第一章 绪论

翁楚良

https://chuliangweng.github.io

2023 春 ECNU

微处理器时代(1980-)

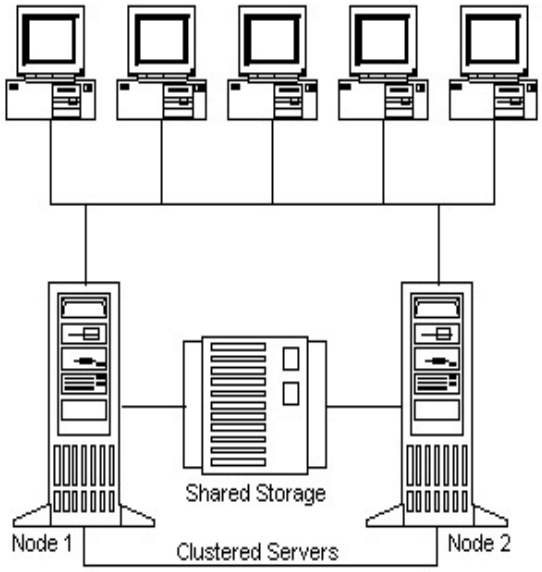
- 随着大规模集成电路的发展，芯片在每平方厘米的硅片上可以集成数千个晶体管，于是个人计算机时代到来了。
- 个人计算机和工作站领域中主流操作系统
  - MS - DOS广泛用于IBM PC及其他采用Intel 80X86芯片的计算机
  - UNIX在工作站和高档计算机领域（如网络服务器）占据了统治地位，尤其对于采用高性能RISC芯片的计算机
- 操作系统的特点：
  - 人机交互性好：在调试和运行程序时由用户自己操作。
  - 共享主机：多个用户同时使用。
  - 用户独立性：对每个用户而言好象独占主机。

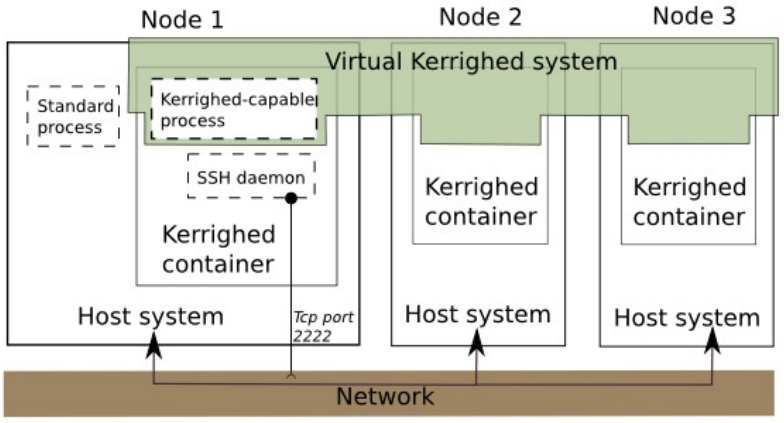
网络/分布式操作系统

- Network Operating System (NOS):
  - provides mainly file sharing.
  - Each computer runs independently from other computers on the network.
- Distributed Operating System (DOS):
  - gives the impression there is a single operating system controlling the network.
  - network is mostly transparent – it's a powerful virtual machine.

集群系统/网格系统

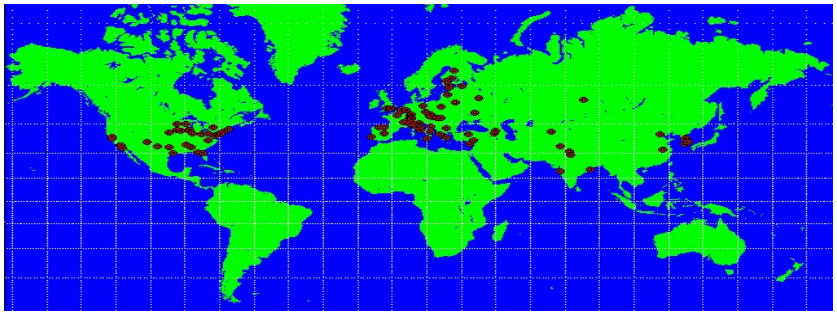
- Clustered Systems
  - Clustering allows two or more systems to share external storage and balance CPU load.
    - *Asymmetric clustering*: one server runs the application while other servers standby.
    - *Symmetric clustering*: all N hosts are running the application.
  - Closely coupled system:
    - processors also have their own external memor
    - communication takes place through high-speed channels.
    - Provides high reliability.
- 实例
  - Kerrighed, OpenSSI, openMosix





