OS Structure

COMP755 Advanced OS

"Today's scientists have substituted mathematics for experiments, and they wander off through equation after equation, and eventually build a structure which has no relation to reality."

Nikola Tesla

Programming Assignment

- A simple programming assignment has been posted on Blackboard under assignments
- You can write the programs in C++, Java or another OO programming language
- Due by midnight on Friday, August 30

Short Term Schedule

	Wednesday, August 21 Introduction read chapter 1
Monday, August 26 OS structure read chapter 2	Wednesday, August 28 Concurrent Programming read chapter 6
Monday, September 2 *Labor Day Holiday* *(no class)*	Wednesday, September 4 Concurrent Programming
Monday, September 9 Concurrent Programming	Wednesday, September 11 Deadlock & thread implementation read chapter 7
Monday, September 16 CPU scheduling & review read chapter 5	Wednesday, September 18 Exam 1

The OS is a BIG program

- The operating system is a large program composed of many modules.
- Writing and updating an OS requires careful software engineering techniques.
- Errors in the OS can cause the whole system to fail.

System Implementation

- Once written in assembly language, operating systems are now written in higher-level languages
- Code written in a high-level language:
  - Can be written faster
  - Is more compact.
  - Is easier to understand and debug
- An operating system is far easier to *port* (move to some other hardware) if it is written in a high-level language

System Design Goals

- **User goals –** operating system should be convenient to use, easy to learn, reliable, safe, and fast
- **System goals –** operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

