现 代 操 作 系 统

Modern Operating Systems

宋虹 songhong@csu. edu. cn

中南大学计算机学院网络空间安全系

Contents

- Operating System Concepts
- MultiProcessor Operating System
- Mobile Operating System
- Embedded Operating System
- Security for Operating System
- Operating System Design

About the course

- Class period: 32
- Grades including three parts
  - Attendance : 30%
  - Paper review: 40%
  - Final Exam: 30%

Course Material

- Two required textbooks
  - Modern Operating Systems (4e)， Andrew S. Tanenbaum , 2014 Prentice-Hall, Inc.
  - 《现代操作系统》,机械工业出版社，Tanenbaum著，陈向群译，2017
- Recommended text books
  - 《操作系统：设计与实现》，电子工业出版社，中译本
  - 《操作系统－内核与设计原理》，电子工业出版社，William Stallings著，英文原版
  - 《Windows操作系统原理》，机械工业出版社，陈向群著

Course Material

- **https://www.usenix.org/conference/osdi24:July 10–12, Santa Clara, CA, USA**

- **OSDI 2024: 53**
  - Memory Management (6 papers)
  - Low-Latency LLM Serving (5 papers)
  - Distributed Systems (4 papers)
  - Deep Learning (5 papers)
  - Operating Systems (5 papers)
  - Cloud Computing (5 papers)
  - Formal Verification (5 papers)
  - Cloud Security (4 papers)
  - Data Management (4 papers)
  - Analysis of Correctness (5 papers)
  - ML Scheduling (5 papers)

- **OSDI 2022: 49**
  - Distributed Storage and Far Memory
  - Bugs
  - Persistent Memory
  - Machine Learning
  - Potpourri
  - Storage
  - Formal Verification
  - Isolation and OS Services
  - Security and Private Messaging
  - Managed Languages
  - Recommenders and Pattern Mining

- **OSDI 2023: 50（35+15）**
  - Make Your Bits Go Faster
  - Secure Your Bits
  - Expanding, Hardening, and Deploying Your Bits
  - Store Your Bits
  - Manage Your Bits
  - Train Your Bits
  - Verify Your Bits
  - Transfer Your Bits

- **OSDI 2021: 31**
  - Optimizations and Scheduling for Machine Learning
  - Storage
  - Data Management
  - Operating Systems and Hardware
  - Security and Privacy
  - Correctness
  - Graph Embeddings and Neural Networks

Course Material

- **SOSP 2023: Oct 23-26, Koblenz, Germany. Total: 43 papers**
- **https://sosp2023.mpi-sws.org/**
  - Kernel Design and Testing(4 papers)

    TreeSLS: A Whole-system Persistent Microkernel with Tree-structured State Checkpoint on NVM, Best Award, Shanghai Jiao Tong University
  - Reliability(4 papers)

    Validating JIT Compilers via Compilation Space Exploration , Best Paper Award, Nanjing University
  - Storage (4 papers)

    Enabling High-Performance and Secure Userspace NVM File Systems with the Trio Architecture , Best Paper Award
  - Cloud (4 papers)
  - Distributed systems (4 papers)
  - Learning (8 papers)
  - Security and Privacy (4 papers)
  - Datacenter (4 papers)
  - Data and databases (4 papers)
  - Distributed and disaggregated memory (4 papers)

Course Material

- **SOSP 2024: November 4-6, Austin, TX USA. Total: 43 papers**
  - Aceso: Achieving Efficient Fault Tolerance in Memory-Disaggregated Key-Value Stores
  - Apparate: Rethinking Early Exits to Tame Latency-Throughput Tensions in ML Serving
  - Autobahn: Seamless high speed BFT
  - BIZA: Design of Self-Governing Block-Interface ZNS AFA for Endurance and Performance
  - CHIME: A Cache-Efficient and High-Performance Hybrid Index on Disaggregated Memory
  - Caribou: Fine-Grained Geospatial Shifting of Serverless Applications for Sustainability
  - Cookie Monster: Efficient On-Device Budgeting for Differentially-Private Ad-Measurement Systems
  - DNS Congestion Control in Adversarial Settings
  - Dirigent: Lightweight Serverless Orchestration
  - Efficient Reproduction of Fault-Induced Failures in Distributed Systems with Feedback-Driven Fault Injection
  - Efficient file-lifetime redundancy management for cluster file systems
  - Enabling Parallelism Hot Switching for Efficient Training of Large Language Models
  - FBDetect: Catching Tiny Performance Regressions at Hyperscale through In-Production Monitoring
  - Fast & Safe IO Memory Protection
  - Fast Core Scheduling with Userspace Process Abstraction
  - Fast, Flexible, and Practical Kernel Extensions
  - Icarus: Trustworthy Just-In-Time Compilers with Symbolic Meta-Execution
  - If At First You Don’t Succeed, Try, Try, Again...? Insights and LLM-informed Tooling for Detecting Retry Bugs in Software Systems

Chapter 1 Operating System Concepts

- 1.1 Some Basic Concepts
- 1.2 Processes and Threads
- 1.3 Memory Management
- 1.4 File System
- 1.5 I/O System
- 1.6 Reading Materials

1.1 Some Basic Concepts

- **What's An Operating System?**
- **Roles of Operating System**
- **History of Operating Systems**
- **Hardwares**
- https://ysyx.oscc.cc/slides/hello-x86.html

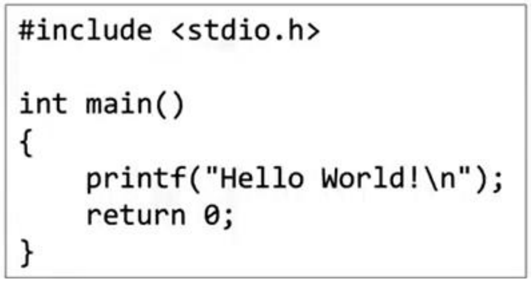
1.1 Some Basic Concepts

- **What's An Operating System?**
- **Roles of Operating System**
- **History of Operating Systems**
- **Hardwares**

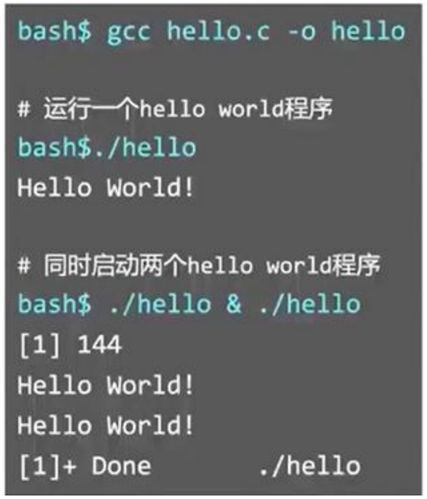
What Is An Operating System?

- Which options are belong to the scope of Operating System?
    - All software in Windows 10
    - Linux kernel and all of device drivers
    - The third-part NTFS system downloaded and installed in Macbook
    - All software of HuaWei Mate 8

What Is An Operating System?



```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```



```
bash$ gcc hello.c -o hello

# 运行一个hello world程序
bash$./hello
Hello World!

# 同时启动两个hello world程序
bash$ ./hello & ./hello
[1] 144
Hello World!
Hello World!
[1]+ Done          ./hello
```

- There is only an application in the machine, which will automatically run and will not exit after the power is on. Do the machine need Operating System?
- Do the machine need to install Operating System if an application hopes to control hardware by itself other than to use the abstractions of Operating System?

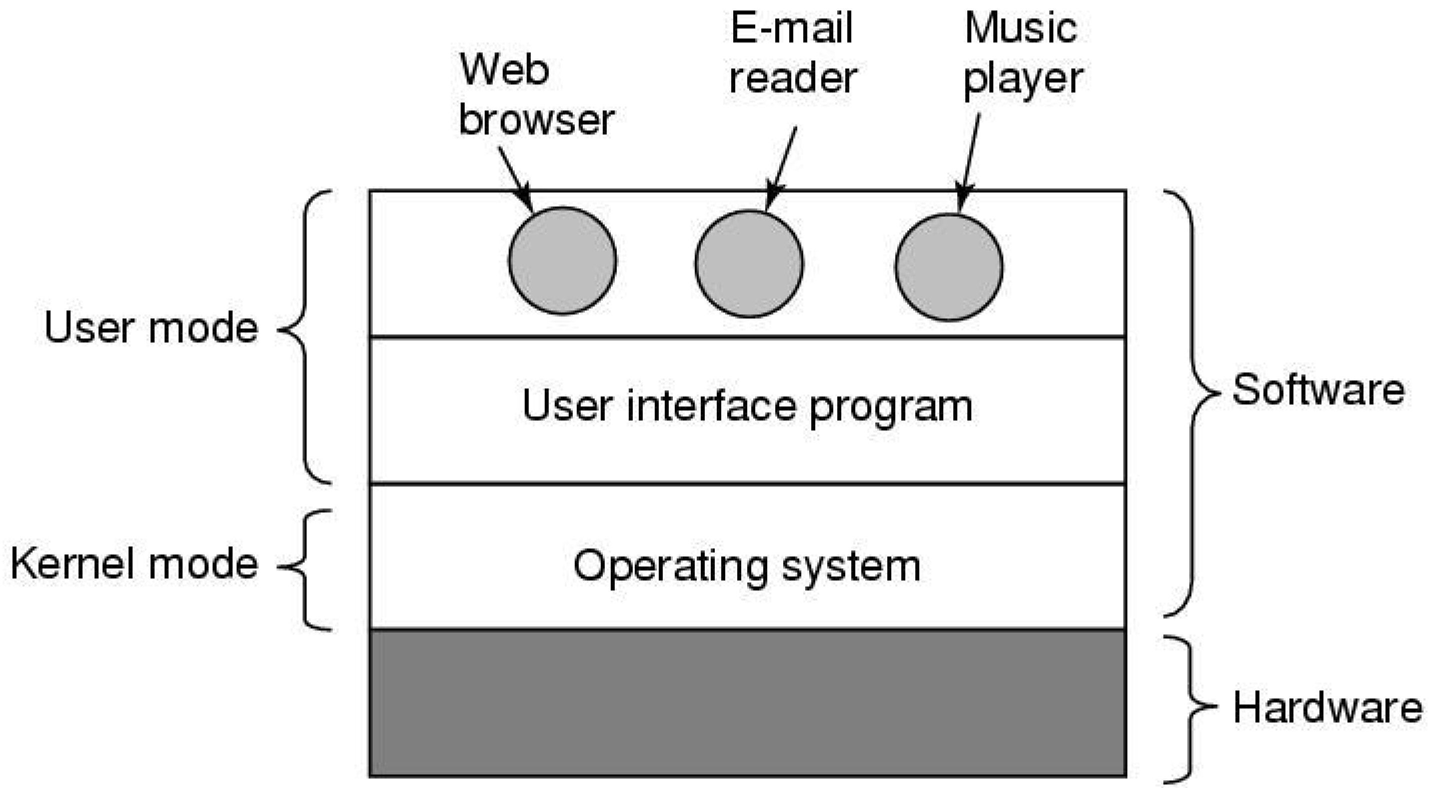
What Is An Operating System?

Lots of hardware !!

- One or more processors
- Main memory
- Disks
- Printers
- Various input/output devices

Managing all these components requires a layer of software – the **operating system**

Where is the software?



Where the operating system fits in.

1.1 Some Basic Concepts

- What’s An Operating System?
- Roles of Operating System
- History of Operating Systems
- Hardwares

Roles of Operating System

- Bridge the “Semantic Gap” between Hardware and Application
- Three View of Operating System
  - OS is an Extended Machine
  - OS is a Virtual Machine
  - OS is a Resource Manager

Semantic Gap

- Hardware capabilities are very low level
  - Arithmetic and logical operators
  - Comparison of two bit-strings
  - Branching, reading, and writing bytes
- User needs to think in terms of problem to be solved
  - High-level data structures and corresponding operations
  - Simple, uniform interfaces to subsystems
  - Treat programs and data files as single entities

Bridging the Semantic Gap

- Use software to bridge this semantic gap
  - Language processors: assemblers, compilers, interpreters,….
  - Editors and text processors, linkers and loaders
  - Application programs, utility and service programs

$$\Downarrow$$

Operating Systems

The Role of the OS

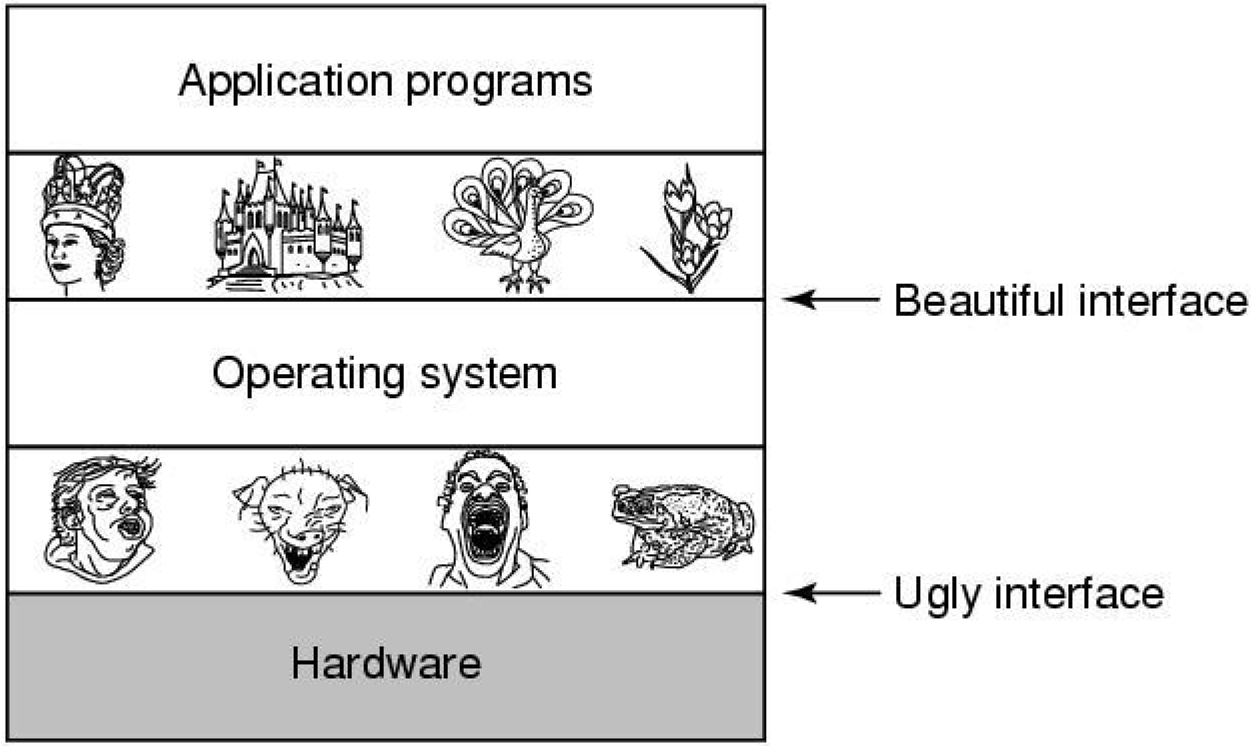
- Bridge Hardware/Application Gap
  - Machine instruction vs. high level operation
  - Linear memory vs. data structures

  } **compiler bridges gap**

  - Limited CPU & memory vs. more needed
  - Secondary memory devices vs. files
  - I/O devices vs. high level I/O commands

  } **OS bridges gap**

The Operating System as an Extended Machine

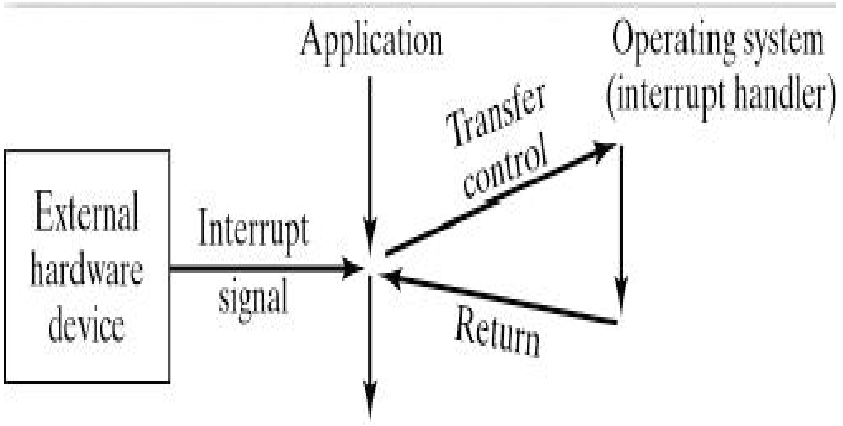


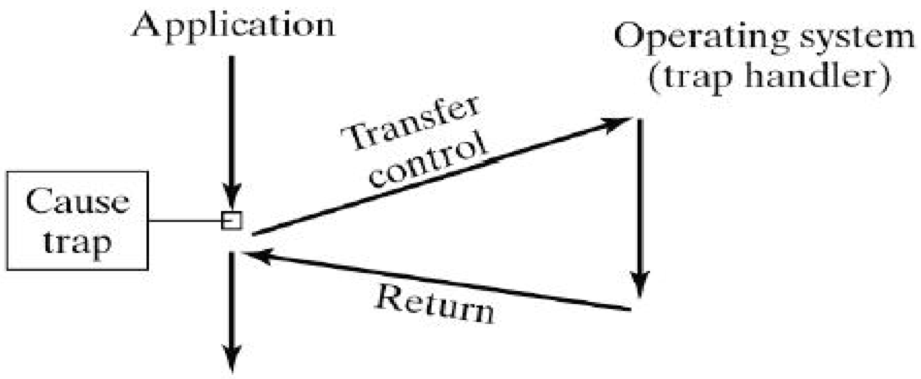
The Operating System as an Extended Machine

- Principle of abstraction hides complexity
- OS provides high level operations using lower level operations

Interface of OS to outside world

- Structural Organization: layered
- Interfaces
  - Hardware interface
    - Applications and OS compiled into machine instructions
    - interrupts and traps:allow OS to seize control





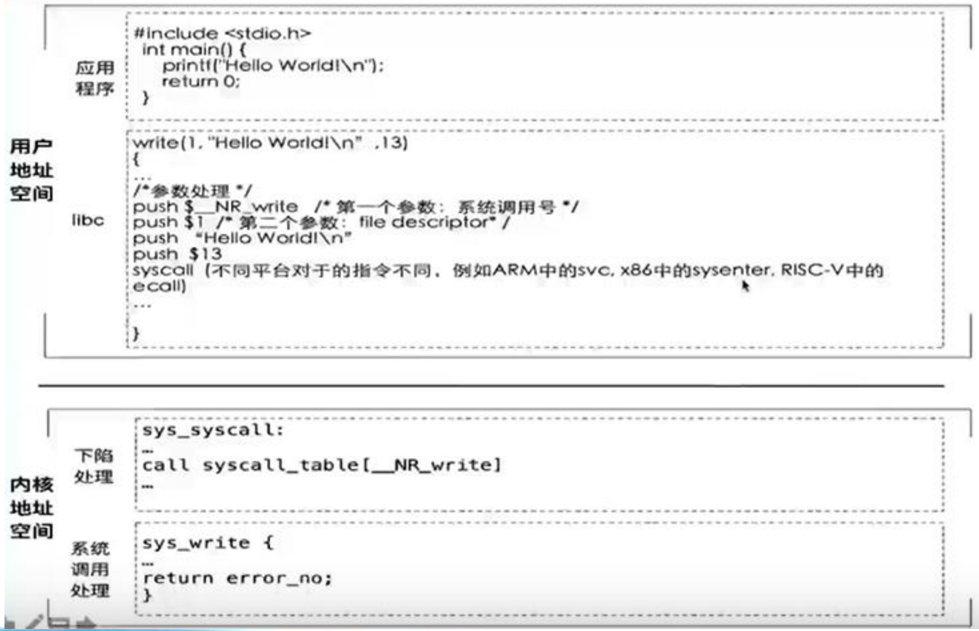
Interface of OS to outside world

- Structural Organization: layered
- Interfaces
  - Hardware interface
  - Program interface
    - Invoking system services
      - Library call——nonprivileged
      - Kernel call——privileged
      - Example: Printf() → write() → sys_write()
  - User interface
    - Text-based shell: command interpreter, shell scripts
    - Graphics-based GUI

Interface of OS to outside world

Hello运行中的系统调用（strace）

