829b <+91>:    cmp    %r12d,0x84(%r13)
  0xffffffff811482a2 <+98>:    mov    %r12d,%ebx
  0xffffffff811482a5 <+101>:   cmovae 0x84(%r13),%ebx

440             fd = start;
441             if (fd < files->next_fd)
442                     fd = files->next_fd;
443
444             if (fd < fdt->max_fds)
  0xffffffff811482ad <+109>:   mov    (%r14),%esi
  0xffffffff811482b0 <+112>:   cmp    %ebx,%esi
  0xffffffff811482b2 <+114>:   jbe    0xffffffff81148280 <alloc_fd+64>

445                     fd = find_next_zero_bit(fdt->open_fds, fdt->max_fds, fd);
  0xffffffff811482b4 <+116>:   mov    0x18(%r14),%rdi
  0xffffffff811482b8 <+120>:   mov    %ebx,%edx
  0xffffffff811482ba <+122>:   callq  0xffffffff812446f0 <find_next_zero_bit>
  0xffffffff811482bf <+127>:   mov    %eax,%ebx
```

```
  0xffffffff811482c1 <+129>:   jmp    0xffffffff81148280 <alloc_fd+64>
  0xffffffff811482c3 <+131>:   nopl   0x0(%rax,%rax,1)

456                     goto repeat;
457
458             if (start <= files->next_fd)
  0xffffffff811482c8 <+136>:   cmp    0x84(%r13),%r12d
  0xffffffff811482cf <+143>:   ja     0xffffffff811482db <alloc_fd+155>

459                     files->next_fd = fd + 1;
  0xffffffff811482d1 <+145>:   lea    0x1(%rbx),%eax
  0xffffffff811482d4 <+148>:   mov    %eax,0x84(%r13)
  0xffffffff811482db <+155>:   mov    0x18(%r14),%rax

/home/scottt/work/aple/linux-stable/arch/x86/include/asm/bitops.h:
84              asm volatile("bts %1,%0" : ADDR : "Ir" (nr) : "memory");
  0xffffffff811482df <+159>:   bts    %ebx,(%rax)

fs/file.c:
462             if (flags & O_CLOEXEC)
  0xffffffff811482e2 <+162>:   testl  $0x80000,-0x3c(%rbp)
  0xffffffff811482e9 <+169>:   mov    0x10(%r14),%rax
  0xffffffff811482ed <+173>:   je     0xffffffff81148348 <alloc_fd+264>

/home/scottt/work/aple/linux-stable/arch/x86/include/asm/bitops.h:
84              asm volatile("bts %1,%0" : ADDR : "Ir" (nr) : "memory");
  0xffffffff811482ef <+175>:   bts    %ebx,(%rax)

fs/file.c:
469             if (rcu_dereference_raw(fdt->fd[fd]) != NULL) {
  0xffffffff811482f2 <+178>:   mov    0x8(%r14),%rax
  0xffffffff811482f6 <+182>:   mov    %ebx,%r12d

447             error = expand_files(files, fd);
  0xffffffff811482f9 <+185>:   mov    %ebx,%r15d

463                     __set_close_on_exec(fd, fdt);
464             else
465                     __clear_close_on_exec(fd, fdt);
466             error = fd;
467     #if 1
468             /* Sanity check */
469             if (rcu_dereference_raw(fdt->fd[fd]) != NULL) {
```

```
   0xffffffff811482fc <+188>:    lea    (%rax,%r12,8),%rax
   0xffffffff81148300 <+192>:    mov    (%rax),%rax
   0xffffffff81148303 <+195>:    test   %rax,%rax
   0xffffffff81148306 <+198>:    je     0xffffffff81148324 <alloc_fd+228>

470                     printk(KERN_WARNING "alloc_fd: slot %d not NULL!\n", fd);
   0xffffffff81148308 <+200>:    mov    %ebx,%esi
   0xffffffff8114830a <+202>:    mov    $0xffffffff816f1a00,%rdi
   0xffffffff81148311 <+209>:    xor    %eax,%eax
   0xffffffff81148313 <+211>:    callq  0xffffffff8103b890 <printk>

