OS Structures

These slides were compiled from the OSC textbook slides (Silberschatz, Galvin, and Gagne) and the instructor’s class materials.

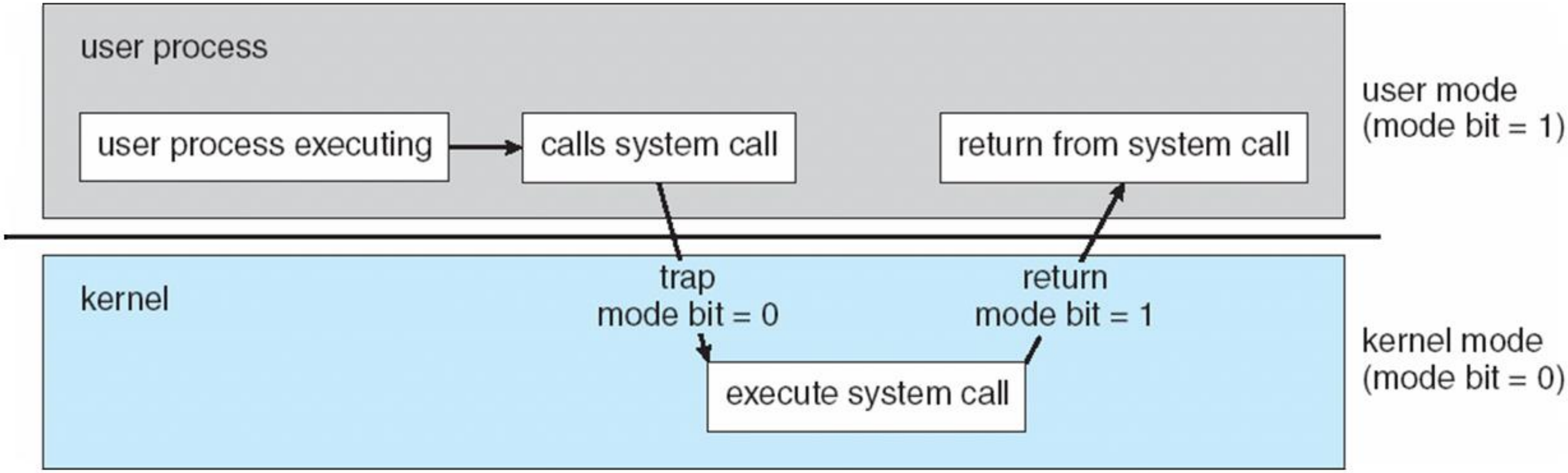
OS Dual-Mode Operations

Provide hardware support to differentiate between at least two modes of operations.

1. *User mode* – execution done on behalf of a user.
2. *Monitor mode* (also *supervisor mode, system mode, or Kernel mode*) – execution done on behalf of operating system.

Switching between two modes: transition from user -> kernel mode

- Device interrupts, hardware traps, system calls cause a trap to the kernel mode
- The operating system returns to the user mode after servicing requests.



user process

user process executing

calls system call

return from system call

user mode
(mode bit = 1)