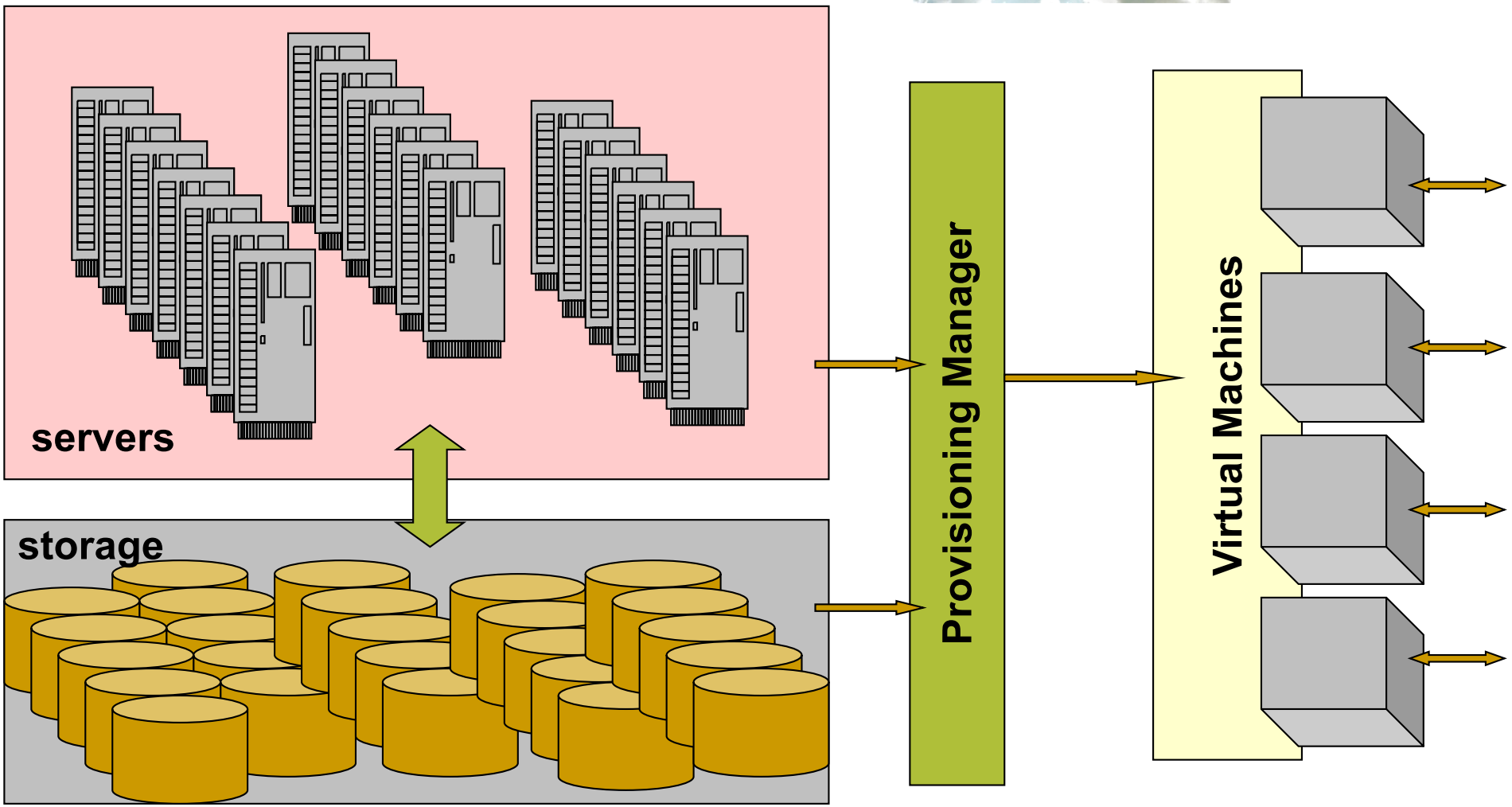
集群系统/网格系统

- Grid Computing System
  - 建立在Internet技术、web技术、高性能计算等技术之上的综合软硬件的基础设施，采用开放标准，为动态参与的多个机构所组成的虚拟组织协同完成某类科学、工程或工业上的应用提供可扩展的、安全、一致的、普及的、高效的大规模资源有效共享。
  - 共享主要不在于文件交换，而在于对计算机、软件、数据和其它资源的直接接入使用，这是工业界、科学界中大量出现的协同解决问题和资源代理策略的需要。
  - 这种共享必须被高度控制，资源提供者和消费者要清晰和详细的定义哪些资源可被共享，谁可享用这些资源，及共享发生的条件。

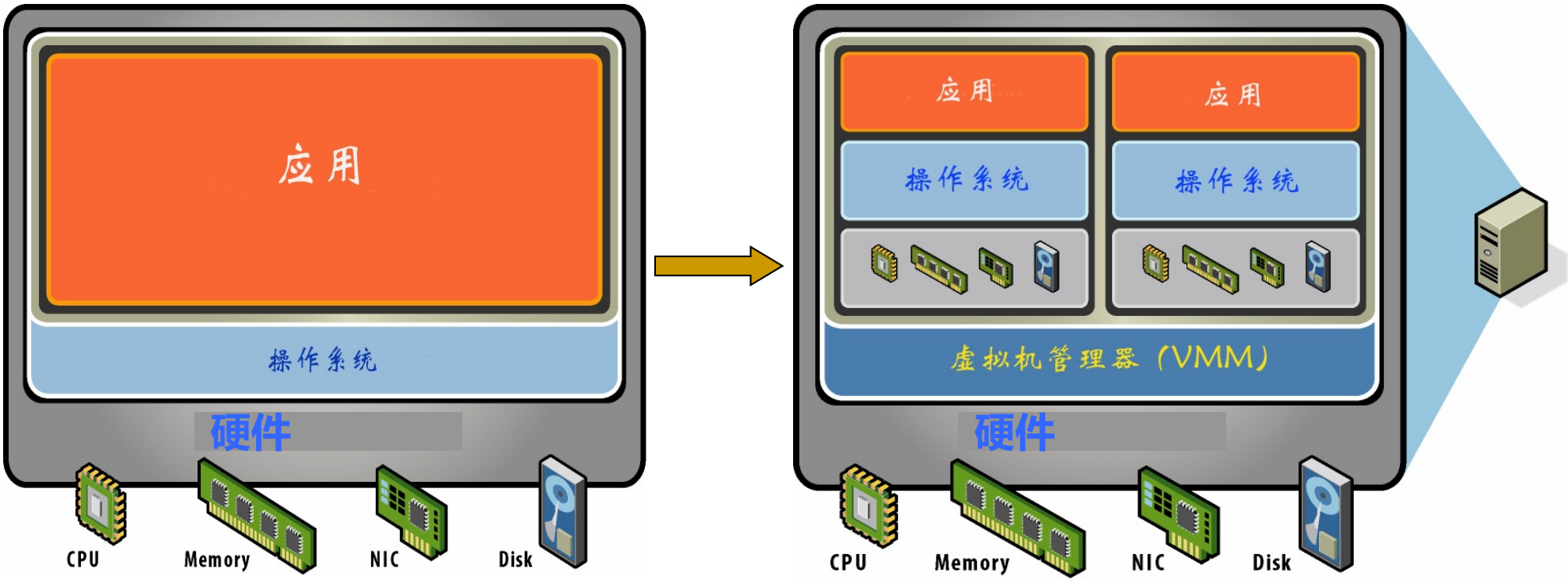


云计算(IaaS)

