计算机操作系统

主讲教师：王　雷

第一章 操作系统引论

- 什么是操作系统
- 操作系统简史
- 计算机硬件简介
- 操作系统的基本类型
- 操作系统的特征和功能
- 操作系统结构
- 目前常用操作系统的介绍

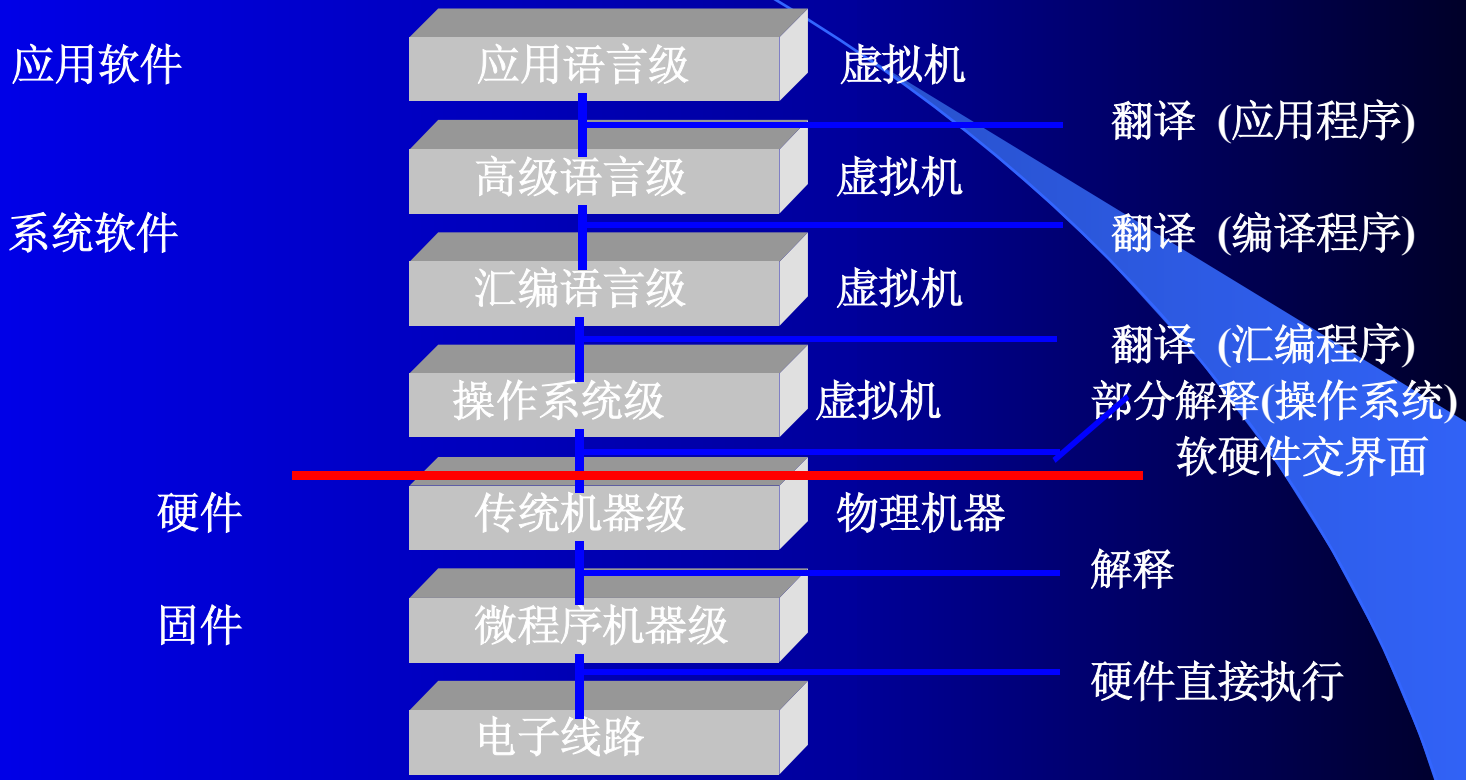
参考书

- “Operating Systems Internals and Design Principles” William Stalling, 清华大学出版社 Prentice Hall, 1998.6, 35.00元。
- “Operating Systems：Design and Implementation Second Edition” A. S. Tanenbaum, A. S. Woodhull, 清华大学出版社 Prentice Hall, 1997.9, 59.00元。

什么是操作系统

- 提供一个计算机用户与计算机硬件系统之间的接口，使计算机系统更易于使用。（使用者）
- 有效地控制和管理计算机系统中的各种硬件和软件资源，使之得到更有效的利用。（资源管理者）
- 合理地组织计算机系统的工作流程，以改善系统性能（如响应时间、系统吞吐量）。
- 虚拟机的概念

操作系统层次

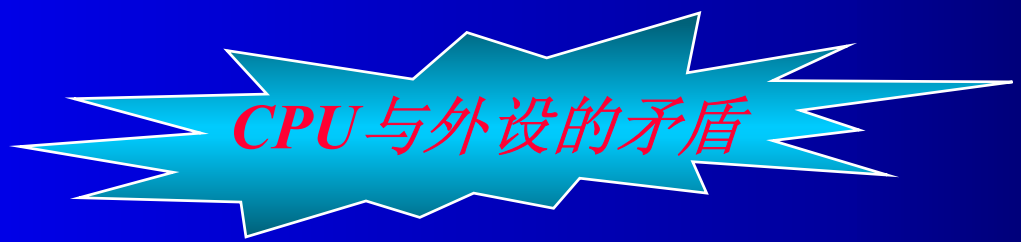

应用软件
系统软件
硬件
固件
应用语言级
高级语言级
汇编语言级
操作系统级
传统机器级
微程序机器级
电子线路
虚拟机
虚拟机
虚拟机
虚拟机
物理机器
翻译 (应用程序)
翻译 (编译程序)
翻译 (汇编程序)
部分解释(操作系统)
软硬件交界面
解释
硬件直接执行

操作系统简史

- 1946~1955年 电子管
- 1955~1965年 晶体管 & 监控系统
- 1965~1980年 集成电路 & 多道程序设计
- 1980~1990年 PC机 & 微机操作系统
- 1990~ 年 分布式与嵌入式系统

批处理技术

- 脱机输入技术
- 脱机输出技术
- 批处理技术：计算机系统对一批作业进行处理的技术。



多道程序设计

- 多道程序设计技术
- 分时系统

多道程序系统需要解决的问题

- 在一个连续的内存空间，同时驻留多道程序。
- 处理机的争夺
- I/O设备的分配
- 有效的组织不同程序的运行
- 系统对各种存储介质的管理

微机操作系统

- PC机
- MSDOS、WINDOWS 95、WINDOWS NT、WINDOWS 2000
- 类UNIX(Linux, Freebsd等)。

分布式与嵌入式系统

- Cluster of Workstation, Network of Workstation
- 实时操作系统： Psos, VRTX, RTLinux,

计算机硬件简介

- 硬件是基础，必须了解硬件。
- 但不必象硬件工程师那样，我们了解的是功能、接口和状态。

操作系统的基本类型

- 批处理系统
- 分时系统
- 实时系统

- 混合型

批处理系统

- 分类：
  - 单道批处理系统：无须作业调度和进程调度；内存仅有一道作业；顺序完成
  - 多道批处理系统：作业调度和进程调度；内存有多道作业；非顺序完成
  - 远程批处理系统
- 优点：
  - 系统吞吐量大
  - 资源利用率高
- 缺点：
  - 平均周转时间长
  - 不能提供交互作用能力

分时系统

- 简单分时系统：内存中只有一道作业；
- 具有前台和后台的分时系统
- 基于多道程序设计的分时系统

分时系统的特征

- 多路性：多路连接；宏观上用户共享，微观上分时；
- 独立性：用户相互不干扰；
- 及时性：响应时间；
- 交互性：人机对话。

影响响应时间的因素

- 系统开销
- 用户数目
- 时间片
- 对换信息量

- 采用可重入文件
- 引入虚存减少对换

实时系统的类型和特征

- 实时控制系统
- 实时信息处理系统

- 及时性
- 交互作用性：比分时系统差，但非常必要。
- 多路性
- 独立性

实时系统的特殊功能

- 高可靠性：双工；原子事务；
- 连续人机对话
- 过载防护

RTOS性能指标

- RTOS性能指标
  - 中断响应时间： Interrupt Response
  - 上下文切换时间：Context Switching Time
  - 确定性： Determinism

    Kernel services should be deterministic by specifying how long each service call will take to execute.
- 调度器的实现算法

Rate Monotonic（发生率单调），优先级与发生率呈正比 [LiuLay 1973]

lottery scheduler (彩票调度， Wald&Weihl94）

操作系统的特征

- 并发
  - 并发：两个或多个事件在同一时间间隔内发生；
  - 并行：两个或多个事件在同一时刻内发生；
- 共享：系统中软、硬件资源不再为某个程序所独占，而是供多个用户同时使用。
  - 互斥共享（打印机、变量）
  - 同时访问（宏观）
- 虚拟：把一个物理实体，变为若干个逻辑上的对应物。
  - 多道程序中的CPU
  - SPOOLING（外围设备同时联机操作）
  - 虚拟存储
- 不确定
  - 程序执行结果不确定
  - 多道程序中执行顺序不确定

操作系统的功能

- 处理机管理
- 存储器管理
- 设备管理
- 文件管理
- 作业控制

处理机管理

- 进程控制；
- 进程同步；
- 进程通信；
- 进程调度。

存储器管理

- 任务：
  - 为多道程序的并发提供良好的环境
  - 便于用户使用存储器
  - 提高存储器利用率
  - 为尽量多的用户提供足够大的存储空间
- 功能：
  - 内存分配：静态和动态分配。
  - 内存保护；
  - 地址影射；
  - 内存扩充。