kernel

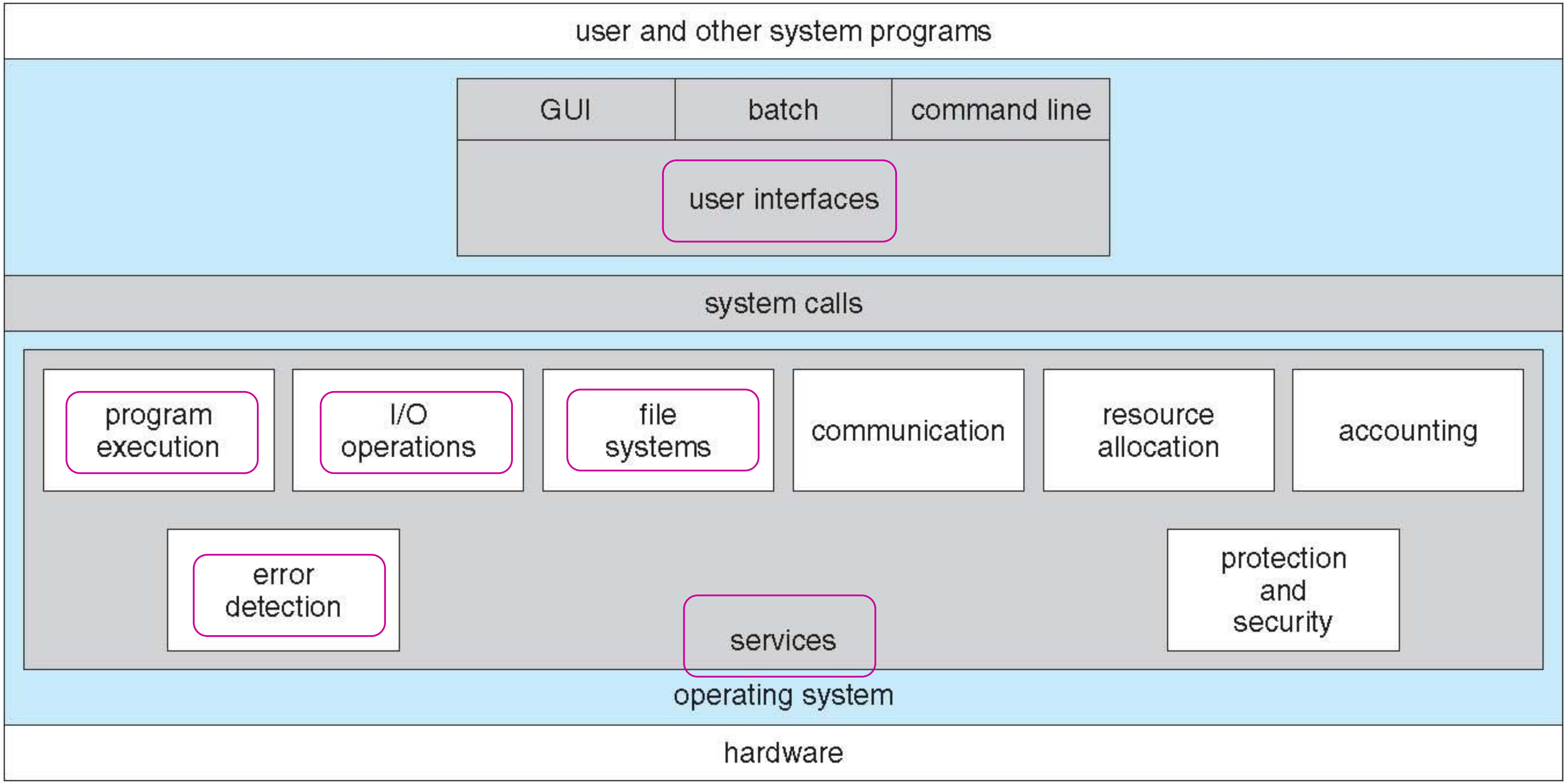
trap
mode bit = 0

return
mode bit = 1

execute system call

kernel mode
(mode bit = 0)

A view of OS Services



System Calls and OS “Managements”

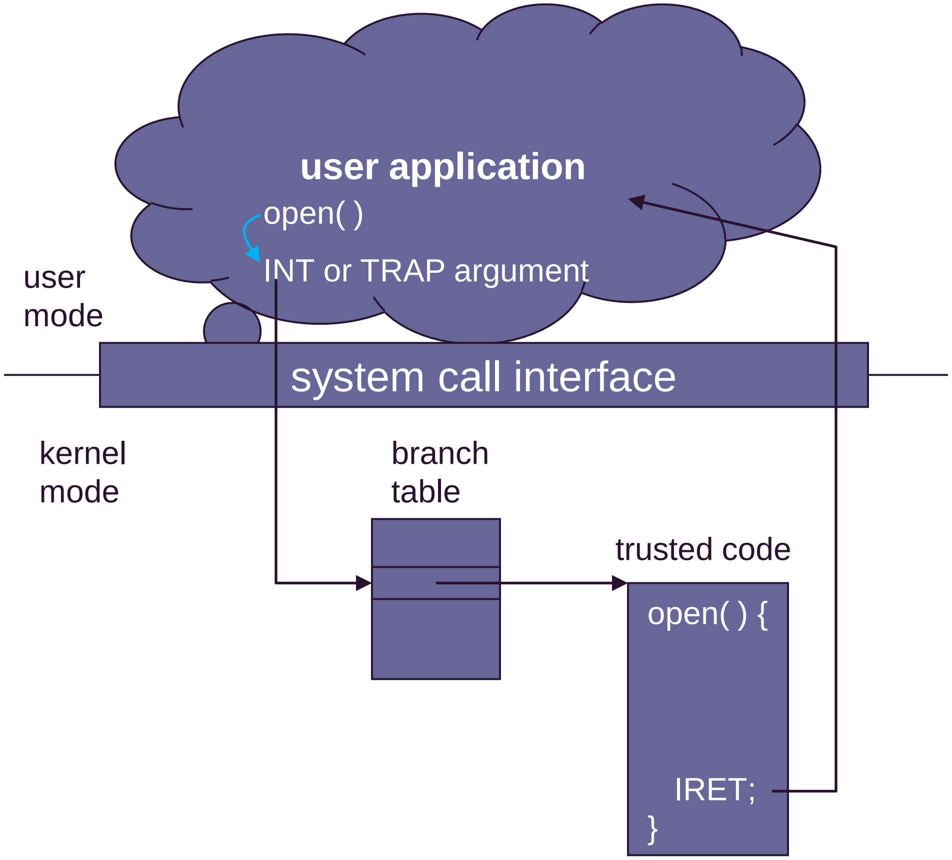
- Big picture
  - Process
  - Memory
  - File
  - Others

System Calls

- Programming interface to the services provided by the OS
- There is a number associated with each system call (used for indexing)
- Typically written in a high-level language (C or C++), accessed via high-level API rather than direct system call use
  - Implementation details hidden
  - Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (for all versions of UNIX), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

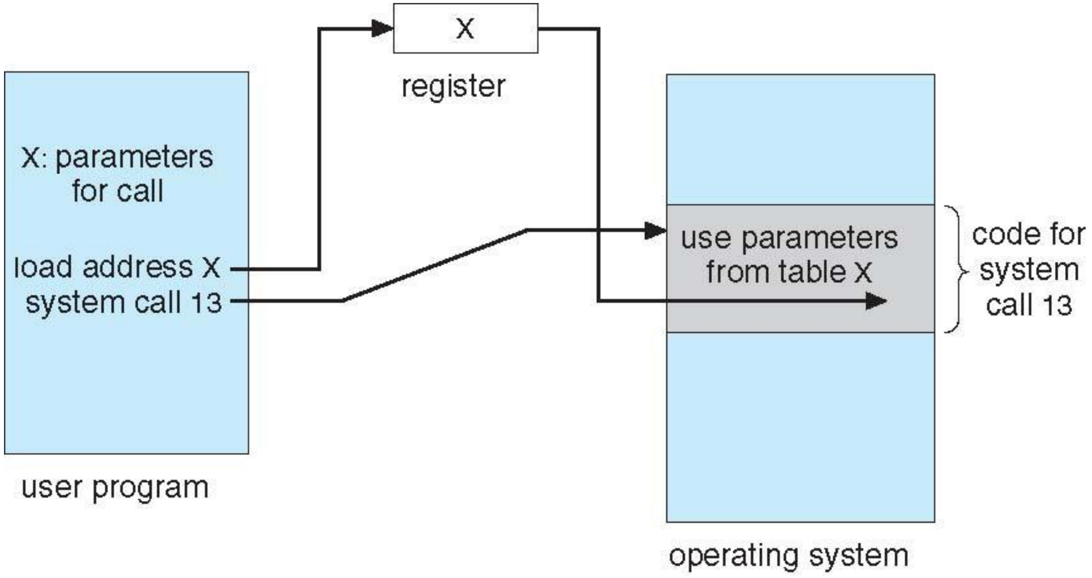
(Note that the system-call names used throughout this text are generic)

