Unigornel
Initializing Go in Mini-OS (Part 4)

Henri Verroken

March 15, 2016

Recap - Remarks on GDT Implementation

- Why initialize CS and DS, but not SS?
  - CS and DS segment descriptors should not be in our GDT
  - CS, DS and SS registers point to GDT entries added by Xen[1]
  - ES is not used
- What about 64-bit base addresses?
  - Use MSR for upper 32-bits
  - We limit ourselves to addresses below `0xFFFFFFFF`

[1] In Mini-OS: `include/xen/arch-x86/xen-x86-64.h`

Recap - Remarks on GDT Implementation[2]

```
 [...]
-#define KERNEL_CS ((1 << 3) | 1)
-#define KERNEL_DS ((2 << 3) | 1)
-#define KERNEL_FS ((3 << 3) | 1)
-
-#define SEG_DESC_CS (KERNEL_CS >> 3)
-#define SEG_DESC_DS (KERNEL_DS >> 3)

+#define KERNEL_FS ((1 << 3) | 1)
 #define SEG_DESC_FS (KERNEL_FS >> 3)
 [...]
 void init_gdt(void) {
     [...]
-    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);
     [...]
 }
```

[2] https://github.ugent.be/unigornel/minios/commit/614dbe70b3968da67d645e23bd5e8ec059fc17dc

Recap - Remarks on GDT Implementation[3]

```
 void switch_fs(unsigned long p) {
+    ASSERT((p & 0xFFFFFFFF) == p,
+           "FS base is limited to 32 bits");

     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)
    );
 }
```

[3] https://github.ugent.be/unigornel/minios/commit/91817cfa8e99299355d220f8f111f02e8fae4c65

Recap - Go Runtime Memory Requirements

- Mini-OS
  - Pseudo-physical = virtual
  - Page tables filled at startup and never changed
    - Small virtual address range
    - Cannot use high addresses
- Go Runtime
  - Uses `mmap` with `PROT_NONE` to reserve around 500GB
  - Allocates from that region when needed
  - Uses high addresses
- Mini-OS and Go are incompatible
  - Try to limit memory requirements in Go

Memory

- Decrease memory requirements in Go[4]

```
const (
    _MHeapMap_TotalBits = 26 // 64 MByte
    _MaxMem = uintptr(1 << _MHeapMap_TotalBits - 1)
)

arenaSize := round(_MaxMem, _PageSize)
```

- Increase VM memory to 256 MB
- Implement basic `mmap` in Mini-OS[5]

[4] https://github.ugent.be/unigornel/go/commit/d37f5677be6a604c41e713ed3d186fc110ce1a06

[5] https://github.ugent.be/unigornel/minios/commit/fcde24adc6db56e43cf48c1e46106b6151661ffc

Synchronization

- Go implements locks using OS primitives
  - On NetBSD using semaphores[6]
    - Platform-specific `semasleep` and `semawake`[7]
    - Using NetBSD `lwp_park` and `lwp_unpark`
    - `lwp` has quite difficult specs
  - On Linux using futex-mechanism[8]
    - `futex` has a very easy specification
- Implement basic `futex` wait/wake in Mini-OS[9]
- Change Go to use futexes for NetBSD[10]

[6] In `runtime/lock_sema.go`

[7] In `runtime/os1_netbsd.go`

[8] In `runtime/lock_futex.go`

[9] `https://github.ugent.be/unigornel/minios/commit/eeacb0f2659ea8dddc155360a15cf3188f2abc5d`

[10] `https://github.ugent.be/unigornel/go/commit/b3d174d7c53ec33327beaf709b70889cf7e56830`

Various System Calls

- Implement some more system calls
  - `thread_id` for thread identification
  - Remove all `signal` stuff
  - Use valid `argc`, `argv` and `envs`
  - Implement `usleep`
  - Implement `now`
  - Implement `write` to print to console
  - Avoid open-ing `/dev/random` for random data[11,12]

[11] The current implementation of `getRandomData` uses only `nanotime` as its source of entropy.

