System Programming

11. B. BootLoader Codes

2019. Fall

Instructor: Joonho Kwon

jhkwon@pusan.ac.kr

Data Science Lab @ PNU

These slides are based on the web page "Writing a boot loader in Assembly and C - Part 1"

datalab data science laboratory

Tool chains

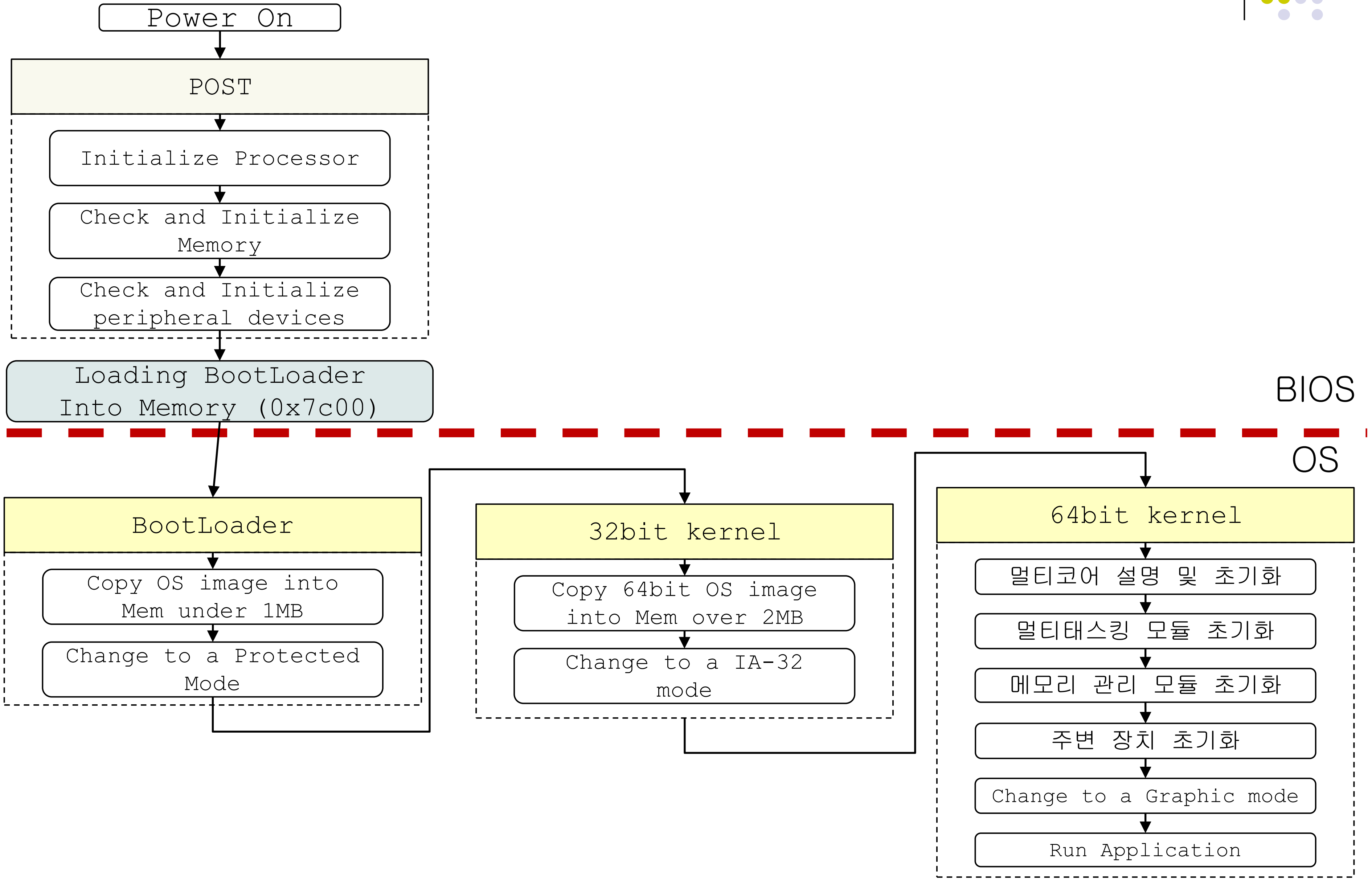
- gcc
- make
- qemu

Qemu intall

- Use an apt-get command on Ubuntu

```
$ sudo apt-get install qemu
[sudo] password for ***:
Reading package lists... Done
Building dependency tree
…
```