```
/* 运行hello程序 */
execve("./hello", ["./hello"], 0x7ffed5a79e80 /* 64 vars */) = 0
...
/*将``Hello World!\n"写到标准输出中，在这里，1代表标准输出（其他的0代表标准输入，2代表标准错误），13代表一共写了13个字符。*/
write(1, "Hello World!\n", 13Hello World!)        = 13
/* 执行结束后，hello程序退出*/
exit_group(0)
```



```
#include <stdio.h>
 int main() {
    printf("Hello World!\n");
    return 0;
 }
```

```
write(1, "Hello World!\n" ,13)
{
...
/*参数处理 */
push $__NR_write  /* 第一个参数: 系统调用号 */
push $1  /* 第二个参数: file descriptor* /
push  "Hello World!\n"
push  $13
syscall (不同平台对于的指令不同, 例如ARM中的svc, x86中的sysenter, RISC-V中的
ecall)
...

}
```

```
sys_syscall:
...
call syscall_table[__NR_write]
...
```

```
sys_write {
...
return error_no;
}
```

The Operating System as a Resource Manager

- Allow multiple programs to run at the same time
- Manage and protect memory, I/O devices, and other resources
- Multiplexes (shares) resources in two different ways:
  - In time
  - In space
- Efficiency: balance overall performance with individual needs (response time, deadlines)

The Operating System as a Resource Manager

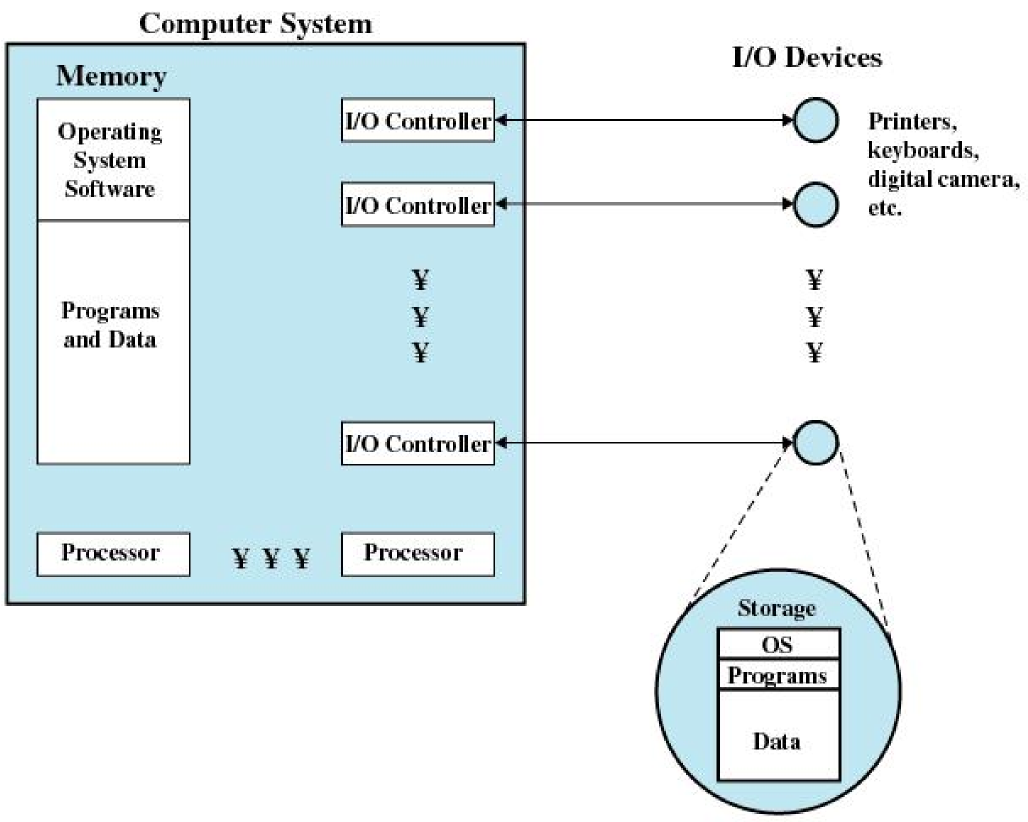
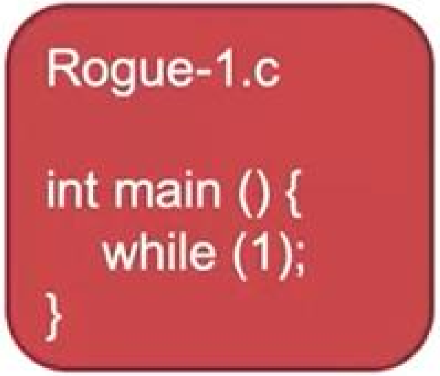
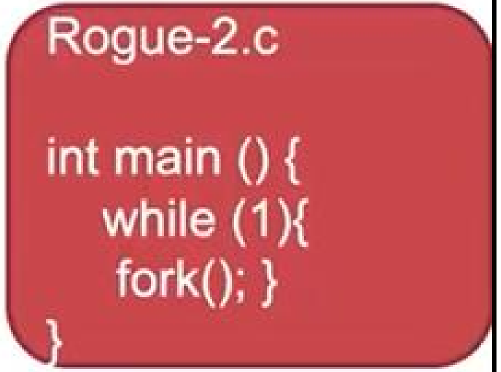


Figure 2.2 The Operating System as Resource Manager

The Operating System as a Resource Manager



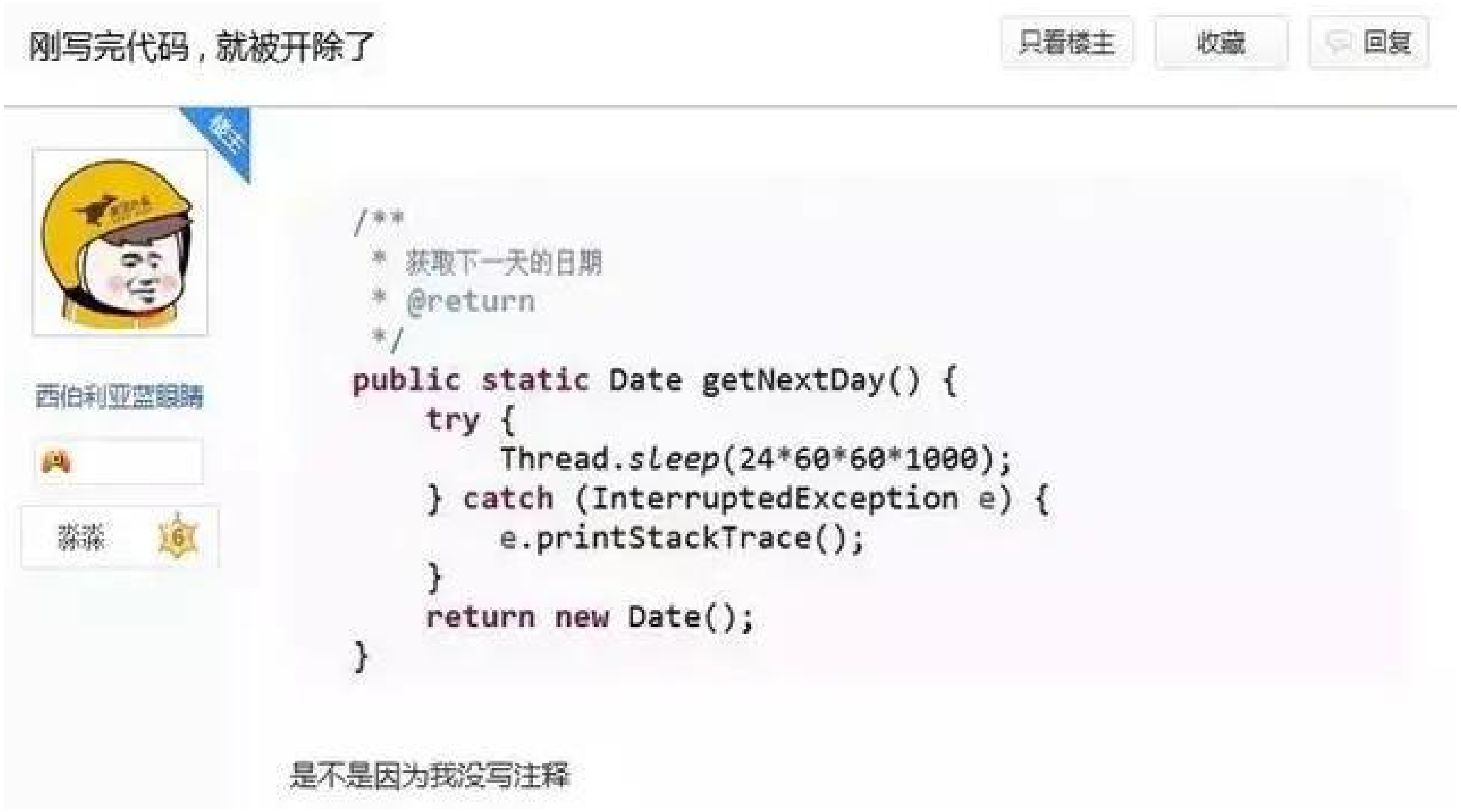
```
Rogue-1.c

int main () {
    while (1);
}
```



```
Rogue-2.c

int main () {
    while (1){
     fork(); }
}
```

- How to avoid a rogue application monopolizing all resources?
- What will be happened when Rogue-2.c runs?
- How to solve this problem?

The Operating System as a Resource Manager



刚写完代码，就被开除了