471                     rcu_assign_pointer(fdt->fd[fd], NULL);
   0xffffffff81148318 <+216>:    mov    0x8(%r14),%rax
   0xffffffff8114831c <+220>:    movq   $0x0,(%rax,%r12,8)

/home/scottt/work/aple/linux-stable/arch/x86/include/asm/paravirt.h:
783             PVOP_VCALL1(pv_lock_ops.spin_unlock, lock);
   0xffffffff81148324 <+228>:    mov    -0x38(%rbp),%rdi
   0xffffffff81148328 <+232>:    callq  *0xffffffff81819bc8

fs/file.c:
478     }
   0xffffffff8114832f <+239>:    add    $0x18,%rsp
   0xffffffff81148333 <+243>:    mov    %r15d,%eax
   0xffffffff81148336 <+246>:    pop    %rbx
   0xffffffff81148337 <+247>:    pop    %r12
   0xffffffff81148339 <+249>:    pop    %r13
   0xffffffff8114833b <+251>:    pop    %r14
   0xffffffff8114833d <+253>:    pop    %r15
   0xffffffff8114833f <+255>:    pop    %rbp
   0xffffffff81148340 <+256>:    retq
   0xffffffff81148341 <+257>:    nopl   0x0(%rax)

/home/scottt/work/aple/linux-stable/arch/x86/include/asm/bitops.h:
127             asm volatile("btr %1,%0" : ADDR : "Ir" (nr));
   0xffffffff81148348 <+264>:    btr    %ebx,(%rax)
   0xffffffff8114834b <+267>:    jmp    0xffffffff811482f2 <alloc_fd+178>
End of assembler dump.

(gdb) disassemble alloc_fd
Dump of assembler code for function alloc_fd:
   0xffffffff81148240 <+0>:      push   %rbp
   0xffffffff81148241 <+1>:      mov    %gs:0xb7c0,%rax # current in RAX
```

```
0xffffffff8114824a <+10>:    mov    %rsp,%rbp
0xffffffff8114824d <+13>:    push   %r15
0xffffffff8114824f <+15>:    push   %r14
0xffffffff81148251 <+17>:    push   %r13
0xffffffff81148253 <+19>:    push   %r12
0xffffffff81148255 <+21>:    mov    %edi,%r12d
0xffffffff81148258 <+24>:    push   %rbx
0xffffffff81148259 <+25>:    sub    $0x18,%rsp
0xffffffff8114825d <+29>:    mov    0x508(%rax),%r13 # current->files in R13
0xffffffff81148264 <+36>:    mov    %esi,-0x3c(%rbp)
0xffffffff81148267 <+39>:    lea    0x80(%r13),%rax
0xffffffff8114826e <+46>:    mov    %rax,%rdi
0xffffffff81148271 <+49>:    mov    %rax,-0x38(%rbp)
0xffffffff81148275 <+53>:    callq  0xffffffff814b27a0 <_raw_spin_lock>
0xffffffff8114827a <+58>:    jmp    0xffffffff81148297 <alloc_fd+87>
0xffffffff8114827c <+60>:    nopl   0x0(%rax)
0xffffffff81148280 <+64>:    mov    %ebx,%esi
0xffffffff81148282 <+66>:    mov    %r13,%rdi
0xffffffff81148285 <+69>:    callq  0xffffffff81147da0 <expand_files>
0xffffffff8114828a <+74>:    test   %eax,%eax
0xffffffff8114828c <+76>:    mov    %eax,%r15d
0xffffffff8114828f <+79>:    js     0xffffffff81148324 <alloc_fd+228>
0xffffffff81148295 <+85>:    je     0xffffffff811482c8 <alloc_fd+136>
0xffffffff81148297 <+87>:    mov    0x8(%r13),%r14
0xffffffff8114829b <+91>:    cmp    %r12d,0x84(%r13)
0xffffffff811482a2 <+98>:    mov    %r12d,%ebx
0xffffffff811482a5 <+101>:   cmovae 0x84(%r13),%ebx
0xffffffff811482ad <+109>:   mov    (%r14),%esi
0xffffffff811482b0 <+112>:   cmp    %ebx,%esi
0xffffffff811482b2 <+114>:   jbe    0xffffffff81148280 <alloc_fd+64>
0xffffffff811482b4 <+116>:   mov    0x18(%r14),%rdi
0xffffffff811482b8 <+120>:   mov    %ebx,%edx
0xffffffff811482ba <+122>:   callq  0xffffffff812446f0 <find_next_zero_bit>
0xffffffff811482bf <+127>:   mov    %eax,%ebx
0xffffffff811482c1 <+129>:   jmp    0xffffffff81148280 <alloc_fd+64>
0xffffffff811482c3 <+131>:   nopl   0x0(%rax,%rax,1)
0xffffffff811482c8 <+136>:   cmp    0x84(%r13),%r12d
0xffffffff811482cf <+143>:   ja     0xffffffff811482db <alloc_fd+155>
0xffffffff811482d1 <+145>:   lea    0x1(%rbx),%eax
0xffffffff811482d4 <+148>:   mov    %eax,0x84(%r13)
0xffffffff811482db <+155>:   mov    0x18(%r14),%rax
0xffffffff811482df <+159>:   bts    %ebx,(%rax)
0xffffffff811482e2 <+162>:   testl  $0x80000,-0x3c(%rbp)
```