设备管理

- 任务：
  - 为用户程序分配I/O设备
  - 完成用户程序请求的I/O操作
  - 提高CPU和I/O设备的利用率：中断；通道。
  - 改善人机界面
- 功能：
  - 缓冲管理；
  - 设备分配；
  - 设备处理；
  - 虚拟设备功能。

文件管理

- 文件存储空间的管理；
- 目录管理；
- 文件读、写管理；
- 文件保护；
- 向用户提供接口。

作业控制

- 作业调度；
- 作业控制。
  - 批量型作业
  - 终端型作业

多处理机操作系统

- 多处理机结构
  - 紧密耦合；**SMP**
  - 松散耦合。
- 多处理机操作系统类型
  - 主-从式；
  - 独立管理程序系统；
  - 浮动式管理程序控制方式。

网络操作系统

- 广域网WAN;
- 局域网LAN。
- 建立在主机操作系统基础上，用于管理网络通信和资源共享，协调各主机上的任务运行，并向用户提供统一的、有效的网络接口的软件集合。

计算机网络的基本特征：

- 自治性
- 分布性：（1）地理分布；（2）功能分布；（3）任务分布。
- 互联性：物理上和逻辑上。
- 统一性：**TCP/IP**

网络操作系统的功能和定义

- 网络通信
- 资源管理
- 提供多种网络服务：**email**、文件传输、远程控制。
- 提供网络接口。

分布式操作系统基本特征

- 分布性
- 自治性
- 模块性
- 并行性

基本功能

- 资源管理
- 任务分配
- 分布式进程同步和通信
- 管理程序浮动

操作系统结构

- 模块接口
- 有序分层法
- 微内核结构

模块接口

- 内聚性：模块内部各部分间联系的紧密程度。
  - 逻辑内聚模块、时间、过程、数据和功能。
- 耦合性：模块间相互联系和相互影响。
  - 数据耦合、控制、病态
- 优点：加速了操作系统的研制过程、增加了操作系统的灵活性、便于修改和维护。
- 缺点：接口定义困难、无序性。

有序分层法

- 自底向上法、自顶向下法
- 层次设置原则
  - 调用关系
  - 与硬件的关系
  - 与虚存的关系
  - 可扩充性
- 调用方式
  - 只能调用直接下层
  - 可以调用所有下层
  - 可以部分调用下层。

微内核结构

- 内核中只包括中断处理、进程通信（IPC）、基本调度等
- 文件系统、网络功能、内存管理、设备管理等作为服务在微内核上运行。
- 优点：
  - 内核易于实现、可移植性好、配置灵活、适应分布式环境（本地内核与远程内核对服务同样的支持）
- 缺点：
  - 速度较慢。（扩大内核减少切换；减少内核提高其他优点）

目前常用操作系统的介绍

- CP/M（Control Program Monitor）
- Windows操作系统
- UNIX操作系统
- Linux操作系统

CP/M（Control Program Monitor）

- CP/M（Control Program Monitor）：1975年Digital Research公司推出的带有软盘系统的8位微机操作系统，配置在以Intel 8080、8085、Z80为芯片的微机上。

DOS操作系统

- Microsoft公司在1981年开发了MS DOS1.0，4000行汇编语句，运行在Intel 8086上。后来IBM推出了带硬盘的PC XT，
- Microsoft公司在1983年开发了MS DOS2.0，2.0有了一些类UNIX的功能，例如I/O redirection。
- 在1984年IBM推出了包含80286的PC AT，Microsoft公司开发了MS DOS3.0，但是DOS 3.0没有发挥出80286的优势。接着Microsoft公司开发了MS DOS3.1，支持共享文件、网络功能。在1987年推出了DOS3.3支持IBM推出了新型机PS/2。

Windows操作系统

- 在486、Pentium芯片问世后，DOS不能充分发挥硬件性能，因此从80年代初，Microsoft开始开发GUI。1990年推出了WINDOWS 3.0，单需要在DOS上运行。1996年推出Windows 95,后来的Windows 98。
- IBM在1987年开发的OS/2，在286保护模式下运行。在与IBM分道扬镳后，Microsoft开发了WINDOWS NT（单用户多任务），
- 1993年推出了第一个版本WINDOWS NT3.1（WINDOWS 3.1风格）。NT4.0(WINDOWS 95)风格。现在的Windows 2000是合二为一的操作系统。

UNIX操作系统

- UNIX系统在1969~70年有贝尔实验室开发，在PDP 7上运行。
- 于MULTICS项目的研究工作。
- 1973年，C语言加入了结构和全局变量。与此同时，Ken和Dennis成功地用C重写了UNIX核心。Shell也被重写了。
- 1975年，第六版UNIX系统发行了。这是第一个在贝尔实验室外广为流传的UNIX系统。

Linux操作系统

- Linux是由Linus Torvalds于1991年开发的。
- 1991年9月，Linux 0.0.1，很不完善。
- 1991年10月， Linux 0.0.2，第一个“正式”版本。两周后0.0.3。
- 1991年12月， Linux 0.1.0，已经有许多人在上面工作了。
- 1994年3月， Linux 1.0

GNU/LINUX

- GNU(GNU not Unix)
- Linux成为了一个操作系统。值得注意的是Linux并没有包括Unix源码。它是按照公开的POSIX标准重新编写的。Linux大量使用了由麻省剑桥免费软件基金的GNU软件，同时Linux自身也是用它们构造而成。

小结

- 什么是操作系统
- 操作系统的历史
- 操作系统的类型
- 操作系统的特征
- 操作系统的功能
- 操作系统的结构
- 常用的操作系统

- 进程的基本概念
- 进程控制
- 进程同步
- 经典进程同步问题
- 进程通信
- 进程调度
- 死锁

2.1 前趋图

- 程序的顺序执行
- 程序的并发执行和特征
- Bernstein条件

2.1.1 程序的顺序执行与特征

- 顺序性
- 封闭性
- 可再现性

2.1.2 前趋图的定义

- 前趋关系（不能表示循环）。

2.1.3 程序的并发执行和特征

- 程序并发执行

程序执行时的特征

- 不可再现性
- 间断性
  - 共享资源
  - 程序之间相互协同，
- 通信性
- 独立性

```
Var N :integer;
  begin
    N:=0;
    parbegin
        program A : begin
                      repeat
                      …
                      N:=N+1;
                      …
                      until false
                    end
        program B : begin
                      repeat
                      …
                      print(N);
                      N:=N+1;
                      …
                      until false
                    end
    parend
end
```

2.1.4 Bernstein条件

- S1：c:=a+b;
- 读集: R(S1)={a, b}
- 写集: W(S1)={c}
- Bernstein条件
- $R(S1) \cap W(S2) \cup R(S2) \cap W(S1) \cup W(S1) \cap W(S2) = \{\}$

2.2 进程的基本概念

- 多道程序设计技术、分时系统要求能够描述程序的动态特性，需要引入新概念。

2.2.1 进程的定义和特征

- 进程是程序的一次执行；
- 进程是可以和别的计算并发执行的计算；
- 进程可定义为一个数据结构，及能在其上进行操作的一个程序；
- 进程是一个程序及其数据，在处理机上顺序执行时所发生的活动；
- 进程是程序在一个数据集合上运行的过程，它是系统进行资源分配和调度的一个独立单位。