Boot Loader 동작



Real Mode(16bit) Boot Loader (1/2)

- test.S

```
.code16                     #generate 16-bit code
.text                       #executable code location
     .globl _start;

_start:                     #code entry point
     . = _start + 510       #mov to 510th byte from 0 pos
     .byte 0x55             #append boot signature
     .byte 0xaa             #append boot signature
```

Real Mode(16bit) Boot Loader (2/2)

- Meaning
  - .code16
    - To avoid assembler and compilers generating 32-bit code, we use this directive.
  - .text:
    - The .text section contains the actual machine instructions, which make up your program.
  - .globl _start:
    - .global <symbol> makes the symbol visible to linker.
  - _start:
    - Entry to the main code and _start is the default entry point for the linker.
  - . = _start + 510:
    - traverse from beginning through 510th byte
  - .byte 0x55:
  - .byte 0xaa:

Compile Bootloader

- Compile using gas

```
$ as test.S -o test.o
```

- Liking

```
$ ld –Ttext 0x7c00 --oformat=binary test.o –o test.bin
```

  - --oformat=binary
    - tells the linker you want your output file to be a plain binary image (no startup code, no relocations, ...).
  - –Ttext 0x7c00
    - tells the linker you want your "text" (code segment) address to be loaded to 0x7c00 and thus it calculates the correct address for absolute addressing.

boot signature

- How does BIOS recognize if a device contains a boot sector or not?
  - a boot sector is 512 bytes long
    - in 510th byte a symbol 0x55 is expected
    - in the 511th byte another symbol 0xaa is expected

Booting Test(1/3)

- copy the executable code to a bootable device and then test it
  - To create a floppy disk image of 1.4mb size,

```
$ dd if=/dev/zero of=floppy.img bs=512 count=2880
```

  - To copy the code to the boot sector of the floppy disk image file

```
$ dd if=test.bin of=floppy.img
```

Booting Test(2/3)

- Makefile

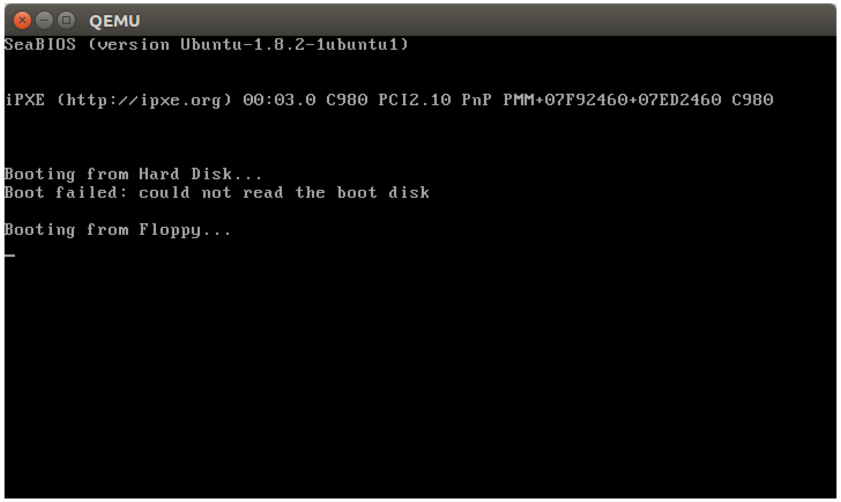
```
all: test.bin
SRC=test.S

test.bin: $(SRC)
	as $(SRC) -o test.o
	ld -Ttext 0x7c00 --oformat=binary -o test.bin test.o
	dd if=/dev/zero of=floppy.img bs=512 count=2880
	dd if=test.bin of=floppy.img

clean:
	rm -f floppy.img test.o test.bin
```

Booting Test(3/3)