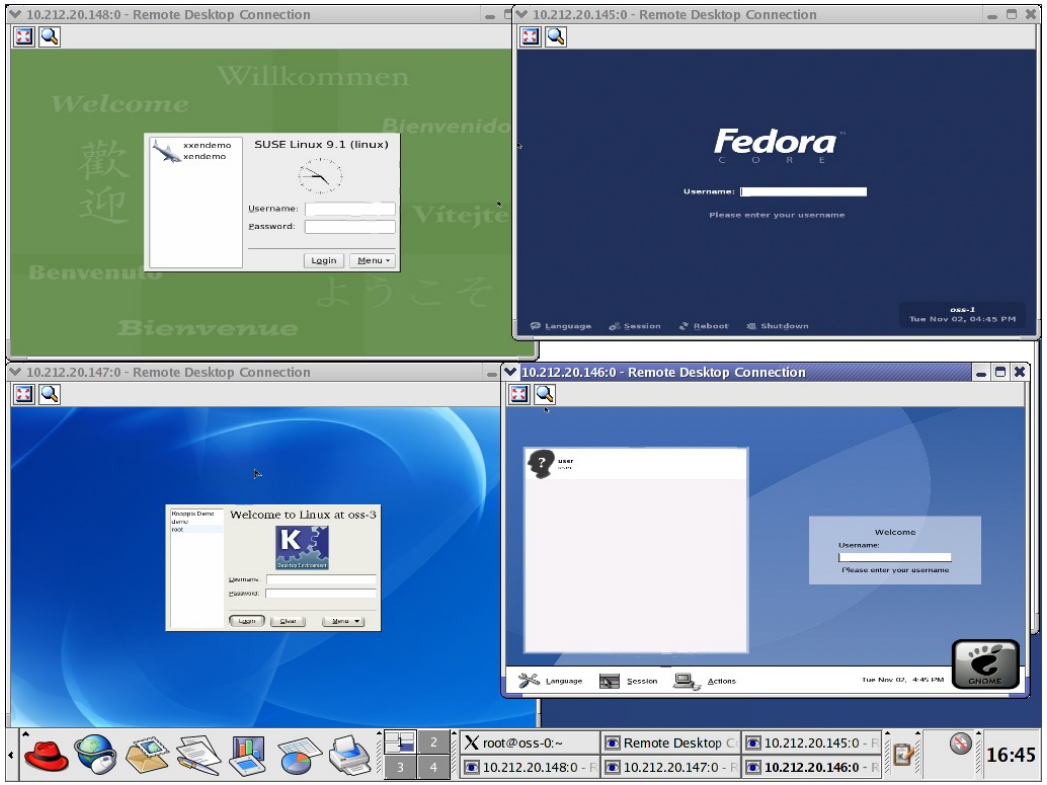
amazon web services™



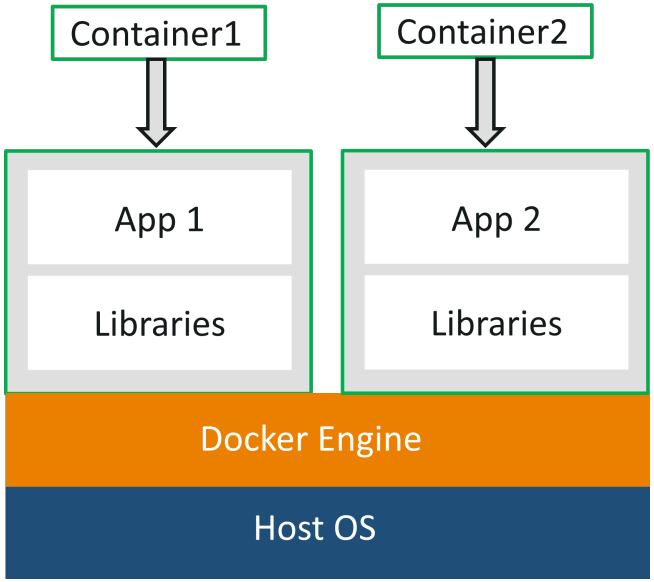
虚拟化技术


应用
操作系统
硬件
CPU
Memory
NIC
Disk
应用
应用
操作系统
操作系统
虚拟机管理器（VMM）
硬件
CPU
Memory
NIC
Disk

虚拟化技术


10.212.20.148:0 - Remote Desktop Connection
Willkommen
Welcome
Bienvenido
歡迎
Vítejte
ようこそ
Benvenuto
Bienvenue
xxendemo
xendemo
SUSE Linux 9.1 (linux)
Username:
Password:
Login
Menu
10.212.20.145:0 - Remote Desktop Connection
Fedora
CORE
Username:
Please enter your username
Language
Session
Reboot
Shutdown
10.212.20.147:0 - Remote Desktop Connection
Welcome to Linux at oss-3
10.212.20.146:0 - Remote Desktop Connection
Welcome
Username:
Please enter your username
Language
Session
Actions
GNOME
root@oss-0:~
Remote Desktop C
10.212.20.145:0
10.212.20.148:0
10.212.20.147:0
10.212.20.146:0
16:45