进程的定义

- 可并发执行的程序，在一个数据集合上的运行过程。

进程的特征

- 动态性
- 并发性
- 独立性
- 异步性
- 结构特征：程序段，数据段，进程控制块PCB

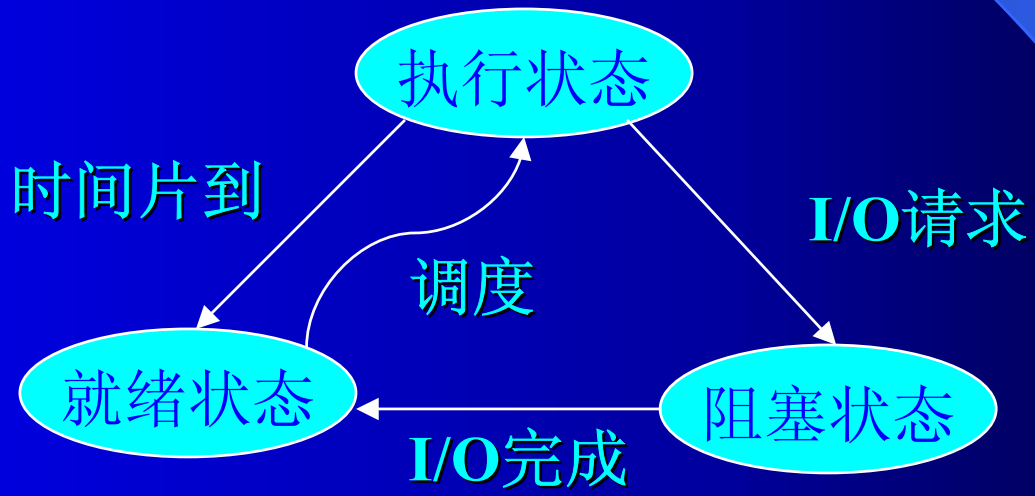
引入进程的利弊

- 利：提高效率
- 弊：空间开销、时间开销。

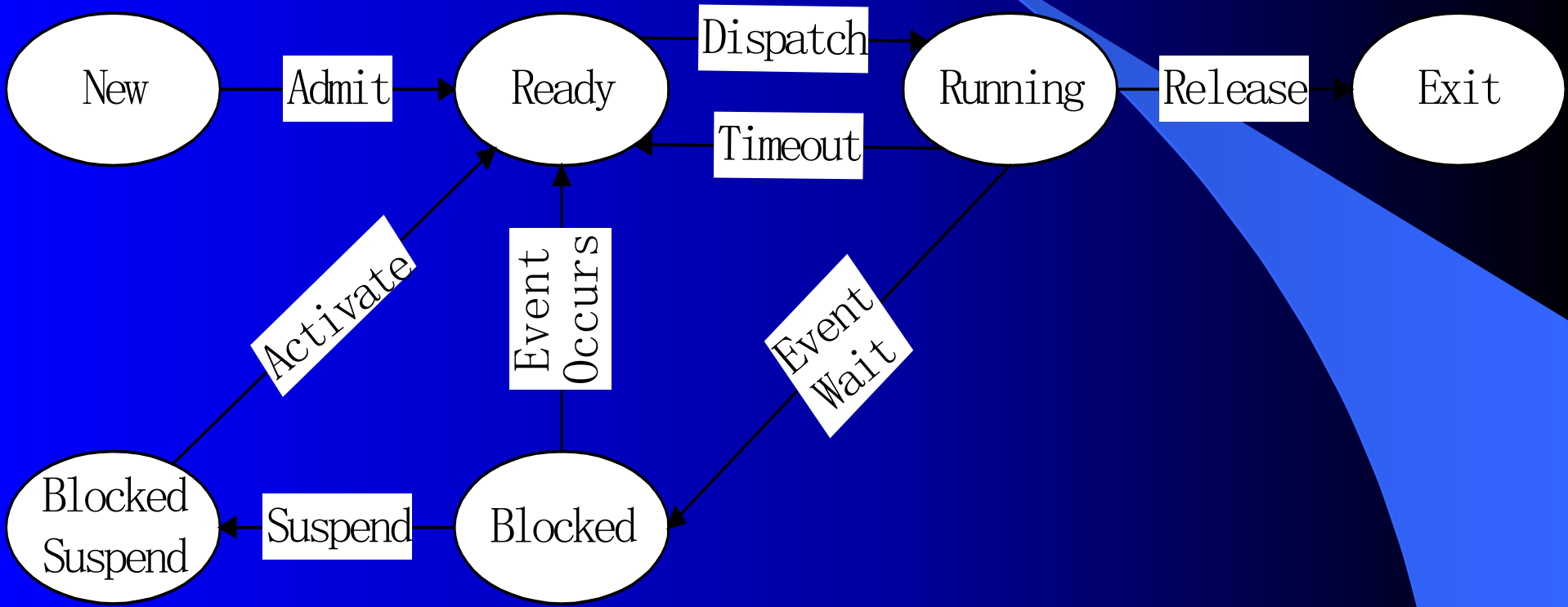
进程与程序的区别

2.2.2 进程状态及其演变

- 进程的三种基本状态
  - 就绪状态：进程已获得除处理机外的所需资源，等待分配处理机资源；只要分配CPU就可执行。
  - 执行状态：占用处理机资源；处于此状态的进程的数目小于等于CPU的数目。在没有其他进程可以执行时（如所有进程都在阻塞状态），通常会自动执行系统的idle进程（相当于空操作）。
  - 阻塞状态：正在执行的进程，由于发生某种事件而暂时无法执行，便放弃处理机处于暂停状态。


执行状态
时间片到
调度
I/O请求
就绪状态
阻塞状态
I/O完成

挂起进程模型


New
Admit
Ready
Dispatch
Running
Release
Exit
Timeout
Activate
Event Occurs
Event Wait
Blocked Suspend
Suspend
Blocked

2.2.2 进程控制块

- 系统为每个进程定义了一个数据结构：进程控制块。
- 作用：
  - 进程创建、撤消
  - 进程唯一标志；
  - 限制系统进程数目。

进程控制块的内容

- 进程标识符
- 程序和数据地址
- 现行状态
- 现场保留区
- 互斥和同步机制
- 进程通信机制
- 优先级
- 资源清单
- 链接字
- 家族关系

PCB的组织方式

- 链接方式
- 索引方式

Linux的进程控制块

北京航空航天大学计算机科学与工程系　王雷

2.3 进程控制

- 进程控制的主要任务是创建和撤消进程，以及实现进程的状态转换。由内核来实现。

2.3.1 内核

- 内核是硬件的第一层软件扩充，可大可小。
- 内核的基本功能：
  . 中断处理 *
  . 进程管理：进程创建和撤消；进程的状态转换；进程调度；控制进程的并发执行。
  . 资源管理中的基本操作
- 原语：是机器指令的延伸，是由若干条机器指令构成用以完成特定功能的一段程序。为保证操作的正确性，它们应当是原子操作。

进程的创建和撤消

- 进程图
- 创建原语(fork, exec)
- 撤消原语(kill)
  - 释放资源、撤消子进程、重新调度。

进程的挂起与激活

- 挂起原语
  将进程置于静止就绪或静止阻塞状态，方式：1）把发命令进程自身挂起。2）挂起具有指定标识符的进程。3）将某进程及其全部或部分子进程挂起。
  留出一段内存保留副本。
- 激活原语
  使处于静止状态的进程变为活动状态。

进程的阻塞和唤醒

- 阻塞原语
  1）停止执行；2）插入等待队列；3）重新调度。
  - 暂停一段时间sleep；暂停并等待信号pause；等待子进程暂停或终止wait
- 唤醒原语
  - 发送信号到某个或一组进程kill

2.4 进程同步

- 间接相互制约。主要原因是资源共享。
- 直接相互制约。主要源于进程合作。
- 进程同步：指多个相关进程在执行次序上的协调，用于保证这种关系的相应机制称为同步机制。

2.4.1 临界区

- 临界资源
  - 这种一次只允许一个进程使用的资源称为临界资源。例如打印机、变量。

```
P1: R1:=count;
    R1:=R1+1;
    count:=R1;

P2: R2:=count;
    R2:=R2+1;
    count:=R2;
```

Count:=2

```
P1: R1:=count;
    R1:=R1+1;
P2: R2:=count;
    count:=R1;
P1: R1:=R1+1;
    count:=R2;
P2: R2:=R2+1;
    count:=R2;
```

Count:=1

临界区的定义

- 每个进程中访问临界资源的那段代码称为临界区。

Repeat

entry section

critical section

exit section

remainder section

until false

同步机制应遵循的准则

- 空闲让进
- 忙则等待
- 有限等待
- 让权等待

2.4.2 硬件指令机制

- Test and Set指令
  - IBM370系列机器中称为TS；在INTEL8086中成为XCHG指令。
- 利用TS实现进程互斥

```
Function TS(var lock: boolean):boolean;
        begin
                TS :=lock;
                lock :=true;
        end

Repeat
        while TS(lock) do skip;
        critical section;
        lock:=false;
        remainder section;
until false
```

软件实现进程互斥 1

- 两个进程Pi, Pj，其中Pi

```
while turn != i do skip;
critical section;
turn := j;
remainder section;
```

- 设立一个公用整型变量 turn：描述允许进入临界区的进程标识
  - 在进入区循环检查是否允许本进程进入：turn为i时，进程Pi可进入；
  - 在退出区修改允许进入进程标识：进程Pi退出时，改turn为进程Pj的标识j；

- 缺点：强制轮流进入临界区，没有考虑进程的实际需要。容易造成资源利用不充分：在**Pi**出让临界区之后，**Pj**使用临界区之前，**Pi**不可能再次使用临界区；

软件实现进程互斥 2

```
while (flag[j]) do skip; <a>
flag[i] := true;          <b>
critical section;
flag[i] := false;
remainder section;
```

- 设立一个标志数组**flag[]**：描述进程是否在临界区，初值均为**FALSE**。
  - 先检查，后修改：在进入区检查另一个进程是否在临界区，不在时修改本进程在临界区的标志；
  - 在退出区修改本进程在临界区的标志；

- 优点：不用交替进入，可连续使用；
- 缺点：**Pi**和**Pj**可能同时进入临界区。按下面序列执行时，会同时进入：“**Pi<a> Pj<a> Pi<b> Pj<b>**”。即在检查对方**flag**之后和切换自己之前有一段时间，结果都检查通过。这里的问题出在检查和修改操作不能连续进行。

软件实现进程互斥 3

```
flag[i] := true;              <a>
while (flag[j]) do skip; <b>
critical section;
flag[i] := false;
remainder section;
```

- 类似于算法2，与互斥算法2的区别在于先修改后检查。可防止两个进程同时进入临界区。
- 缺点：Pi和Pj可能都进入不了临界区。按下面序列执行时，会都进不了临界区：“Pi<a> Pj<a> Pi<b> Pj<b>”。即在切换自己flag之后和检查对方flag之前有一段时间，结果都切换flag，都检查不通过。

软件实现进程互斥 4

```
flag[i] := true;     turn := j;
while (flag[j] && turn == j) do skip;
critical section;
flag[i] := false;
remainder section;
```

- 在进入区先修改后检查，并检查并发修改的先后：(Peterson's Algorithm)
  - 检查对方flag，如果不在临界区则自己进入——空闲则入
  - 否则再检查turn：保存的是较晚的一次赋值，则较晚的进程等待，较早的进程进入——先到先入，后到等待

2.4.3 信号量机制

- 信号量只能通过初始化和两个标准的原语来访问——作为OS核心代码执行，不受进程调度打断

经典信号量机制

P(S) :while S<=0 do skip
S:=S-1;

V(S) :S:=S+1;

计数信号量机制

```
Type semaphore = record
        value : integer;
        L : list of process;
    end

procedure P(S)
        var S : semaphore;
        begin
                S.value := S.value -1;
                if S.value<0 then block(S.L);
        end

procedure V(S)
        var S : semaphore;
        begin
                S.value := S.value + 1;
                if S.value<=0 then wakeup(S.L)
        end
```