- Executing by QEMU

```
$ qemu-system-i386 -fda floppy.img
```

  - Booting from floppy Success
    - But, nothing happens.



```
SeaBIOS (version Ubuntu-1.8.2-1ubuntu1)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F92460+07ED2460 C980



Booting from Hard Disk...
Boot failed: could not read the boot disk

Booting from Floppy...
_
```

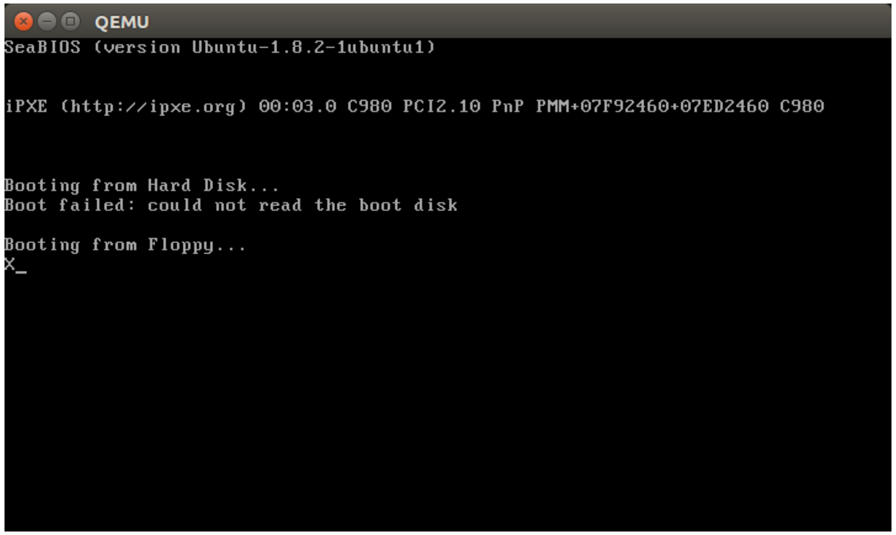
BootLoader v2 (1/2)

- test2.S

```
.code16                         #generate 16-bit code
.text                           #executable code location
     .globl _start;

_start:                         #code entry point
     movb $'X' , %al             #character to print
     movb $0x0e, %ah             #bios service code to print
     int  $0x10                  #interrupt the cpu now

     . = _start + 510            #mov to 510th byte from 0 pos
     .byte 0x55                  #append boot signature
     .byte 0xaa                  #append boot signature
```

BootLoader v2 (2/2)

- See a letter “X”



```
SeaBIOS (version Ubuntu-1.8.2-1ubuntu1)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F92460+07ED2460 C980



Booting from Hard Disk...
Boot failed: could not read the boot disk

Booting from Floppy...
X_
```

Boot Loader V3 (1/4)

- test3.S

```
.code16                           #generate 16-bit code
.text                             #executable code location
     .globl _start;

_start:                           #code entry point
     #print letter 'H' onto the screen
     movb $'H' , %al
     movb $0x0e, %ah
     int  $0x10

     #print letter 'e' onto the screen
     movb $'e' , %al
     movb $0x0e, %ah
     int  $0x10

     #print letter 'l' onto the screen
     movb $'l' , %al
     movb $0x0e, %ah
     int  $0x10

     #print letter 'l' onto the screen
     movb $'l' , %al
     movb $0x0e, %ah
     int  $0x10
```

Boot Loader V3 (2/4)

- test3.S

```
#print letter 'o' onto the screen
movb $'o' , %al
movb $0x0e, %ah
int  $0x10

#print letter ',' onto the screen
movb $',' , %al
movb $0x0e, %ah
int  $0x10

#print space onto the screen
movb $' ' , %al
movb $0x0e, %ah
int  $0x10

#print letter 'W' onto the screen
movb $'W' , %al
movb $0x0e, %ah
int  $0x10

#print letter 'o' onto the screen
movb $'o' , %al
movb $0x0e, %ah
int  $0x10

#print letter 'r' onto the screen
movb $'r' , %al
movb $0x0e, %ah
int  $0x10
```

