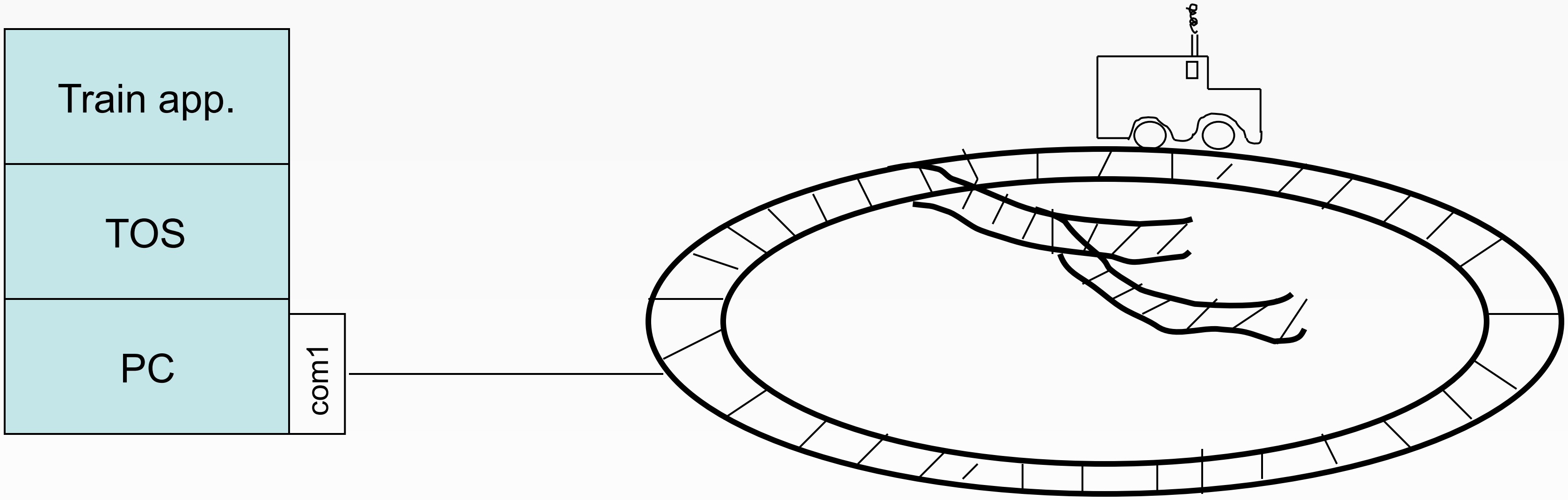
Welcome to CSC 720

- In this course you will:
  - apply your knowledge of Operating Systems to build your own OS
  - learn about the PC architecture
  - learn the basics of the Intel x86 CPU
  - learn how to program I/O devices
- In short: at the end of this term, you will have written your own Operating System that will run on any PC!