System Calls



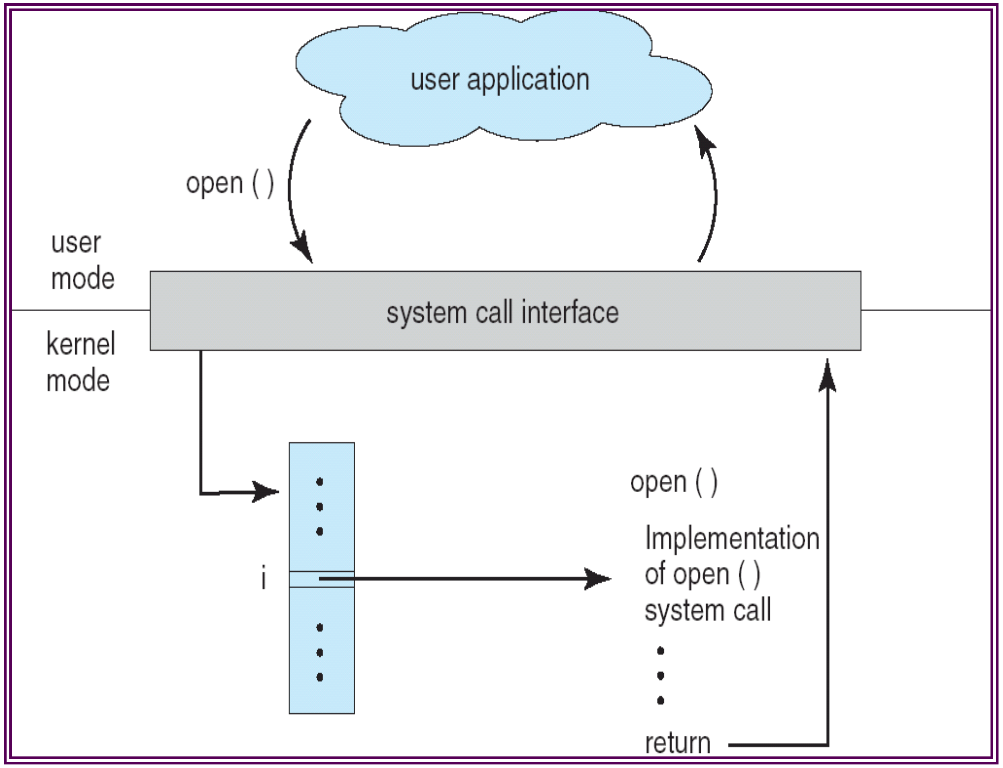
- Software interface to OS
  - A System function is compiled into “trap args”
  - “Trap” changes CPU mode from user to kernel.
  - CPU checks the OS branch table.
  - Jumps to the code pointed to by the table entry.
  - Work on the requested call.
  - IRET switches from kernel to user

System Calls (Cont'd)

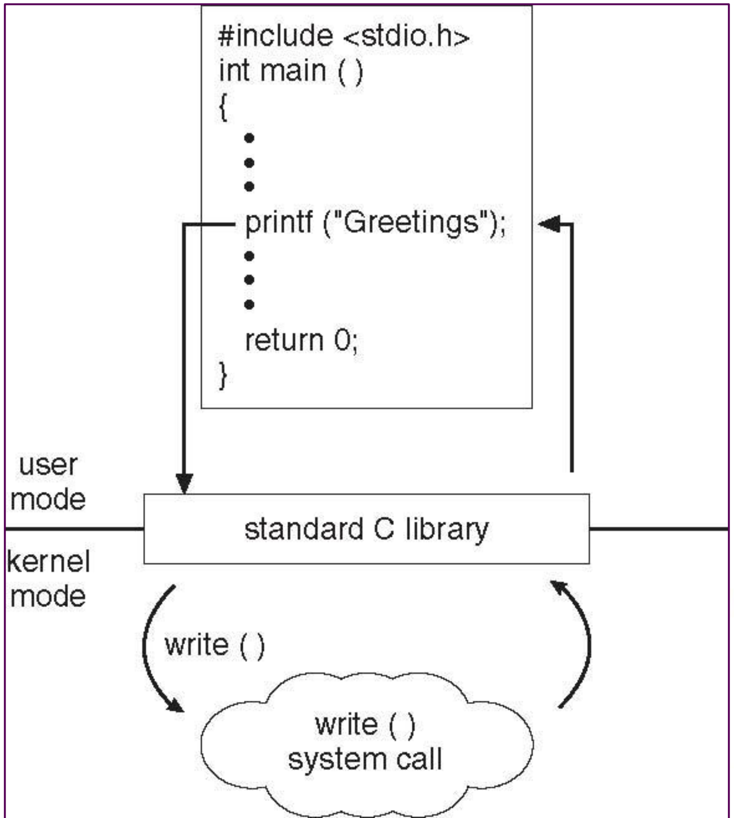


- When a user program executes a special instruction like trap
  - CPU recognizes it as a (software) interrupt.
  - The mode turns in kernel mode.
  - Control jumps to a given vector (e.g. 13)
  - The OS saves the user program status.
  - It then begins to handle the system call.
  - The OS resumes the registers.
  - It finally returns back to a user program