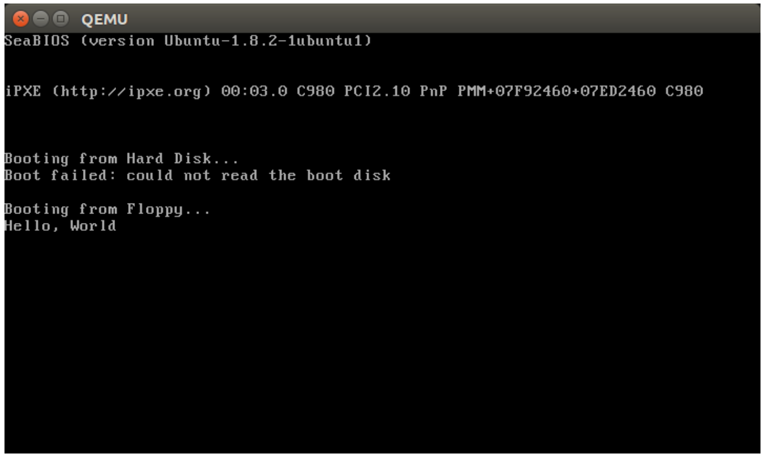
Boot Loader V3 (3/4)

- test3.S

```
      #print letter 'l' onto the screen
      movb $'l' , %al
      movb $0x0e, %ah
      int  $0x10

      #print letter 'd' onto the screen
      movb $'d' , %al
      movb $0x0e, %ah
      int  $0x10

      . = _start + 510      #mov to 510th byte from 0 pos
      .byte 0x55            #append boot signature
      .byte 0xaa            #append boot signature
```

Boot Loader V3 (4/4)

- See “Hello World”



```
SeaBIOS (version Ubuntu-1.8.2-1ubuntu1)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F92460+07ED2460 C980



Booting from Hard Disk...
Boot failed: could not read the boot disk

Booting from Floppy...
Hello, World
```

Boot Loader V4 (1/3)

- test4.S
  - Macro

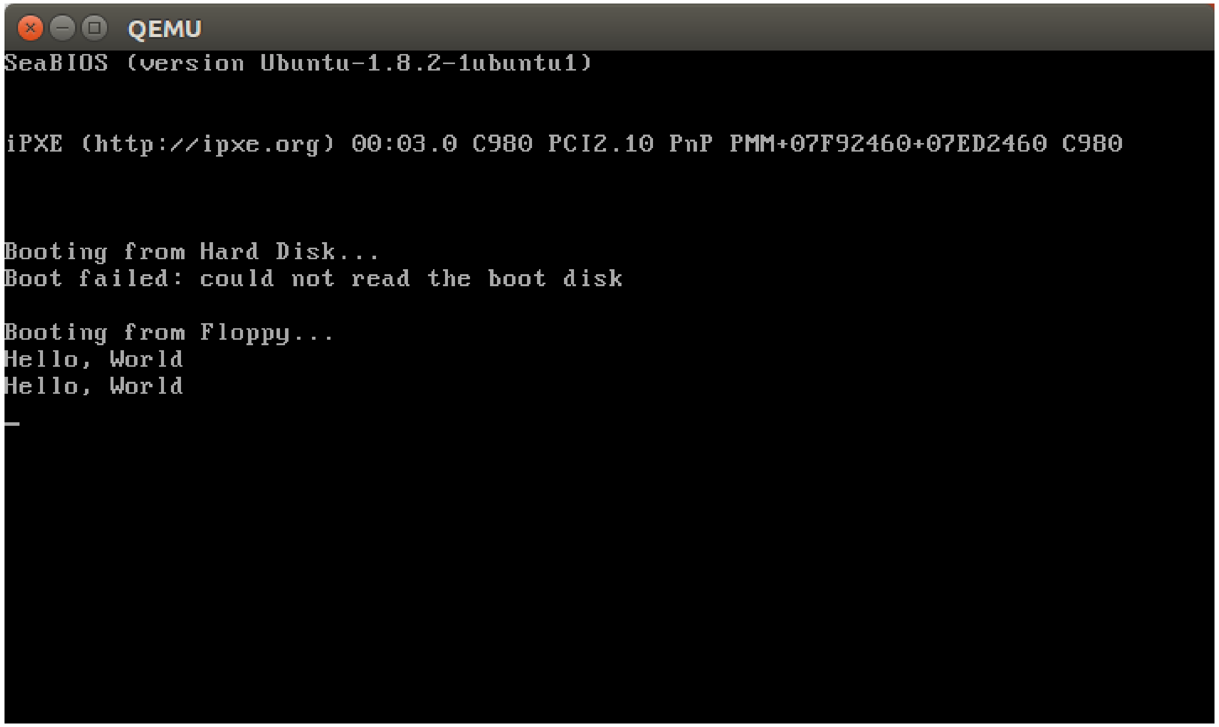
```
.code16           #generate 16-bit code
                  #hint the assembler that here is the executable code located
.text
.globl _start;
#boot code entry
_start:
      jmp _boot                                 #jump to boot code
      welcome: .asciz "Hello, World\n\r"  #here we define the string

     .macro mWriteString str         #macro which calls a function to print a string
          leaw  \str, %si
          call .writeStringIn
     .endm
```