```
/**
 * 获取下一天的日期
 * @return
 */
public static Date getNextDay() {
    try {
        Thread.sleep(24*60*60*1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    return new Date();
}
```

是不是因为我没写注释

1.1 Some Basic Concepts

- **What's An Operating System?**
- **Roles of Operating System**
- **History of Operating Systems**
- **Hardwares**

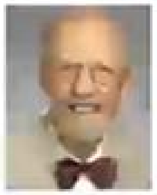
图灵奖与操作系统的演变

Maurice Wilkes
1967年图灵奖

EDSAC,1949
Multi-programming
第一台存储程序式电子计算机

Frederick Brooks
1999年图灵奖

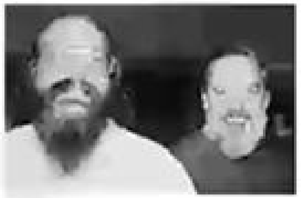
IBM System/360,1964



Fernando J. Corbató
1990年图灵奖

CTSS,1961 & Multics,1969
分时操作系统

Unix,1971
多任务多用户操作系统



Ken Thompson & Dennis Ritchie
1983年图灵奖

Venus,1972
小型低成本交互式分时操作系统

Barbara Liskov
2008年图灵奖

2019
分布式操作系统

History of Operating Systems

Generations:

- (1945–55) Vacuum Tubes
- (1955–65) Transistors and Batch Systems
- (1965–1980) ICs and Multiprogramming
- (1980–Present) Personal Computers, Tablets, Phones

Transistors and Batch Systems (1)

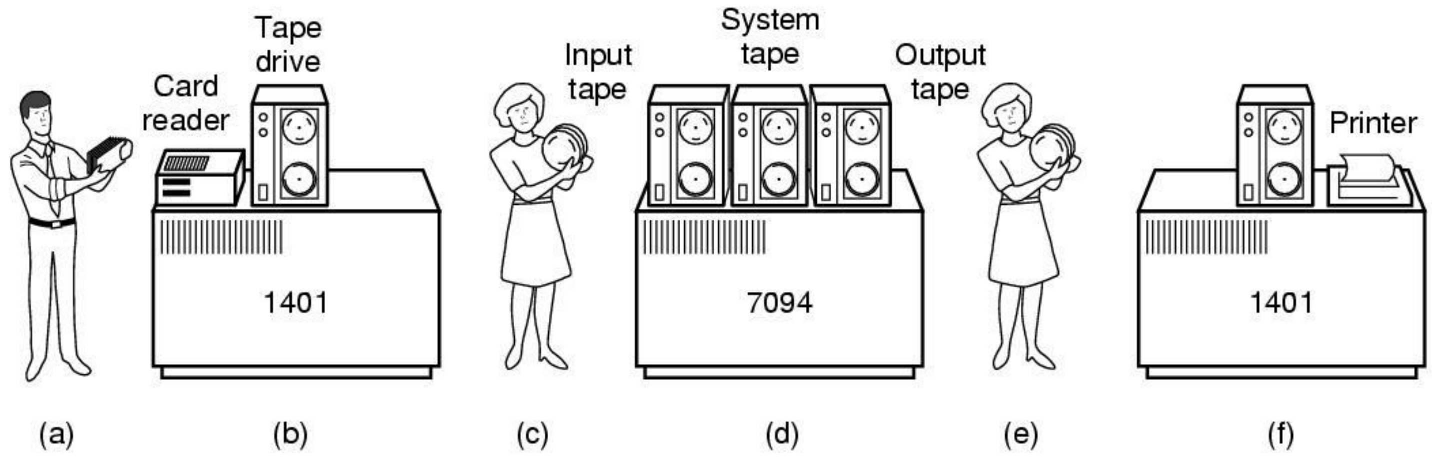


Figure 1-3. An early batch system.

(a) Programmers bring cards to 1401.

(b)1401 reads batch of jobs onto tape.

Transistors and Batch Systems (2)

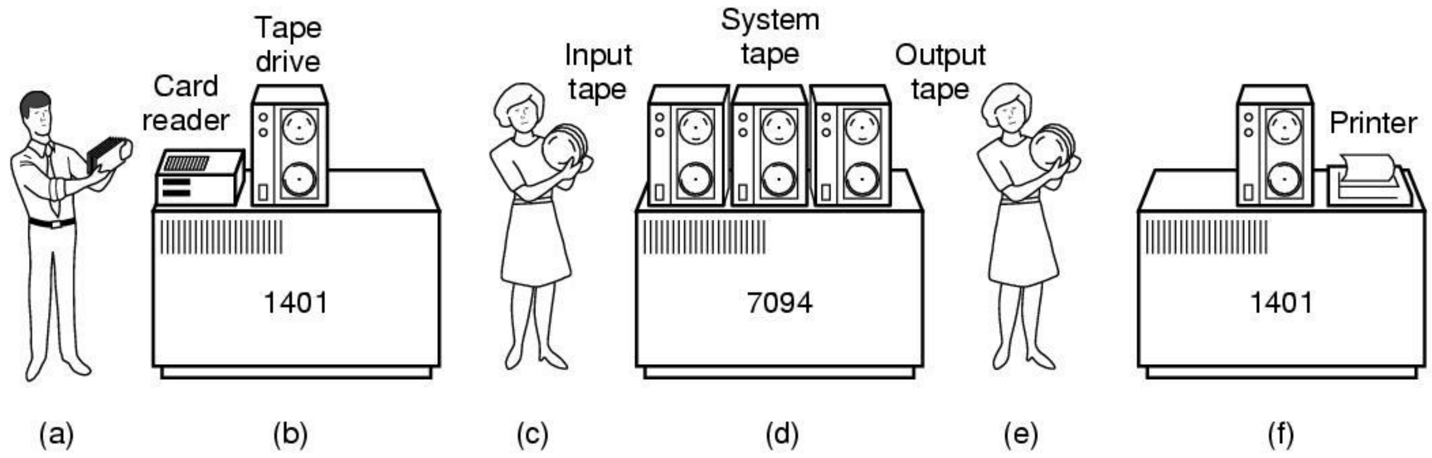


Figure 1-3. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

Transistors and Batch Systems (4)

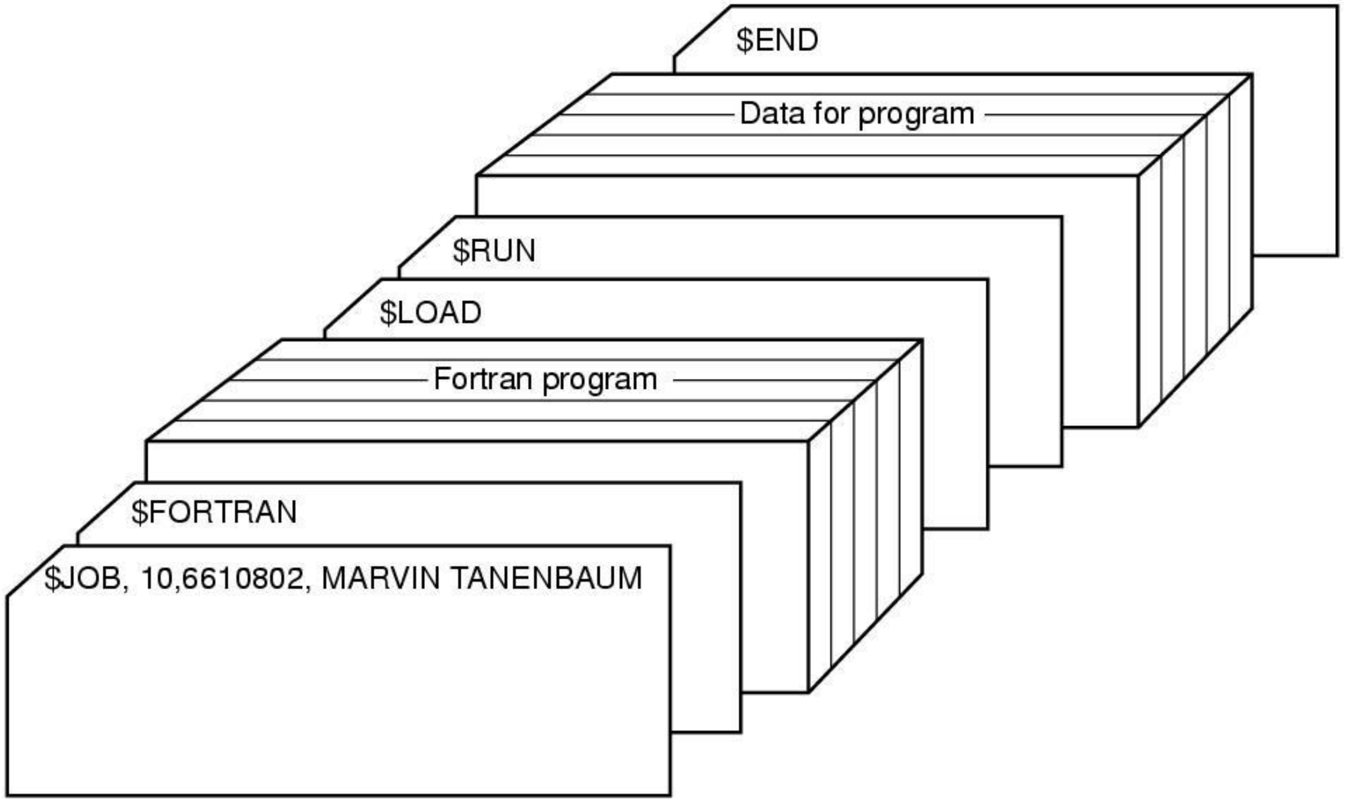


Figure 1-4. Structure of a typical FMS job.

ICs and Multiprogramming

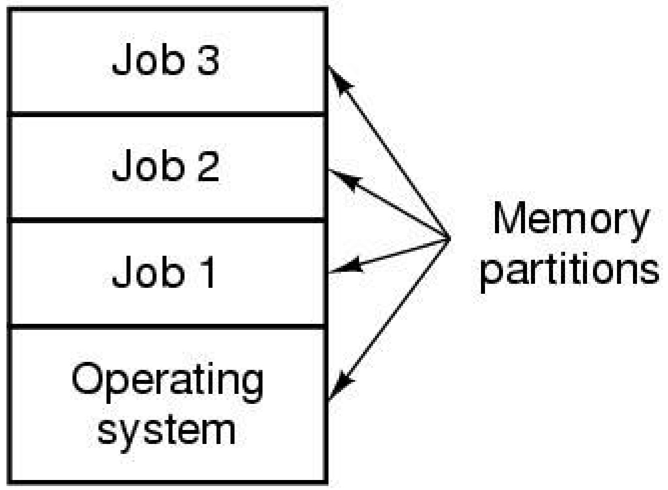
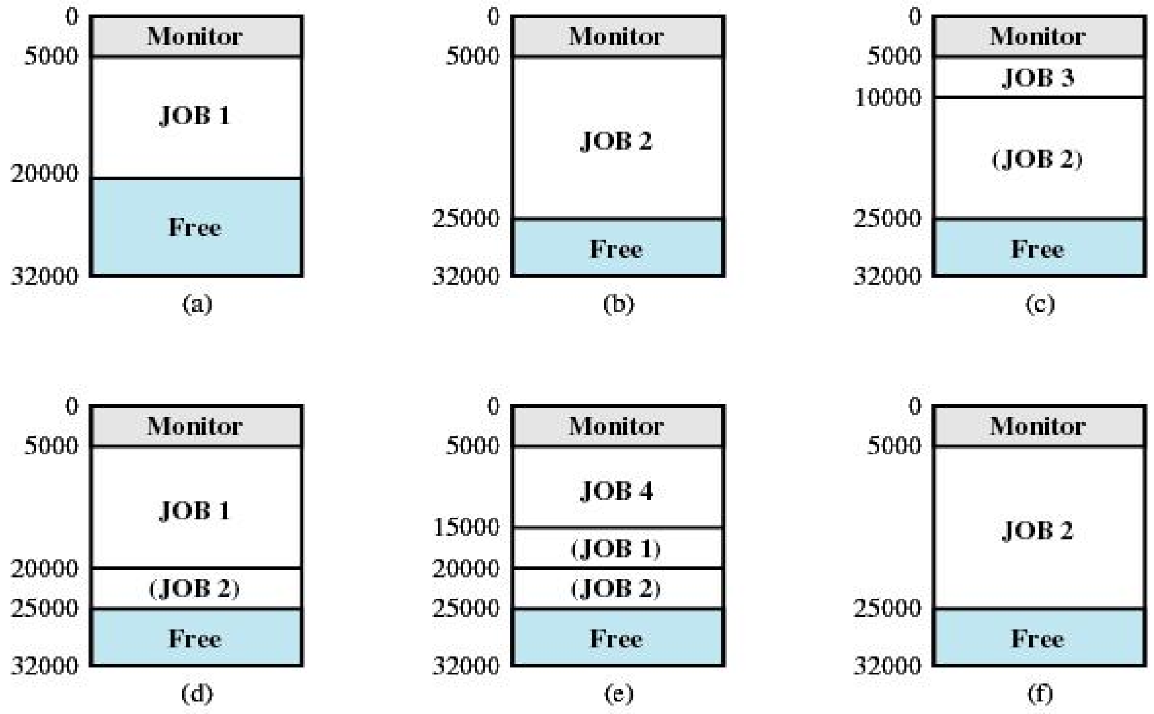


Figure 1-5. A multiprogramming system with three jobs in memory.

More third generation

- Time Sharing (CTSS)
- Multics
- Unix
- Linux



.

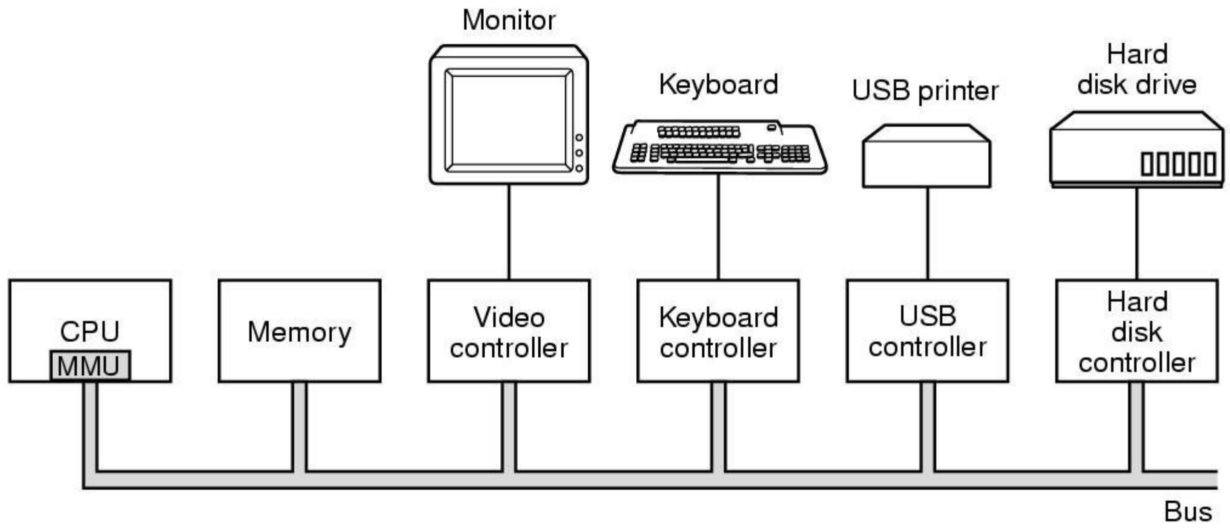
Figure 2.7 CTSS Operation

Fourth generation

- PCs
- Network Operating Systems
- Distributed Operating Systems

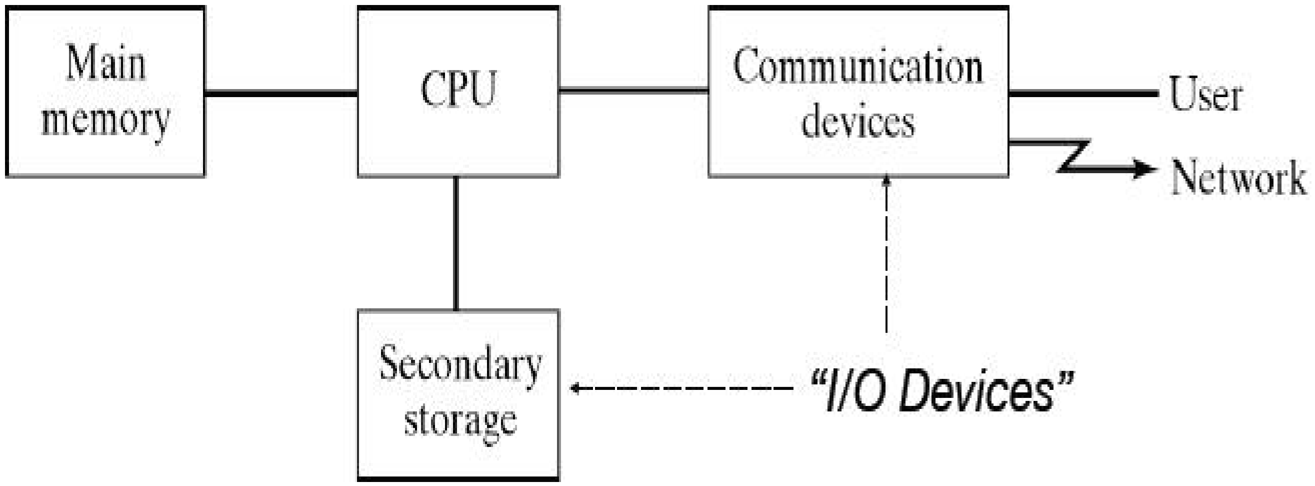
.

Computer Hardware Review



Some of the components
of a simple personal computer.

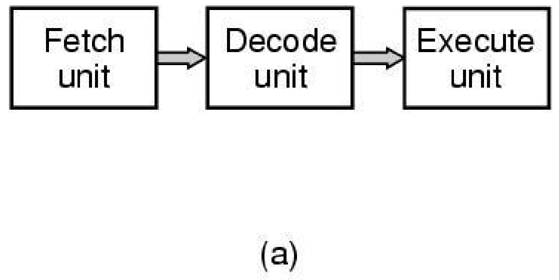
Single CPU System



- Classic von Neumann stored-program computer
  - Memory is linear sequence of directly addressable cells
- Central Processing Unit (CPU) cycles:

(1) Fetch Instruction (2) Increment Program Counter
(3) Decode Instruction (4) Fetch Operands
(5) Execute Instruction

CPU Pipelining



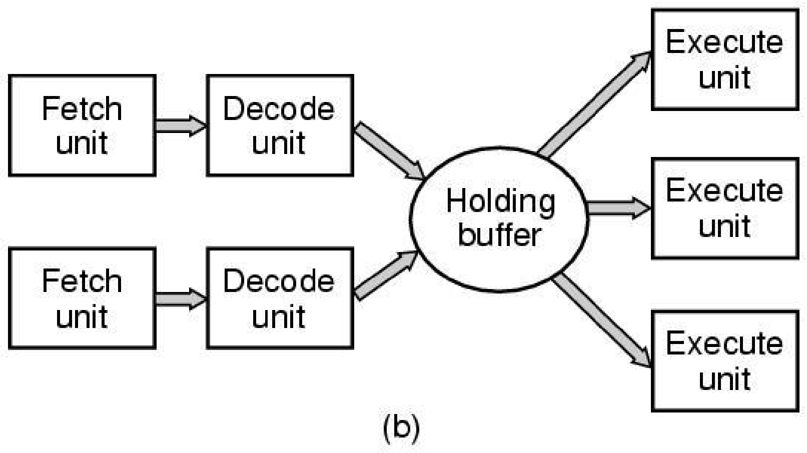
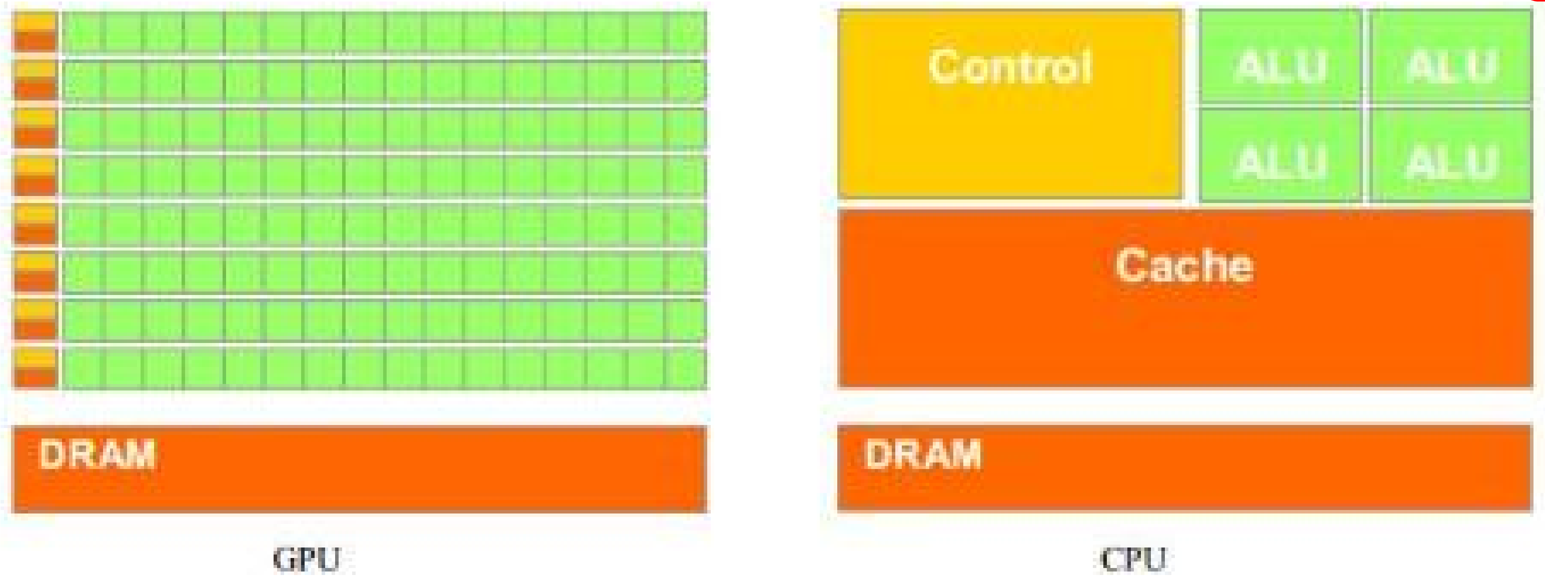


Figure 1-7. (a) A three-stage pipeline. (b) A superscalar CPU.

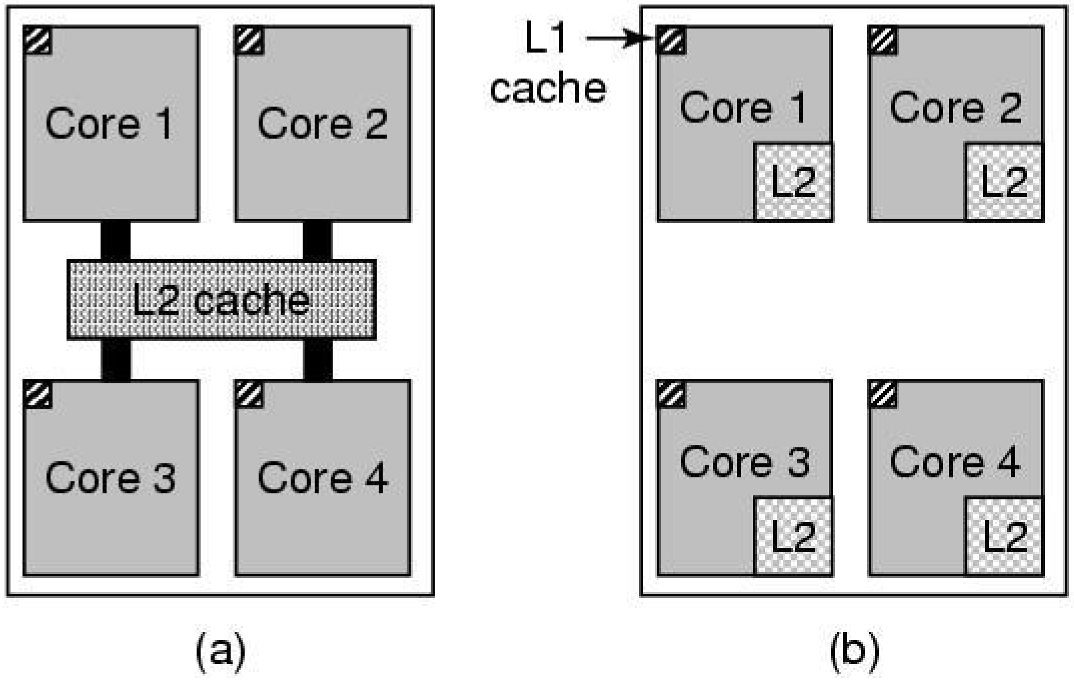
GPU—— Graphic Pipelining & Directive-Based Partitioning



- CUDA: Computed Unified Device Architecture
- OpenMP、OpenACC、OpenCL

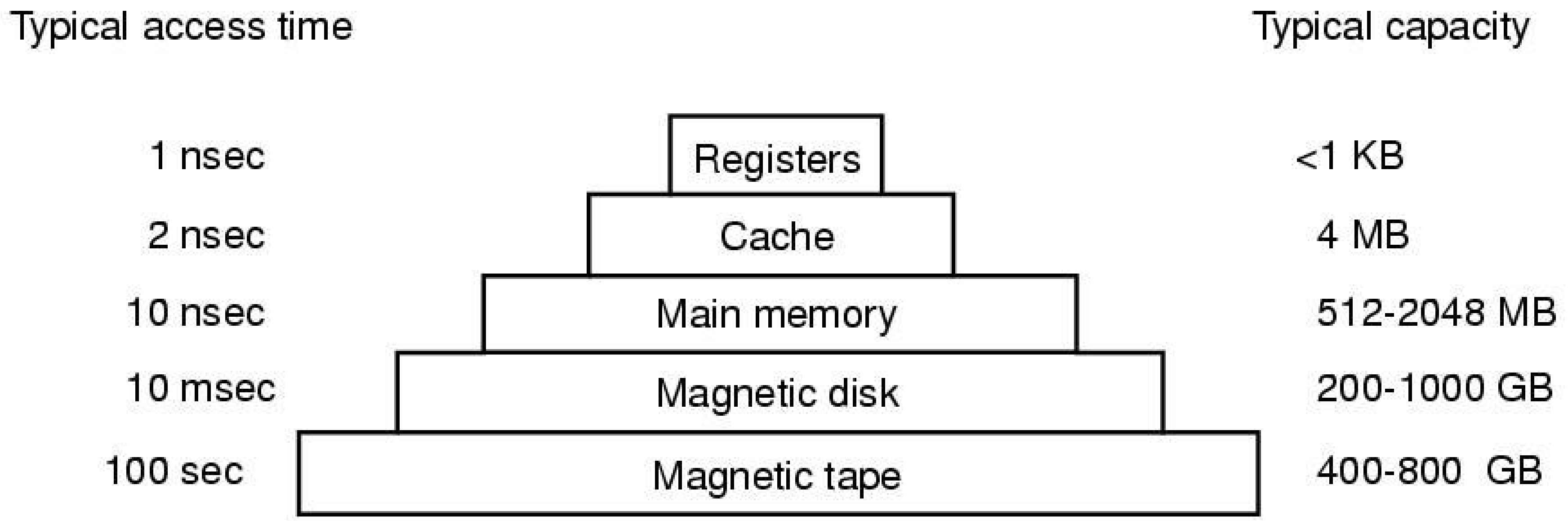
- H. Wang, S. Potluri, D. Bureddy, C. Rosales, and D. K. Panda. GPU-Aware MPI on RDMA-Enabled Clusters: Design, Implementation and Evaluation. IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 10, pp. 2595-2605, 2014.

Multithreaded and Multicore Chips



(a) A quad-core chip with a shared L2 cache.
(b) A quad-core chip with separate L2 caches.

Memory Hierarchy



A typical memory hierarchy.
The numbers are very rough approximations.

Caches

- Main memory is divided into cache lines (64 bytes)
  - 0-63 in line 1, 64-127 in line 2
- When program reads a word-cache hardware checks to see if in cache.
  - If so, then have a cache hit (2 cycles).
  - Otherwise, make request of main memory over the bus (expensive)
- Cache is expensive and is therefore limited in size
- Can have cache hierarchies
- Cache other things, like URL addresses

Four Cache Questions

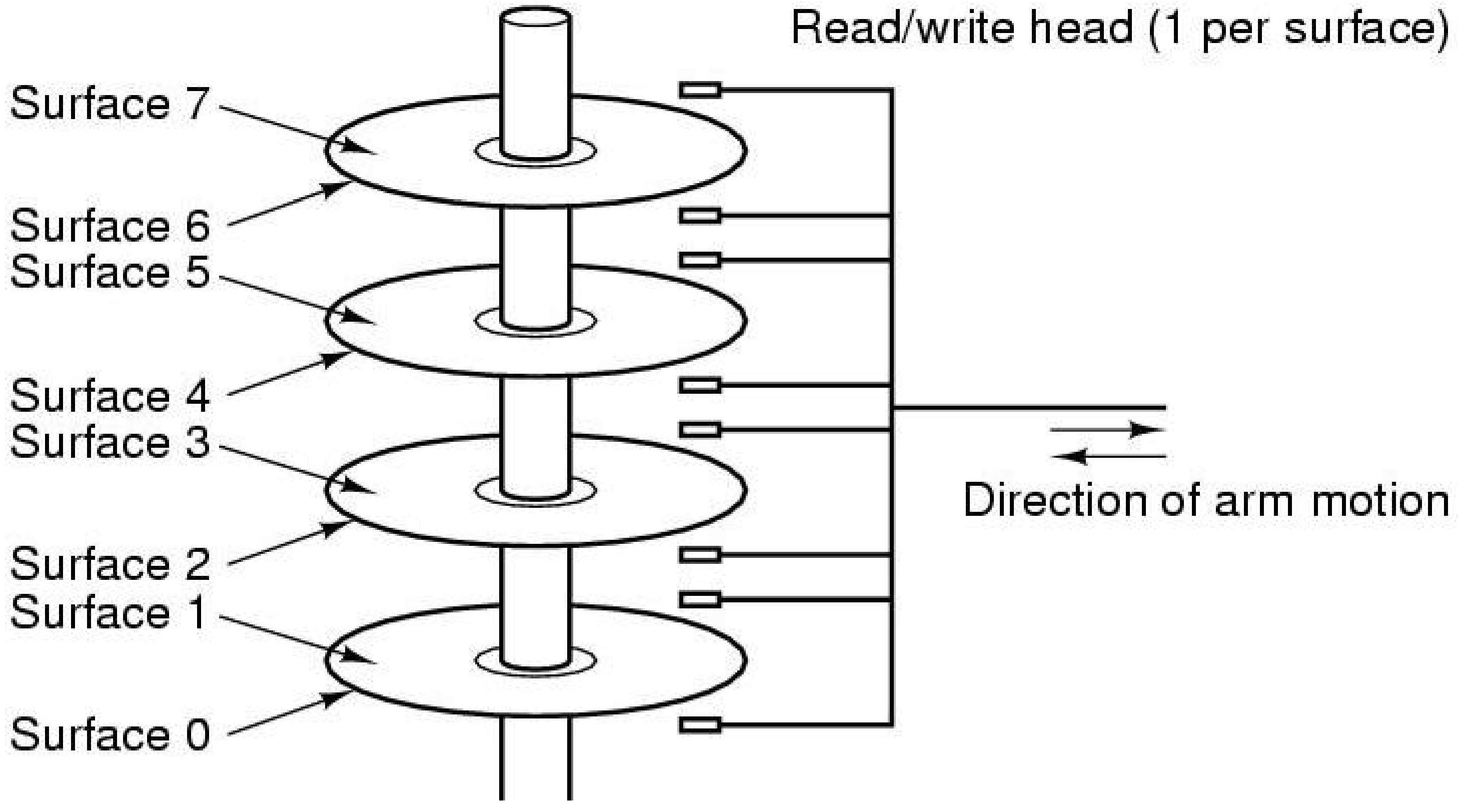
- When to put a new item into the cache? (on a cache miss)
- Which cache line to put the new item in? (memory word determines which line)
- Which item to remove from the cache when a slot is needed? (same line new data goes into)
- Where to put a newly evicted item in main memory? (memory address determines this)

Main Memory

- RAM -> SDRAM ->DDR1->DDR2->DDR3->DDR4->…
- ROM-can’t be changed. Fast, cheap
  - eg.-keeps bootstrap OS loader
- EEPROM (Electrically Erasable PROM) Can be re-written, but slowly
  - E.g.-serves as films in digital cameras, as disk in portable music players