System Calls

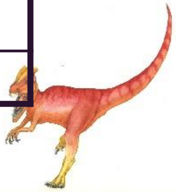


Example: C program invoking printf() library call, which calls write() system call


```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

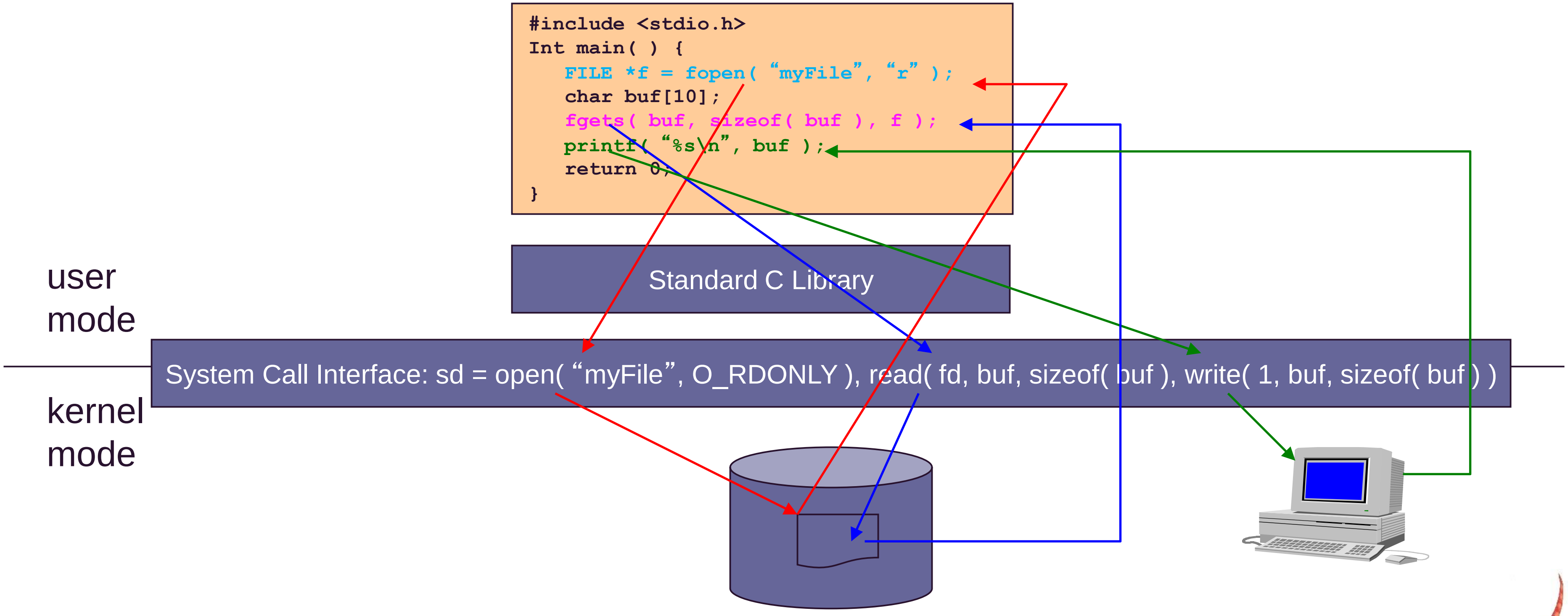
System Calls (Cont’d)

	Windows	Unix
Process Control	CreateProcess( )	fork( )
	ExitProcess( )	exit( )
	WaitForSingleObject( )	wait( )
File Manipulation	CreateFile( )	open( )
	ReadFile( )	read( )
	WriteFile( )	write( )
	CloseHandle( )	close( )
Device Manipulation	SetCosoleMode( )	ioctl( )
	ReadConsole( )	read( )
	WriteConsole( )	write( )
Information Maintenance	GetCurrentProcessID( )	getpid( )
	SetTimer( )	aAlarm( )
	Sleep( )	sSleep( )
Communication	CreatePipe( )	pipe( )
	CreateFileMapping( )	shmget( )
	MapViewOfFile( )	mmap( )
Protection	SetFileSecurity( )	cChmod( )
	InitializeSecurityDescriptor( )	umask( )
	SetSecurityDescriptorGroup( )	chown( )



System Calls Examples

- C program invoking fopen(), fgets(), and printf(), which call open(), read() and write() system calls