TOS Project

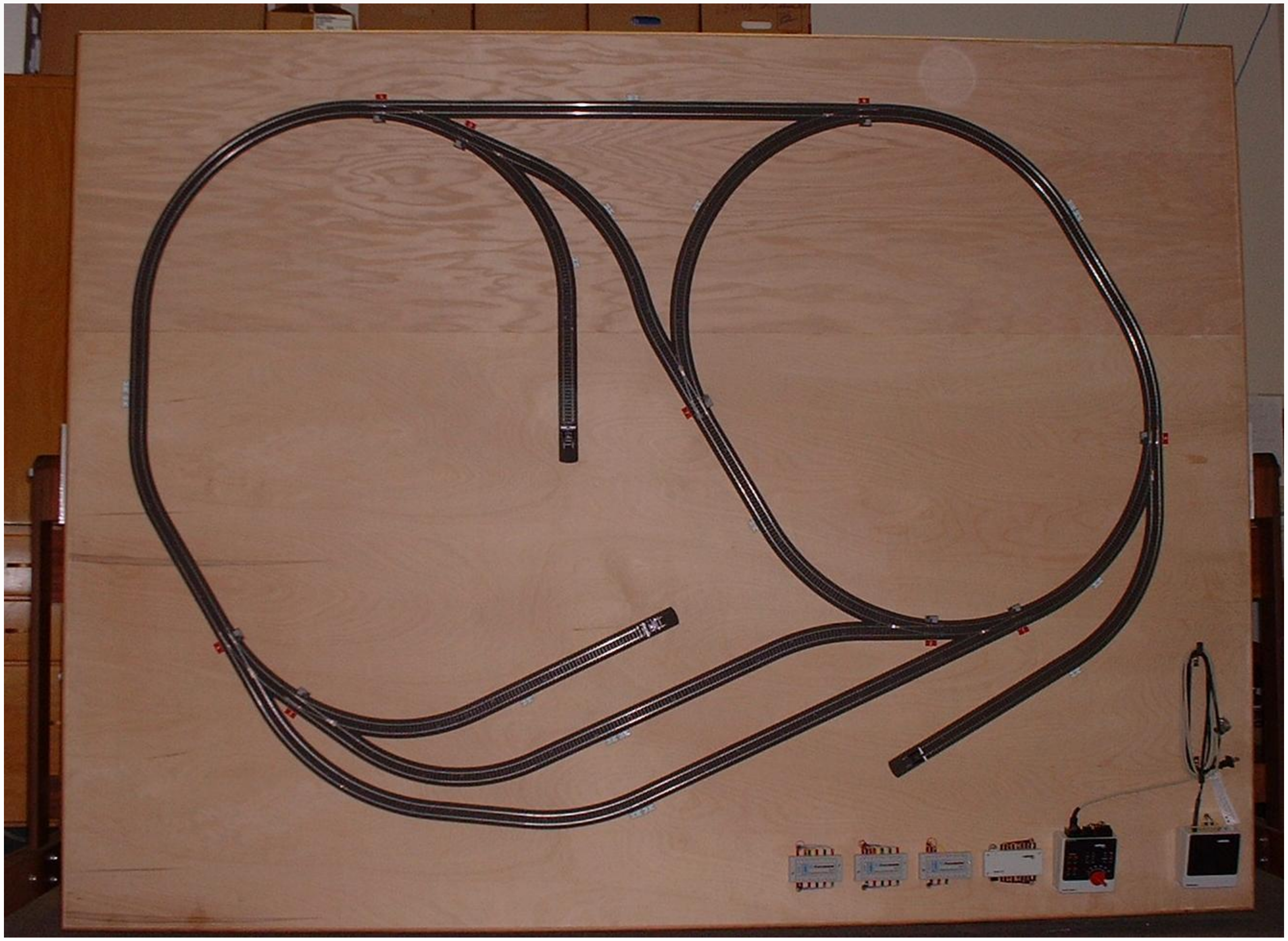
- TOS: Train (as in ‘training’ or ‘train’) Operating System
- You will implement all the key pieces of TOS:
  - process management
  - I/O
  - IPC
  - etc…
- This is a lot of programming! Think twice before taking this class concurrently with other programming intensive courses!

Train Setup



- Train application runs on top of TOS
- TOS implements various operating system functions including a serial line device driver
- Serial line (com1) of the PC is connected to the train
- Commands that control the train are sent via the serial line

Model Train



Course Details

Instructor: Arno Puder
E-mail: arno@sfsu.edu
Office Hours: Th 17:00 – 18:00 (TH 909)

Course homepage:
http://pear.sfsu.edu/csc720/

Course Details

Prerequisites Grade B or better in CSC 415 or consent of instructor

Optional book A. Tanenbaum, A. Woodhull: *“Operating Systems Design and Implementation”*, 3rd edition, Prentice Hall.

Syllabus Course Overview:

- I/O Structures
- Process Management (CPU scheduling, synchronization, threads)
- Memory management and virtual memory

The class is accompanied by an extensive programming project where the students will have to write their own operating system.

