CS 450 Module R0
Getting Started

West Virginia University

Goals

- Prepare your system for MPX development
- Establish group development using GitHub
- Become familiar with building, running, and modifying MPX and sharing changes
- Initialize the low-level functionality of MPX

System Preparation

- Preparation varies based on your computer's host operating system
- Guides are provided for Windows, macOS, and Ubuntu
- Let the TA know immediately if your computer runs a different OS than those listed

Collaboration with GitHub

- We'll be using GitHub to collaborate among your group – and turn in assignments to the TA
- If you don't already have one, sign up for a GitHub account at `https://github.com`
- Have one member of your group email the TA with your group's name and GitHub usernames so you can be added to the appropriate GitHub teams
- You will be added to a private repository containing your group's copy of MPX

Configuring SSH Keys

- You'll need to use SSH keys to connect with GitHub
- Create a key by running `ssh-keygen`
  - Pay close attention to where it outputs "Your public key has been saved in"
- In the GitHub web interface, click your user icon in the top right, then Settings from the drop-down
- Select "SSH and GPG keys" on the left, then click the "New SSH key" button
- Copy the contents of your public key into the Key box and click "Add SSH key"

Obtaining MPX

- Once your SSH key is enabled, you can clone the repository
- Open the repository in GitHub's web interface
- Click the green Code button to get a repository URL you can copy and paste
- Run `git clone` *`URL you copied`*

run git clone from the command line

Compiling MPX

- Once your system is prepared, you should be able to compile MPX
- To compile, `cd` to the top level directory and run `make`
- You should see output similar to the following:

```
~/mpx$ make
nasm -f elf -g  -o kernel/boot.o kernel/boot.s
nasm -f elf -g  -o kernel/irq.o kernel/irq.s
clang -std=c18 --target=i386-elf -Wall -Wextra -Werror -ffreestanding -g -Iinclude   -c -o kernel/gdt.o kernel/gdt.c
clang -std=c18 --target=i386-elf -Wall -Wextra -Werror -ffreestanding -g -Iinclude   -c -o kernel/interrupts.o kernel/interrupts.c
clang -std=c18 --target=i386-elf -Wall -Wextra -Werror -ffreestanding -g -Iinclude   -c -o kernel/kmain.o kernel/kmain.c
clang -std=c18 --target=i386-elf -Wall -Wextra -Werror -ffreestanding -g -Iinclude   -c -o kernel/panic.o kernel/panic.c
clang -std=c18 --target=i386-elf -Wall -Wextra -Werror -ffreestanding -g -Iinclude   -c -o kernel/serial.o kernel/serial.c
clang -std=c18 --target=i386-elf -Wall -Wextra -Werror -ffreestanding -g -Iinclude   -c -o kernel/vm.o kernel/vm.c
clang -std=c18 --target=i386-elf -Wall -Wextra -Werror -ffreestanding -g -Iinclude   -c -o lib/ctype.o lib/ctype.c
clang -std=c18 --target=i386-elf -Wall -Wextra -Werror -ffreestanding -g -Iinclude   -c -o lib/stdlib.o lib/stdlib.c
clang -std=c18 --target=i386-elf -Wall -Wextra -Werror -ffreestanding -g -Iinclude   -c -o lib/string.o lib/string.c
clang -std=c18 --target=i386-elf -Wall -Wextra -Werror -ffreestanding -g -Iinclude   -c -o user/system.o user/system.c
i686-linux-gnu-ld -melf_i386 -znoexecstack -T kernel/link.ld -o kernel.bin kernel/boot.o kernel/irq.o kernel/gdt.o kernel/interrupts.o kernel/kmain.o kernel/panic.o kernel/serial.o kernel/vm.o lib/ctype.o lib/stdlib.o lib/string.o user/system.o
```

Running MPX

- To execute MPX, run `./mpx.sh`
- This will launch QEMU with MPX as the kernel
- Initially, not much is happening:

```
~/mpx$ ./mpx.sh
SeaBIOS (version 1.14.0-2)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8F360+07ECF360 CA00
Press Ctrl-B to configure iPXE (PCI 00:03.0)...


Booting from ROM..
```

Background Knowledge

- MPX is implemented in *freestanding* C – there is no standard library
- You do have access to the following standard headers:

- `<float.h>`
- `<iso646.h>`
- `<limits.h>`
- `<stdalign.h>`
- `<stdarg.h>`
- `<stdbool.h>`
- `<stddef.h>`
- `<stdint.h>`
- `<stdnoreturn.h>`

Limited Library

- We have provided minimal implementations of the following:
  - `atoi()` (`<stdlib.h>`) – convert a string to an integer
  - `isspace()` (`<ctype.h>`) – determine if a character is whitespace
  - `memcpy()` (`<string.h>`) – copy a region of memory
  - `memset()` (`<string.h>`) – fill a region of memory
  - `strcmp()` (`<string.h>`) – compare two NUL-terminated strings
  - `strlen()` (`<string.h>`) – determine the length of NUL-terminated string
  - `strtok()` (`<string.h>`) – tokenize a NUL-terminated string
- Behavior *should* match the C standard, but is not guaranteed

A good early task is to also implement an itoa function

MPX Initialization – Serial Communications

- Initialization is done in `kmain()` (in `kernel/kmain.c`)
- Initialize COM1 (see `include/mpx/serial.h` for constants and functions)

MPX Initialization – GDT

- The Global Descriptor Table (GDT) describes the various memory segments used by the x86 architecture
- Structures and functions are in `include/mpx/gdt.h`
- The GDT must be initialized before you can continue initializing interrupt support

MPX Initialization – Interrupts

- The Interrupt Descriptor Table (IDT) contains pointers to functions that need to be called when interrupts occur
- For proper functionality, the x86 architecture requires interrupt handlers for the first 32 IRQs
- The x86 architecture uses a Programmable Interrupt Controller (PIC) to marshal and dispatch interrupts
- See `include/mpx/interrupts.h` for functions relating to interrupts
- Once the IDT, PIC and interrupt handlers are initialized, enable hardware interrupts with the `sti()` macro

MPX Initialization – Virtual Memory

- Virtual memory (AKA memory paging) allows the CPU to translate logical memory addresses to physical memory addresses (this is what allows each process to behave as though it has exclusive access to memory)
- Virtual memory is managed using *page tables*
- Structures, constants, and functions related to virtual memory are in `include/mpx/vm.h`
- Virtual memory must also be initialized for a fully functional kernel

MPX Initialization – Wrap Up

- Once the GDT, IDT, interrupt handlers, PIC, and virtual memory have been initialized, you can enable interrupts with `sti()`
- All of these changes should be in `kmain()` (in `kernel/kmain.c`)
- Rebuild with `make` (**fix warnings and errors**)
- Boot your new kernel with `./mpx.sh`
- If it still boots, you are ready to begin R1