```
#include <stdio.h>
Int main( ) {
   FILE *f = fopen( “myFile”, “r” );
   char buf[10];
   fgets( buf, sizeof( buf ), f );
   printf( “%s\n”, buf );
   return 0;
}
```

System Call Interface: sd = open(“myFile”, O_RDONLY), read(fd, buf, sizeof(buf), write(1, buf, sizeof(buf))

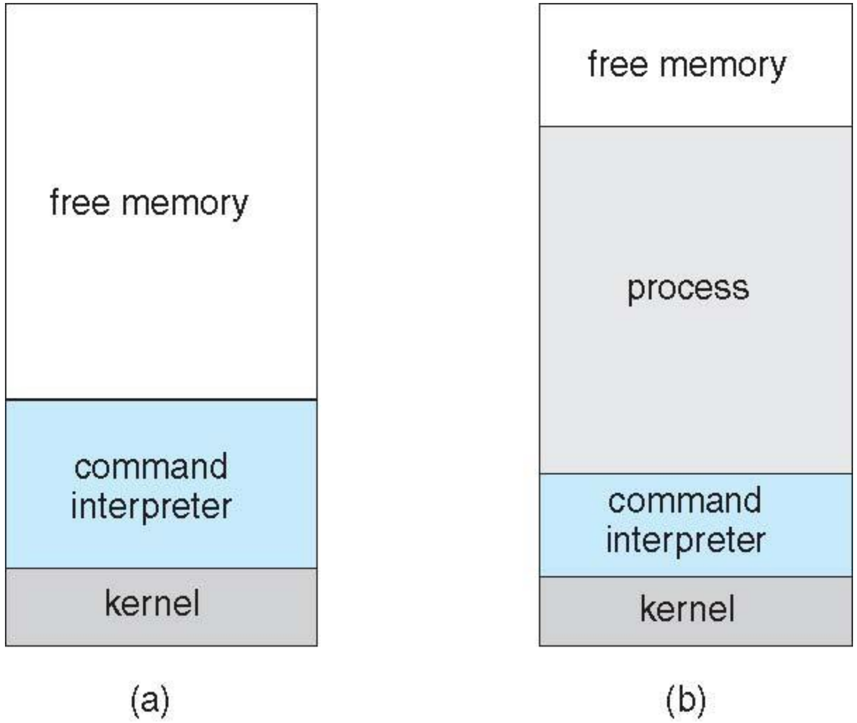
Command Interpreters

- The program that reads and interprets control statements
  - command-line interpreter (in DOS)
  - shell (in UNIX)
  - Mouse-based window and menu system (Windows, Linux, MacOS)

- What control statements can you pass the command interpreter?

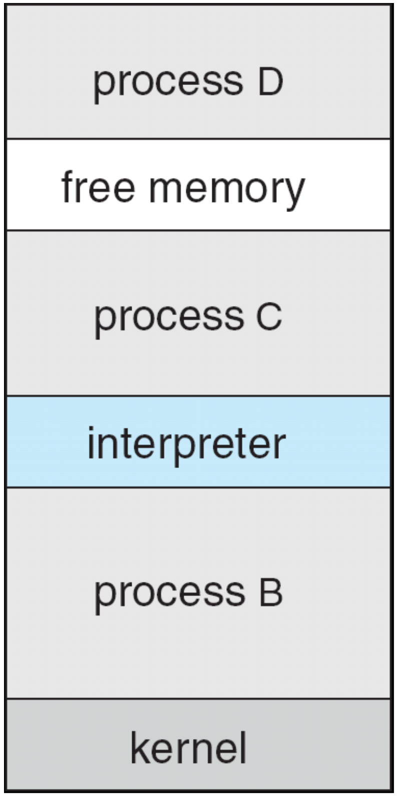
Program execution:	a.out, g++, emacs
Process management:	ps, kill, sleep, top, nice, pstack
I/O operations:	lpr, clear, lprm, mt
File-system manipulation:	ls, mkdir, mv, rm, chmod, [u]mount
Communication:	write, ping, mesg

Command Interpreters



- MS-DOS

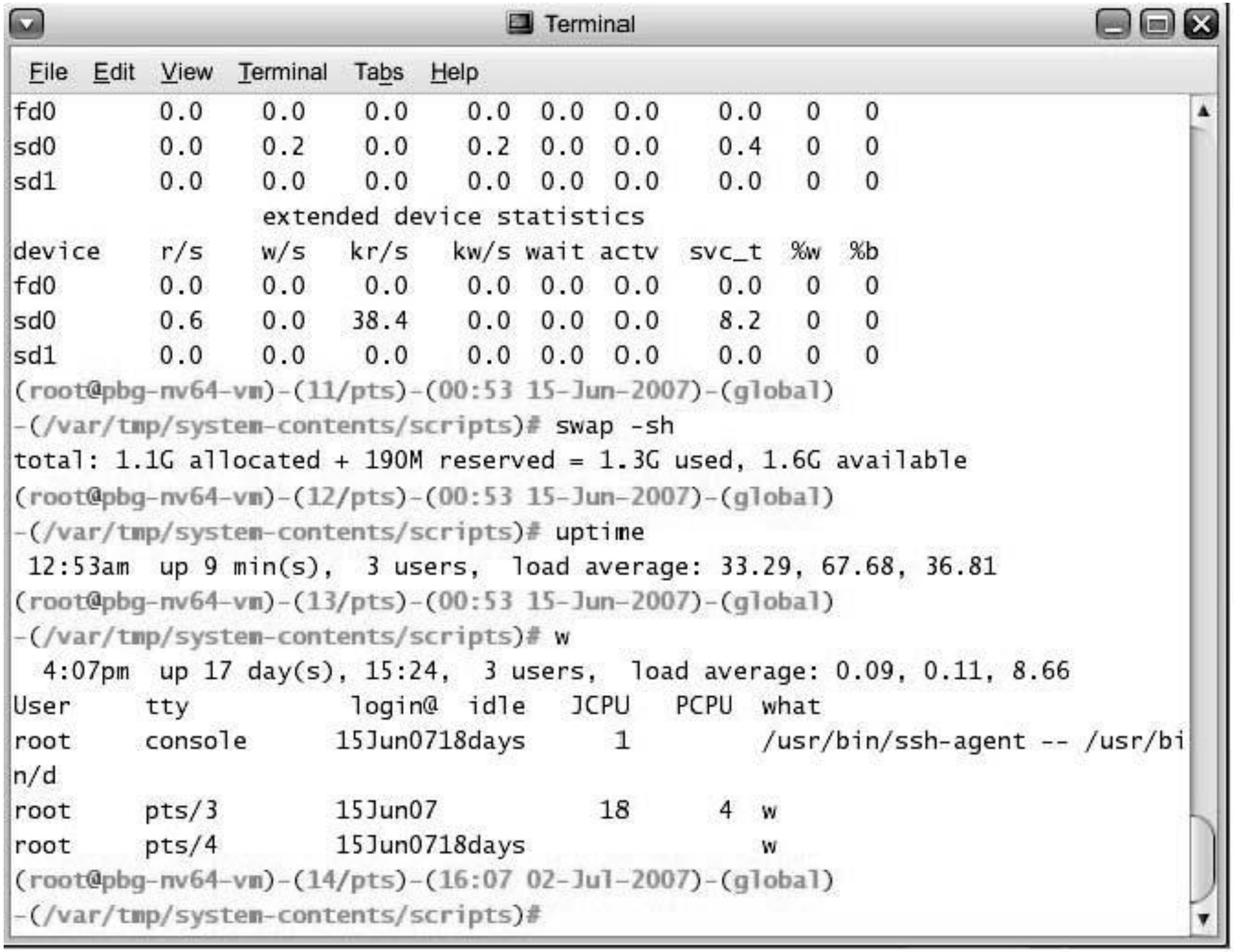