物理意义

- **S.value**表示资源的个数
- **P**操作分配资源
- **V**操作释放资源。

信号量机制的实现

- 原子性问题；
- **PCB**链表形式。

补充练习

```
movl eax, sem ？ ？
decl eax
js  down_failed
```

```
movl sem, %ecx
decl 0(%ecx)
js  down_failed
```

2.4.4 信号量的应用

- 互斥
  - 利用信号量实现进程互斥（S=1）
- 同步
  - 利用信号量实现进程同步（S=0）
- 描述前趋关系。
  - 利用信号量实现描述前趋关系（S=0）

互斥

同步

前趋关系

2.4.5 “信号量集”机制

```
Process A:                    Process B:
  P(Dmutex);                    P(Emutex);
  P(Emutex);                    P(Dmutex);
```

Dmutex, Emutex = 1;

Process A: P(Dmutex);
Process B: P(Emutex);
Process A: P(Emutex);
Process B: P(Dmutex);

AND型信号量集机制

- 基本思想：将进程需要的所有共享资源一次全部分配给它；待该进程使用完后再一起释放。

```
SP(S1, S2, … ,Sn)
  if S1=>1 and … and Sn=>1 then
    for I :=1 to n do
      Si := Si - 1;
    endfor
  else
   wait in Si;
  endif
```

```
SV(S1, S2, … ,Sn)
  for I :=1 to n do
      Si := Si + 1;
      wake waited process
  endfor
```

一般“信号量集”机制

- 一次需要N个某类临界资源时，就要进行N次wait操作——低效又可能死锁
- 基本思想：在AND型信号量集的基础上进行扩充：进程对信号量Si的测试值为ti（用于信号量的判断，即Si >= ti，表示资源数量低于ti时，便不予分配），占用值为di（用于信号量的增减，即Si = Si - di和Si = Si +di）

```
SP(S1, t1, d1, … , Sn, tn, dn)
  if S1=>t1 and … and Sn=>tn then
    for I :=1 to n do
      Si := Si - di;
    endfor
  else
   wait in Si;
  endif
```

```
SV(S1, d1, … ,Sn, dn)
  for I :=1 to n do
      Si := Si + di;
      wake waited process
  endfor
```

- SP(S, d, d)
- SP(S, 1, 1)
- SP(S, 1, 0)

2.4.6 管程 (Monitor)

- 用信号量可实现进程间的同步，但由于信号量的控制分布在整个程序中，其正确性分析很困难。管程是管理进程间同步的机制，它保证进程互斥地访问共享变量，并方便地阻塞和唤醒进程。管程可以函数库的形式实现。相比之下，管程比信号量好控制。

管程的引入

- 1973年，Hoare和Hanson所提出
- 信号量机制中，同步操作分散在各个进程中，使用不当就可能导致各进程死锁（如P、操作的次序错误、重复或遗漏）
- 一个管程定义了一个数据结构和能为并发进程所执行（在该数据结构上）的一组操作，这组操作能同步进程和改变管程中的数据。

管程的组成

- 为每个共享资源设立一个管程，由用户编写——类似于“面向对象”的观点

- 局部控制变量：一组局部于管程的控制变量
- 操作原语：对控制变量和临界资源进行操作的一组原语过程（程序代码），是访问该管程的唯一途径。这些原语本身是互斥的，任一时刻只允许一个进程去调用，其余需要访问的进程就等待。
- 初始化代码：对控制变量进行初始化的代码

同步、互斥和条件变量

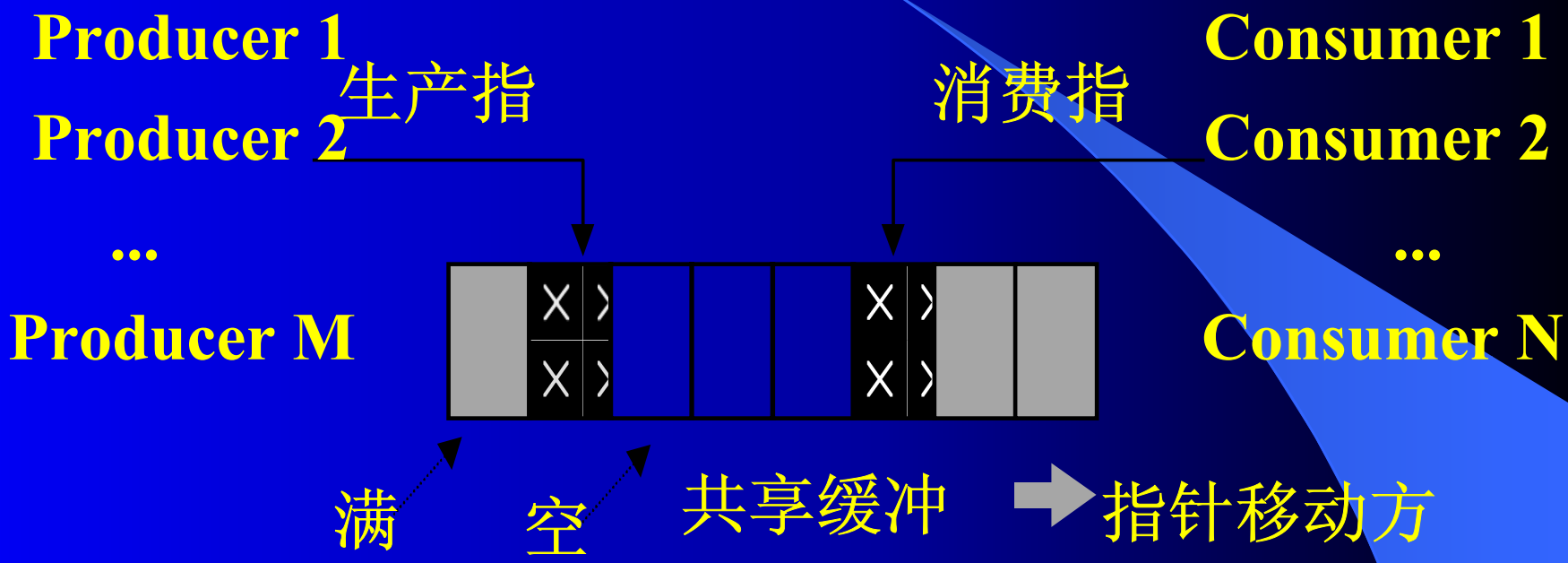
- 互斥
- 同步：wait和signal
- 条件变量(condition)
  - 每个表示一种等待原因，并不取具体数值——相当于每个原因对应一个队列
  - 同步操作原语：针对条件变量x，c.wait(x)将自己阻塞在x队列中，c.signal(x)将x队列中的一个进程唤醒。若进程P唤醒进程Q，则可有两种执行方式：
    - P等待，直到执行Q离开管程或下一次等待。
    - Q送入Ready队列，直到执行P离开管程或下一等待。

2.5 经典进程同步问题

- 生产者－消费者问题(the producer-consumer problem)
- 读者－写者问题(the readers-writers problem)
- 哲学家进餐问题(the dining philosophers problem)

生产者—消费者问题(the producer-consumer problem)

- 问题描述：若干进程通过有限的共享缓冲区交换数据。其中，“生产者”进程不断写入，而“消费者”进程不断读出；共享缓冲区共有N个；任何时刻只能有一个进程可对共享缓冲区进行操作。


Producer 1
Producer 2
...
Producer M
生产指
消费指
Consumer 1
Consumer 2
...
Consumer N
满
空
共享缓冲
指针移动方

- full是“满”数目，初值为0，empty是“空”数目，初值为N。实际上，full和empty是同一个含义：full + empty == N
- mutex用于访问缓冲区时的互斥，初值是1

生产者	消费者
P(empty);	P(full);
P(mutex);	P(mutex);
one >> buffer	one << buffer
V(mutex)	V(mutex)
V(full)	V(empty)

采用AND信号量集

- SP(empty, mutex), SP(full, mutex)

采用管程解决生产者-消费者问题

读者－写者问题(the readers-writers problem)

- 问题描述：对共享资源的读写操作，任一时刻“写者”最多只允许一个，而“读者”则允许多个——“读－写”互斥，“写－写”互斥，“读－读”允许

采用信号量机制

- wmutex表示“允许写”，初值是1。
- 公共变量readcount表示“正在读”的进程数，值是0；
- rmutex表示对Rcount的互斥操作，初值是1。

Writer

```
P(wmutex);
   write
V(wmutex);
```

Reader

```
P(rmutex);
if readcount=0 then P(wmutex);
V(rmutex);
   read
P(rmutex)
readcount := readcount -1;
if readcount=0 then V(wmutex);
V(rmutex)
```

采用一般“信号量集”机制

- 增加一个限制条件：同时读的“读者”最多RN个
- mx表示“允许写”，初值是1
- L表示“允许读者数目”，初值为RN

Writer

```
SP(mx, 1, 1; L, RN, 0);
   write
SV(mx, 1);
```

Reader

```
SP(L, 1, 1; mx, 1, 0);
   write
SV(L, 1);
```

哲学家进餐问题(the dining philosophers problem)

- 问题描述：（由Dijkstra首先提出并解决）5个哲学家围绕一张圆桌而坐，桌子上放着5支筷子，每两个哲学家之间放一支；哲学家的动作包括思考和进餐，进餐时需要同时拿起他左边和右边的两支筷子，思考时则同时将两支筷子放回原处。如何保证哲学家们的动作有序进行？如：不出现相邻者同时要求进餐；不出现有人永远拿不到筷子；

```
Var chopstick : array[0..4] of semaphore;

P(chopstick[i]);
P(chopstick[(i+1)mod 5]);
    eat
V(chopstick[i]);
V(chopstick [(i+1)mod 5]);
   think
```

```
Var chopstick : array[0..4] of semaphore;

    think
SP(chopstick[(i+1)mod 5], chopstick[i]);
    eat
SV(chopstick[(i+1)mod 5], chopstick[i]);
```