Boot Loader V4 (2/3)

```
     #function to print the string
     .writeStringIn:
          lodsb
          orb  %al, %al
          jz   .writeStringOut
          movb $0x0e, %ah
          int  $0x10
          jmp  .writeStringIn
     .writeStringOut:
     ret

_boot:
     mWriteString welcome

     #move to 510th byte from the start and append boot signature
     . = _start + 510
     .byte 0x55
     .byte 0xaa
```

Boot Loader V4 (3/3)

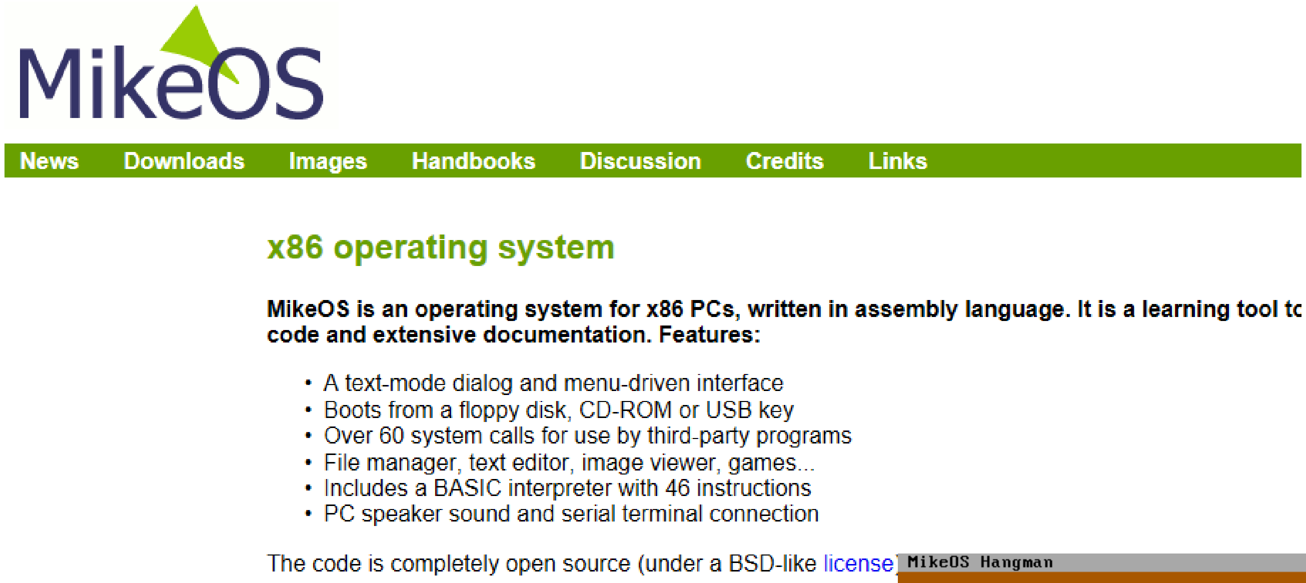
- See “Hello World”



```
SeaBIOS (version Ubuntu-1.8.2-1ubuntu1)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F92460+07ED2460 C980



Booting from Hard Disk...
Boot failed: could not read the boot disk

Booting from Floppy...
Hello, World
Hello, World
_
```

