MEMOS-2

Timothy Borunov (Slides by Shriram Raja)

Ph.D. Student, Computer Science

MEMOS-2

- Same as MEMOS-1; print the total memory and the different ranges of memory
- Instead of writing a Master Boot Record, we now write the actual kernel
- GRUB detects memory for you and puts you in Protected Mode:
  - can access up to 4 GB of memory
  - segmentation for rings of separation
  - 32-bit Ring 0
- For now, no need to worry about segmentation; upcoming assignments will focus on that

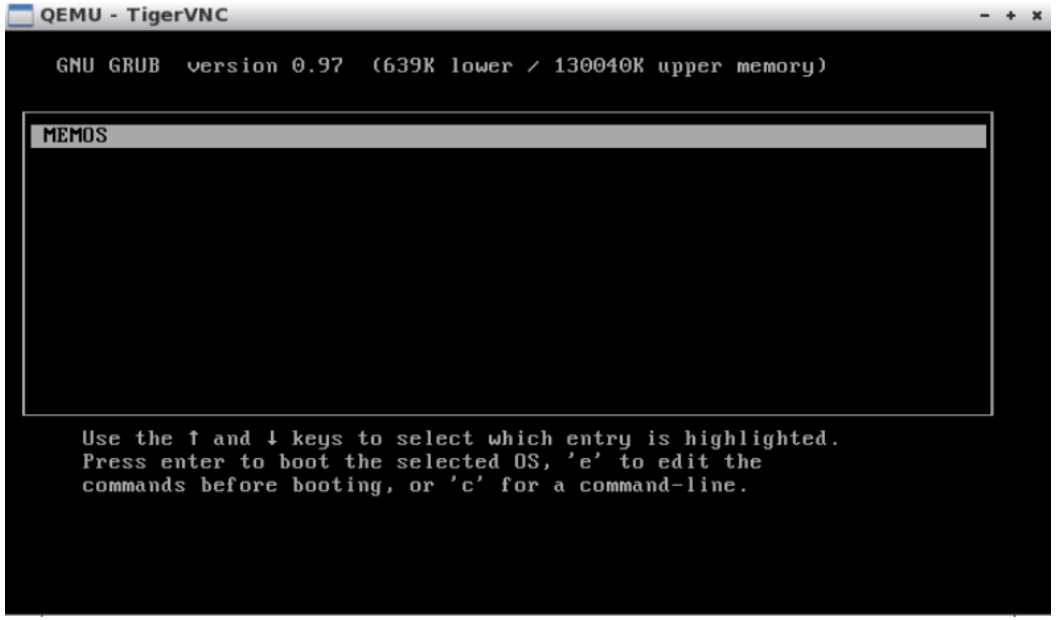
GNU GRUB

- **GR**and **U**nified **B**ootloader (a play on Grand Unified Theory)
- We will use GRUB Legacy (Version 0)
- Stage 1: cannot do much in 512 B (as you should know now); boots stage 1.5
- Stage 1.5: contains file system drivers, enabling it to directly load stage 2 from any known location in the filesystem, for example from /boot/grub
- Stage 2: holds the bulk of GRUB functionality

GRUB Stage 2

- Can execute commands related to loading the OS
- Processes menu.lst to get the menu description; kernel should be elf
- Displays the menu items to the user and gets input from the keyboard

```
title MEMOS
root (hd0,0)
kernel /path/to/memos2.elf
```



GRUB Stage 2

- Can execute commands related to loading the OS
- Processes menu.lst to get the menu description; kernel should be elf
- Displays the menu items to the user and gets input from the keyboard
- Loads the kernel at 0x100000 (1 MB), passes information about current state and jumps to 0x100000

Linker Script (link.ld)

```
OUTPUT_FORMAT("elf32-i386")
OUTPUT_ARCH(i386)
ENTRY(_start)
SECTIONS {
 . = 0x100000;
 .ksection : {
 *(.*);
 . = ALIGN(0x1000);
 }
}
```

- Where should the bootloader write the boot information for the kernel to access?