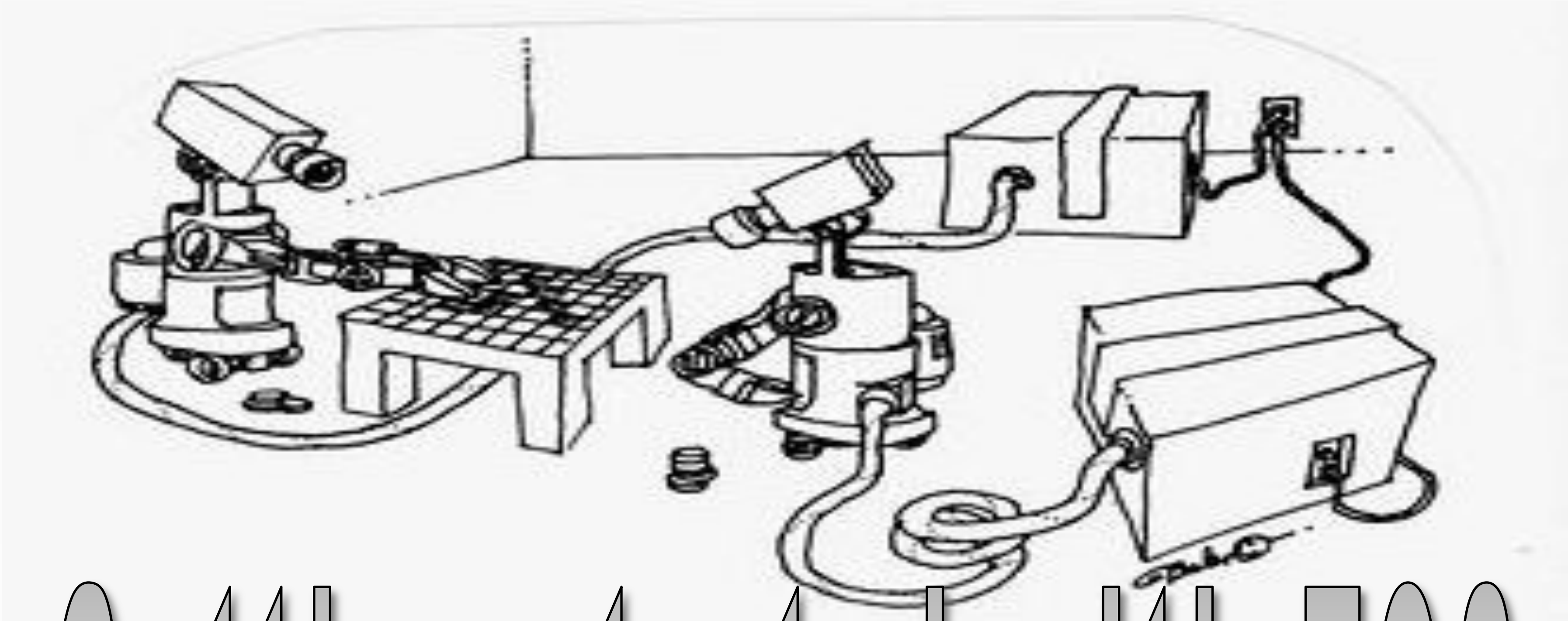
Grading

Grading

Quizzes	**60%**
Programming project	**40%**

Grading Scale

Total	Grade
> 90%	A
> 85%	A-
> 80%	B+
> 75%	B
> 70%	B-
> 65%	C+
> 60%	C
> 55%	C-
> 50%	D+
> 45%	D
> 40%	D-
<= 40%	F