(a) At system startup (b) running a program



- Unix

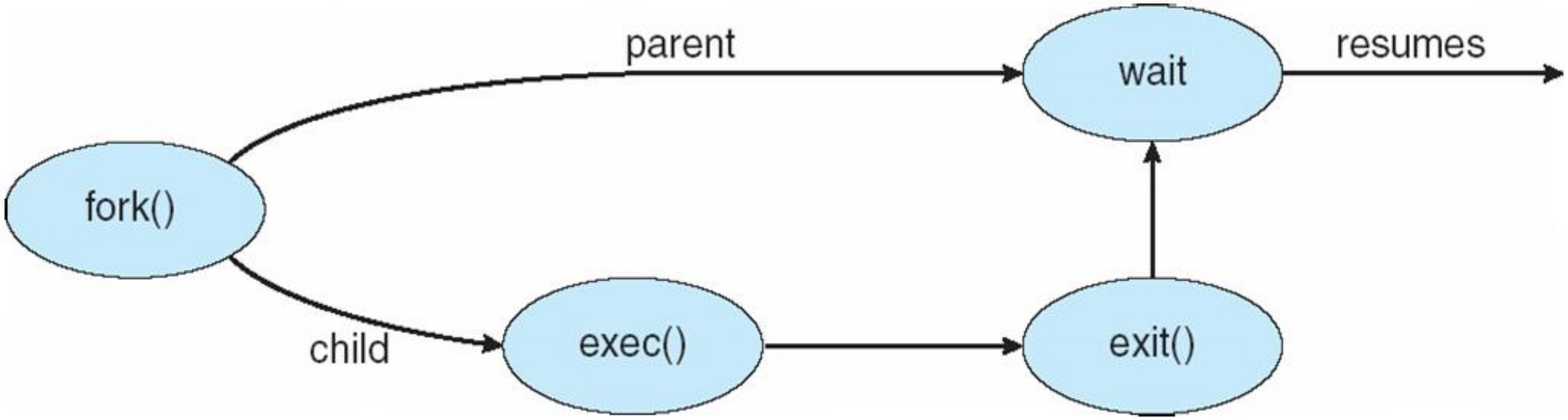
Shell (interpreter) as one of user processes

Bourne Shell Command Interpreter



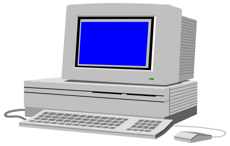
```
fd0         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0         0.0    0.2    0.0    0.2  0.0  0.0    0.4   0   0
sd1         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
                   extended device statistics
device      r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
fd0         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0         0.6    0.0   38.4    0.0  0.0  0.0    8.2   0   0
sd1         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# uptime
 12:53am  up 9 min(s),  3 users,  load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# w
  4:07pm  up 17 day(s), 15:24,  3 users,  load average: 0.09, 0.11, 8.66