虚拟机


Container1
Container2
App 1
App 2
Libraries
Libraries
Docker Engine
Host OS

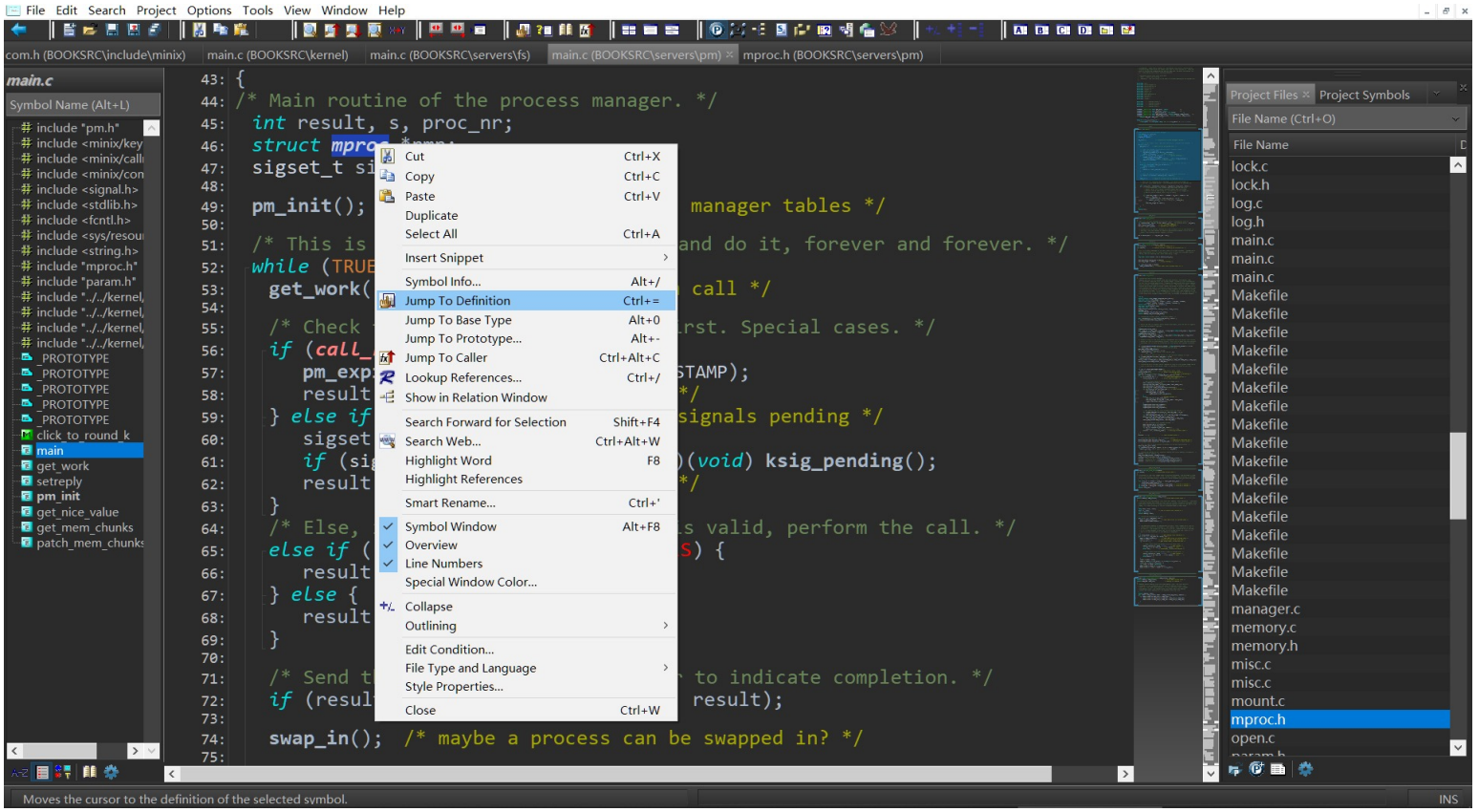
容器

MINIX历史

- 坦尼鲍姆等编写一个在用户看来与UNIX完全兼容，然而内核全新的操作系统
  - 名字由来： mini-UNIX
  - 1987年发布第一版
  - 不受任何商业许可证的限制，适用于教学
  - 内核代码量：4000
  - 比UNIX晚出现十年，并且其代码采用了一种更加模块化的组织方法
  - 是Linux的“前辈”
- 目前的版本：MINIX3
  - 本课程以MINIX 3作为例子

分析代码工具

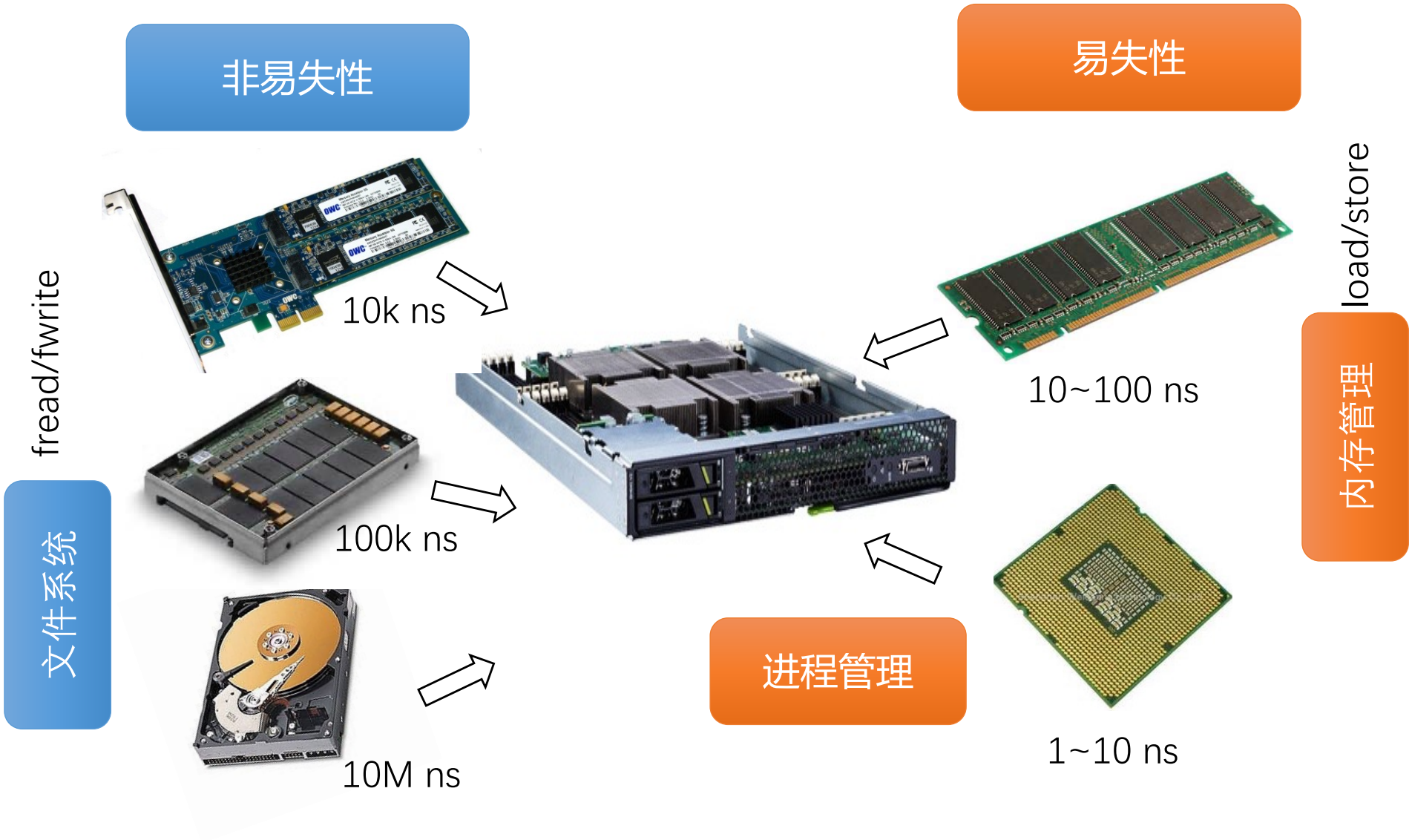
- Sourceinsight：分析源码工具



第一章绪论 提纲

- 1.1 什么是操作系统
- 1.2 操作系统的发展历史
- 1.3 操作系统基本概念
- 1.4 操作系统系统调用
- 1.5 操作系统组织结构
- 1.6 常用操作系统简介

