Unigornel
Initializing Go in Mini-OS (Part 3)
Memory

Henri Verroken

March 7, 2016

Memory Requirements for Go

- Go needs a contiguous address space
  - Uses `mmap` with `PROT_NONE` to reserve around 500GB of virtual memory
  - Uses `mmap` with `PROT_READ | PROT_WRITE` to actually allocate memory when needed.
- Go tries to uses high addresses

Virtual Memory in Mini-OS

- Pseudo-physical = virtual
- Page tables filled at startup and never changed
  - Virtual address range from `0x0` to available amount of memory (e.g. 32MB)
  - Cannot use high addresses
- Incompatible with Go runtime

Short Term Solution - Edit Go

- Change memory requirements for Go
  - Reserve much less address space than 500GB
  - Smaller memory management structures
  - Lower requested virtual addresses
- Memory initialized in `mallocinit` in `runtime/malloc.go`

Short Term Solution - Edit Go

```
func mallocinit() {
    [...]
    arenaSize := round(_MaxMem, _PageSize)
    bitmapSize = arenaSize / (sys.PtrSize * 8 / 4)
    spansSize = arenaSize / _PageSize * sys.PtrSize
    spansSize = round(spansSize, _PageSize)
    [...]
    pSize = bitmapSize + spansSize + arenaSize + _PageSize
    p = uintptr(sysReserve(unsafe.Pointer(p),
                           pSize, &reserved)) // mmap
    [...]
    p1 := round(p, _PageSize)
    mheap_.spans = (**mspan)(unsafe.Pointer(p1))
    mheap_.bitmap = p1 + spansSize
    mheap_.arena_start = p1 + (spansSize + bitmapSize)
    mheap_.arena_used = mheap_.arena_start
    mheap_.arena_end = p + pSize
    mheap_.arena_reserved = reserved
}
```

Long Term Solution - Edit Mini-OS

- Give Mini-OS mature virtual memory management
  - Use modifiable page tables
  - Allow use of high memory addresses
  - Virtual $\neq$ pseudo-physical
- More flexibility for future development
  - Reduce/increase memory provided by Xen
  - Dynamic memory allocation in guest

Table of Contents