How does an operating system boot?

- firstly we need to understand the minimum details as to how the IBM-PC boots

- secondly we need to understand the desired final outcome at the end of the boot phase

- thirdly we can examine a specific example to better understand the steps taken to achieve the outcome

How does an IBM-PC boot?

- the bios settings dictates the boot device order

- the bios attempts to load in the first 512 bytes (boot sector) from the various devices in order

- not all devices may be present: usb memory stick, floppy disk

- the bios loads in the 512 bytes from the first found device at 0x7c00
  - it sets register: `dl` to the device number
  - the bios then jumps to location 0x7c00

Boot sector code charactistics and functionality

- it remembers the bios boot device (usb/floppy/harddisk) in a processor register `dl`
- it copies itself into a sensible location (typically out of the way in high memory)
- it reassigns the stack to a consistant location
- it loads in the secondary boot stage of the operating system (a sector at a time)

Boot sector code charactistics and functionality

- it may perform very limited checking as each subsequent sector/track is read from the device into memory

- finally it jumps to the start of the secondary code.

The language used to implement the bootsector (first)

- the boot sector (`first`) is not normally written in a high level language as it needs the ability to:
    - copy its code segment, reassign the stack (change the `SP` and stack segment registers)
    - the ability to jump to a physical location and it must fit in 512 bytes

Final desired outcome after all the boot phases are complete

- line from 1.750,8.500 to 1.750,9.500 to 3.750,9.500 to 3.750,8.500 dashwid = 0.050 line dashed from 1.750,8.500 to 1.750,8.000 line dashed from 3.750,8.500 to 3.750,8.000 line from 1.750,8.000 to 1.750,7.000 to 3.750,7.000 to 3.750,8.000 line from 1.750,8.500 to 3.750,8.500 line from 1.750,8.000 to 3.750,8.000 box with .sw at (1.750,4.000) width 2.000 height 2.500 line from 1.750,4.250 to 3.750,4.250 line from 1.750,4.750 to 3.750,4.750 line from 1.750,5.250 to 3.750,5.250 line from 1.750,5.750 to 3.750,5.750 "Max memory" at 0.750,9.283 ljust "n Mb" at 0.750,9.033 ljust "Stack" at 2.500,9.033 ljust "(grows downwards)" at 2.250,8.783 ljust "1 Mb" at 0.750,7.033 ljust "Heap" at 2.500,7.533 ljust "(grows upwards)" at 2.250,7.283 ljust "Interrupt Vectors" at 2.250,4.096 ljust "640k" at 0.750,6.533 ljust "0" at 0.750,4.033 ljust "10000H" at 0.750,4.283 ljust "(loaded)" at 3.000,4.533 ljust "BSS" at 2.438,5.471 ljust "initially zero" at 2.812,5.471 ljust "Data" at 2.438,4.971 ljust "(loaded)" at 3.000,4.971 ljust "Code" at 2.438,4.533 ljust "unused" at 2.438,6.033 ljust

Example: LuK booting

- LuK consists of a collection of modules
- the microkernel only links the modules actually required at runtime
- the mixture of the modules required for different targets and applications may be different

Linker

- uses the file init to generate a list of modules and generates an ELF 32 bit x86 executable which contains data, code and symbol information
    - for example we will name this, *application*.third

Boot phases

- in the build directory you would see
  - first, second

first

- first is a tiny model 8086 executable, written in assembly language
- see luk-1.0.3/boot/BAS/boot.S
  - watch out as the assembler uses: `mov dest, src`
- its total size (data + code) must not exceed 512 bytes
- its duty is threefold
  - pretend to be a fat12 file system!
  - move itself to a sane location
  - load in second

second

- is written in Modula-2, which is compiled and linked into a tiny model 8086 executable

- tiny model
  - sets all segment registers to the same value
  - total size of data + code + stack must not exceed 64K

- in fact due to legacy booting via the floppy disk it cannot be more than 7K

second

- its duty is to load in the *application*.third
- set up protected mode and move from tiny model into 32 bits
- pass various system parameters into *application*.third
  - such as memory size, video memory start
- finally jump to the start of *application*.third