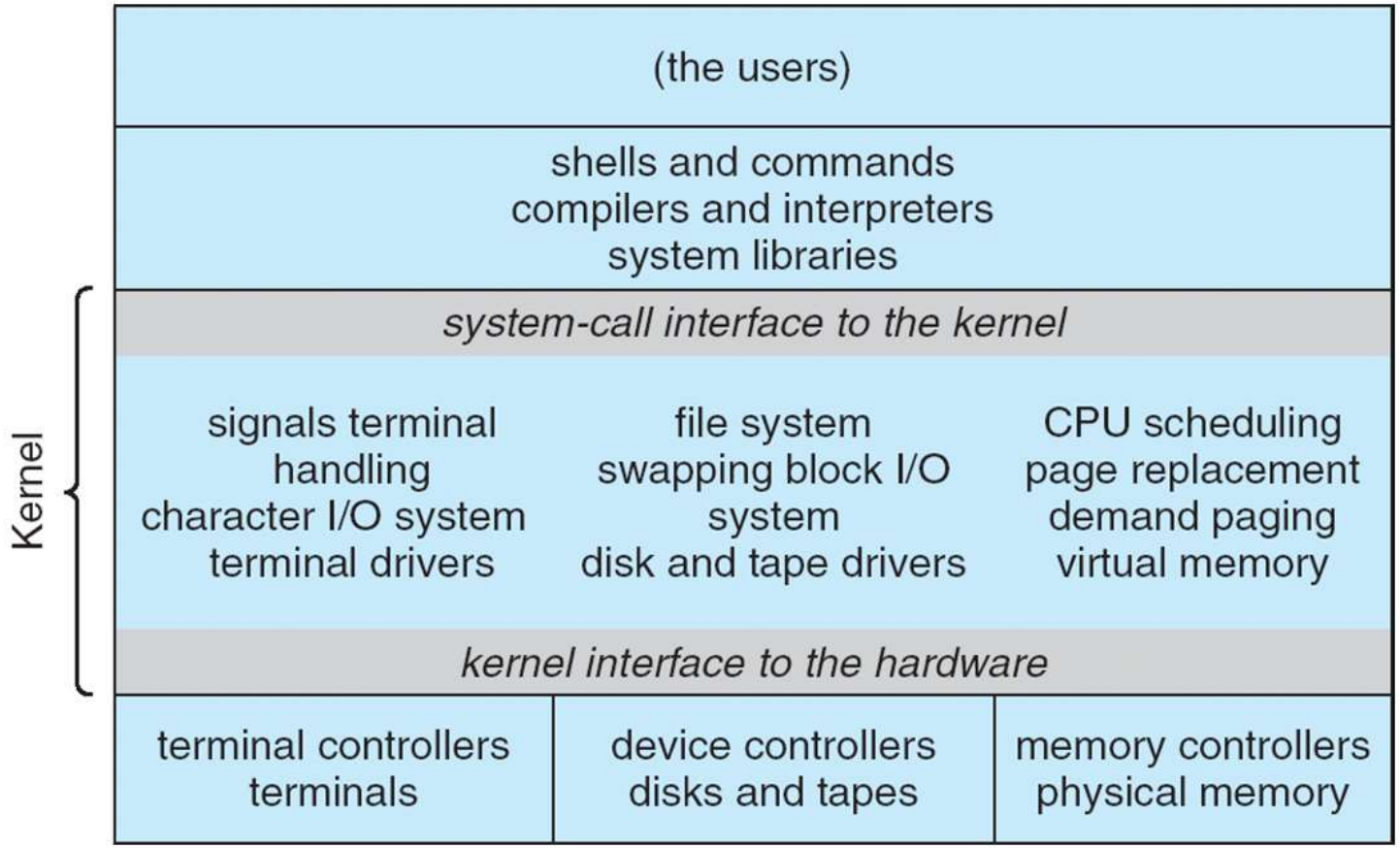
System Components

- Process management
- Memory management
- File management
- I/O management
- Security manager
- Shell

OS kernel

- The kernel of the OS is that portion providing privileged high priority functions.
  - Dispatcher
  - Interrupt handling
  - I/O control
  - Memory management
- Some parts of the operating system do not reside in the kernel.