2.6 进程间通信(IPC, Inter-Process Communication)

- 低级通信：只能传递状态和整数值（控制信息），包括进程互斥和同步所采用的信号量和管程机制。缺点：
  - 传送信息量小：效率低，每次通信传递的信息量固定，若传递较多信息则需要进行多次通信。
  - 编程复杂：用户直接实现通信的细节，编程复杂，容易出错。
- 高级通信：能够传送任意数量的数据，包括三类：共享存储器系统、消息系统、共享文件。

共享存储系统

- 共享数据结构
- 共享存储区

共享存储区(shared memory)

- 相当于内存，可以任意读写和使用任意数据结构（当然，对指针要注意），需要进程互斥和同步的辅助来确保数据一致性。

消息系统

- 直接通信方式
  - send, receive
  - 原理图
- 间接通信方式
  - email

消息(message)

- 与窗口系统中的“消息”不同。通常是不定长数据块。消息的发送不需要接收方准备好，随时可发送。

管道(pipe)

- 管道是一条在进程间以字节流方式传送的通信通道。它由OS核心的缓冲区（通常几十KB）来实现，是单向的；常用于命令行所指定的输入输出重定向和管道命令。在使用管道前要建立相应的管道，然后才可使用。
- 互斥、同步、对方是否存在。

进程与线程

- 进程包含了两个概念：资源拥有者和可执行单元。
- 现代操作系统将资源拥有者称为进程（process, task）
- 可执行单元称为线程。

- 进程拥有虚空间、进程映像、处理机保护、文件、I/O空间。
- 线程额外的：运行状态、保存上下文（程序计数器）、执行栈、资源共享机制。

计算机操作系统

主讲教师：王　雷

文件管理

- 什么是文件系统
- 文件的存取方式
- 文件的物理组织
- 文件的存取控制
- 对文件的操作
- 文件系统的层次结构

- 硬件资源：

  中央处理机、主存储器及各种输入输出设备。
- 软件资源：

  各种系统程序、标准子程序和某些常用的应用程序。

文件、文件系统

- 文件
- 逻辑结构和物理结构
- 文件类型
- 文件系统

- 文件系统是文件管理系统的简称，它的管理功能是通过把它所管理的信息（程序和数据）组织成一个个文件的方式来实现的。
- 文件：是一个具有文件名的一组相关联元素的有序序列，通常由若干个记录组成。
- 记录：是一组相关数据项的集合，是作为对文件进行操作的基本单位。关键字。
- 一些低速的字符设备也被看成一个“文件”。

逻辑文件和物理文件

- 逻辑文件：

  用户看到的建立在逻辑结构基础上的文件。

- 物理文件：

  存储在物理设备上的文件。

文件类型

- 按性质和用途分：

  系统文件、库文件、用户文件

- 按数据形式：

  源文件、目标文件、可执行文件

- 按对文件实施的保护级别分：

  只读文件、读写文件、执行文件

- 按逻辑结构分：

  有结构文件、无结构文件

- 按文件中物理结构分：

  顺序文件、链接文件、索引文件

文件系统

1. 文件系统定义
2. 采用文件系统方式来管理各种软件资源和其它信息的优点
3. 一个文件系统必须解决的几个主要问题
4. 一个理想文件系统应具有的特性

1. 文件系统定义

- 文件系统定义

  是指操作系统中与文件管理有关的那部分软件和被管理的文件以及实施管理所需要的一些数据结构的总体。

2. 采用文件系统的优点

- 使用方便
- 安全性
- 统一性

3. 文件系统必须解决的几个主要问题

- 如何有效的分配文件存储器的存储空间
- 提供合适的存取方法
- 命名的冲突和文件的共享

4.理想文件系统应具有的特性

1. 有效的分配文件存储器的存储空间；
2. 文件结构和存取的灵活性和多样性；
3. 具有对用户来说尽可能是透明的机制；
4. 尽可能达到对文件存储装置的独立性；
5. 存储在文件中的信息的安全性；
6. 能方便的共享公用的文件；
7. 有效的实现各种文件操作的命令。

文件系统模型

文件系统的接口

- 命令接口
- 程序接口

文件操作

- 创建文件
- 删除文件
- 读文件
- 写文件
- 截断文件
- 设置文件的读写位置

文件组织

1. 文件逻辑结构和文件物理结构
2. 顺序存取
3. 直接存取
4. 索引顺序存取

1.文件逻辑结构和文件物理结构

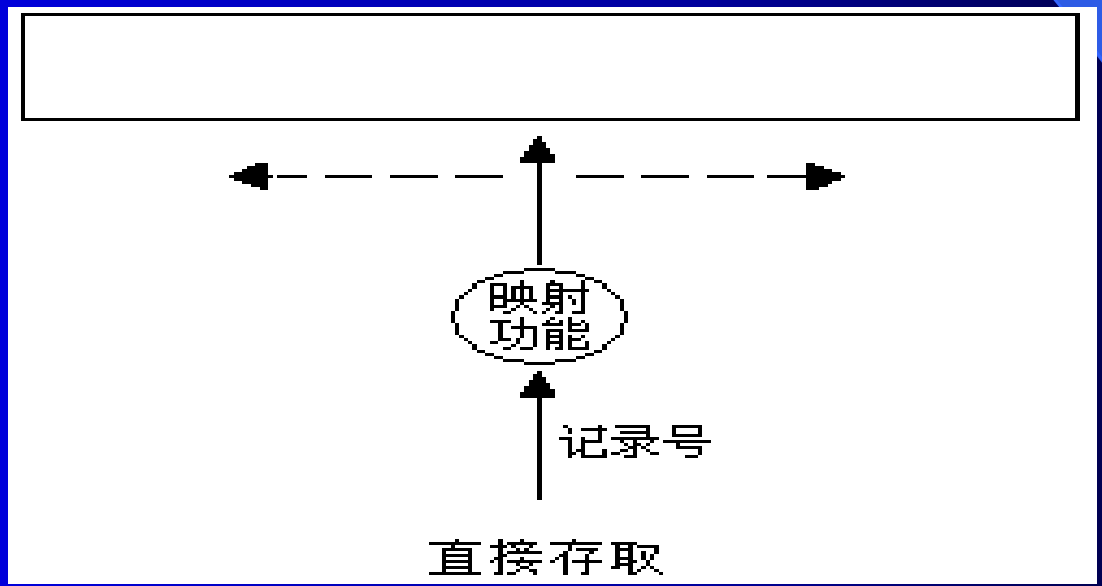
- 文件逻辑结构（文件组织）
  - 提高检索效率
  - 便于修改
  - 降低文件存储费用
- 文件物理结构