[12] `https://github.ugent.be/unigornel/go/commit/6cbdecf80df6a62516faccf0ff2b966d7dd5370e`

Deadlock - Stuck in the GC

- Stuck in the GC thread
- All other threads were blocking[13]
  - Fix memory bug in `futex`-implementation
  - Implement variant of pthreads conditions

[13] `https://github.ugent.be/unigornel/minios/commit/f7a9121e29cbf108b4a188c06060cdd6a365df5c`

Hello World - Hello Sum[14,15]

```
package main

import "C"

func main() {}

//export Sum
func Sum(a, b int) int {
        return a + b
}
```

[14] Mini-OS at f7a9121e29cbf108b4a188c06060cdd6a365df5c

[15] Go at d7d2d60a91bd0cb79b09e07a4468c0fcf4c7d711

Hello World - Hello Sum[16,17]

```
static void *initialize_go_thread(void *ctx) {
    _rt0_amd64_netbsd_lib();
    return NULL;
}

static void *main_thread(void *ctx) {
    GoInt i = Sum(3, 4);
    printk("main_thread: result: %lld\n", i);
    CRASH("main thread must not return");
    return NULL;
}

int app_main(start_info_t *si) {
    pthread_t t;
    pthread_create_name(&t, "main", NULL,
                        main_thread, NULL);
    pthread_create_name(&t, "initialize_go", NULL,
                        initialize_go_thread, NULL);
    return 0;
}
```

[16] Mini-OS at f7a9121e29cbf108b4a188c06060cdd6a365df5c

[17] Go at d7d2d60a91bd0cb79b09e07a4468c0fcf4c7d711

Hello World - Hello Sum

```
[...]
go_main.c: app_main(0xffe40)
Thread "main": pointer: 0x2b2220, stack: 0x2f0000
Thread "initialize_go": pointer: 0x2b2300, stack: 0x300000
mallocinit: arenaSize 64 MByte
mmap(addr=[...],len=0x4412000,prot=0x0, flags=0x1002)
mheap_.spans = 0x310000
mheap_.bitmap = 0x320000
mheap_.arena_start = 0x720000
mheap_.arena_used = 0x720000
mheap_.arena_end = 0x4722000
mheap_.arena_reserved = true
mmap(addr=0x0,len=0x40000,prot=0x3, flags=0x1002)
mmap(addr=0x720000,len=0x100000,prot=0x3, flags=0x1002)
mmap(addr=0x718000,len=0x8000,prot=0x3, flags=0x1002)
mmap(addr=0x310000,len=0x1000,prot=0x3, flags=0x1002)
mmap(addr=0x0,len=0x10000,prot=0x3, flags=0x1002)
Thread "pthread-0": pointer: 02b2488, stack: 04780000
mmap(addr=0x0,len=0x40000,prot=0x3, flags=0x1002)
Thread "pthread-1": pointer: 02b25a0, stack: 047d0000
main_thread: result: 7
crash: go_main.c, line 25: main thread must not return
```

Hello World[18,19]

```
package main

import "C"
import "fmt"

func main() {}

//export Main
func Main(unused int) {
        fmt.Println("Hello World!")
}
```

[18] Mini-OS at c1b830d41e4612c49c95bb4c8b906137361a92c6

[19] Go at 6cbdecf80df6a62516faccf0ff2b966d7dd5370e

Hello World[20,21]

```
static void *initialize_go_thread(void *ctx) {
    _rt0_amd64_netbsd_lib();
    return NULL;
}

static void *main_thread(void *ctx) {
    Main(0);
    CRASH("main thread must not return");
    return NULL;
}

int app_main(start_info_t *si) {
    pthread_t t;
    pthread_create_name(&t, "main", NULL,
                        main_thread, NULL);
    pthread_create_name(&t, "initialize_go", NULL,
                        initialize_go_thread, NULL);
    return 0;
}
```

[20] Mini-OS at c1b830d41e4612c49c95bb4c8b906137361a92c6

[21] Go at 6cbdecf80df6a62516faccf0ff2b966d7dd5370e

Hello World

```
[...]
mheap_.spans = 0x380000
mheap_.bitmap = 0x390000
mheap_.arena_start = 0x790000
mheap_.arena_used = 0x790000
mheap_.arena_end = 0x4792000
mheap_.arena_reserved = true
mmap(addr=[...],len=0x40000,prot=0x3, flags=0x1002)
mmap(addr=[...],len=0x100000,prot=0x3, flags=0x1002)
mmap(addr=[...],len=0x8000,prot=0x3, flags=0x1002)
mmap(addr=[...],len=0x1000,prot=0x3, flags=0x1002)
mmap(addr=[...],len=0x10000,prot=0x3, flags=0x1002)
Thread "pthread-0": pointer: 0x329488, stack: 0x47f0000
mmap(addr=[...],len=0x40000,prot=0x3, flags=0x1002)
Thread "pthread-1": pointer: 0x3295a0, stack: 0x4840000
Hello World!
crash: go_main.c, line 21: main thread must not return
```

What's Next

- Clean up code
- Rewrite parts in Go (e.g. `futex`)
- Test with more complex applications
  - Do some memory intensive things
  - Use channels and multiple goroutines
- Support full virtual address space
- Explore networking in Xen
- Automated testing

Table of Contents