- NVM: NonVolatile Memory
  - Write-Optimized and High-Performance Hasing Index Scheme for Persistent Memory, OSDI 2018

.

Disks



- Tracks are divided into sectors (512 bytes)
- Multiple tracks form a cylinder.
- SSD: Solid State Disk/Solid State Drive

I/O Devices

- Controller runs a device-accepts commands from the OS and executes them
- Complicated business
  - Eg. Gets command to read sector x on disk y. Must convert to (cylinder, sector, head) address, move arm to correct cylinder, wait for sector to rotate under the head, read and store bits coming off the drive, compute checksum, store bits as words in memory
  - Controller contains a computer to run its device

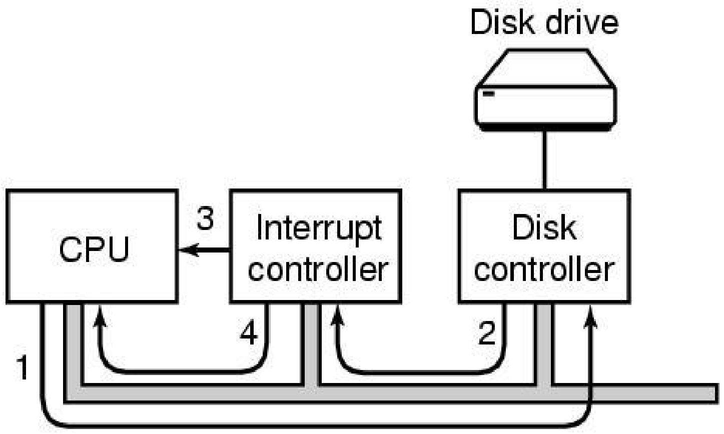
Device Driver

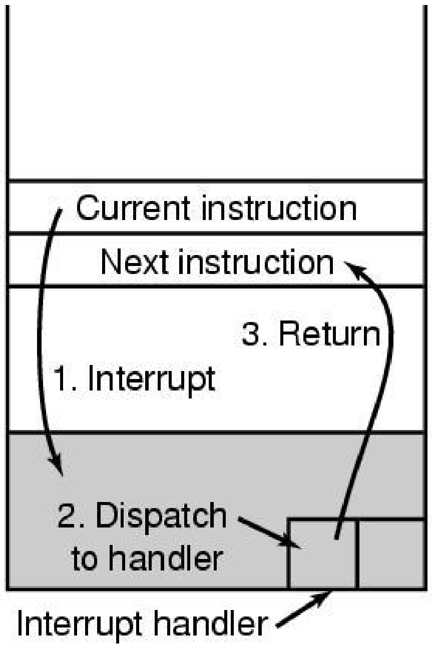
- OS software that talks to controller-gives commands, accepts responses
- Each controller manufacturer supplies a driver for each OS
- Driver runs in kernel mode
- Controller has registers which are used to communicate with the driver
- Three modes of communication
  - Polling
  - Interrupts
  - DMA

I/O by polling device

- Driver issues command to controller
- Driver polls device until it is ready
- Eg Send character to printer controller and poll until it is ready to accept the next character
- Big use of CPU
- Called programmed I/O-not really used any more

I/O by Interrupts



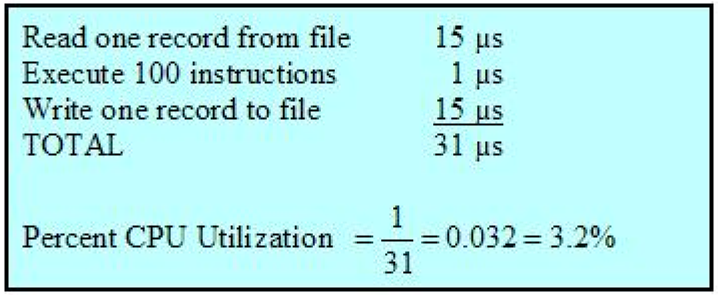


Generate an interrupt when I/O is finished.

Eg When character is finished being printed, interrupt CPU.
    Allows CPU to do something else while character is being printed

I/O by DMA

- Special (controller) chip
- Avoids using the CPU as part of the transfer to/from memory
- CPU tells chip to set up transfer and take care of it
- Chip does as it it told and interrupts CPU when it is finished



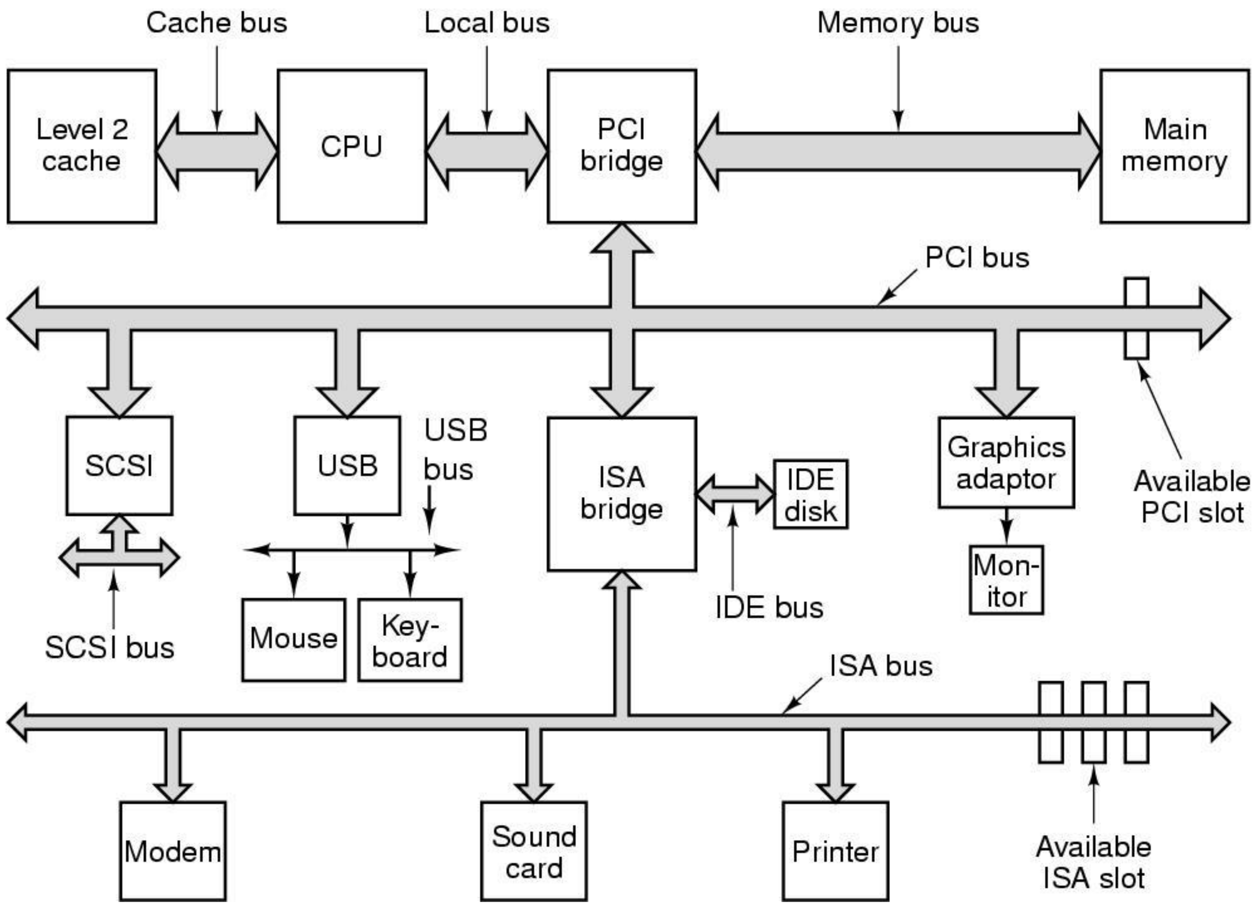
Read one record from file	15 µs
Execute 100 instructions	1 µs
Write one record to file	15 µs
TOTAL	31 µs

$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

The bus hierarchy

- In the beginning there was one bus-couldn’t handle the traffic when cpu and memories got faster and bigger
- Create a hierarchy of faster buses (PCI) and specialized buses (SCSI, USB)

Pentium System Buses



The structure of a large Pentium system

Hardware Features

- Privileged instructions
  - Certain machine level instructions can only be executed by the monitor
- Interrupts
  - Early computer models did not have this capability
- Memory protection
  - User program executes in user mode
  - Monitor executes in system mode

The Operating System Zoo

- Mainframe operating systems
  - Big-thousands of disks….
  - Lots of jobs with lots of I/O
  - Services-batch (payroll) transactions (airline reservations, timesharing (query database)
  - Elderly-Unix, Linux replacing them

The Operating System Zoo

- Server operating systems
  - Workstations
  - File, print, web servers
  - BSD, Linux, Windows

  - Multiprocessor operating systems
    - Use multiple cores

The Operating System Zoo

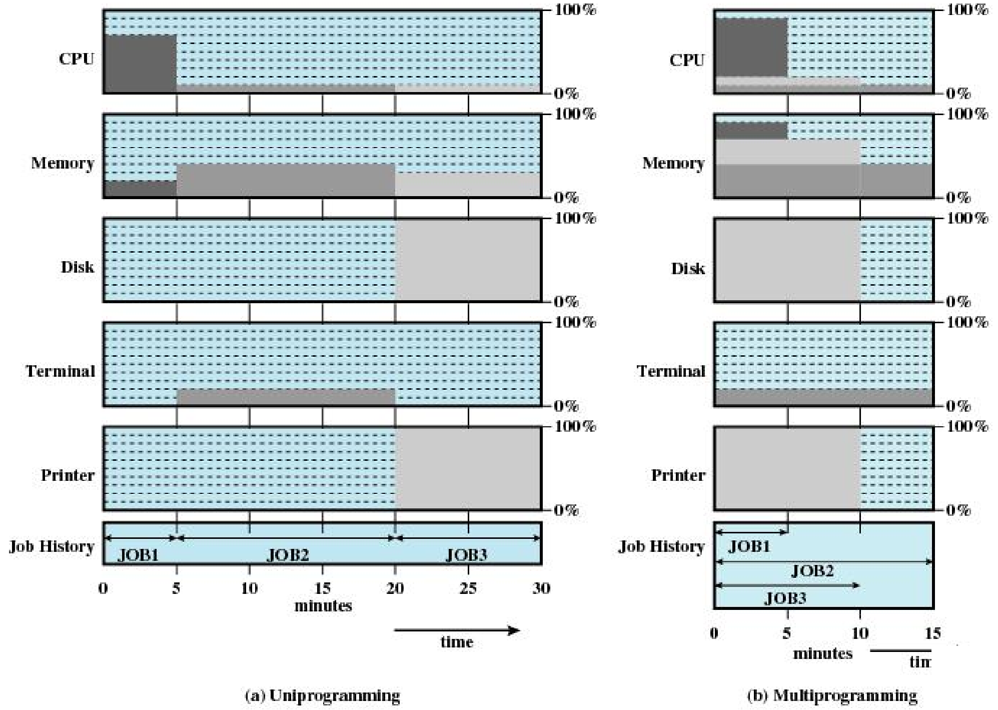


Figure 2.6 Utilization Histograms

Table 2.1 Sample Program Execution Attributes

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

The Operating System Zoo

- PC operating systems-Linux, Mac, Windows
- Smart phone operating systems- Android, iPhone, Blackberry
  - No hard disk
  - Palm, Symbian popular OS’s

The Operating System Zoo

- Embedded operating systems-TV sets, cars, DVDs, MP3s
    - Everything is in ROM (no apps can run on it)
    - QNx, Vxworks
- Real time operating systems
    - Hard (eg. factory) deadline
    - Soft (eg. multi-media) deadline

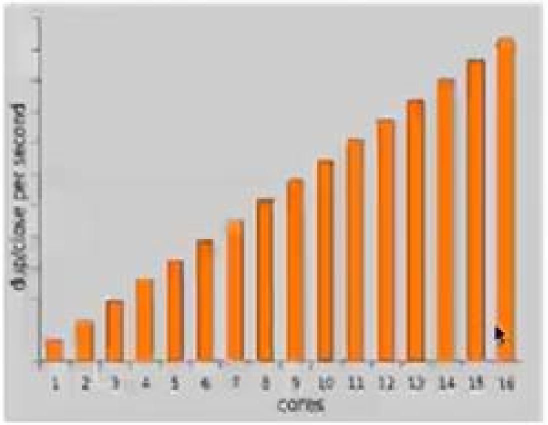
Smart card OS (eg border crossing cards)

- Java in ROM

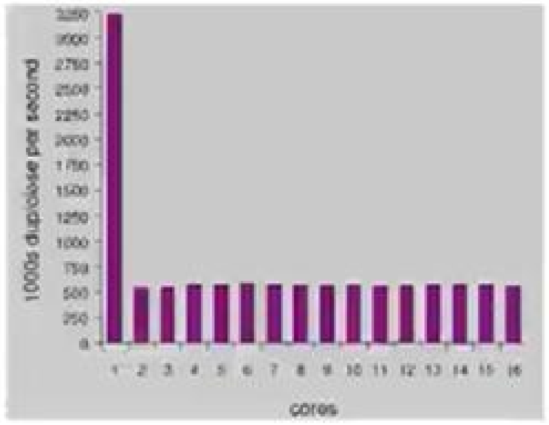
Challenges of Operating System

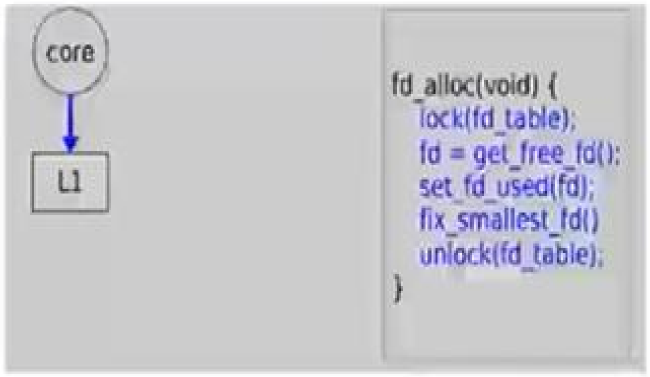
- Scale-up Problem

理想性能

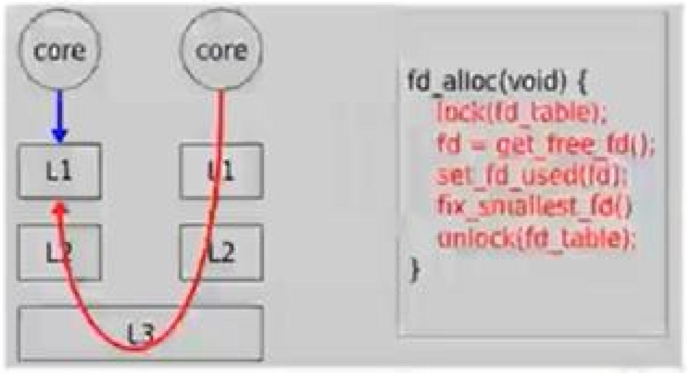


实际性能





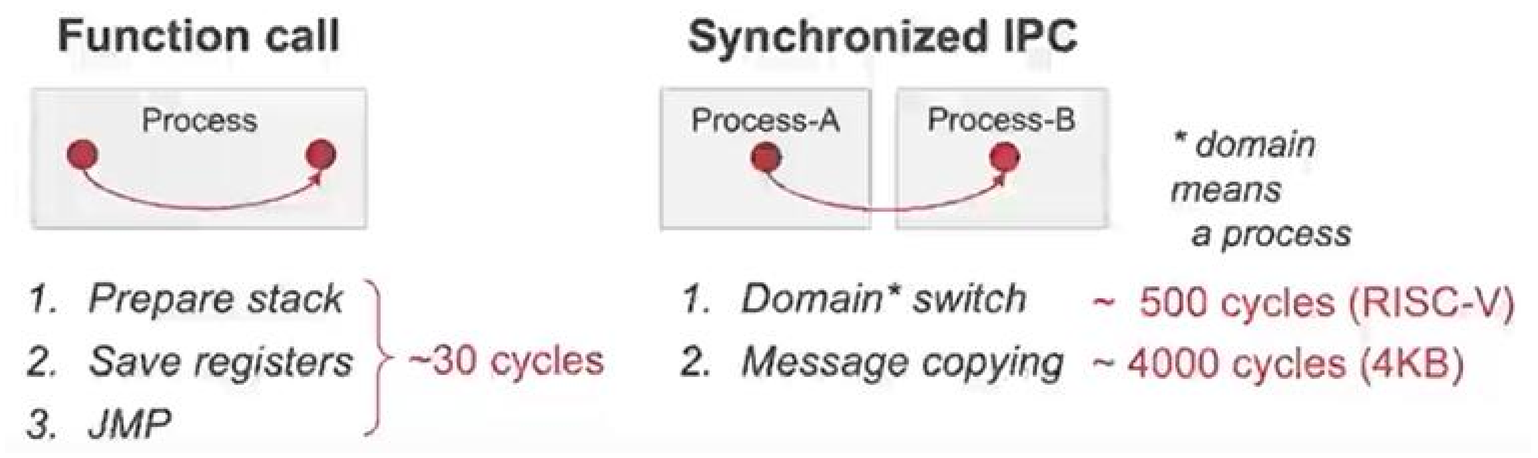
```
fd_alloc(void) {
  lock(fd_table);
  fd = get_free_fd();
  set_fd_used(fd);
  fix_smallest_fd()
  unlock(fd_table);
}
```