2. 顺序存取

- 顺序存取：记录是按某种方式排序的，并按该顺序一个接着一个存取。

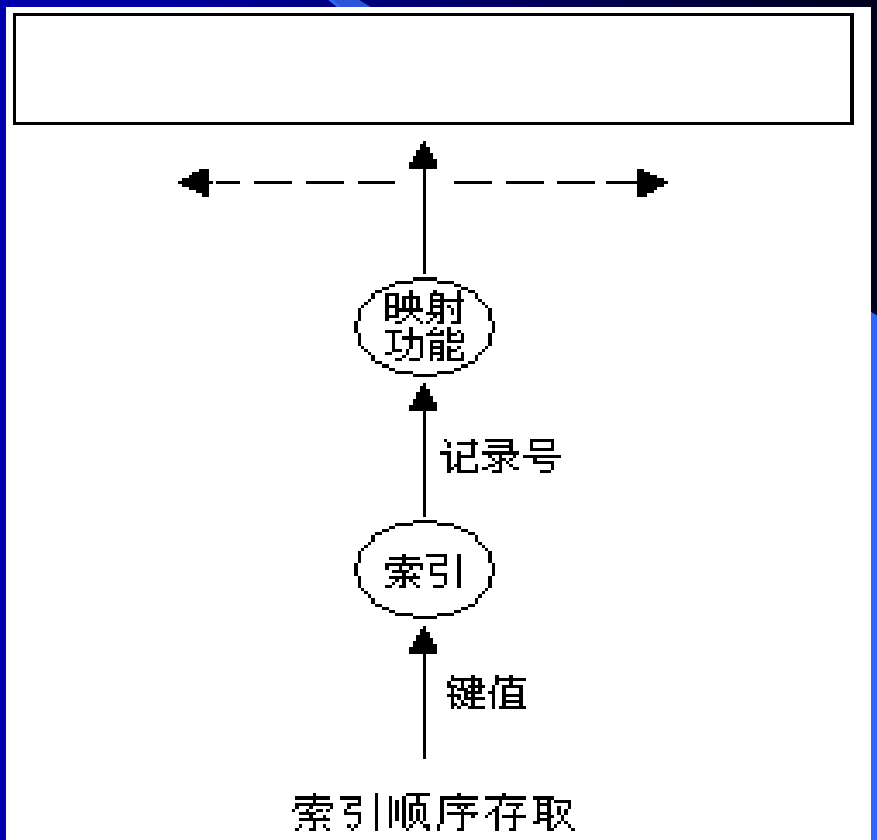
3. 直接存取

- 直接存取（随机存取）：能直接定位到文件中的任意一个记录，而无需存取其前面的记录。



4. 索引顺序存取

- 索引顺序存取：
  文件的记录不是按它们在文件中的位置来编址，而是按逻辑记录中的某个数据项的值来编址。



索引顺序存取

文件的物理组织

- 文件的物理组织：

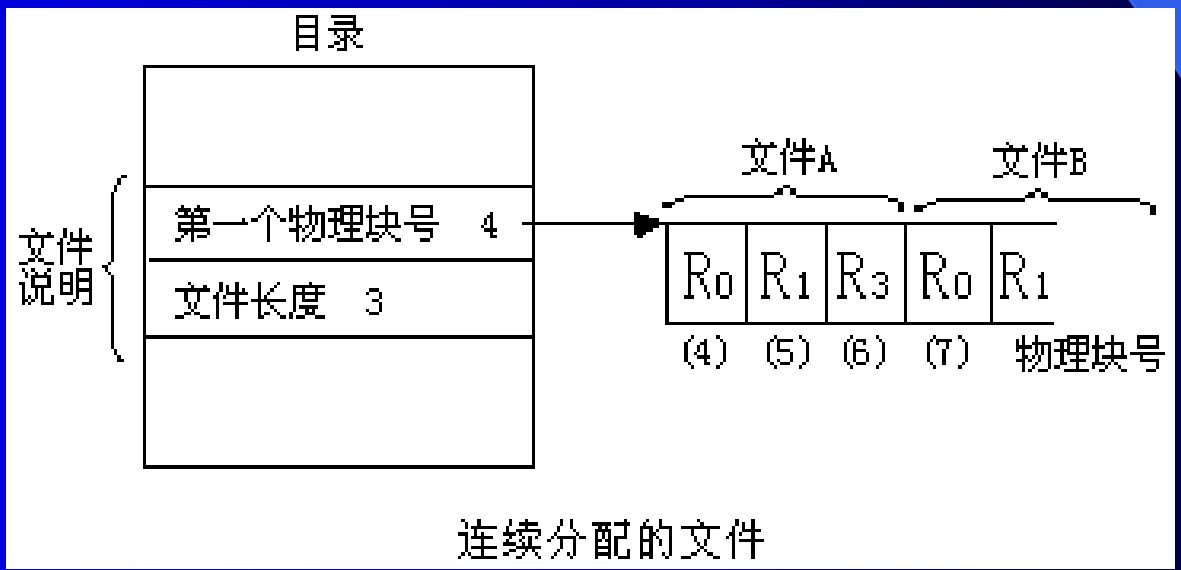
  表示了一个文件在文件存储器上的位置、链接和编目的方法。

文件物理组织的不同方式

1. 连续文件
2. 串联文件
3. 索引文件
4. **Hash文件**

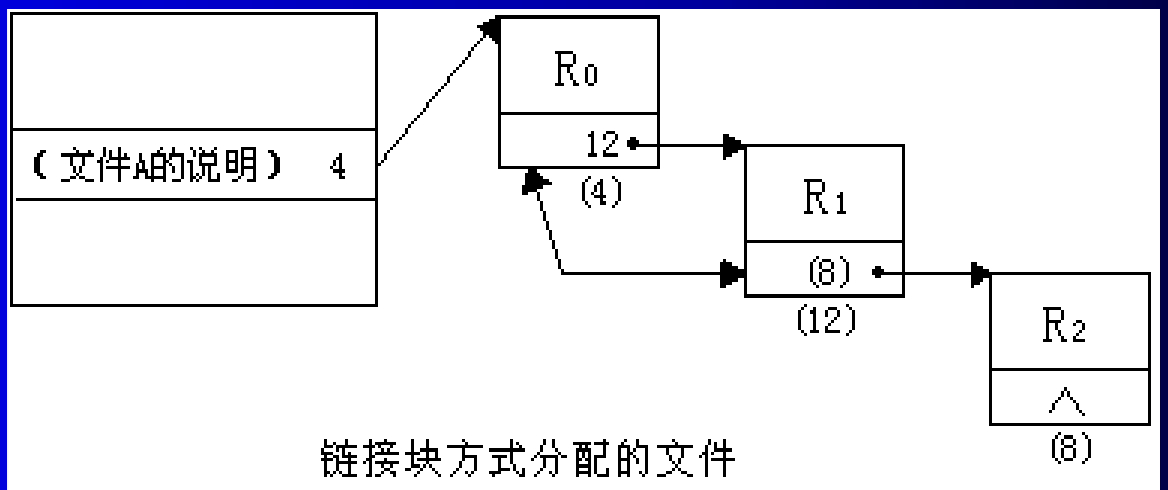
1. 连续文件

- 连续文件：把一个由逻辑上连续的记录构成的文件分配到依次连续的物理块中，这样组织的文件称为连续文件。



2. 串联文件

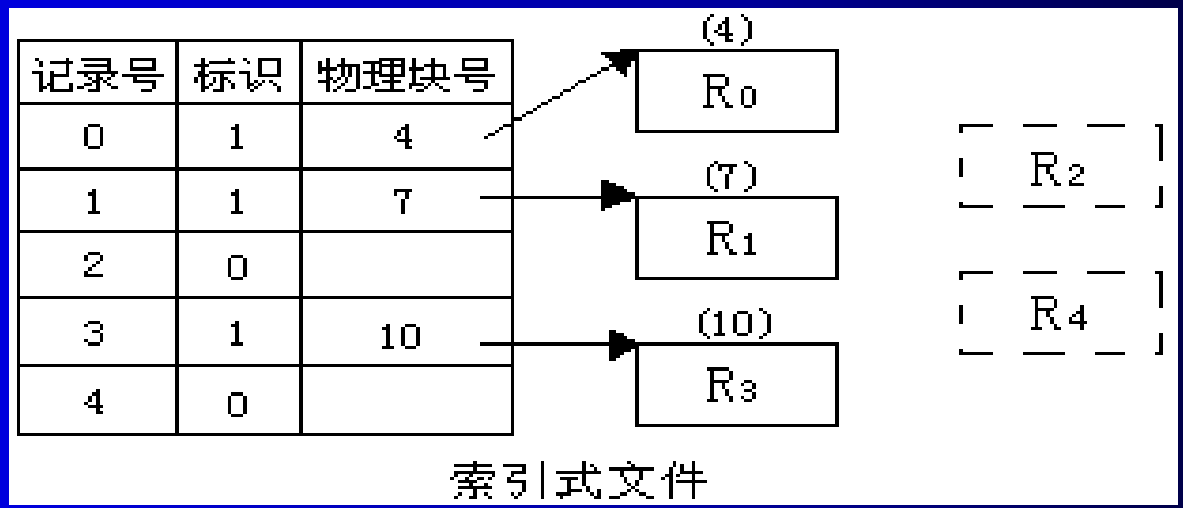
- 串联文件：物理块可以不连续，也不必顺序排列，在每个块中设有一个指针，它指向该文件的下一个物理块，采用这种分配方式的文件称为串联文件。



链接块方式分配的文件

3. 索引文件

- 索引文件：为每个文件建立一张索引表，其中每一个表目指出文件中记录所在的物理块号。文件的某个记录是否已存入，由索引表中相应的标识位指出。



4. Hash文件

- **Hash文件：**

  采用计算寻址结构的一种随机文件 。

文件类型与文件存储器、存取方法的关系

存储设备	磁盘、磁鼓				磁带
文件类型	连续文件	串联文件	索引文件	Hash文件	连续文件
文件长度	固定	固定、可变	固定、可变	固定、可变	固定
存取方法	直接、顺序	顺序	直接、顺序	直接、顺序	顺序

目录管理

- 目录是由文件说明索引组成的用于文件的检索特殊文件。文件目录的内容主要是文件访问和控制的信息（不包括文件内容）。

文件控制块

- 1. 基本信息
  - 文件名：字符串，通常在不同系统中允许不同的最大长度。可以修改。
  - 物理位置
  - 文件逻辑结构：有/无结构（记录文件，流式文件）
  - 文件物理结构（如顺序，索引等）
- 访问控制信息
  - 文件所有者（属主）：通常是创建文件的用户，或者改变已有文件的属主；
  - 访问权限（控制各用户可使用的访问方式）：如读、写、执行、删除等；

*一个典型的文件说明

文件名	
文件所在的物理地址	
记录长度	记录个数
文件占有者的存取权限	
其它用户的存取权限	
……	
……	
文件建立时间	
上次存取时间	
暂存文件/永久文件	

索引结点

文件存储器存储空间的管理

- 空白文件目录
- 空白物理块链
- 位视图

空白文件目录

一个连续的未分配区域称为“空白文件”,系统为所有这些“空白文件”单独建立一个目录。对应于每个空白文件，在这个目录中建立一个表目。表目的内容包括：第一空白物理块的物理块号、空白块的数目。

空白物理块链

- 成组链接法：

  把空白物理块分成组，在通过指针把组与组之间链接起来，，这种管理空白块的方法称为成组链接法。

- 成组链接法的优点：

1. 空白块号登记不占用额外空间；
2. 节省时间；
3. 采用后进先出的栈结构思想 。

文件目录

- 单级文件目录
- 多级文件目录
- 便于共享的目录组织
- 符号文件目录的查询技术
- 向用户提供接口

单级文件目录

- 文件目录的每个表目应包含：
  1. 文件的符号名
  2. 文件所在物理地址
  3. 文件结构信息
  4. 存取控制信息
  5. 管理信息

特点

- 结构简单；
- 文件多时，目录检索时间长；
- 有命名冲突：如多个文件有相同的文件名（不同用户的相同作用的文件）或一个文件有多个不同的文件名（不同用户对同一文件的命名）；
- 不便于实现共享