Multiboot Specification

- Standard interface between bootloader and kernel
- Multiboot compliant kernel binary should have the Multiboot header as early as possible in the first page
- Bootloader writes the boot info in a struct defined in the multiboot header
- The address of this structure to be passed to the kernel is in %ebx

Kernel asm starter code (stub.S)

```
.text

.globl _start
_start:
    jmp real_start
    # Multiboot header – Must be in 1st page of memory for GRUB
    .align 4
    .long 0x1BADB002 # Multiboot magic number
    .long 0x00000003 # Align modules to 4KB, req. mem size
    # See 'info multiboot' for further info
    .long 0xE4524FFB # Checksum

real_start:
    #TODO: Setup a proper stack for C
    #TODO: Prepare the boot information to pass to kmain
    call kmain
    hlt
```

You can pass arguments by pushing them to the stack.

refer C calling convention [3]

Now, you C me!

- Finally, we are in C! But don't get too happy
- No library functions
- But printing is still easier with familiar programming techniques

Makefile

```
as --32 stub.S -o stub.o
gcc -m32 -fno-stack-protector -fno-builtin -nostdinc -c kentry.c -o kentry.o
ld -m elf_i386 -T link.ld stub.o kentry.o -o memos2.elf
```

Kernel C starter code (kentry.c)

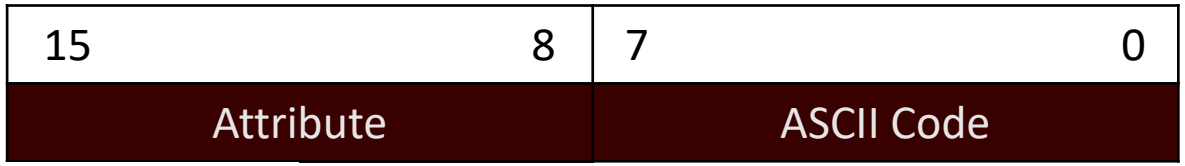
```c
#include "memos.h" /* this should include multiboot.h */

void puts (char *text) {
    for (int i = 0; i < my_strlen ((const char*)text); i++)
        /* You know how to implement strlen ;) */
        putc (text[i]);
}

/* this is a custom struct; you need to decide what makes sense here */
void kmain (boot_info_t* binfo) {
    puts ("MemOS: Welcome *** Total Free Memory: ");
}
```

Printing to the Screen

- Video Memory base address: 0xB8000
- For each print: first byte is the ASCII code and the second is the attribute
- Thus, in x86, since it uses little endian:



Depends on hardware configuration;
Refer Osdev for more info

putc()

```
/* Base address of the VGA frame buffer */
static unsigned short *videoram = (unsigned short *) 0xB8000;
static int attrib = 0x0F; /* black background, white foreground */
static int csr_x = 0, csr_y = 0;
#define COLS 80

void putc (unsigned char c) {
    if (c == 0x09) { /* Tab (move to next multiple of 8) */
        csr_x = (csr_x + 8) & ~(8 - 1);
    } else if (c == '\r') { /* Carriage Return */
        csr_x = 0;
    } else if (c == '\n') { /* Line Feed (unix-like) */
        csr_x = 0; csr_y++;
    } else if(c >= ' ') { /* Printable characters */
        /* Put the character w/attributes */
        *(videoram + (csr_y * COLS + csr_x)) = c | (attrib << 8);
        csr_x++;
    }
    if(csr_x >= COLS){ csr_x = 0; csr_y++;} /* wrap around */
}
```

References

1. GRUB Legacy - Wiki, Working, Features
2. Multiboot - Wiki, Specification, header file (older than the current version in the specification)
3. Calling C functions from Assembly
4. Printing to Screen in Protected Mode
5. Linker Commands
6. More info about E820