User     tty           login@  idle   JCPU   PCPU  what
root     console      15Jun0718days      1         /usr/bin/ssh-agent -- /usr/bi
n/d
root     pts/3        15Jun07           18      4  w
root     pts/4        15Jun0718days                w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#
```

Process Creation in a Nutshell



Linux Shell

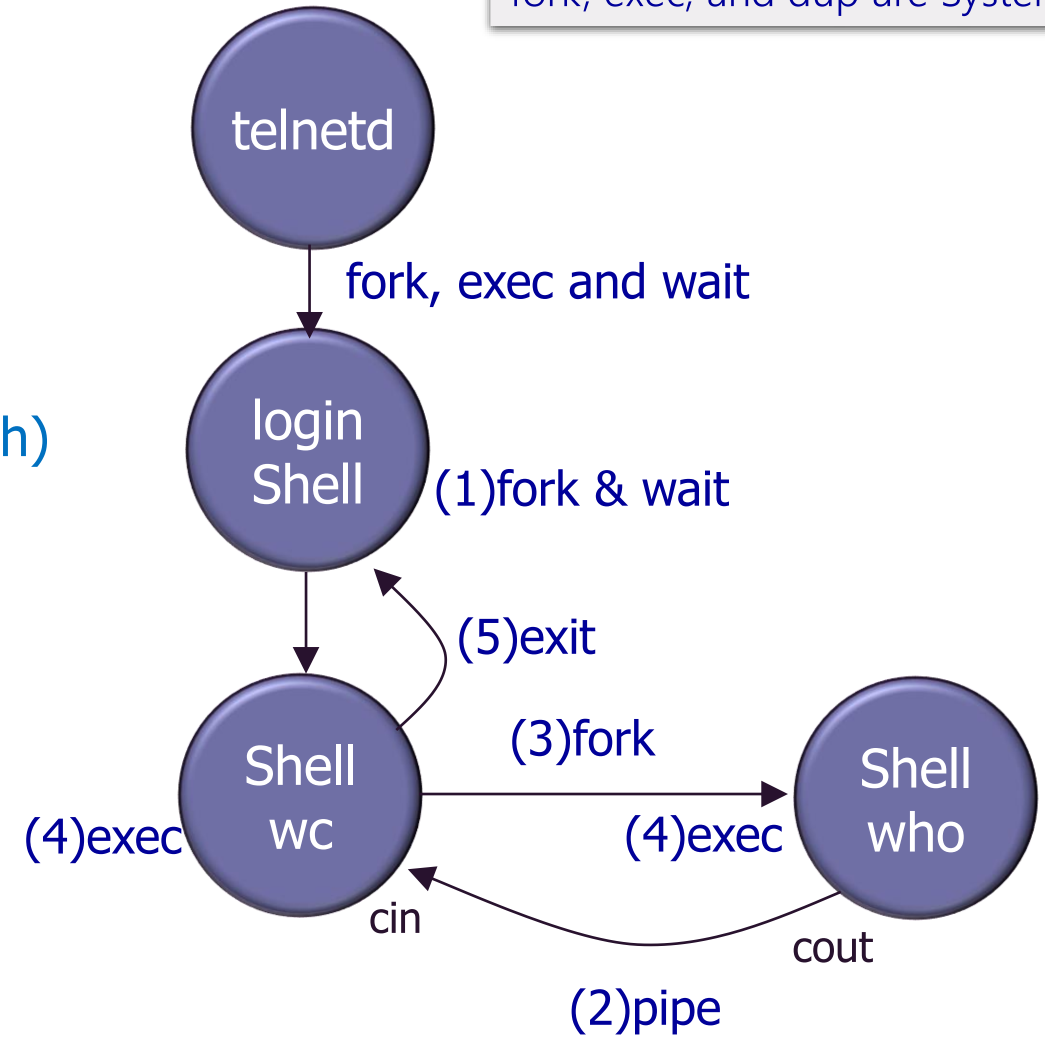
fork, exec, and dup are System calls



goodall login: chinchia

goodall[1]% (you type sth)

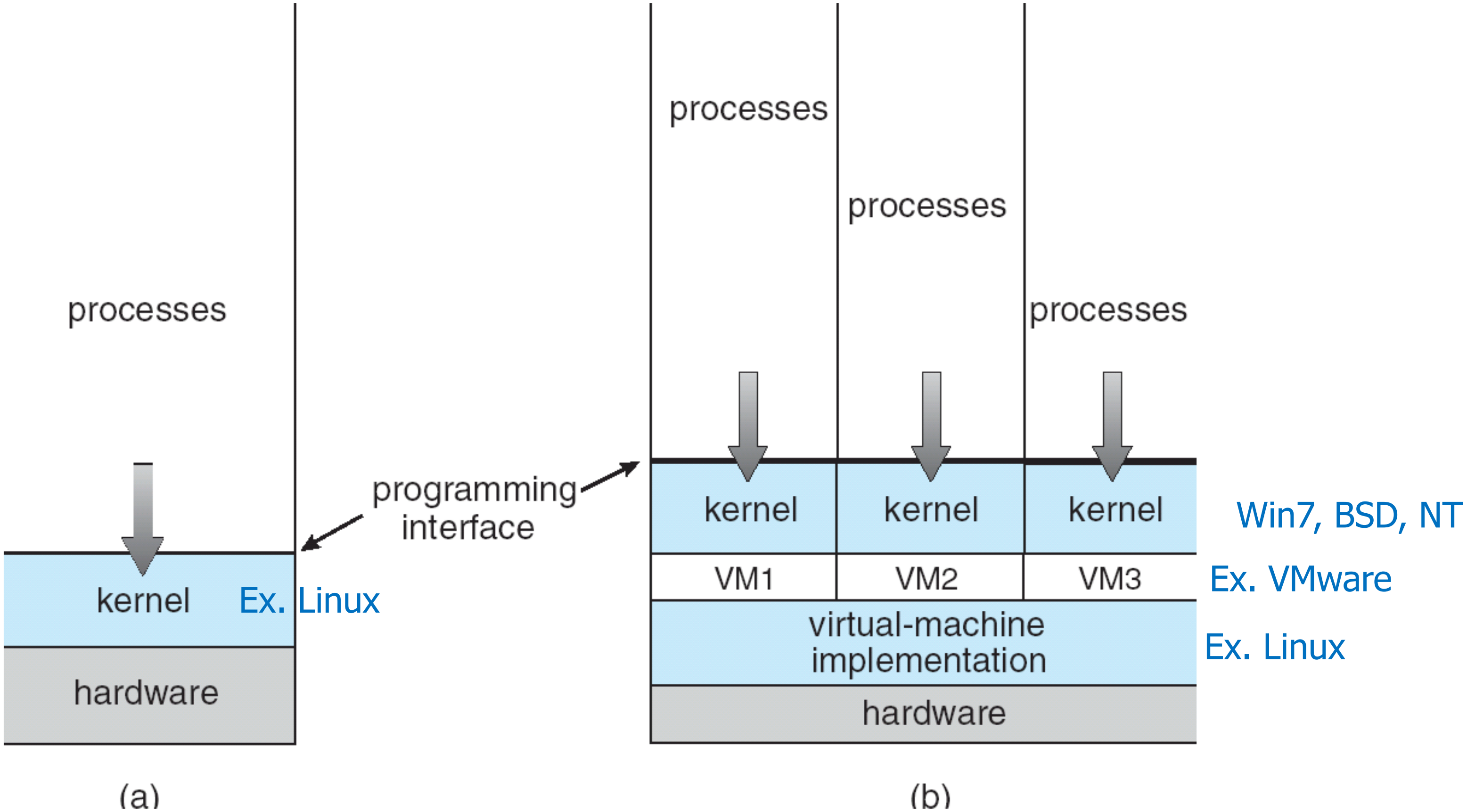
goodall[1]% who | wc -l



Virtual Machines

- A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.
- A virtual machine provides an interface *identical* to the underlying bare hardware.
- The operating system host creates the illusion that a process has its own processor and (virtual memory).
- Each guest is provided with a (virtual) copy of underlying computer.