计算机系统

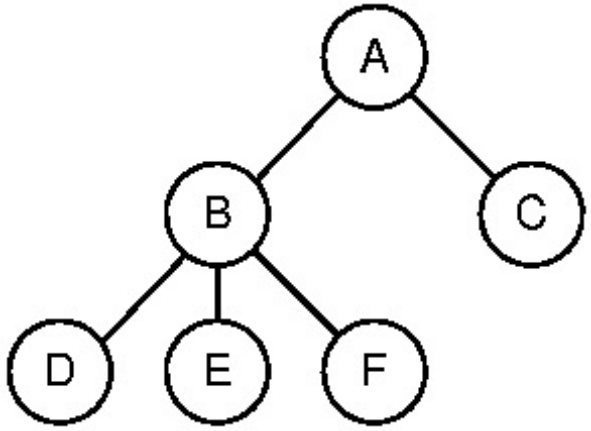

非易失性
易失性
fread/fwrite
load/store
文件系统
内存管理
进程管理
10k ns
100k ns
10M ns
10~100 ns
1~10 ns

系统调用

- 操作系统与用户程序的界面由操作系统提供的“扩展指令”集定义，即系统调用
  - 与进程相关的系统调用
    - 创建进程、进程间通信、终止进程
  - 与文件系统相关的系统调用
    - mount文件系统、读写文件等

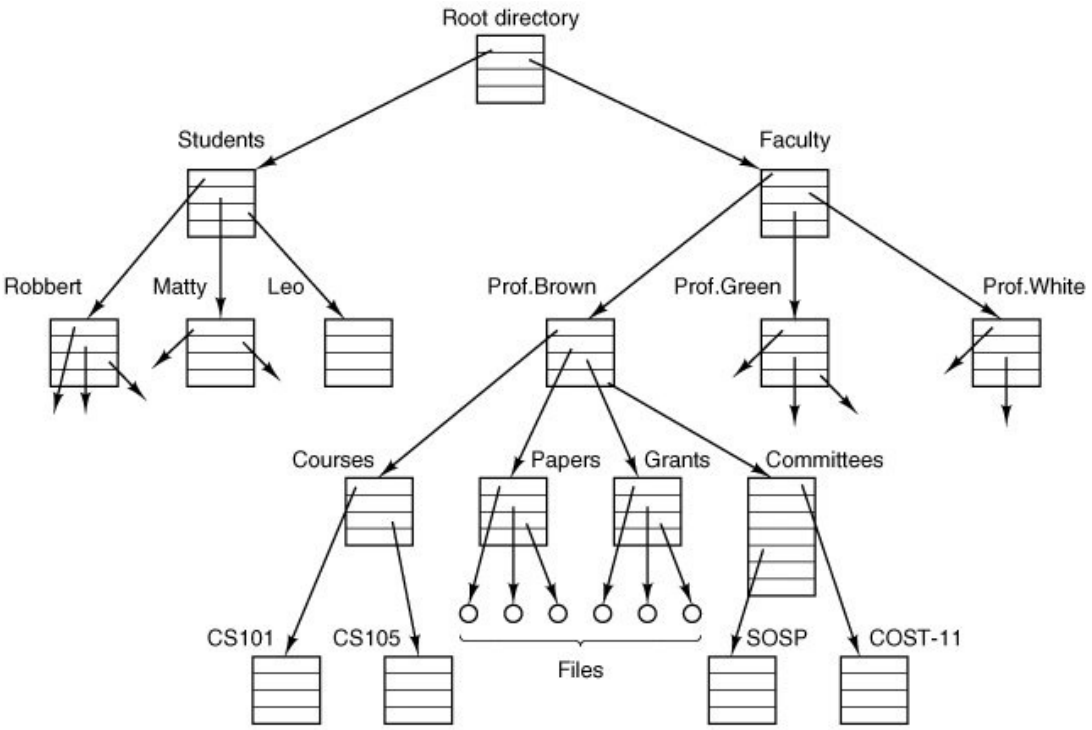
进程

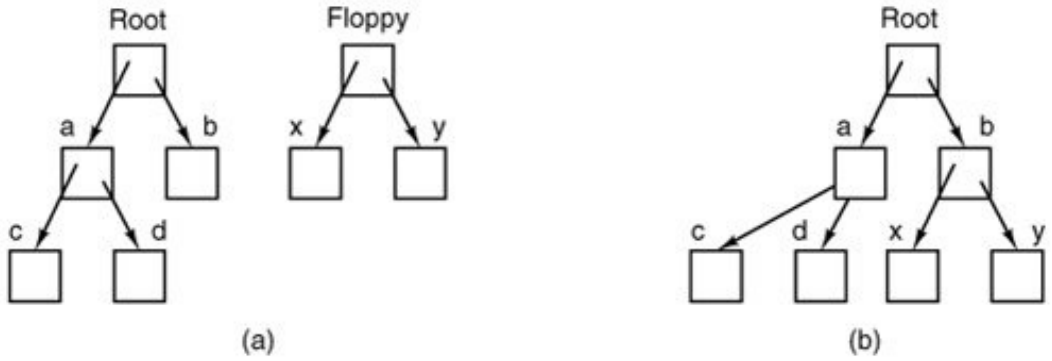
- 进程：正在执行的程序
  - 地址空间：其中包括可执行程序、程序数据和栈
  - 进程表项：寄存器、程序计数器、堆栈指针等信息
- 一个进程可以创建一个或多个子进程
- 进程间的协同由系统提供的进程间通信(IPC)机制实现