```
fd_alloc(void) {
  lock(fd_table);
  fd = get_free_fd();
  set_fd_used(fd);
  fix_smallest_fd()
  unlock(fd_table);
}
```

Challenges of Operating System

- Tradeoff between performance and isolation



1. *Prepare stack*
2. *Save registers*
3. *JMP*

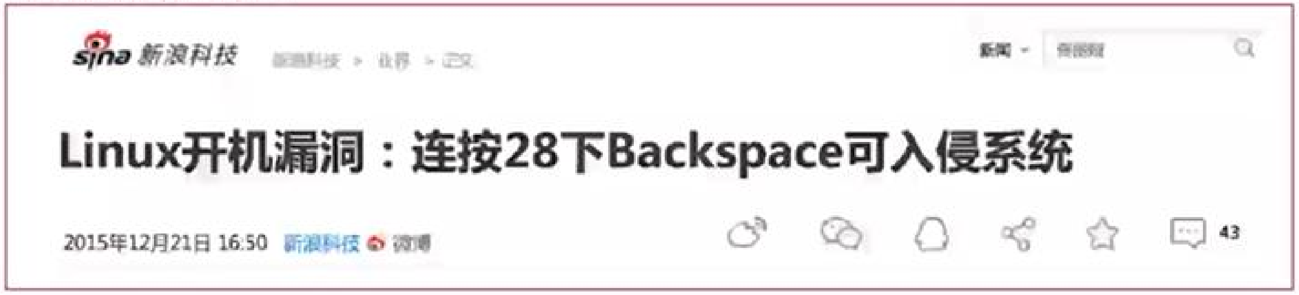
~30 cycles

1. *Domain* switch* ~ 500 cycles (RISC-V)
2. *Message copying* ~ 4000 cycles (4KB)

Challenges of Operating System

- security

CVE-2015-8370

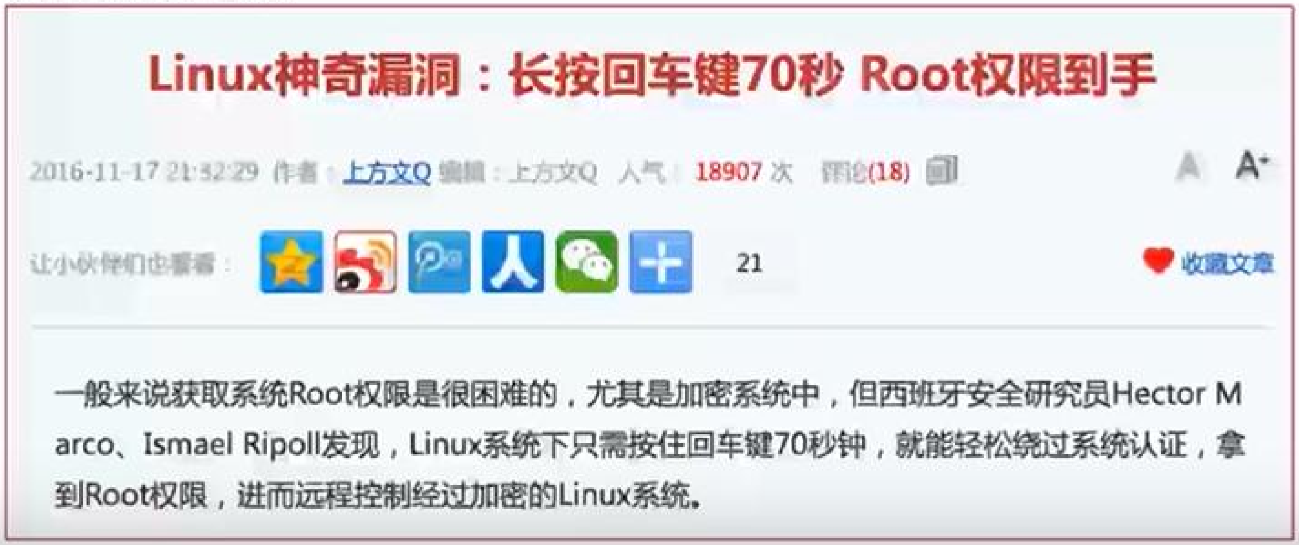


新浪科技 新浪科技 > 业界 > 正文

Linux开机漏洞：连按28下Backspace可入侵系统

2015年12月21日 16:50 新浪科技 微博 43

CVE-2016-4484



Linux神奇漏洞：长按回车键70秒 Root权限到手

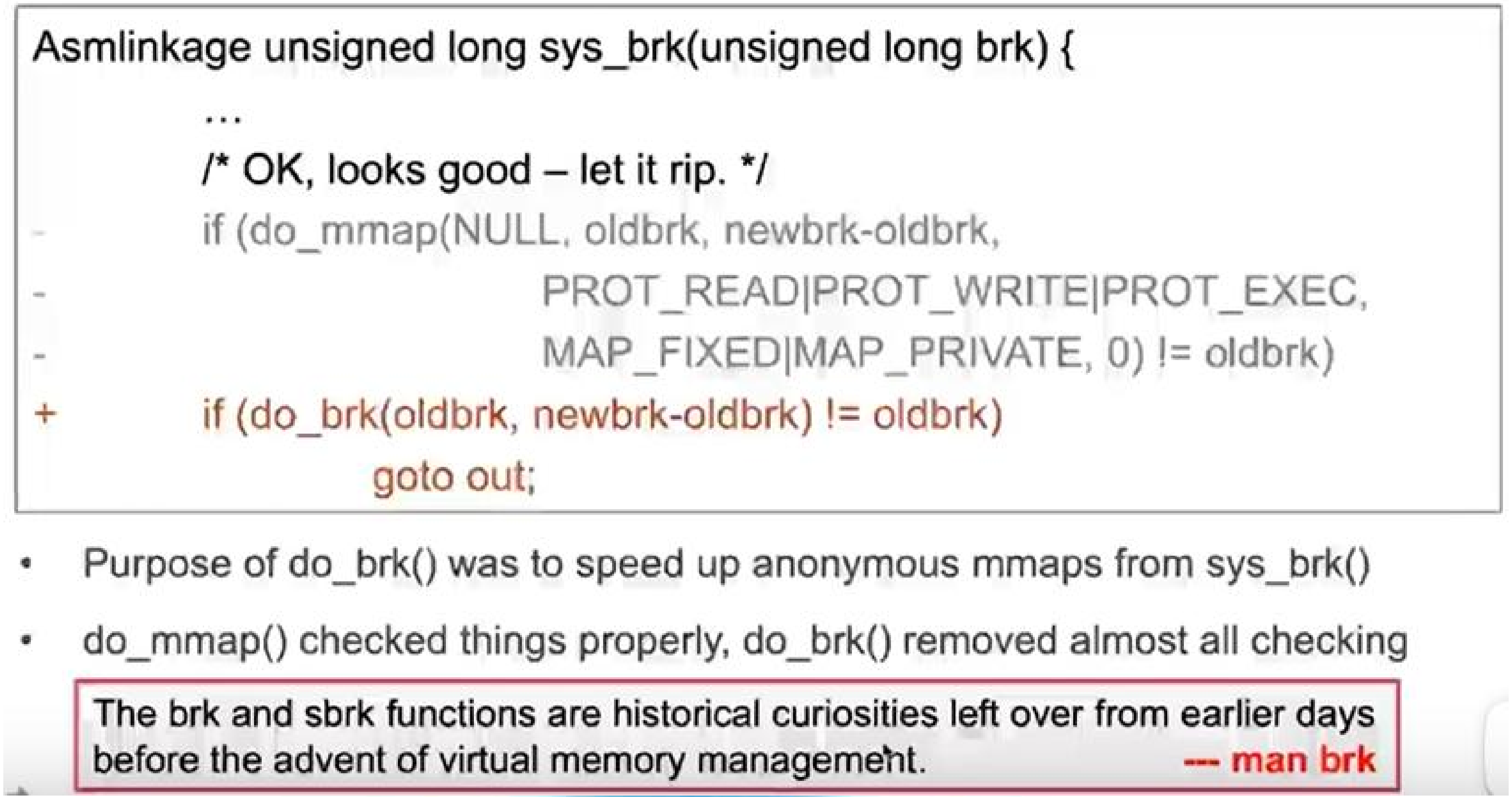
2016-11-17 21:32:29 作者：上方文Q 编辑：上方文Q 人气：18907 次 评论(18)

让小伙伴们也看看： 21 收藏文章

一般来说获取系统Root权限是很困难的，尤其是加密系统中，但西班牙安全研究员Hector Marco、Ismael Ripoll发现，Linux系统下只需按住回车键70秒钟，就能轻松绕过系统认证，拿到Root权限，进而远程控制经过加密的Linux系统。

Challenges of Operating System

- security