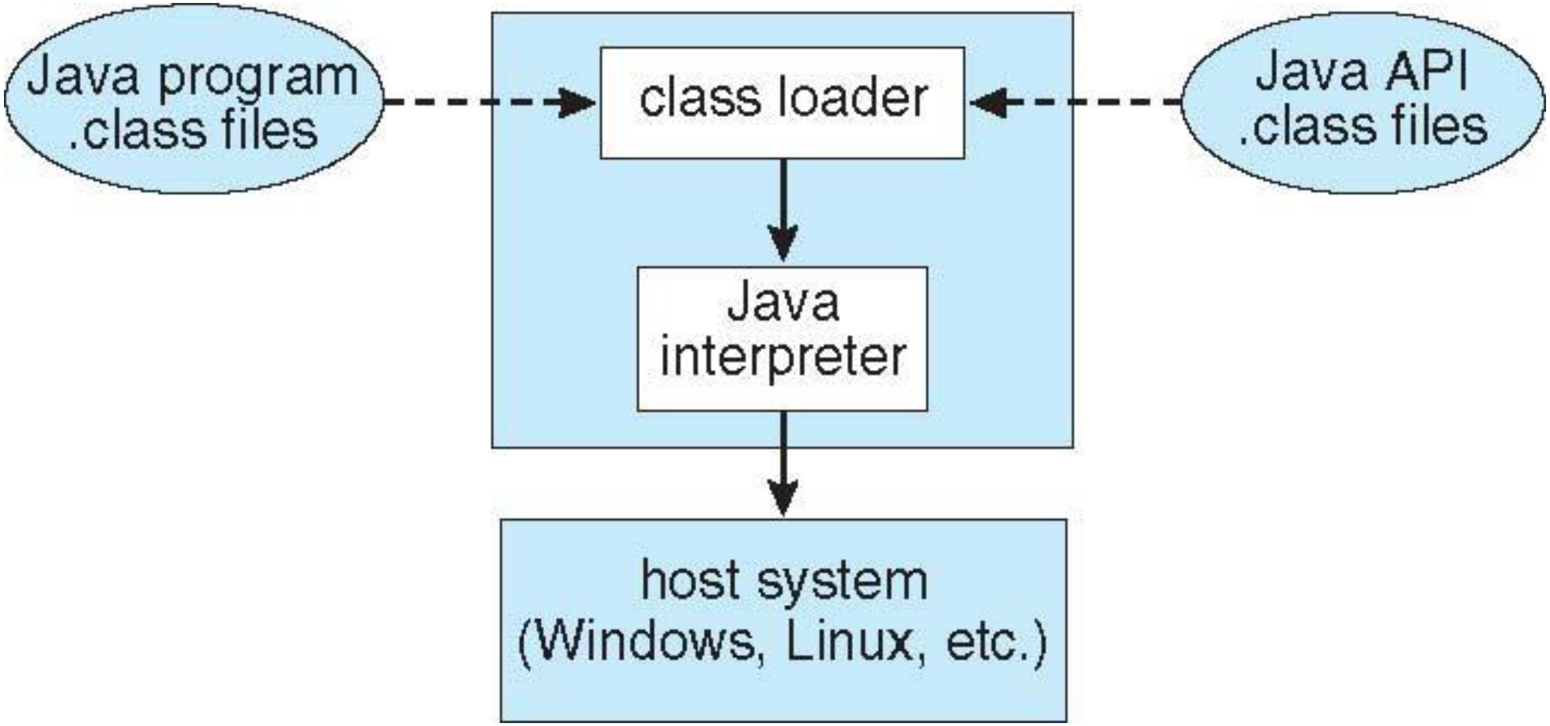
Virtualization



(a) non-virtual machine

(b) virtual machine

Java Virtual Machine



Discussions 1: TBD

1. In what particular situation have your program received a *segmentation fault*?

2. To read data from a **file**, why do we need to call open and close the file? In other words, why doesn’t OS allow read(filename, data, size)?

3. If your C++ program terminates upon an **exception**, it may not print out a **`cout`** statement that must have been executed before the exception. Why?

System Boot

- An operating system must be made available to hardware so hardware can start it.
  - Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it.
  - Sometimes two-step process where **boot block** at fixed location loads bootstrap loader.
  - When power initialized on system, execution starts at a fixed memory location.
    - Firmware is used to hold initial boot code.