```
   0xffffffff811482e9 <+169>:   mov    0x10(%r14),%rax
   0xffffffff811482ed <+173>:   je     0xffffffff81148348 <alloc_fd+264>
   0xffffffff811482ef <+175>:   bts    %ebx,(%rax)
   0xffffffff811482f2 <+178>:   mov    0x8(%r14),%rax
   0xffffffff811482f6 <+182>:   mov    %ebx,%r12d
   0xffffffff811482f9 <+185>:   mov    %ebx,%r15d
   0xffffffff811482fc <+188>:   lea    (%rax,%r12,8),%rax
   0xffffffff81148300 <+192>:   mov    (%rax),%rax
   0xffffffff81148303 <+195>:   test   %rax,%rax
   0xffffffff81148306 <+198>:   je     0xffffffff81148324 <alloc_fd+228>
   0xffffffff81148308 <+200>:   mov    %ebx,%esi
   0xffffffff8114830a <+202>:   mov    $0xffffffff816f1a00,%rdi
   0xffffffff81148311 <+209>:   xor    %eax,%eax
   0xffffffff81148313 <+211>:   callq  0xffffffff8103b890 <printk>
   0xffffffff81148318 <+216>:   mov    0x8(%r14),%rax
   0xffffffff8114831c <+220>:   movq   $0x0,(%rax,%r12,8)
   0xffffffff81148324 <+228>:   mov    -0x38(%rbp),%rdi
   0xffffffff81148328 <+232>:   callq  *0xffffffff81819bc8
   0xffffffff8114832f <+239>:   add    $0x18,%rsp
   0xffffffff81148333 <+243>:   mov    %r15d,%eax
   0xffffffff81148336 <+246>:   pop    %rbx
   0xffffffff81148337 <+247>:   pop    %r12
   0xffffffff81148339 <+249>:   pop    %r13
   0xffffffff8114833b <+251>:   pop    %r14
   0xffffffff8114833d <+253>:   pop    %r15
   0xffffffff8114833f <+255>:   pop    %rbp
   0xffffffff81148340 <+256>:   retq
   0xffffffff81148341 <+257>:   nopl   0x0(%rax)
   0xffffffff81148348 <+264>:   btr    %ebx,(%rax)
   0xffffffff8114834b <+267>:   jmp    0xffffffff811482f2 <alloc_fd+178>
End of assembler dump.
```