Unigornel
Initializing Go in Mini-OS (Part 2)

Maxim Bonnaerens

March 7, 2016

Recap - Let's crash it![1]

```
Breakpoint 1, [...] at 0x6505f: x_cgo_sys_thread_create
(gdb) disas
Dump [...] for function x_cgo_sys_thread_create:
   0x65050 <+0>:   sub    $0x18,%rsp
   0x65054 <+4>:   mov    %rdi,%rdx
   0x65057 <+7>:   mov    %rsi,%rcx
   0x6505a <+10>:  mov    %rsp,%rdi
   0x6505d <+13>:  xor    %esi,%esi
=> 0x6505f <+15>:  mov    %fs:0x28,%rax
   0x65068 <+24>:  mov    %rax,0x8(%rsp)
   0x6506d <+29>:  xor    %eax,%eax
   0x6506f <+31>:  callq  0x9f3f <pthread_create>
   0x65074 <+36>:  test   %eax,%eax
   0x65076 <+38>:  jne    0x6508d
                            <x_cgo_sys_thread_create+61>
```

[1] On Ubuntu Wily with gcc 5.2.1/4.9.3

Recap - Let's crash it![2]

```
Breakpoint 1, [...] at 0x64fc0: x_cgo_sys_thread_create
(gdb) disas
Dump [...] for function x_cgo_sys_thread_create:
=> 0x64fc0 <+0>:   sub    $0x18,%rsp
   0x64fc4 <+4>:   mov    %rdi,%rdx
   0x64fc7 <+7>:   mov    %rsi,%rcx
   0x64fca <+10>:  lea    0x8(%rsp),%rdi
   0x64fcf <+15>:  xor    %esi,%esi
   0x64fd1 <+17>:  callq  0x9f13 <pthread_create>
   0x64fd6 <+22>:  test   %eax,%eax
   0x64fd8 <+24>:  jne    0x64fdf
                            <x_cgo_sys_thread_create+31>
```

[2] On Debian Jessie with gcc 4.9.2

Fix crashes

- Crash on Ubuntu: Using uninitialized %fs-segment
- Crash on Debian: Stack cookies
  - edit src/cmd/go/build.go:

```
case "netbsd":
        a = append(a, "-fno-stack-protector")
}
```

Go Runtime Initialization - Implement GDT

```
//seg_desc_t represents one entry in the GDT.

struct _segment_descriptor {
    unsigned lolimit:16;     /* low bits of maximum segment
    unsigned lobase:24;      /* low bits of segment starting
    unsigned accessed:1;     /* segment accessed */
    unsigned rw:1;           /* readable for code, writable
    unsigned dc:1;           /* direction for data, conformi
    unsigned ex:1;           /* executable for code */
    unsigned system:1;       /* system critical descriptor *
    unsigned dpl:2;          /* descriptor privilege level *
    unsigned present:1;      /* present bit, must be 1 */
    unsigned hilimit:4;      /* high bits of maximum segment
    unsigned zero:2;         /* always zero */
    unsigned size:1;         /* size of p-mode segment */
    unsigned gran:1;         /* granularity of segment 1b fo
    unsigned hibase:8;       /* high bits of maximum segment
} __attribute__((__packed__));
```

Implementing Global Descriptor Table - Initialize GDT

```
void init_gdt(void) {
    pte_t pte; unsigned long frames[1];

    seg_desc_fill(&gdt[SEG_DESC_CS], seg_desc_type_era);
    seg_desc_fill(&gdt[SEG_DESC_DS], seg_desc_type_rwa);
    seg_desc_fill(&gdt[SEG_DESC_FS], seg_desc_type_rwa);

    pte = __pte((virt_to_mach(&gdt)) | L1_PROT_RO);
    HYPERVISOR_update_va_mapping((unsigned long)&gdt,
        pte, UVMF_INVLPG)

    frames[0] = virt_to_mfn(&gdt);
    HYPERVISOR_set_gdt(frames, NUM_GDT_ENTRIES)

    memcpy(&fs, &gdt[SEG_DESC_FS], sizeof(fs));
    fs_pa = virt_to_mach(&gdt[SEG_DESC_FS]);
}
```

Implementing Global Descriptor Table - Initialize GDT

- Initialize GDT when kernel boots
  - Call `init_gdt()` in kernel.c

```
void start_kernel(void) {
    [...]

    /* Init memory management. */
    init_mm();

    /* Init GDT */
    init_gdt();

    [...]
}
```

Implementing Global Descriptor Table - Update `fs`

```
#define KERNEL_FS ((3 << 3) | 1)

void switch_fs(unsigned long p) {
    fs.desc.lobase = p & 0xFFFFFF;
    fs.desc.hibase = (p >> 24) & 0xFF;
    HYPERVISOR_update_descriptor(fs_pa, fs.raw)

    asm volatile("mov %0, %%fs" :: "r"(KERNEL_FS));
}
```

Using fs-segment without crashing

```
//- go_main.c
void thread_main(void *ctx) {
    [...]
    tls_page = alloc_page();
    switch_fs(tls_page + PAGE_SIZE);

    /* TEST TLS */
    *(int *)(tls_page + PAGE_SIZE - 8) = 9876;
    asm volatile(
        "mov %%fs:0xfffffffffffffff8, %0" : "=r"(v)
    );

    if(v != 9876) {
        printk("Could not successfully set up TLS\n");
        *(char *)0 = 0;
    }
    [...]
}
```

Stubbed `pthread` functions give trouble

- Runtime initialization calls `pthread_attr_getstacksize()`[3]
  - Stubbed
  - Runtime thinks stacksize is 0
  - Calls functions for more stack $\Longrightarrow$ crashes
- Solution:
  - Return `STACK_SIZE` used by Mini-OS scheduler[4]

[3] `go/src/runtime/cgo/gcc_netbsd_amd64.c`

[4] `minios/arch/x86/sched.c`

Table of Contents