```
Asmlinkage unsigned long sys_brk(unsigned long brk) {
              …
              /* OK, looks good – let it rip. */
-             if (do_mmap(NULL, oldbrk, newbrk-oldbrk,
-                                    PROT_READ|PROT_WRITE|PROT_EXEC,
-                                    MAP_FIXED|MAP_PRIVATE, 0) != oldbrk)
+             if (do_brk(oldbrk, newbrk-oldbrk) != oldbrk)
                     goto out;
```

- Purpose of do_brk() was to speed up anonymous mmaps from sys_brk()
- do_mmap() checked things properly, do_brk() removed almost all checking

The brk and sbrk functions are historical curiosities left over from earlier days before the advent of virtual memory management. --- man brk

Challenges of Operating System

- New hardware is appeared.
  - Example: non-volatile memory
  - Is fsck() still good for using?
  - Better I/O performance?

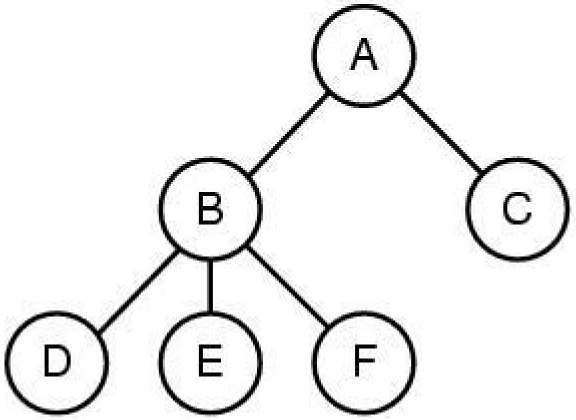
Operating System Concepts

- Processes
- Address spaces
- Files
- Input/Output
- Protection
- The shell

Processes

- Program in execution
- Lives in address space
- Process table
  - Keeps info about process
  - Used to re-start process
- Shell (command interpreter) reads commands from terminal

A Process Tree



Process creates child processes

Tree structure

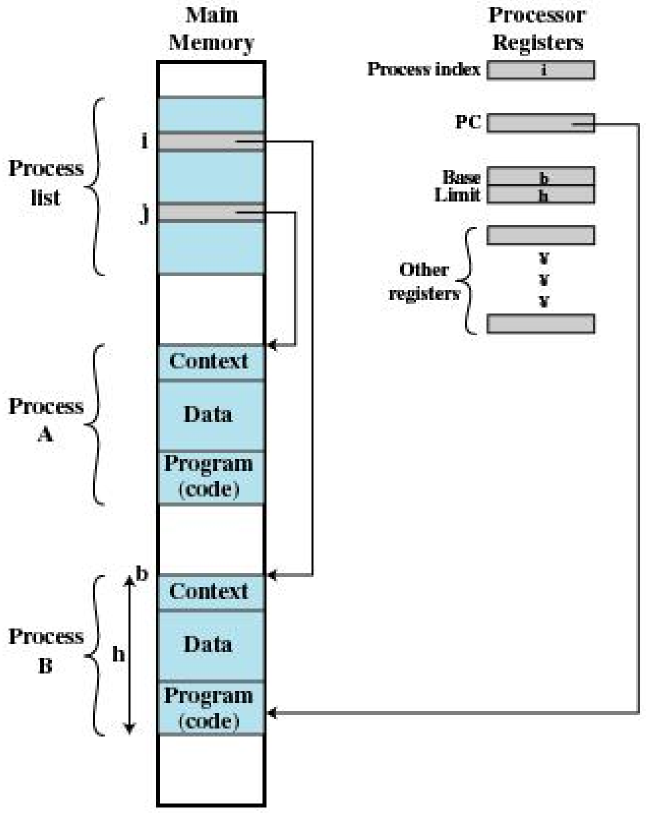
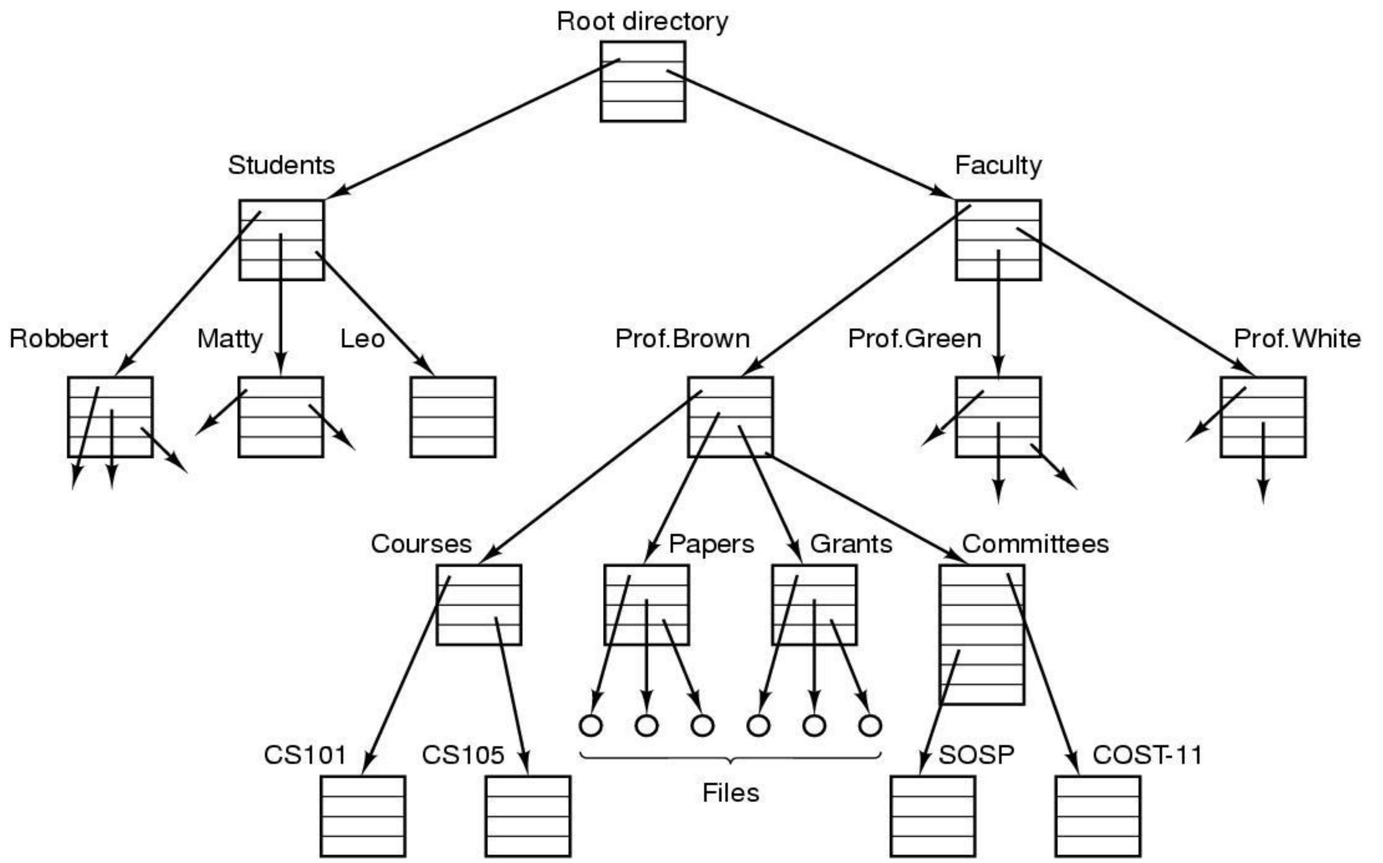


Figure 2.8 Typical Process Implementation

Process

- Have UID’s and group ID’s (GID)
- Can communicate with one another (IPC)
- Difficulties with Designing System Software
  - Improper synchronization: Ensure a process waiting for an I/O device receives the signal
  - Failed mutual exclusion
  - Nondeterminate program operation: Program should only depend on input to it, not on the activities of other programs
  - Deadlocks

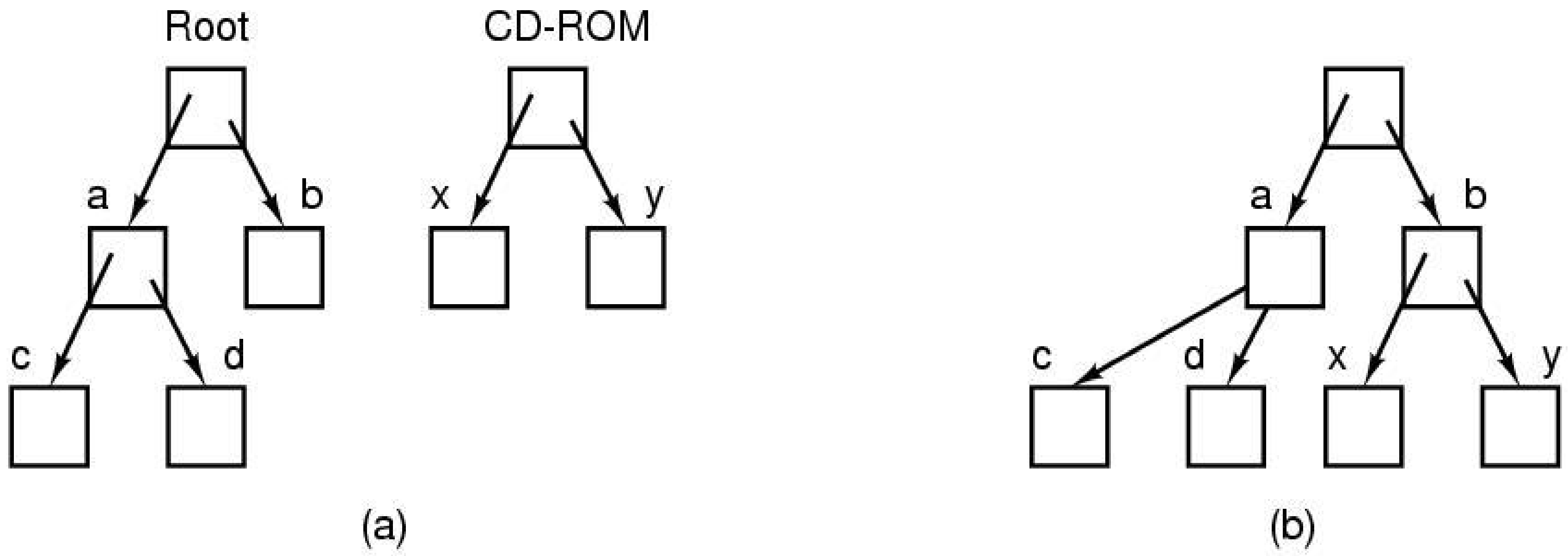
File directory



The directory is organized as a tree

Root directory at the top. Path proceeds from the root (e.g. faculty/prof brown/courses)

Mounting Files in UNIX



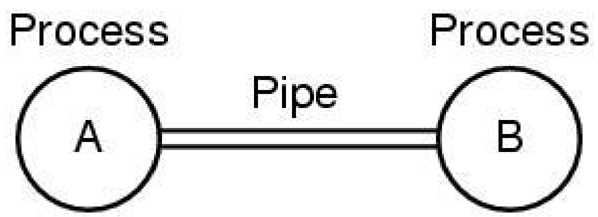
A CD-ROM is mounted on directory b.

Special files

- Special files-can use same calls for I/O as for files. OS treats them as files.
  - Block special files (disks)
  - Character special files (line printers, modems)
  - Kept in /dev directory, e.g. /dev/lp is line printer

Unix Pipe

Processes communicate by writing into/reading from a file in Unix



A and B write into the pipe and read from the pipe.

File Systems

- Kernel-level File Systems——Kernel-FS
- User-level File Systems—— User-FS
- Firmware-level File Systems——Firmware-FS
- CrossFS——OSDI 2020
- SplitFS——SOSP2019

Information Protection and Security

- Data integrity
  - Protection of data from unauthorized modification
- Authenticity
  - Concerned with the proper verification of the identity of users and the validity of messages or data

Scheduling and Resource Management

- Fairness
  - Give equal and fair access to resources
- Differential responsiveness
  - Discriminate among different classes of jobs
- Efficiency
  - Maximize throughput, minimize response time, and accommodate as many uses as possible

I/O, Protection/Shell

- I/O-big part of OS
- Protection-UNIX uses rwx bits for each file
  - 3 bits for owner, 3 for group, 3 for everyone else
- Shell (command interpreter)
  - UNIX
  - Has lots of flavors-sh,bash,csh,ssh…..
  - Sort<file1>file2
  - Cat file 1 file 2 file3 | sort > /dev/lp

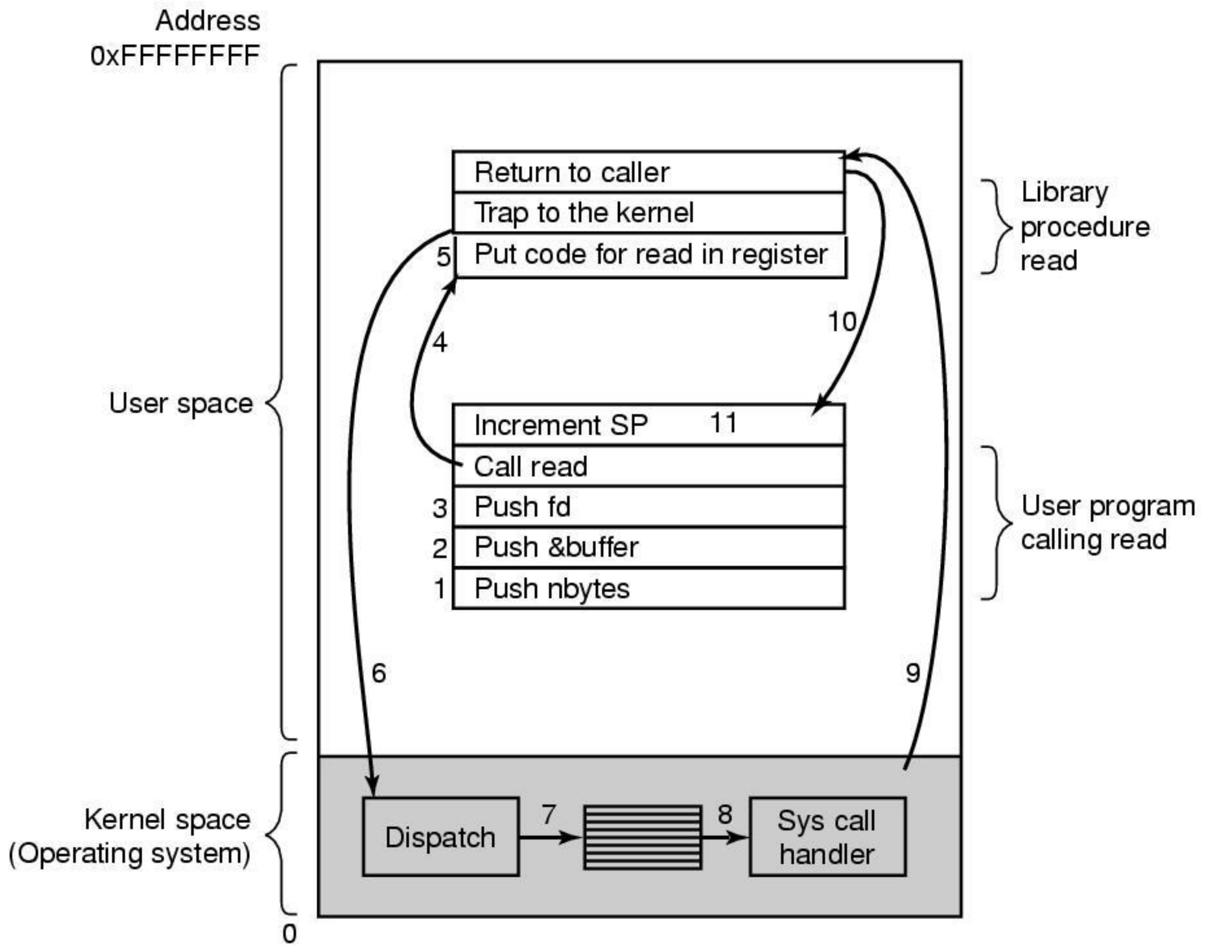
System Calls

- Interface between user programs and OS
- Varies from OS to OS
- System call issued by user program
- Call uses a library call of the same name
- Library routine puts machine into kernel modes (by issuing a special instruction)
- Finds actual routine for system call in a table
- Does the work involved in the call
- Returns to user program

Unix Read System Call

- Count=read(fd, buffer, nbytes)
  - fd is a file descriptor. When a file is opened, permissions are checked. If access is allowed, a number (fd) is returned. Then file can be read/written
  - nbytes is number of bytes in file
  - buffer is where read deposits the bytes

System Calls



read(fd, buffer, nbytes).

Look at System Calls

- Start with process management
  - See how they are used in shell
  - Then you write a shell
- Then do a brief summary of calls for
  - File management
  - Other useful calls

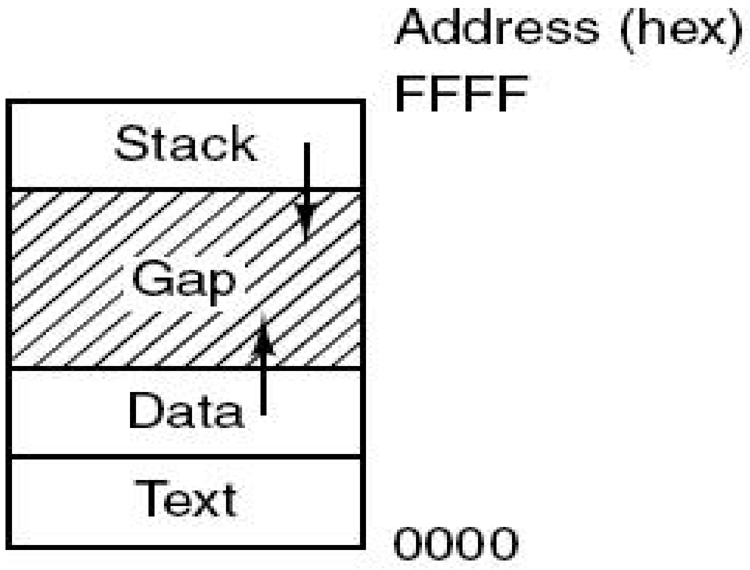
System Calls for Process Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Memory Layout

Processes have three segments:
text, data, and stack.



Shell