文件

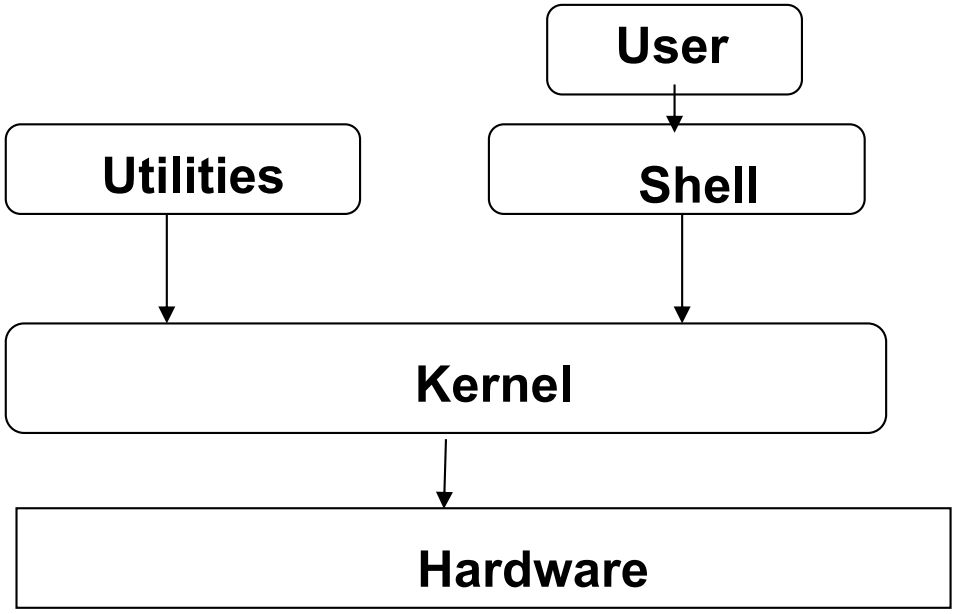
- 以文件的方式组织计算机中的存储数据
  - 操作过程：创建、打开、读写、关闭、删除
- 文件的存放通过目录完成，一个目录下可以存放一组文件





命令解释器Shell

- 本身不是操作系统的一部分，但体现了操作系统的许多特性并很好地说明了系统调用的具体用法，同时它也是终端用户与操作系统之间的界面



操作系统的特征

- 并发(concurrency)
- 共享(sharing)
- 虚拟(virtual)
- 异步性(asynchronism)

并发(concurrency)

- 多个事件在同一时间段内发生。操作系统是一个并发系统，各进程间的并发，系统与应用间的并发。操作系统要完成这些并发过程的管理。并行(parallel)是指在同一时刻发生。
  - 在多道程序处理时，宏观上并发，微观上交替执行（在单处理器情况下）。
  - 程序的静态实体是可执行文件，而动态实体是进程（或称作任务），并发指的是进程。

共享(sharing)

- 多个进程共享有限的计算机系统资源，操作系统要对系统资源进行合理分配和使用，资源在一个时间段内交替被多个进程所用。
  - 互斥共享（如音频设备）：资源分配后到释放前，不能被其他进程所用。
  - 同时访问（如可重入代码，磁盘文件）
  - 资源分配难以达到最优化

虚拟(virtual)

- 一个物理实体映射为若干个对应的逻辑实体：分时或分空间。虚拟是操作系统管理系统资源的重要手段，可提高资源利用率。
  - CPU —— 每个用户（进程）的"虚处理机"
  - 存储器 —— 每个进程都占有的地址空间（指令 + 数据 + 堆栈）
  - 显示设备 —— 多窗口或虚拟终端(virtual terminal)

异步性(asynchronism)

- 也称不确定性，指进程的执行顺序和执行时间的不确定性。
  - 进程的运行速度不可预知：分时系统中，多个进程并发执行，"时走时停"，不可预知每个进程的运行推进快慢
  - 判据：无论快慢，应该结果相同——通过进程互斥和同步手段来保证
  - 难以重现系统在某个时刻的状态（包括重现运行中的错误）
  - 性能保证：实时系统与分时系统相似，但通过资源预留以保证性能

第一章绪论 提纲

- 1.1 什么是操作系统
- 1.2 操作系统的发展历史
- 1.3 操作系统基本概念
- 1.4 操作系统系统调用
- 1.5 操作系统组织结构
- 1.6 常用操作系统简介

操作系统提供的服务

- 服务类型
  - 程序执行和终止（包括分配和回收资源）
  - I/O操作
  - 文件系统操作
  - 通信：本机内，计算机之间（通常通信服务的使用者为进程，而不是笼统说"主机"）
  - 配置管理：硬件、OS本身、其他软件
  - 差错检测
- 服务提供方式：系统命令和系统调用

类UNIX的系统调用

- 示例：READ系统调用
  - 三个参数：第一个指定所操作的文件，第二个指定使用的缓冲区，第三个指定要读的字节数。在C程序中调用该系统调用的方法如下：

```
count = read(file, buffer, nbytes);
```

  - 该系统调用将真正读到的字节数返回给count变量，正常情况下这个值与nbytes相等，但当读至文件结尾符时则可能比nbytes小。
  - 若由于参数非法或磁盘操作错误导致该系统调用无法执行，则count被置为-1，同时错误码被放在一个全局变量errno中。程序应该经常检查系统调用的返回值以确定其是否正确地执行。

系统调用的分类

- 进程管理: fork, waitpid, getpid, …
- 信号管理: kill, alarm, pause, sigaction, …
- 文件管理: creat, open, close, read, write, dup, …
- 目录管理: mkdir, mount, link, umount, chdir, …
- 安全管理: chmod, chwon, umask, getuid, …
- 时间管理: time, stime, utime, times