A first step for building Your own OS

- A Project to Build Your own OS

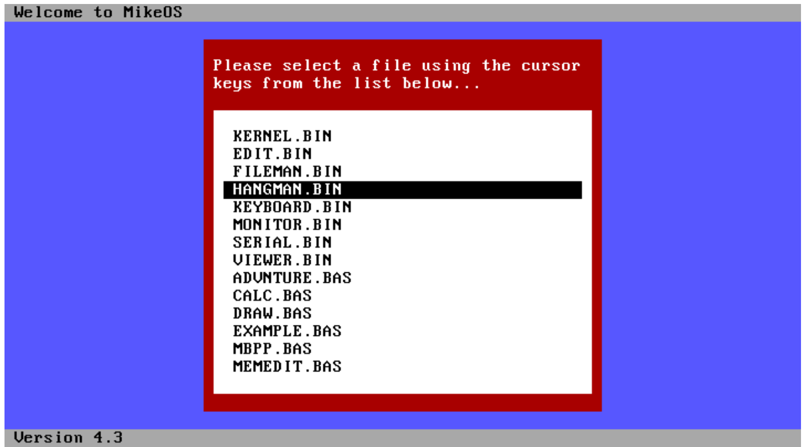


x86 operating system

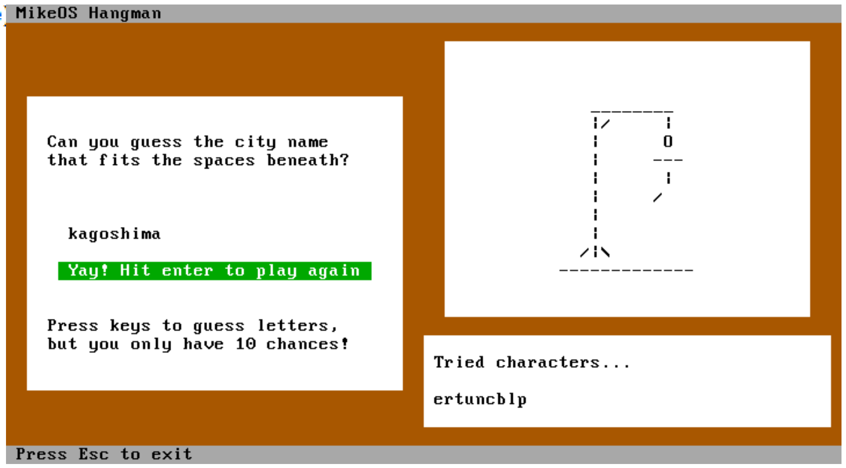
MikeOS is an operating system for x86 PCs, written in assembly language. It is a learning tool tc code and extensive documentation. Features:

- A text-mode dialog and menu-driven interface
- Boots from a floppy disk, CD-ROM or USB key
- Over 60 system calls for use by third-party programs
- File manager, text editor, image viewer, games...
- Includes a BASIC interpreter with 46 instructions
- PC speaker sound and serial terminal connection

The code is completely open source (under a BSD-like license)



```
Welcome to MikeOS

Please select a file using the cursor
keys from the list below...

KERNEL.BIN
EDIT.BIN
FILEMAN.BIN
HANGMAN.BIN
KEYBOARD.BIN
MONITOR.BIN
SERIAL.BIN
VIEWER.BIN
ADVNTURE.BAS
CALC.BAS
DRAW.BAS
EXAMPLE.BAS
MBPP.BAS
MEMEDIT.BAS

Version 4.3
```



```
MikeOS Hangman

Can you guess the city name
that fits the spaces beneath?

kagoshima

Yay! Hit enter to play again

Press keys to guess letters,
but you only have 10 chances!

Tried characters...

ertuncblp

Press Esc to exit
```

Q&A