二级目录

- 在根目录（第一级目录）下，每个用户对应一个目录（第二级目录），在用户目录下是该用户的文件，而不再有下级目录。适用于多用户系统，各用户可有自己的专用目录。

多级目录

- 在较高的目录级，其目录表目为下一级目录名以及一个指向其目录的指针。
- 在最后一级目录，这个指针指向文件的物理地址。

便于共享的目录组织

文件的共享，要求系统能提供某种手段，使存储空间内置保存一份副本，而所有要共享该文件的用户可用相同的或不同的文件名来访问它。

符号文件目录的查询技术

- 顺序查寻法：

  依次扫描符号文件目录中的表目，将表目中的名字字段与查找的符号名NAME进行比较（只在表目不多时适用）。

- Hash方法：

  即一种“散列法”或成“杂凑法”，是一种构造符号表、查询符号表常用的技术。其基本思想是：利用一个易于实现的变换函数（即Hash函数），把每个符号名唯一的变换成符号表中的表目索引。

文件的存取控制

- 文件保护机制
- 存取权限验证步骤
- 存取控制的实现方案

1. 文件保护机制

- 文件保护机制应做到

1. 防止未被核准的用户存取文件；
2. 防止一个用户冒充另一个用户来存取；
3. 防止核准用户（包括文件主）误用文件；

2. 存取权限验证步骤

1. 审定用户的权限；
2. 比较用户的权限与本次存取要求是否一致；
3. 将存取要求和被访问文件的保密性比较，看是否有冲突。

3. 存取控制的实现方案

- 存取控制矩阵
- 存取控制表
- 用户权限表
- 口令

对文件的各种操作

- 文件的使用
- 文件控制块

文件的使用

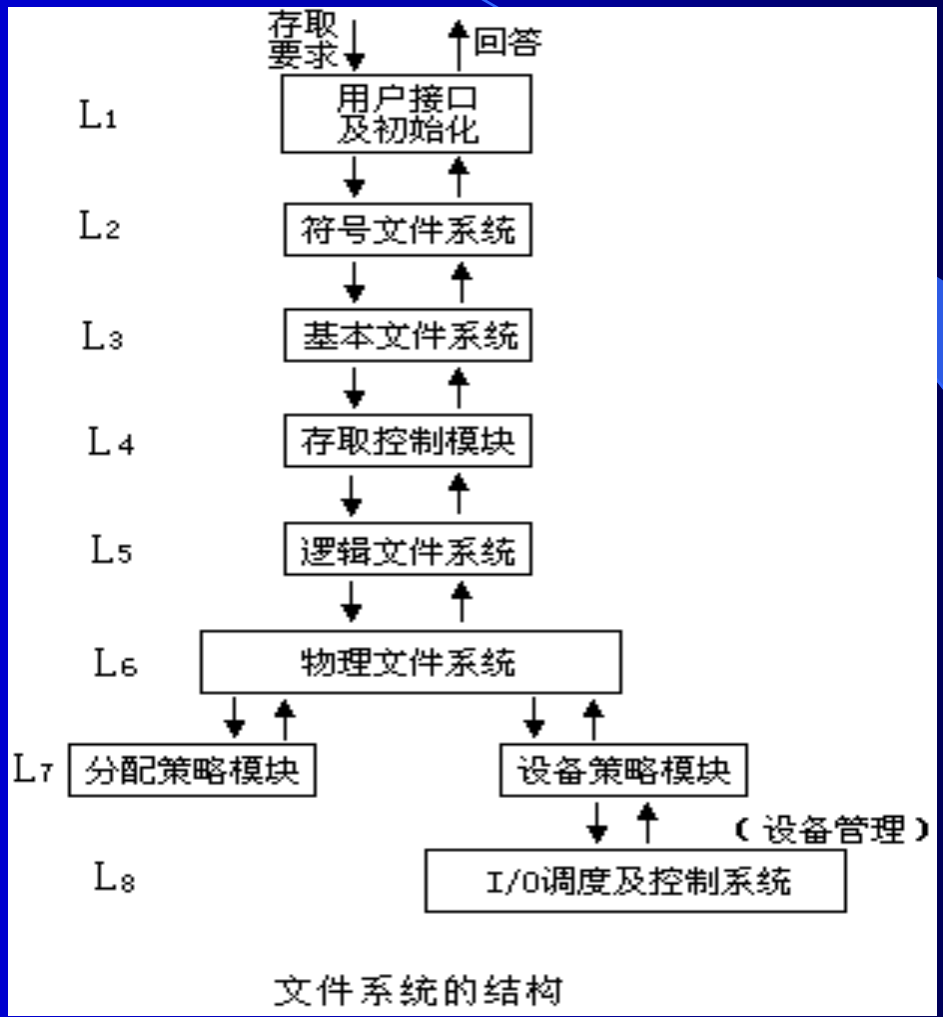
- 建立文件
- 打开文件
- 写文件
- “添加”命令
- 读文件
- 修改纪录
- 拷贝文件
- “搬家”命令
- 删除文件
- 关闭文件

6.8.2 文件控制块

- 文件控制块：

  当一个用户进程使用文件时，由文件系统把文件说明中全部或大多数信息，再加上有关当前使用文件的有关信息，抄到主存中的一个数据结构中，这个数据结构称为文件控制块（FCB）。

6.9 文件系统的一般模型



文件系统的结构

6.9.1 用户接口及初始化模块

- 主要功能：
  1. 对用户给出的文件命令进行语法检查；
  2. 把文件命令按系统要求加以改造，使之变成内部的调用格式；
  3. 补充用户未给出而系统可提供的信息，并存入约定的工作单元；
  4. 使文件系统初始化。

6.9.2 符号文件系统

- 符号系统负责：
  1. 管理符号文件目录；
  2. 管理活动名字表；
  3. 与基本文件系统通信 ；

6.9.3 基本文件系统

- 基本文件系统负责：
  1. 管理基本文件目录；
  2. 管理活动文件表；
  3. 与存取控制验证模块通信 。

6.9.4 存取控制验证

- 主要功能：

  实现文件保护。

6.9.5 逻辑文件系统

- 逻辑文件系统负责：
  1. 根据文件的逻辑结构计算存取记录的相对地址；
  2. 与物理文件系统通信 。

6.9.6 物理文件系统

- 物理文件系统负责：
  1. 把存取记录所在的相对块号转换为物理块地址；
  2. 负责把系统缓冲区中的记录搬到用户缓冲区；
  3. 与设备策略模块和分配策略模块通信 。

6.9.7 设备策略模块

- 主要功能：

  把物理块号转换成相应设备所要求的地址格式。

6.9.8 I/O调度和控制系统

- 主要功能：

  实现所有I/O请求的排队、调度、启动、I/O操作的控制，最终完成所需数据块从文件所在的设备传输到系统缓冲区中。

6.9.9 分配策略模块

- 主要功能：

  负责记住每个存储设备上的空白情况，并进行分配。

6.9.10 主要文件命令的算法

- 见课本

6.10 小结

计算机操作系统

主讲教师：王　雷

第四章 存储器的管理

- 存储分配：主要是讨论和解决多道作业之间共享主存的存储空间的问题。

- 存储器资源的组织（如内存的组织方式）
- 地址变换（逻辑地址与物理地址的对应关系维护）
- 虚拟存储的调度算法

存储组织

- 存储器的功能保存数据，存储器的发展方向是高速、大容量和小体积。如：内存在访问速度方面的发展：DRAM、SDRAM、SRAM等；硬盘技术在大容量方面的发展：接口标准、存储密度等；
- 存储组织的功能是在存储技术和CPU寻址技术许可的范围内组织合理的存储结构，其依据是访问速度匹配关系、容量要求和价格。如：“寄存器-内存-外存”结构和“寄存器-缓存-内存-外存”结构；
- 现在微机中的存储层次组织：访问速度越来越慢，容量越来越大，价格越来越便宜；最佳状态应是各层次的存储器都处于均衡的繁忙状态（如：缓存命中率正好使主存读写保持繁忙）；

存储层次结构


寄存器(register)
快速缓存(cache)
主存(primary storage)
外存(secondary storage)

存储管理的功能

- 存储分配和回收：是存储管理的主要内容。讨论其算法和相应的数据结构。
- 地址变换：可执行文件生成中的链接技术、程序加载时的重定位技术，进程运行时硬件和软件的地址变换技术和机构。
- 存储共享和保护：代码和数据共享，对地址空间的访问权限（读、写、执行）。
- 存储器扩充：它涉及存储器的逻辑组织和物理组织；
  - 由应用程序控制：覆盖；
  - 由OS控制：交换（整个进程空间），请求调入和预调入（部分进程空间）