Traditional UNIX System Structure



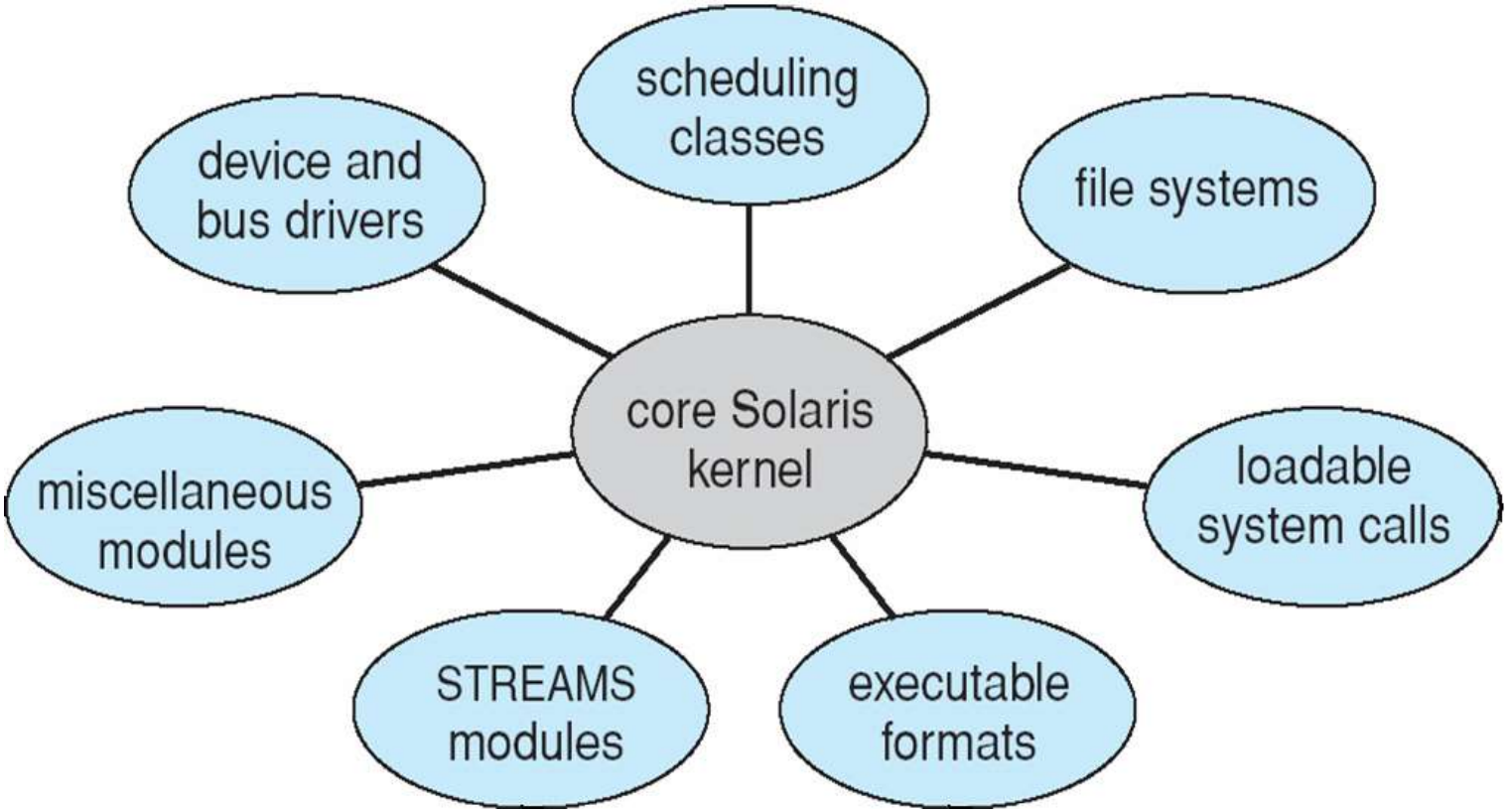
Microkernel System Structure

- Moves as much from the kernel into "*user*" space
- Communication takes place between user modules using message passing

Microkernel System Structure

- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

Solaris Modular Approach



Shell Interface Program

- The shell is the user interface.
- It is part of the OS
- Runs at the user level *(not kernel level)*
- The shell reads the user's commands and then causes the applications to be loaded and executed.

Shell Features

- Reads user input and executes programs.
- Provides the prompt.
- Implements redirection
- Implements parallel execution with &
- May implement simple commands
- Shells may provide a graphical user interface.

Human Interface

- The shell is the part of the OS that is most important to most users.
- Considerable effort has gone into trying to make the interface intuitive and easy to use

Popular Shells

- command.com in DOS
- C shell in Unix
- Korn shell in Unix
- Borne shell in Unix
- explorer.exe – Windows user interface
- Cygwin – Unix shell for Windows

Shell Outline

```
do forever {
      print prompt;
      read command;
      if (fork()==0) { // child process
            use exec() to load program;
      }
      wait for child process to terminate;
}
```

execv function

int _execv(const char * *cmdname*,
const char * *argv[ ]* **);**

where:

- *cmdname* is the name of the program file.
- *argv* is an array of pointer to strings containing the parameters. By convention argv[0] = cmdname; The last argv entry must be NULL.

The execv function causes the specified program to overlay the calling program. This function does not return if successful.

Simple C Program

```
int main ( int argc, char *argv[] )
{
     ...
     return 5;
}
```

The **`argv`** array passed to the main function when you start a C programs is the **`argv`** array passed to **`execv`** by the shell.

Standard I/O Streams

When a program is started, it has three I/O streams open:

0 stdin – Standard input, usually keyboard

1 stdout – Standard output, usually screen

2 stderr – Standard error, usually screen

Redirection

Standard I/O streams can be redirected from the command line

```
ls > myfile.txt
myprog < usualstuff.txt
cc nogood.c >&2 Errmsg.Txt
```

Redirection Implementation

- The freopen(“filename”, mode, *stream) function will direct output to stream to the specified filename.
- The shell can redirect stdin or stdout to the filename specified on the command line.
- The shell forks a new process, redirects stdin and/or stdout then does the exec()

Multiple Processes

- If you put a “&” at the end of a command, the shell will not wait for the process to terminate before printing the next prompt.
- You can run a process “in the background” by putting a “&” at the end of the line.
- You can put a “&” between commands to execute them in parallel.

Example Multiple Processes

```
makefile bigthing  &

xterm &

ls & cc myprog.c

ls & cc myprog.c & ps
```

Shell Outline

```
do forever {
      print prompt;
      read command;
      if (fork()==0) { // child process
            use exec() to load program;
      }
      if (no “&”)
            wait for child to terminate;
}
```

Scripts

- The shell can execute script files to produce commands for the shell.
- Scripts are a program whose output is a series of commands to the shell.
- DOS batch files are a simple example of scripting.
- Unix scripts can be much more elaborate.

Utilities

- An operating system is more than just the kernel, there are many utilities.
- The utilities are just user applications.
- Different user interfaces require different utilities