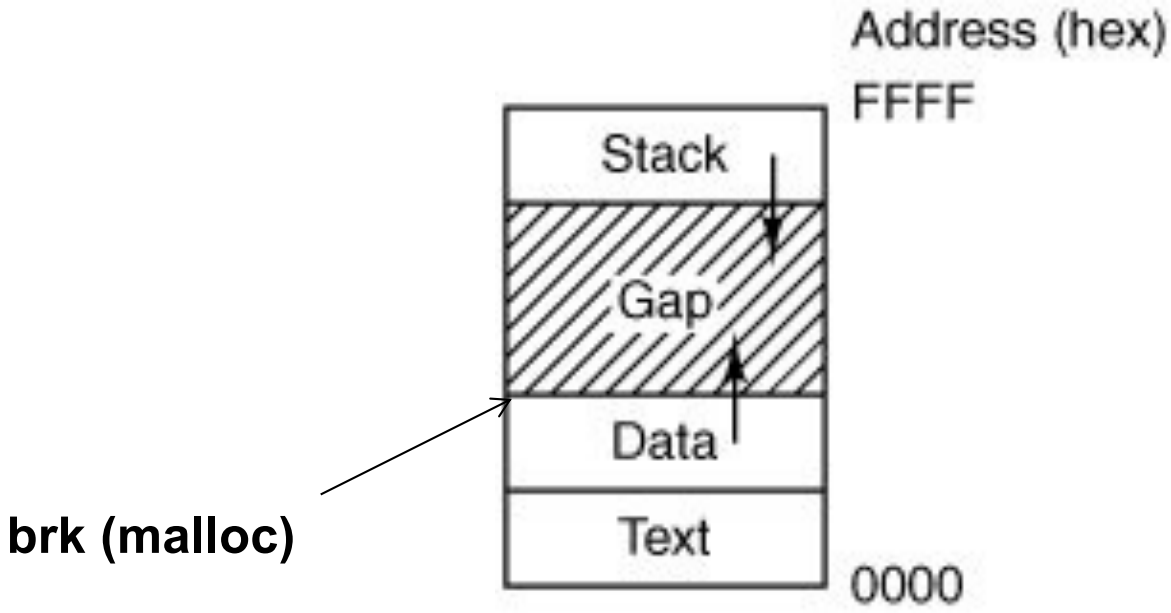
进程管理系统调用

```
#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 */
    }
}
```

MINIX中进程的内存空间

- 代码段
- 数据段
- 栈段



系统调用的分类

- 进程管理: fork, waitpid, getpid, …
- 信号管理: kill, alarm, pause, sigaction, …
- 文件管理: creat, open, close, read, write, dup, …
- 目录管理: mkdir, mount, link, umount, chdir, …
- 安全管理: chmod, chwon, umask, getuid, …
- 时间管理: time, stime, utime, times

信号管理系统调用

- 信号是软件中断，信号提供了一种处理异步事件的方法：终端用户键入中断键，则会通过信号机构停止一个运行的程序。
- 定义针对信号的处理操作

```
sigaction(sig, &act, &oldact);
```

```
sigaction(SIGINT, SIG_IGN, NULL);
```

- 发送信号给进程

```
kill(pid, sig)
```

系统调用的分类

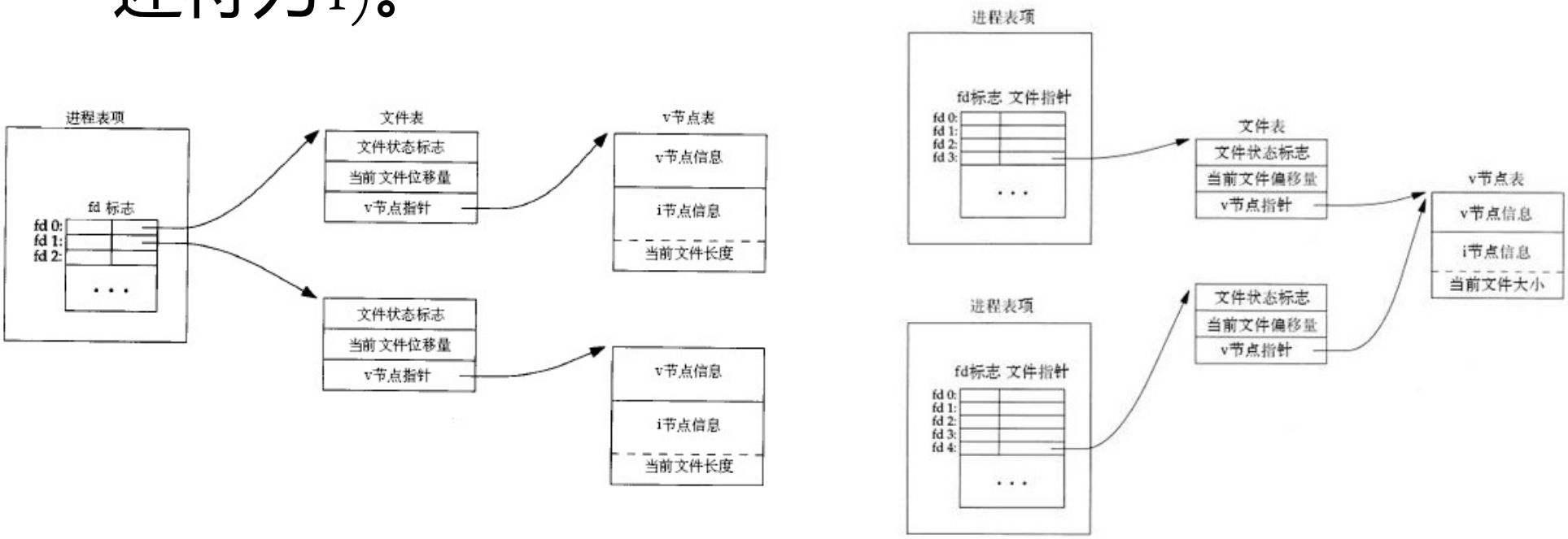
- 进程管理: fork, waitpid, getpid, …
- 信号管理: kill, alarm, pause, sigaction, …
- 文件管理: creat, open, close, read, write, dup, …
- 目录管理: mkdir, mount, link, umount, chdir, …
- 安全管理: chmod, chwon, umask, getuid, …
- 时间管理: time, stime, utime, times

