Implementation lecture 1

Languages for OS implementation

Device control

Choosing a language:

- can it do it?
- does it help?
    - data structures we need
    - control structures we need
    - provide level of safety
    - (does our code do what we think it should?)
- does it hinder?
- what about documentation?

Data structures

An operating system is a large collection of tables joined together by little bits of code.

tables, queues, messages, semaphores, files

Program structures

communication

thread control

control of critical sections

and synchronisation

exception handling

interrupt handling

Programming environments

Some of the requirements we listed above can be helped by the programming environment rather than just the programming language.

It could be argued that UNIX with its standard collection of editors, compiler compilers, code checkers (lint), source code control systems etc, etc makes an ideal environment in which to develop operating systems.

(It has been the standard for several years, after all.)

What language do you know which you wouldn't mind writing an operating system with?

Common OS programming languages

Assembly language

- anything which can be done

Forth

- easily extensible
- incredibly compact - embedded systems

C/C++

- very expressive
- excellent compilers

Oberon

- safely extensible

ADA

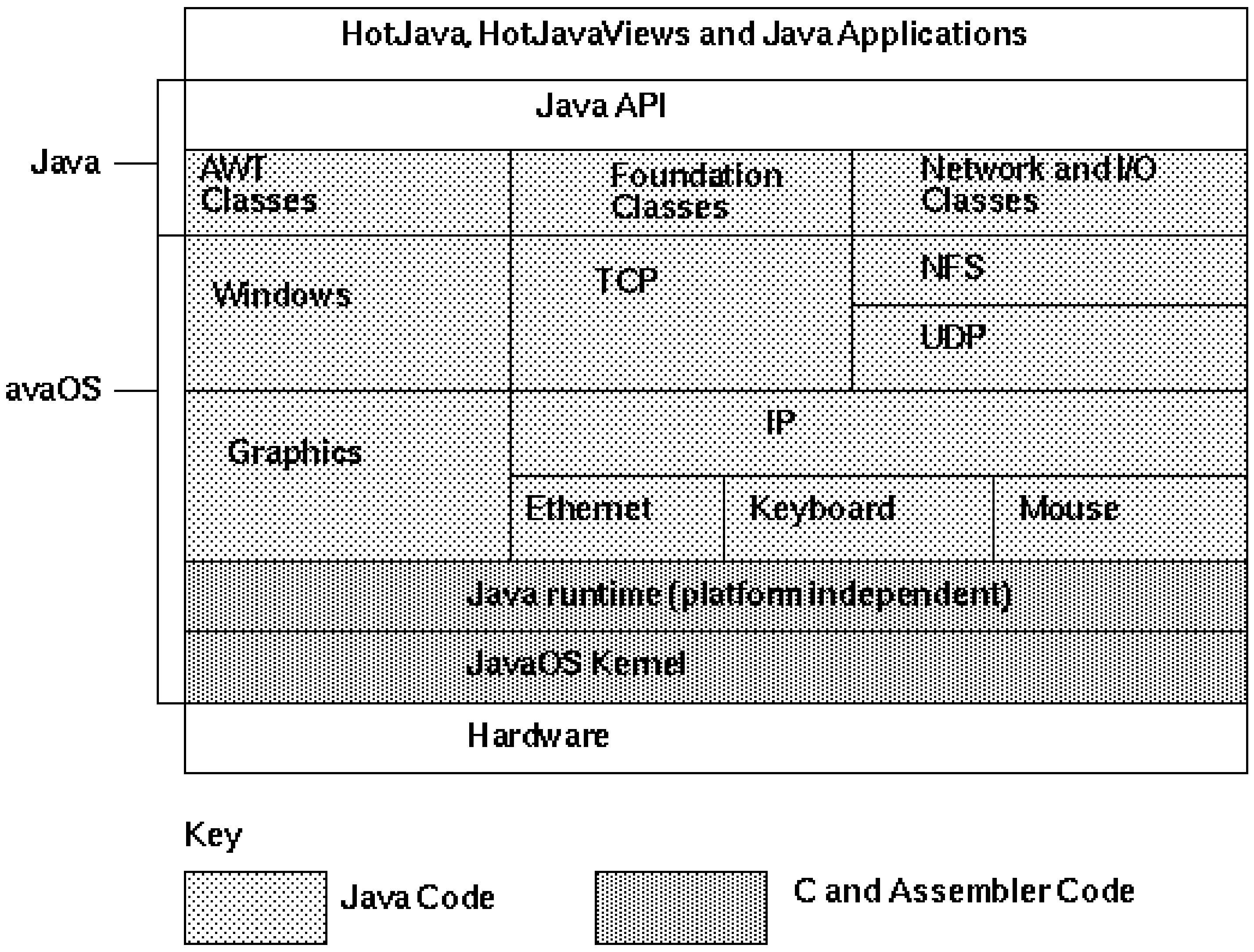
- real-time features

Why stop with one?

OOOS - object oriented operating systems

Java?

JavaOS



The kernel for Java contains the low-level functions required by the Java Virtual Machine.

BootingExceptions	Threads
Memory Management	Monitors
File System	Timing
Native Code Library Management	
Interrupts	DMA
Debugging	
Miscellaneous Platform Control	

Why Implement JavaOS in Java?

- Java is portable, easy to write, and easy to debug
- Runtime checking provides a robust system
- Services can be extended at runtime
- Single address space enables high performance with full security
- Minimal overhead

Devices

Scanner, mouse, stereo sound system, virtual reality goggles, various disk / tape drives, windows, keyboard etc.

Very different:

- speed
  - few chars/sec to many millions chars/sec
- unit of transfer
  - bits, bytes, records, blocks
- data representation
  - e.g. different encodings on tape
  - protocols of network cards

- operations
    - input or output certainly but also
    - rewind a mouse?
    - recalibrate a window?
- control
    - different signals to get/put data
    - control and status operations
- error conditions

<u>We would like some uniform method to deal with devices.</u>

Device independence.

All deal with streams of data

<u>Device table</u>

It makes sense to store device information in a table.

Each device entry (device descriptor), leads to routines which control the device.

Typically reading and writing, initializing or opening the device, shutting down or closing

Also status, what it is doing (for what process?)

Request queue.

Typically a call to open a file or a device makes

- a connection between the file information block and the device descriptor

Need the device name, and possibly volume (or file name) in the open call.

May need other information to set the device up in a particular way.

  what size buffers, access method etc

In UNIX devices are included in the file system

/dev/tty, /dev/cdrom, /dev/modem

The file attributes include information saying it is a device and what type of device.

Reading or writing commands make sense for all devices.

Other control information is specific and is nornally handled with some catch-all command - DOIO.

UNIX ioctl