```
#define TRUE 1

while (TRUE) {                                      /* repeat forever */
      type_prompt( );                               /* display prompt on the screen */
      read_command(command, parameters);            /* read input from terminal */

      if (fork( ) != 0) {                           /* fork off child process */
          /* Parent code. */
          waitpid(−1, &status, 0);                  /* wait for child to exit */
      } else {
          /* Child code. */
          execve(command, parameters, 0);           /* execute command */
      }
}
```

Fork

- Pid=fork( )
  - Duplicates parent process completely Everything duplicated -data,registers,fd’s
  - Fork returns a value (pid)
    - Zero in child
    - Child’s PID in parent
    - Used to differentiate child from parent

Execve

- Replaces process' core image by the file named in its invocation
- Execve(name,arg,environp) has 3 parameters
    - Name of file (e.g. cp command)
    - Arg-pointer to argument array
    - Environp-pointer to environment array

Excve (example)

- Cp f1 f2 is located by execve and parameters are passed to it by execve

  - Main(argc, argv, envp) is main prog in cp
    - Argc is #items on command line (=3 in cp)
    - Argv points to array (arg[0] points to cp, arg[1] points to f1,
    - Envp points to environment, an array of name=value with info like terminal type

System Calls for File Management

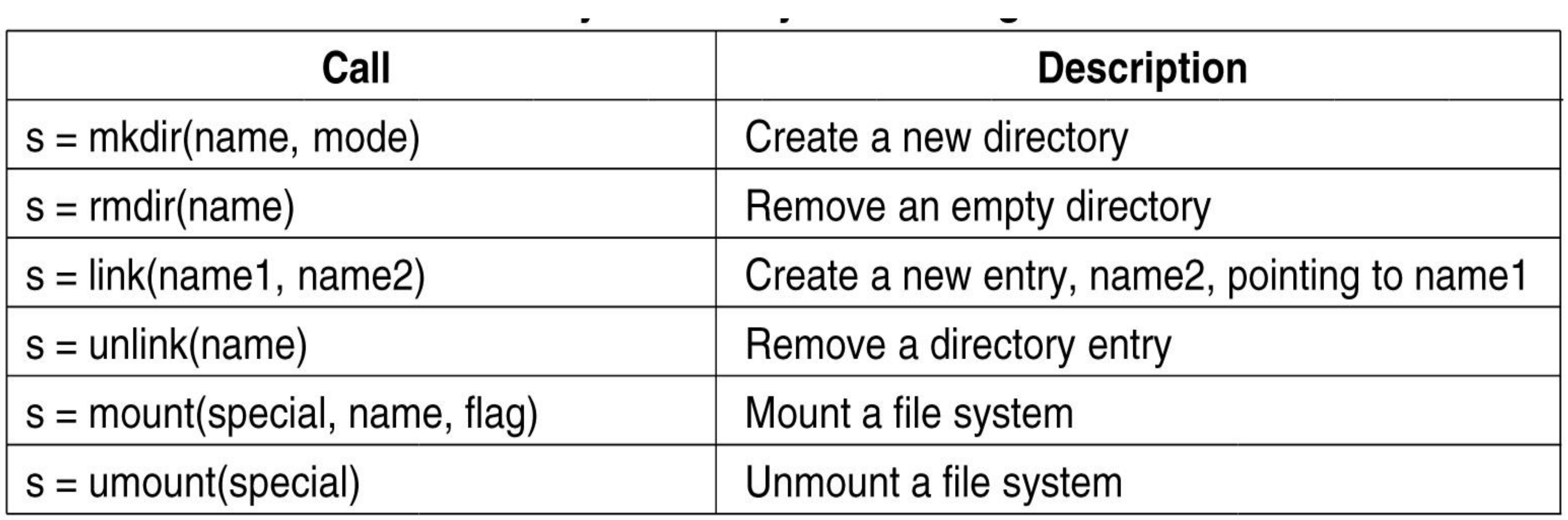
File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

File mode

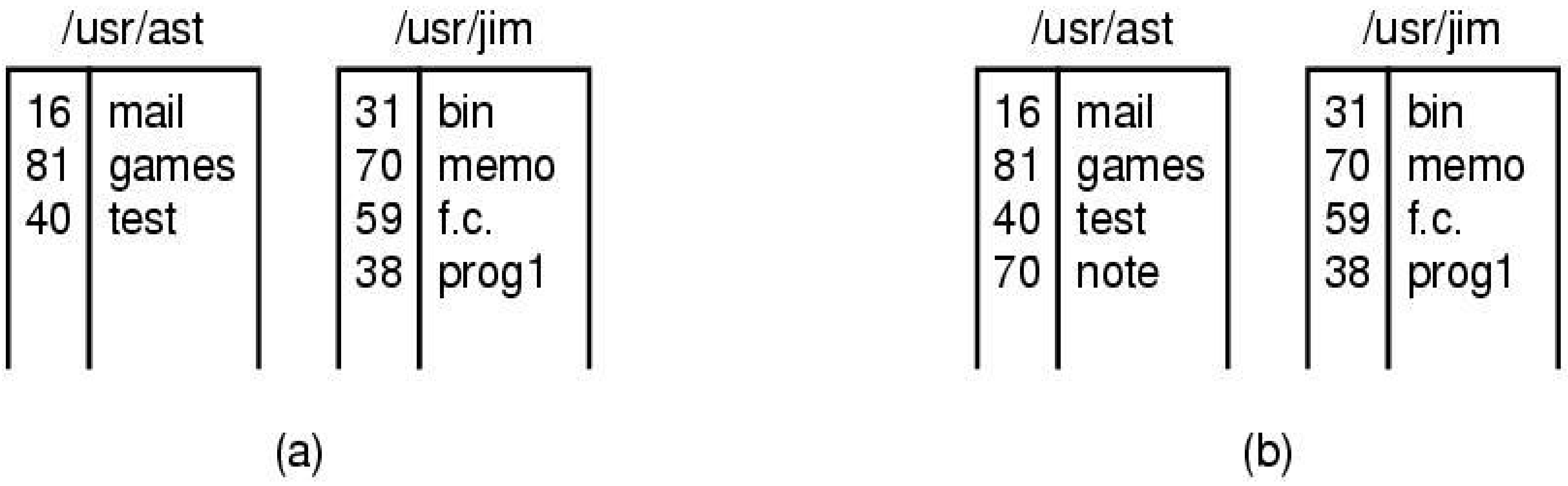
- System keeps track of it
  - Regular, special
  - Date of creation
  - Size
  - Access status of file via stat command

System Calls for Directory and File Management



Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Linking



/usr/ast

16	mail
81	games
40	test

/usr/jim

31	bin
70	memo
59	f.c.
38	prog1

(a)

/usr/ast

16	mail
81	games
40	test
70	note

/usr/jim

31	bin
70	memo
59	f.c.
38	prog1

(b)

- In Unix, each file is identified by an i-number
- i-number indexes into i-node table
  - Link creates a new directory entry with same i-number
  - Has a new name (note instead of memo)

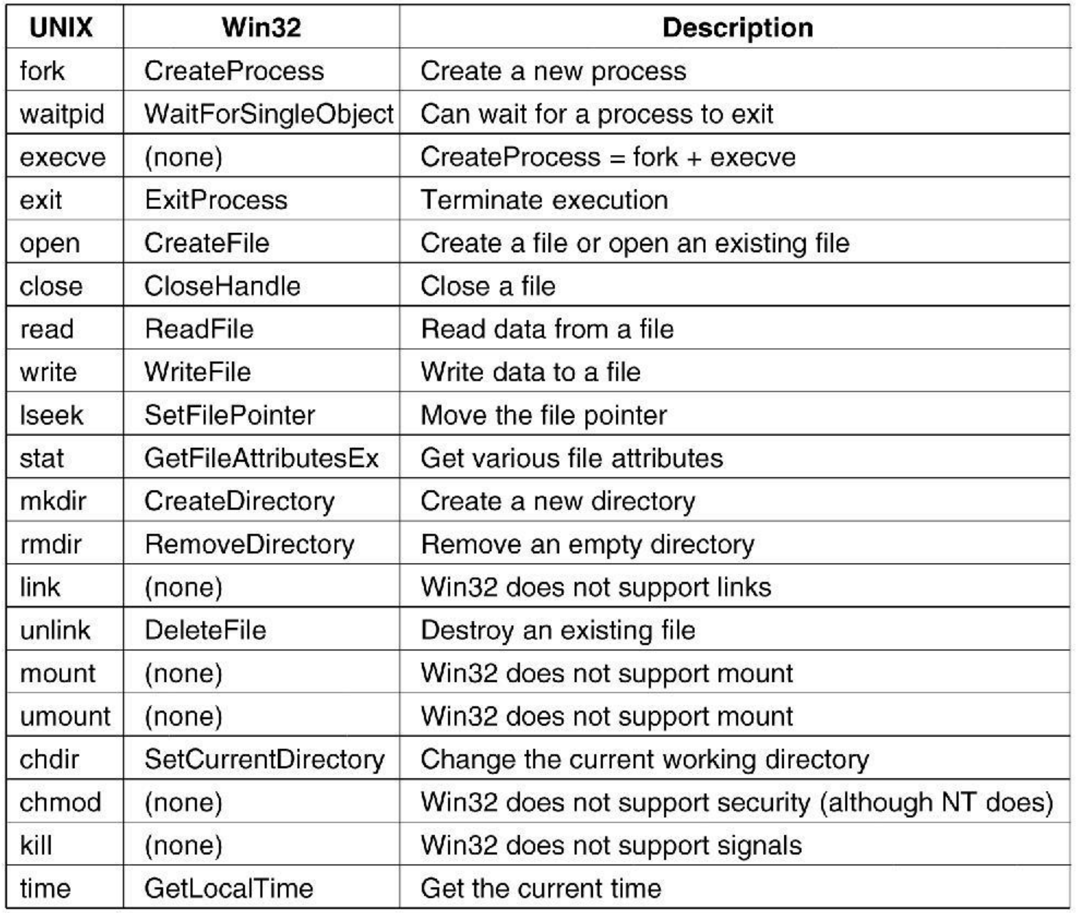
Mount System Call

- mount(“/dev/fd0”, “/mnt”, 0) is system call
- File comes from drive 0 and is mounted on binary file /mnt.
- Third parameter tells if it is read or read-write
- Result is that file on drive zero can be accessed from a directory
- Saw this example before with CD-same mechanism for memory sticks and portions of hard drives

Other System Calls

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Windows Win32 API



UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

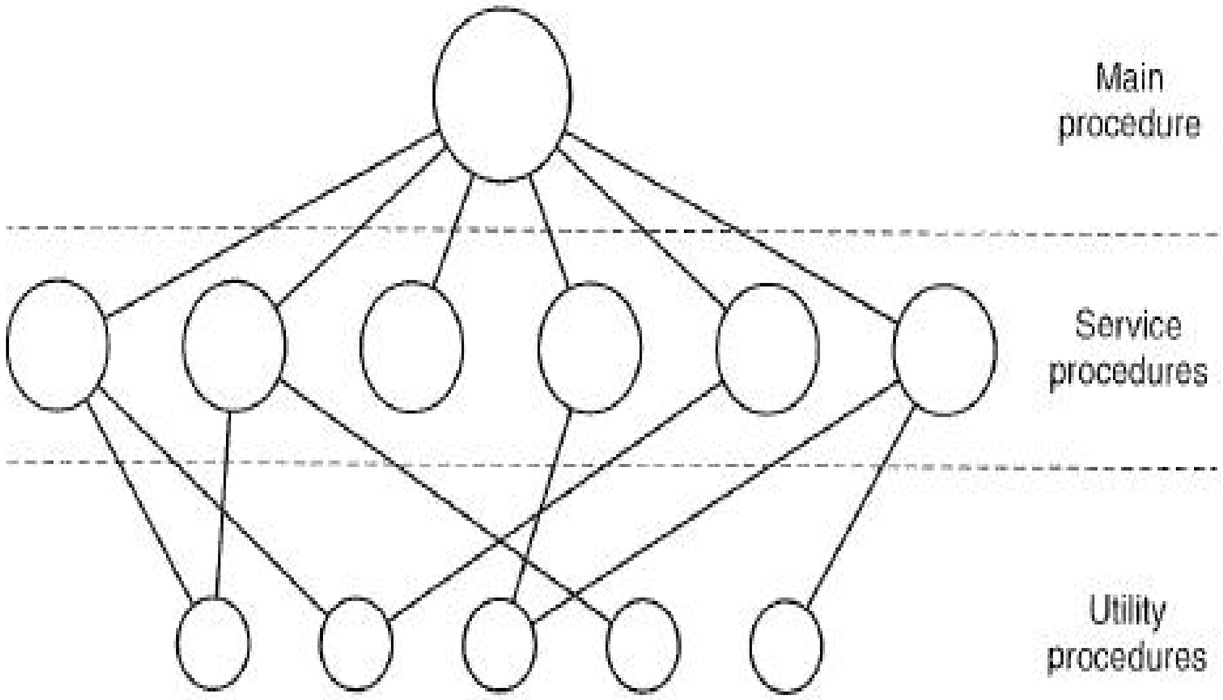
The Win32 API calls that roughly correspond to the UNIX calls

Operating Systems Structure

1. Monolithic systems

- A main program that invokes the requested service procedure.
- A set of service procedures that carry out the system calls.
- A set of utility procedures that help the service procedures.

1. Monolithic Systems



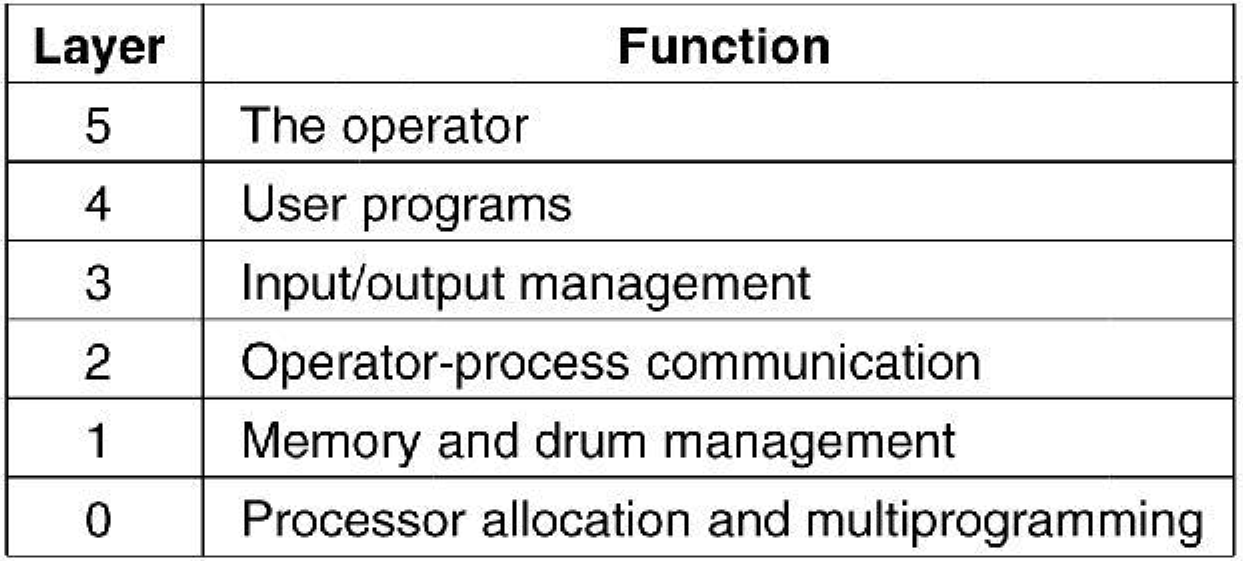
A simple structuring model for a monolithic system.

1. Monolithic Systems