Goal of the overall boot procedure

- line from 1.750,8.500 to 1.750,9.500 to 3.750,9.500 to 3.750,8.500 dashwid = 0.050 line dashed from 1.750,8.500 to 1.750,8.000 line dashed from 3.750,8.500 to 3.750,8.000 line from 1.750,8.000 to 1.750,7.000 to 3.750,7.000 to 3.750,8.000 line from 1.750,8.500 to 3.750,8.500 line from 1.750,8.000 to 3.750,8.000 box with .sw at (1.750,4.000) width 2.000 height 2.500 line from 1.750,4.250 to 3.750,4.250 line from 1.750,4.750 to 3.750,4.750 line from 1.750,5.250 to 3.750,5.250 line from 1.750,5.750 to 3.750,5.750 "Max memory" at 0.750,9.283 ljust "n Mb" at 0.750,9.033 ljust "Stack" at 2.500,9.033 ljust "(grows downwards)" at 2.250,8.783 ljust "1 Mb" at 0.750,7.033 ljust "Heap" at 2.500,7.533 ljust "(grows upwards)" at 2.250,7.283 ljust "Interrupt Vectors" at 2.250,4.096 ljust "640k" at 0.750,6.533 ljust "0" at 0.750,4.033 ljust "10000H" at 0.750,4.283 ljust "(loaded)" at 3.000,4.533 ljust "BSS" at 2.438,5.471 ljust "initially zero" at 2.812,5.471 ljust "Data" at 2.438,4.971 ljust "(loaded)" at 3.000,4.971 ljust "Code" at 2.438,4.533 ljust "unused" at 2.438,6.033 ljust

Goal of the overall boot procedure

- notice that no tiny model code will exist in the end
- all code is 32 bit and belongs to the core microkernel
- `first` and `second` will be overwritten

Overview of the boot stages

- three boot phases
  - *first* boot stage (boot sector, 1 sector, assembly language)
  - *second* boot stage (up to 14 sectors 8088 small mode Modula-2)
  - *LuK* (up to 512K of 32 bit code, Modula-2 and C)

- dashwid = 0.038 line dotted <-> from 3.000,7.750 to 6.000,7.750 line dotted <-> from 1.750,7.750 to 3.000,7.750 line dotted <-> from 1.000,7.750 to 1.750,7.750 dashwid = 0.050 box dashed with .sw at (3.000,8.000) width 3.000 height 1.250 line from 1.750,9.250 to 1.750,8.000 box with .sw at (1.000,8.000) width 2.000 height 1.250 "up to 512 K bytes" at 4.062,7.283 ljust "14*512" at 2.000,7.283 ljust "512" at 1.188,7.283 ljust "Modula-2 and C" at 3.875,8.533 ljust "Modula-2" at 2.000,8.533 ljust "8088" at 2.000,8.283 ljust "boot" at 2.000,8.783 ljust "Secondary" at 2.000,9.033 ljust "8088" at 1.125,8.283 ljust "512 bytes" at 1.125,8.533 ljust "sector" at 1.125,8.783 ljust "Boot" at 1.125,8.971 ljust "80586" at 4.000,8.283 ljust "LuK" at 3.750,8.783 ljust

LuK boot first

- (program first)

- 512 bytes boot sector is small! Just enough space to place an assembly language program which loads in a larger program
  - loads in *secondary* boot stage at 0x90200
  - jumps to 0x90200

- *secondary* boot stage (program second)
  - consists of limited amounts of assembly language
  - most of the code is written in Modula-2 but compiled to small mode 8088
  - the secondary stage may be up to 14 sectors in size (14 * 512 bytes)

Secondary boot stage

- purpose of *secondary* boot stage is to load in your *application*.third code as quickly as possible
  - it uses whole track reads whenever possible (fast)
  - the *primary* boot stage only used single sector loads (slow)
  - it loads in the LuK 32 bit executable (*application*.third) into location 0x10000
  - collects vital statistics about the PC (how much memory the PC contains and where video memory starts)
  - saves this information
  - turns the floppy disk motor off

- finally *second* puts the microprocessor into 32 bit mode and calls *application*.third

Boot phase in more detail

- *how* do you put LuK in the right place?
  - tip, think backwards

- start with the final position you desire
  - and consider how you can achieve it
  - draw memory maps of the different LuK bootstage intermediate positions