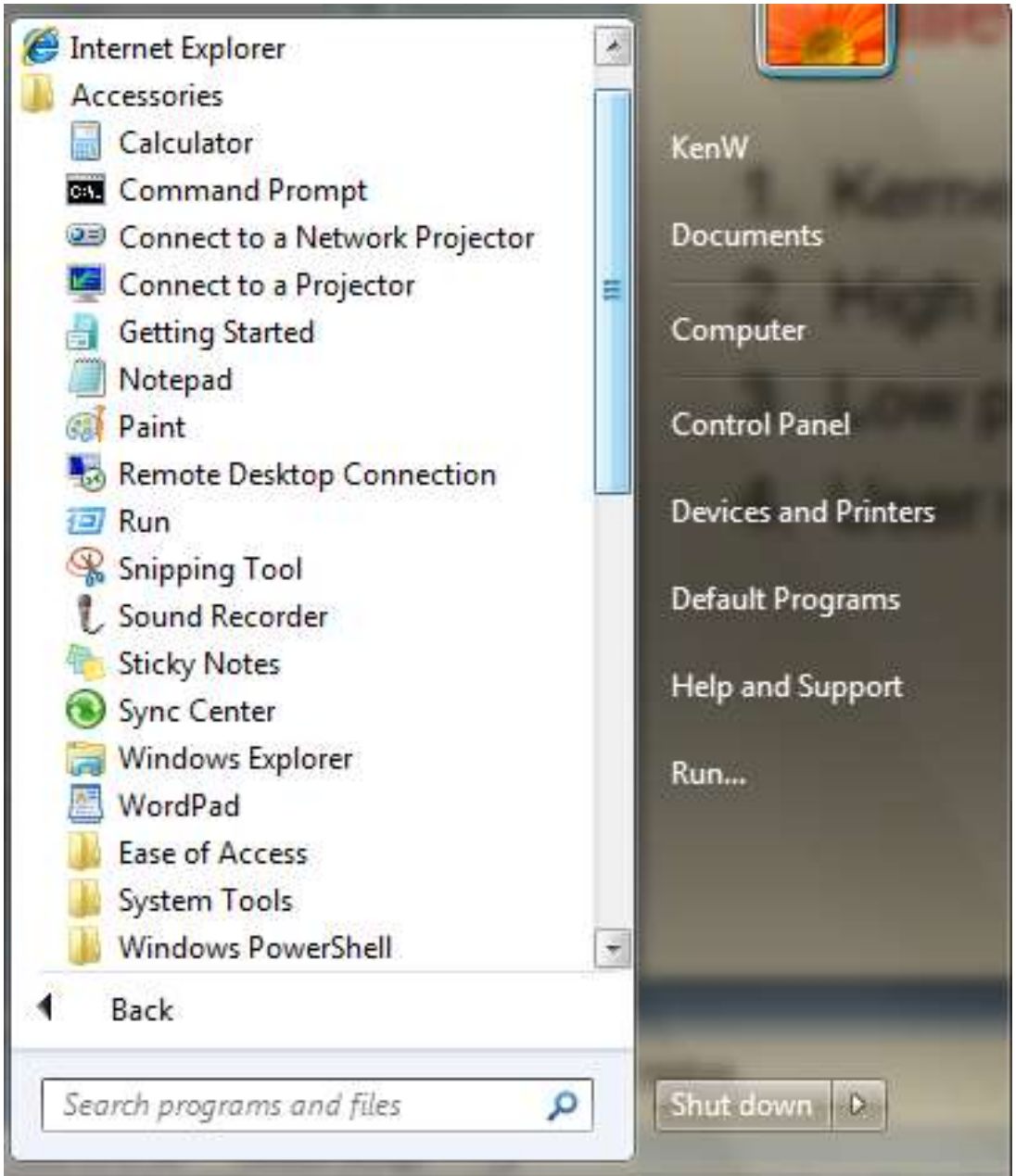
DOS Commands

attrib
cd or chdir
chkdsk
cls
copy
ctty
defrag
del or erase
deltree
dir
echo
fdisk
find
format
help
join
label
md or mkdir
mem
more
move
rd or rmdir
ren
scandisk
set
setver
share
smartdrive
sort
sys
time and date
tree
type
xcopy