```
#define FALSE  0
#define TRUE   1
#define N      2                       /* number of processes */

int turn;                              /* whose turn is it? */
int interested[N];                     /* all values initially 0 (FALSE) */

void enter_region(int process);        /* process is 0 or 1 */
{
    int other;                         /* number of the other process */

    other = 1 – process;               /* the opposite of process */
    interested[process] = TRUE;        /* show that you are interested */
    turn = process;                    /* set flag */
    while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)         /* process: who is leaving */
{
    interested[process] = FALSE;       /* indicate departure from critical region */
}
```

A simple structuring model for a monolithic system.

2. Layered Systems-THE operating system

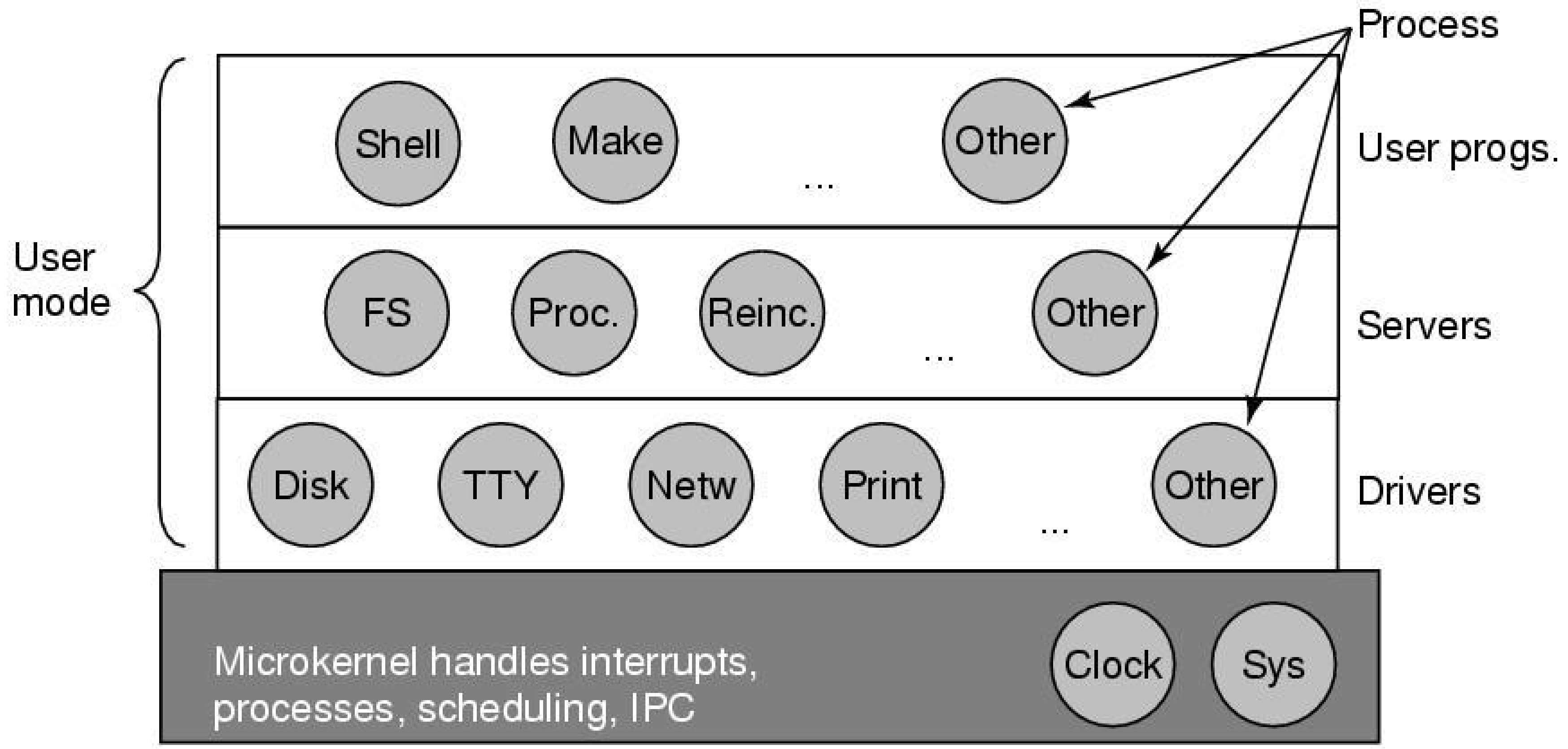


Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

3. Microkernels

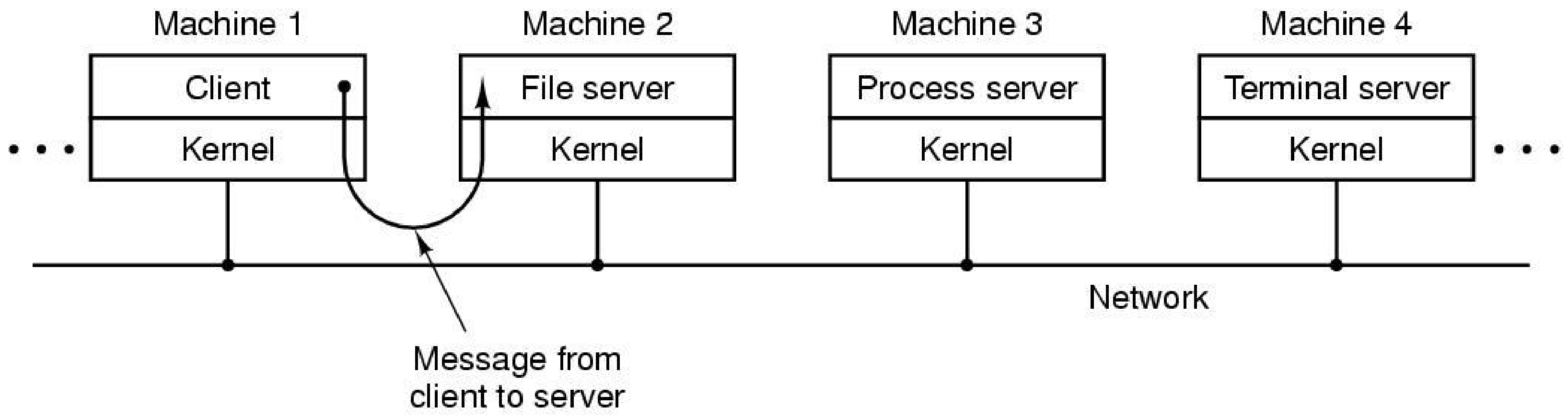
- Small number of processes are allowed in the kernel
- Minimizes effects of bugs
    - Don’t want bug in driver to crash system
- Put mechanism in kernel and policy outside the kernel
    - Mechanism- schedule processes by priority scheduling algorithm
    - Policy-assign process priorities in user space

3. Microkernels



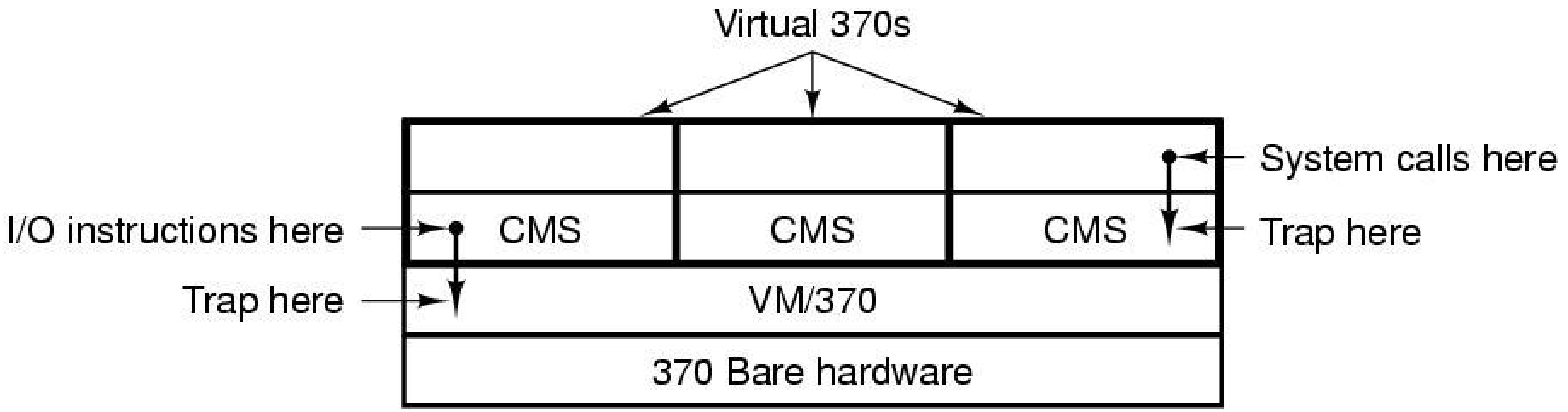
Structure of the MINIX 3 system.

4. Client-Server Model



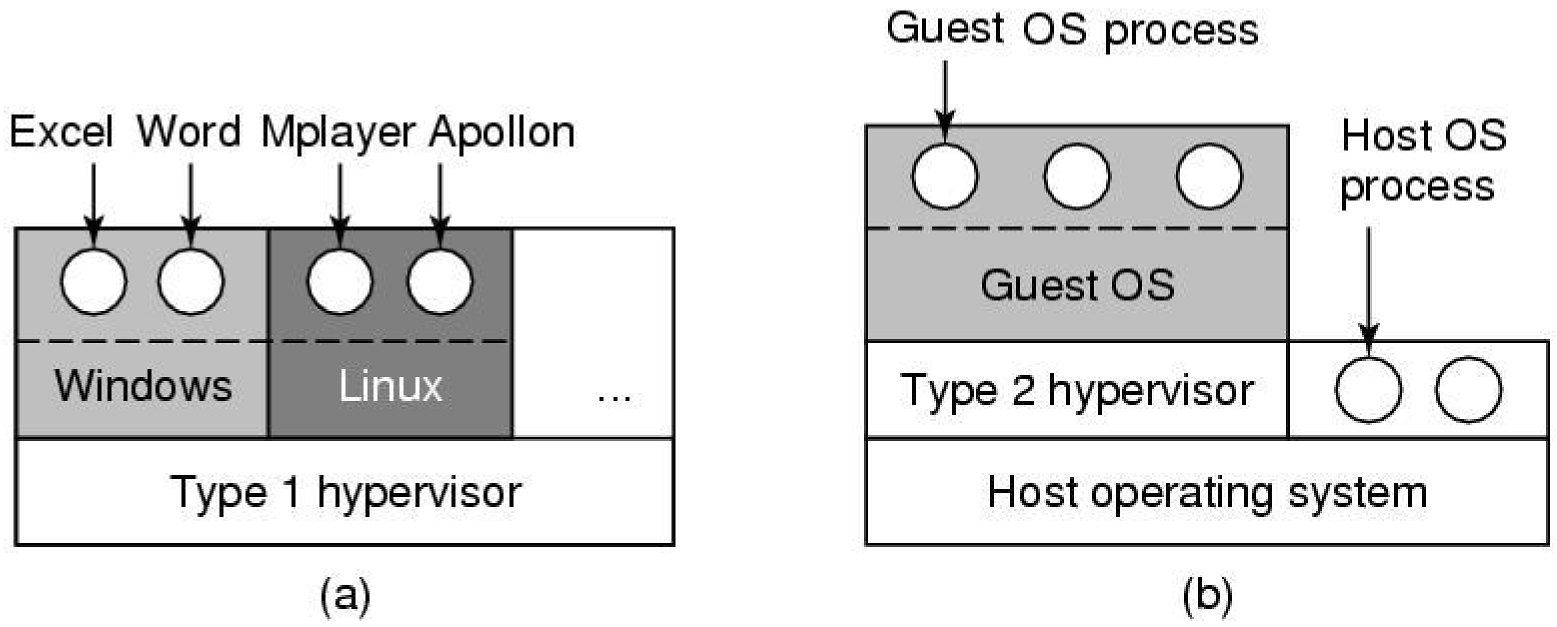
The client-server model over a network.

5. Virtual Machines



The structure of VM/370 with CMS.

5. Virtual Machines



(a) A type 1 hypervisor. (b) A type 2 hypervisor.

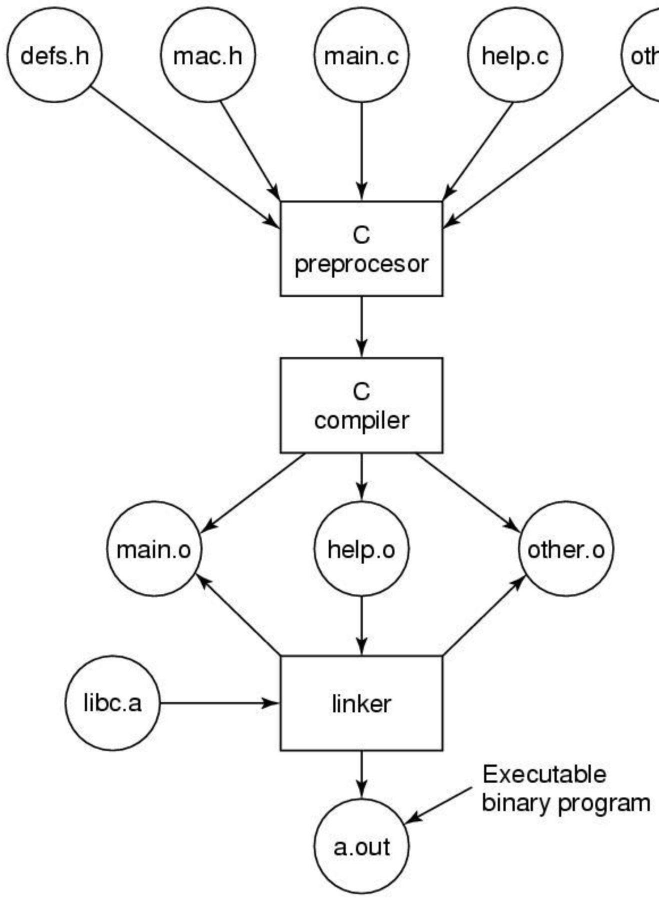
6.exokernel

- Idea: give each user a subset of the resources of the actual machine.
- Exokernel (1995): allocate resources to the virtual machines and then check attempts to use them to make sure no machine is trying to use some system.
- Advantage
  - save a layer of mapping.
  - Separating the multiprogramming from the user operating system code, but with less overhead.

The World According to C

- The C language
- Header files
- Large programming projects
- The model of run time

The Model of Run Time



The process of compiling C and header files to make an executable.

Reading Materials

- New Operating Systems——LegoOS，LinnOS
- Principles of Virtual Machines
- Reliability and Correctness of operating systems
- Other topics——Storage, Consistency，Machine Learning, Bugs, Consensus, Scheduling, Hardware(FPGA),Security, Clusters

Problems

- Which of the following instructions should be allowed only in kernel mode?
  - (a) Disable all interrupts.
  - (b) Read the time-of-day clock.
  - (c) Set the time-of-day clock.
  - (d) Change the memory map.
- A file whose file descriptor is fd contains the following sequence of bytes: 3, 1, 4, 1, 5,9, 2, 6, 5, 3, 5. The following system calls are made:

  lseek(fd, 3, SEEK_SET);

  read(fd, &buffer, 4);

where the lseek call makes a seek to byte 3 of the file. What does buffer contain after the read has completed?

Problems

- Suppose that a 10-MB file is stored on a disk on the same track (track 50) in consecutive sectors. The disk arm is currently situated over track number 100. How long will it take to retrieve this file from the disk? Assume that it takes about 1ms to move the arm from one cylinder to the next and about 5 ms for the sector where the beginning of the file is stored to rotate under the head. Also, assume that reading occurs at a rate of 200 MB/s.
- A portable operating system is one that can be ported from one system architecture to another without any modification. Explain why it is infeasible to build an operating system that is completely portable. Describe two high-level layers that you will have in designing an operating system that is highly portable.

谢 谢！