Final memory map for LuK

- line from 1.750,8.500 to 1.750,9.500 to 3.750,9.500 to 3.750,8.500 dashwid = 0.050 line dashed from 1.750,8.500 to 1.750,8.000 line dashed from 3.750,8.500 to 3.750,8.000 line from 1.750,8.000 to 1.750,7.000 to 3.750,7.000 to 3.750,8.000 line from 1.750,8.500 to 3.750,8.500 line from 1.750,8.000 to 3.750,8.000 box with .sw at (1.750,4.000) width 2.000 height 2.500 line from 1.750,4.250 to 3.750,4.250 line from 1.750,4.750 to 3.750,4.750 line from 1.750,5.250 to 3.750,5.250 line from 1.750,5.750 to 3.750,5.750 "Max memory" at 0.750,9.283 ljust "n Mb" at 0.750,9.033 ljust "Stack" at 2.500,9.033 ljust "(grows downwards)" at 2.250,8.783 ljust "1 Mb" at 0.750,7.033 ljust "Heap" at 2.500,7.533 ljust "(grows upwards)" at 2.250,7.283 ljust "Interrupt Vectors" at 2.250,4.096 ljust "640k" at 0.750,6.533 ljust "0" at 0.750,4.033 ljust "10000H" at 0.750,4.283 ljust "(loaded)" at 3.000,4.533 ljust "BSS" at 2.438,5.471 ljust "initially zero" at 2.812,5.471 ljust "Data" at 2.438,4.971 ljust "(loaded)" at 3.000,4.971 ljust "Code" at 2.438,4.533 ljust "unused" at 2.438,6.033 ljust

Second memory map for LuK

- line from 1.750,8.500 to 1.750,9.500 to 3.750,9.500 to 3.750,8.500 dashwid = 0.050 line dashed from 1.750,8.500 to 1.750,8.000 line dashed from 3.750,8.500 to 3.750,8.000 line from 1.750,8.000 to 1.750,7.000 to 3.750,7.000 to 3.750,8.000 line from 1.750,8.500 to 3.750,8.500 line from 1.750,8.000 to 3.750,8.000 box with .sw at (1.750,4.000) width 2.000 height 2.500 line from 1.750,4.250 to 3.750,4.250 line from 1.750,6.000 to 3.750,6.000 "Max memory" at 0.750,9.283 ljust "n Mb" at 0.750,9.033 ljust "1 Mb" at 0.750,7.033 ljust "Interrupt Vectors" at 2.250,4.096 ljust "640k" at 0.750,6.533 ljust "0" at 0.750,4.033 ljust "10000H" at 0.750,4.283 ljust "90200H" at 0.750,6.033 ljust "second.mod" at 2.250,6.283 ljust "(code+data+stack)" at 2.250,6.096 ljust

Boot memory map for LuK

- line from 1.750,8.500 to 1.750,9.500 to 3.750,9.500 to 3.750,8.500 dashwid = 0.050 line dashed from 1.750,8.500 to 1.750,8.000 line dashed from 3.750,8.500 to 3.750,8.000 line from 1.750,8.000 to 1.750,7.000 to 3.750,7.000 to 3.750,8.000 line from 1.750,8.500 to 3.750,8.500 line from 1.750,8.000 to 3.750,8.000 box with .sw at (1.750,4.000) width 2.000 height 2.500 line from 1.750,6.000 to 3.750,6.000 line from 1.750,5.750 to 3.750,5.750 line from 1.750,4.125 to 3.750,4.125 "Max memory" at 0.750,9.283 ljust "n Mb" at 0.750,9.033 ljust "1 Mb" at 0.750,7.033 ljust "640k" at 0.750,6.533 ljust "0" at 0.750,4.033 ljust "90200H" at 0.750,6.033 ljust "90000H" at 0.750,5.783 ljust "copy of boot sector" at 2.250,5.783 ljust "7c00" at 0.750,4.158 ljust "initial boot sector (BIOS)" at 2.000,4.283 ljust

Conclusion

- this technique works
- it is not the most efficient, it might be possible to make first perform the actions of second
- however the approach presented here allows us to:
  - execute high level language code sooner
- some of the older limits should be removed now that booting floppy disks is no longer needed
- maybe it would be sensible to move LuK to start at 1MB upwards
  - would allow LuK to expand