Unix Commands

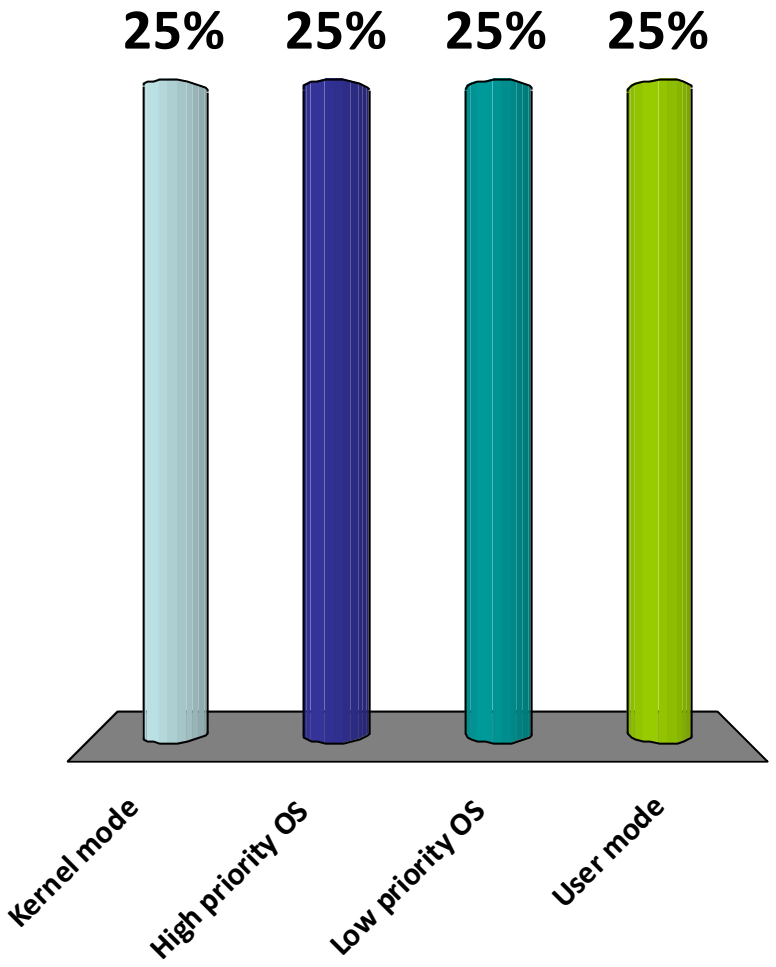
```
at
cat
cd
chmod
chown
cmp
compres
cp
date
du
env
grep
head
kill
lp
make
man
mkdir
more
mv
nice
ps
rm
rmdir
sort
tail
touch
uncompress
who
```

Windows Utilities



In Windows the user interface is called the Explorer. It runs in

1. Kernel mode
2. High priority OS
3. Low priority OS
4. User mode



Process Management

- A *process* is a program in execution
  - A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task
- The operating system is responsible for the following activities in connection with process management
  - creation, deletion
  - suspension, resumption
  - synchronization, communication

Main-Memory Management

- *Memory* is a large array of words or bytes, each with its own address
- RAM is a volatile storage device. It loses its contents without power.
- The operating system is responsible for the following activities in connection with memory management
  - Keep track of which parts of memory are currently being used and by whom
  - Decide which processes to load when memory space becomes available
  - Allocate and deallocate memory space as needed

File Management

- The operating system is responsible for the following activities in connections with file management:
  - File creation and deletion
  - Directory creation and deletion
  - Support of primitives for manipulating files and directories
  - Mapping files onto *secondary storage*

Secondary-Storage Management

- Since RAM is volatile and too small to accommodate all data and programs permanently, the computer system must provide *secondary storage* to back up main memory
- Most modern computer systems use disks *(or simulated disks)* as the primary storage medium, for both programs and data
- The operating system is responsible for:
  - Free space management
  - Storage allocation
  - Disk scheduling

I/O System Management

- The I/O system consists of:
  - A buffer-caching system
  - A general device-driver interface
  - Drivers for specific hardware devices

Security manager

- The Security manager provides a mechanism for controlling access by programs, processes, or users to both system and user resources
- The protection mechanism must:
  - distinguish between authorized and unauthorized usage
  - specify the controls to be imposed
  - provide a means of enforcement

Priorities

- The many different parts of the OS may run at different priorities
  - Non-process
  - High priority
  - Lower priority
  - User level

OS Services

- Program Execution
- I/O control
- Communications
- Error detection
- Resource allocation
- Protection

Privileged Instructions

- Most machines allow certain instructions only in "OS mode"
- When a user wants to do something that they cannot do directly, the user program requests the OS to do the task
- The Intel Pentium supports 4 levels of privilege.

Supervisor Calls

- Supervisor Calls (SVC) are a means of making a function call to an OS service.
- The normal function call mechanism will not work since you have to change to the OS environment.
- An SVC or INT call generates an interrupt. The OS gets control and executes the desired function.