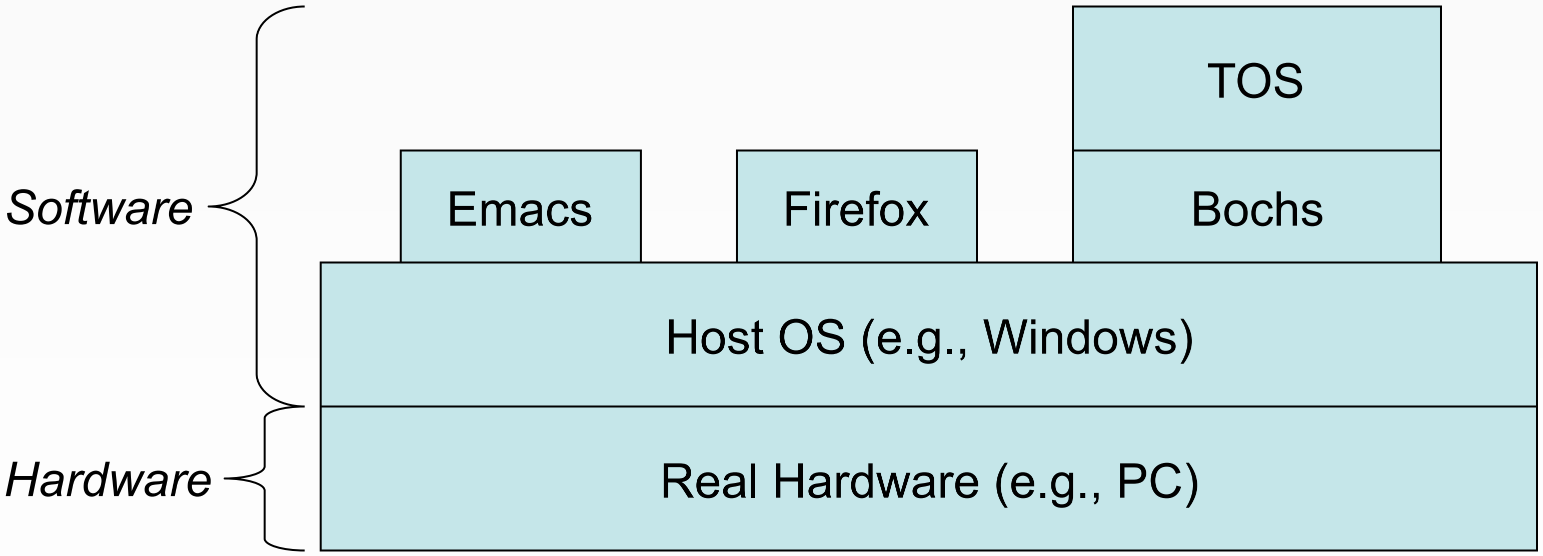


Getting started with TOS

Overview of TOS

- TOS = Train Operating System (Train == Training || Model Train ☺)
- An educational operating system running on a PC
- Written in C (99%) and x86 assembly (1%)
- All the files and Makefiles are provided for you
- You just need to implement the core functions.

Running TOS in Bochs



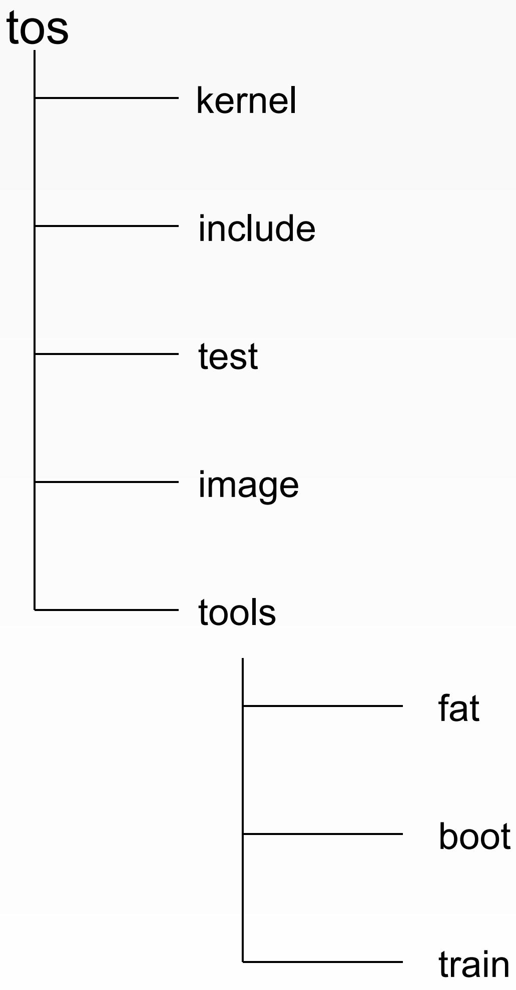
Virtual Hardware

- How does Bochs emulate hardware of the guest OS?
- The ‘virtual’ Hardware is mapped to resources on the Host OS.
- E.g. the floppy drive A: of the guest OS is mapped to a regular file located in the filesystem of the host OS.
- This mapping between virtual and real resources is done with the configuration file `~/.bochsrc` which contains the line:

  ```
  floppya: 1_44=image/disk_image
  ```

- This means that the drive A: of the guest OS is mapped to a 1.44 MB file located in `image/disk_image`
- Whenever the guest OS accesses A:, the operation is redirected by Bochs to this file.