文件系统系统调用

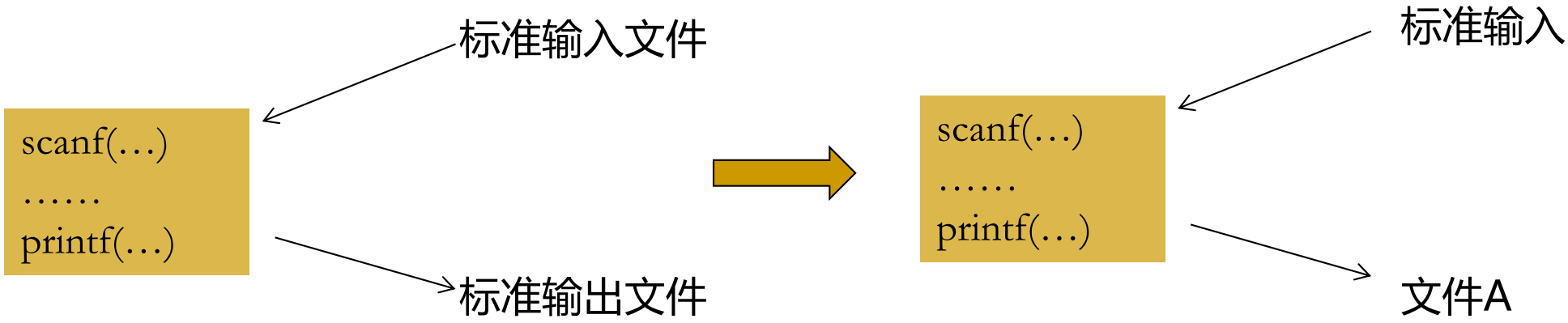
- 对于内核而言，所有打开文件都由文件描述符引用。文件描述符是一个非负整数。
- 当打开一个现存文件或创建一个新文件时，内核向进程返回一个文件描述符。当读、写一个文件时，用open或creat返回的文件描述符标识该文件，将其作为参数传送给read或write。
- 通常情况，UNIX shell使文件描述符0与进程的标准输入相结合，文件描述符1与标准输出相结合，文件描述符2与标准出错输出相结合。

进程表项、文件表、v节点表

- 该进程有两个不同的打开文件,一个为标准输入(文件描述符为0),另一个为标准输出(文件描述符为1)。



文件系统示例

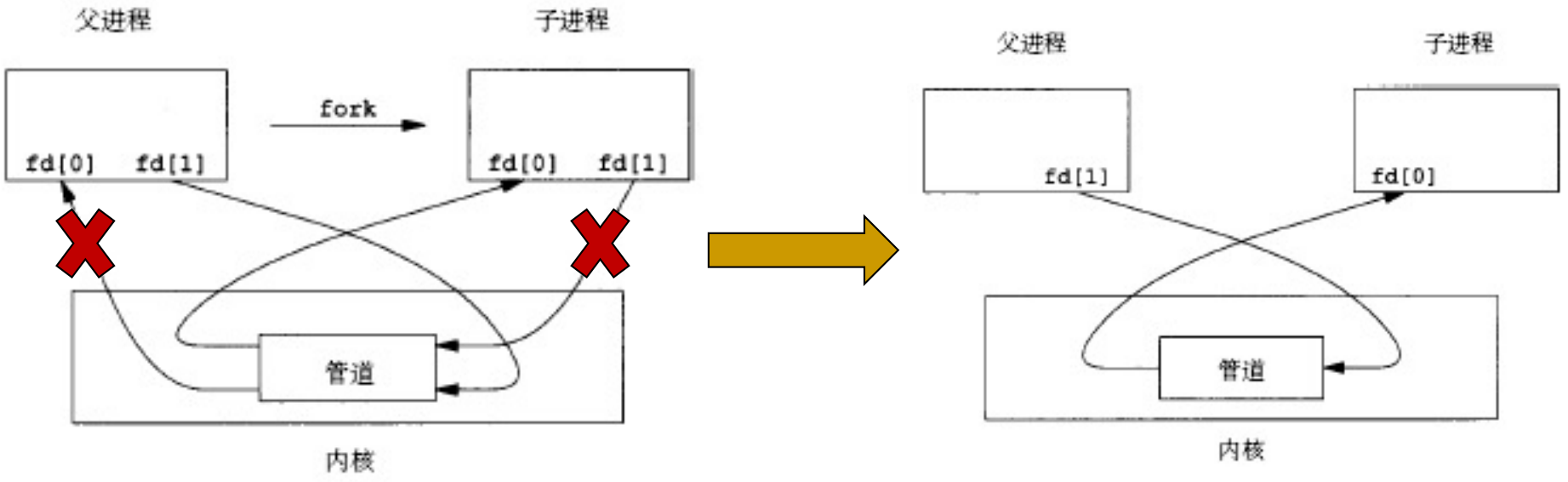



```
fd=dup(1);                  /*复制文件描述符*/
close(1);                   /* 关闭文件描述符1*/
open(文件A,……);    /*打开文件A*/
……
close(1);                   /*关闭描述符1*/
n=dup(fd);      /*复制文件描述符，恢复标准输出*/
close(fd);
```

dup复制的新文件描述符一定是当前可用文件描述符中的最小数值

管道

```
int fd[2];
pipe(&fd[0]);
fork();
close(fd[0]);
close(fd[1]);
```



进程管道实例

```
#define STD_INPUT 0                    /* file descriptor for standard input */
#define STD_OUTPUT 1                   /* file descriptor for standard output */
pipeline(process1, process2)
char *process1, *process2;             /* pointers to program names */
{
 int fd[2];
 pipe(&fd[0]);                         /* create a pipe */
 if (fork() != 0) {
    /* The parent process executes these statements. */
    close(fd[0]);                      /* process 1 does not need to read from pipe */
    close(STD_OUTPUT);                 /* prepare for new standard output */
    dup(fd[1]);                        /* set standard output to fd[1] */
    close(fd[1]);                      /* this file descriptor not needed any more */
    execl(process1, process1, 0);
 } else {
    /* The child process executes these statements. */
    close(fd[1]);                      /* process 2 does not need to write to pipe */
    close(STD_INPUT);                  /* prepare for new standard input */
    dup(fd[0]);                        /* set standard input to fd[0] */
    close(fd[0]);                      /* this file descriptor not needed any more */
    execl(process2, process2, 0);
  }
}
```