Virtual Machines

- A virtual machine provides an interface *identical* to the underlying bare hardware
- Programs running in a virtual machine appear to be running on their own computer
- An OS can run in a virtual machine and run application programs
- Multiple virtual machines can run on the same computer at the same time

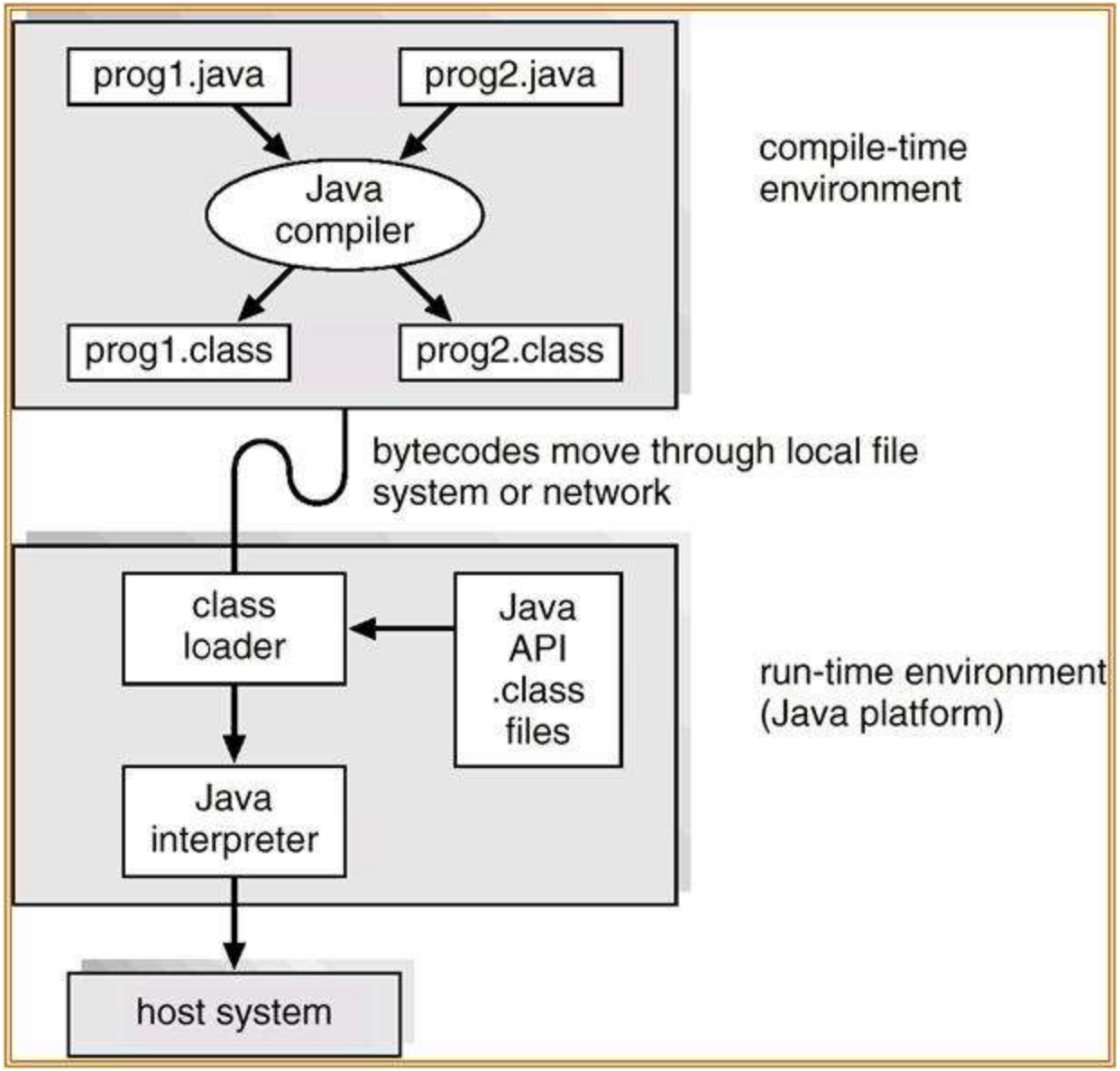
Virtual Machines (Cont.)

- The resources of the physical computer are shared to create the virtual machines
  - CPU scheduling can create the appearance that users have their own processor
  - A virtual disk drive might be simulated as a file on the host OS

Java Virtual Machine

- Compiled Java programs are platform-neutral bytecodes executed by a Java Virtual Machine (JVM)
- JVM consists of
  - Class loader
  - Class verifier
  - Runtime interpreter
- Just-In-Time (JIT) compilers increase performance

Java Environment



System Generation (SYSGEN)

- Older OS had to be recompiled whenever you made a configuration change.
- Most modern OS contain code for every possible configuration. Code is dynamically added when new hardware is found.