Directory structure of TOS



tos		
kernel		The main sources of TOS
include		Header file for TOS
test		Test programs
image		Contains the floppy image from which to boot
tools		
	fat	FAT-tools (fatformat, fatdir, fatcopy, fatdel, rawwrite.exe)
	boot	TOS boot loader
	train	Train simulation

Files in ~/tos/kernel

Files	Contents
assert.c	Assert-function. Does not need to be edited.
com.c	COMs interface.
dispatch.c	Dispatcher and scheduler.
intr.c	Interrupt handling.
main.c	Contains main entry point kernel_main()
null.c	Null process.
train.c	Train application.
demo.c	Empty. Does not need to be edited.
inout.c	Low level input/output routines for COM1.
ipc.c	Inter-process communications.
mem.c	Memory access functions.
pacman.c	PacMan implementation.
process.c	Process management.
timer.c	Timer interrupt handling.
keyb.c	Keyboard interface. Does not need to be edited.
shell.c	Mini-shell for typing in commands. Can be extended for own commands.
window.c	Mini-windowing system for text-mode.

Recompiling TOS

- The only files you will be editing are `tos/kernel/*.c`
- Use your preferred editor to make the changes
- Two ways to compile TOS, both from the main tos directory:
  - `make tests` (build a testing kernel)
  - `make` (build a regular kernel)
- For now, always build a test kernel -- we’ll build “regular” kernels later

Recompiling TOS

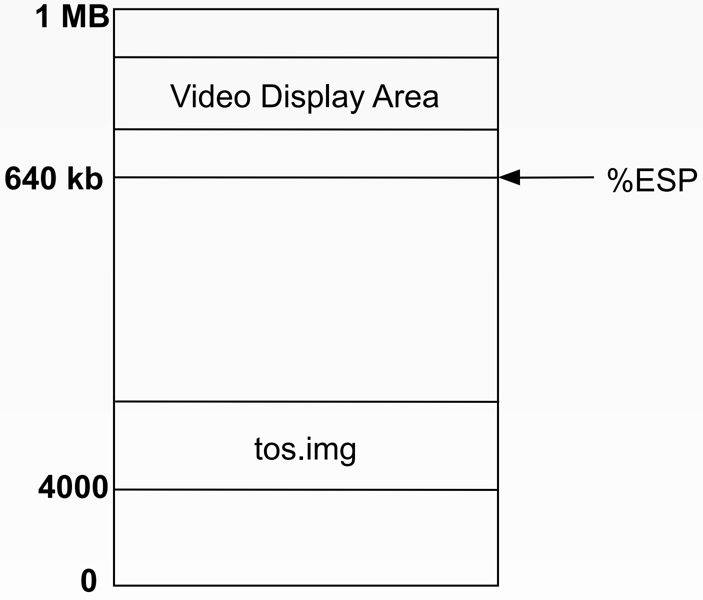
- No need to write or edit Makefiles
- If the build is successful, the new boot image will be located in `tos/image/disk_image`
- Other useful make targets:
  - `make clean` removes all object files and executables
  - `make clean-kernel` removes just kernel-specific object files

Some Guidelines

- Only modify C-files in `tos/kernel`
- No need to change Makefiles or C-header files.
- You can (and are encouraged to) look at and understand other files.
- You can not use any C-library functions: no `malloc()`, no `free()`!! (remember, we don't have an OS yet)

TOS Boot Sequence

- Sequence of events during boot:
  - PC is turned on (i.e. Bochs is executed)
  - PC loads the boot sector (the first sector of the floppy disk)
  - The boot-loader loads TOS at address 4000, initializes %ESP just below 640 kB and then jumps to `kernel_main()`
- The entry point of TOS is function `void kernel_main()` in file `tos/kernel/main.c` or `tos/test/run_tests.c`


1 MB
Video Display Area
640 kb
%ESP
tos.img
4000
0