程序的装入和链接

- 编译
- 链接
- 装入

重定位

- 在装入时对目标程序中的指令和数据地址的修改过程。

程序的装入

- 绝对装入方式
- 可重定位方式
- 动态运行时装入方式

程序的链接

- 静态链接
- 装入时动态链接
- 运行时动态链接

构造动态链接库

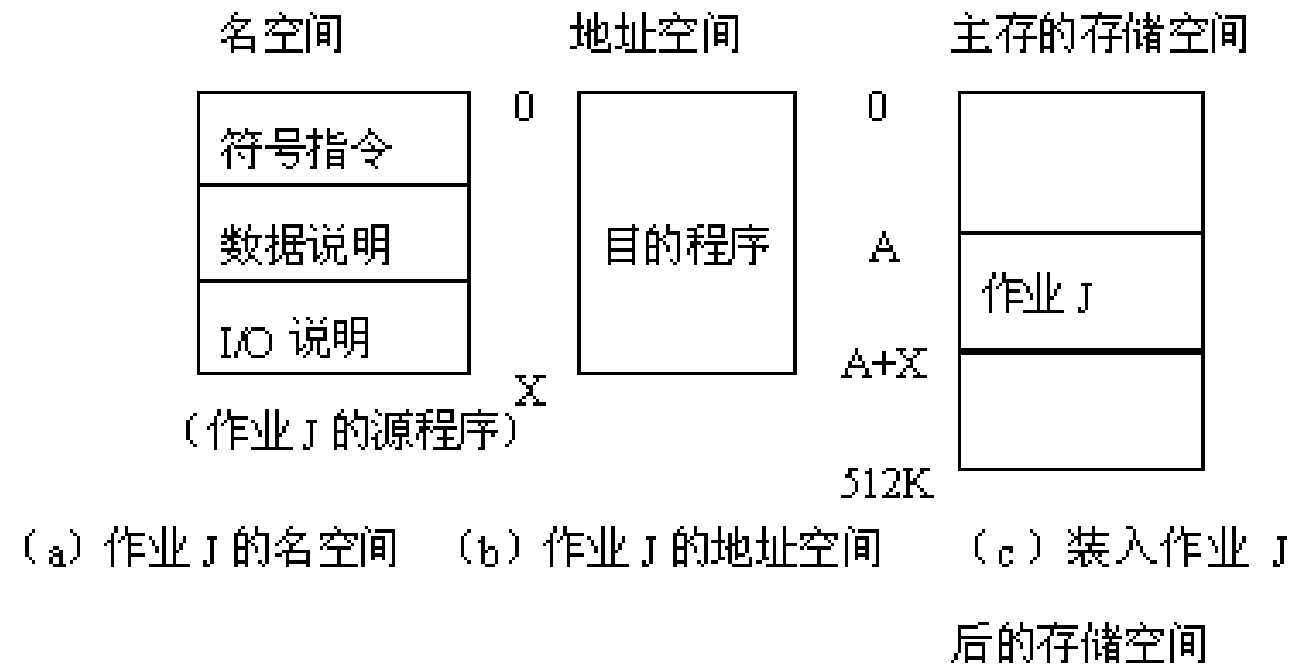
- DLL是包含函数和数据的模块，它的调用模块可为EXE或DLL，它由调用模块在运行时加载；加载时，它被映射到调用进程的地址空间。在VC中有一类工程用于创建DLL。
- 库程序文件 .C：相当于给出一组函数定义的源代码；
- 模块定义文件 .DEF：相当于定义链接选项，也可在源代码中定义；如：DLL中函数的引入和引出。

存储分配的三种方式

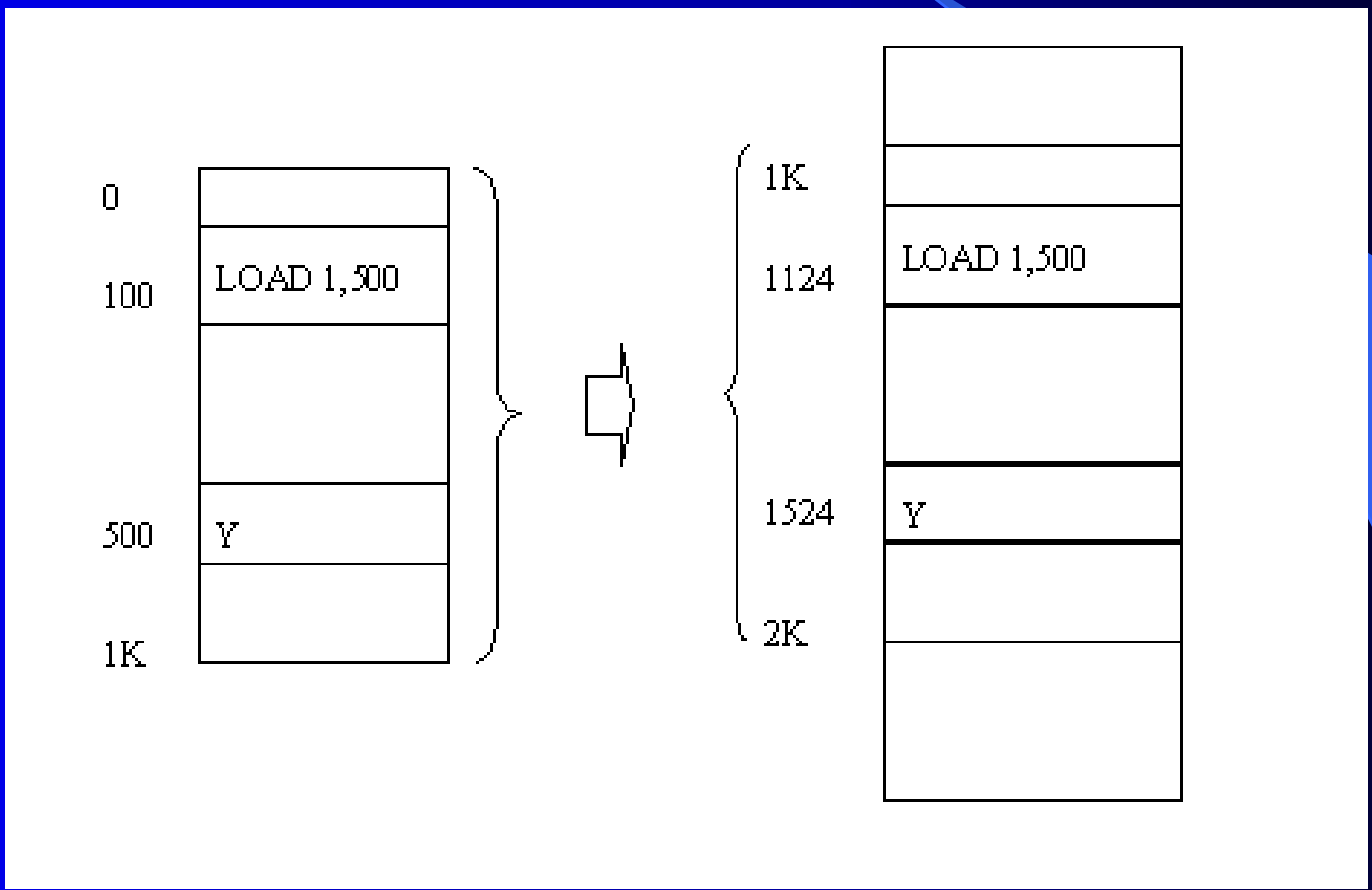
- 1.直接指定方式：程序员在编程序时,或编译程序(汇编程序)对源程序进行编译(汇编)时,所用的是实际存储地址。
- 2.静态分配(Static Allocation)：程序员编程时,或由编译程序产生的目的程序,均可从其地址空间的零地址开始；当装配程序对其进行连接装入时才确定它们在主存中的地址。
- 3.动态分配(Static Allocation)：作业在存储空间中的位置,在其装入时确定,在其执行过程中可根据需要申请附加的存储空间,而且一个作业已占用的部分区域不再需要时,可以要求归还给系统。

- 1．　地址空间：源程序经过编译后得到的目标程序，存在于它所限定的地址范围内，这个范围称为地址空间。简言之，地址空间是逻辑地址的集合。
- 2．　存储空间：存储空间是指主存中一系列存储信息的物理单元的集合，这些单元的编号称为物理地址或绝对地址。简言之，存储空间是物理地址的集合。

作业J存在于不同的空间中



作业由地址空间装入存储空间



连续分配存储管理方式

- 单一连续分配方式
- 分区分配方式
  - 固定分区
  - 动态分区

单一连续区存储管理

- 内存分为两个区域：系统区，用户区。应用程序装入到用户区，可使用用户区全部空间。
- 最简单，适用于单用户、单任务的OS。CP/M和DOS
- 优点：易于管理。
- 缺点：对要求内存空间少的程序，造成内存浪费；程序全部装入，很少使用的程序部分也占用内存。

多用户系统存储器管理----分区式分配

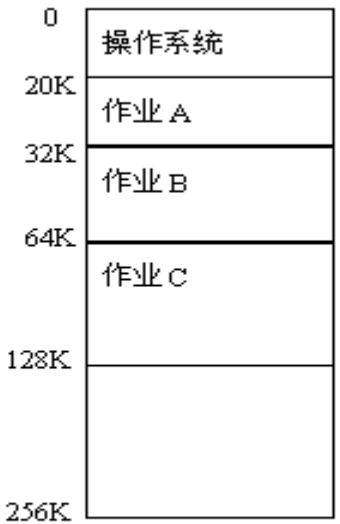
- 把内存分为一些大小相等或不等的分区(partition)，每个应用程序占用一个或几个分区。操作系统占用其中一个分区。
- 适用于多道程序系统和分时系统，支持多个程序并发执行，但难以进行内存分区的共享。

固定式分区

- 固定式分区（静态存储区域）：当系统初始化式，把存储空间划分成若干个任意大小的区域；然后，把这些区域分配给每个用户作业。

分区号	大小	起址	状态
1	12K	20K	已分配
2	32K	32K	已分配
3	64K	64K	已分配
4	128K	128K	未分配

分区说明表



存储空间分配情况

- 把内存划分为若干个固定大小的连续分区。
  - 分区大小相等：只适合于多个相同程序的并发执行（处理多个类型相同的对象）。
  - 分区大小不等：多个小分区、适量的中等分区、少量的大分区。根据程序的大小，分配当前空闲的、适当大小的分区。
- 优点：易于实现，开销小。
- 缺点：内碎片造成浪费，分区总数固定，限制了并发执行的程序数目。
- 采用的数据结构：分区表——记录分区的大小和使用情况

可变式分区：

- 可变式分区：分区的边界可以移动，即分区的大小可变。
- 优点：没有内碎片。缺点：有外碎片。

数据结构

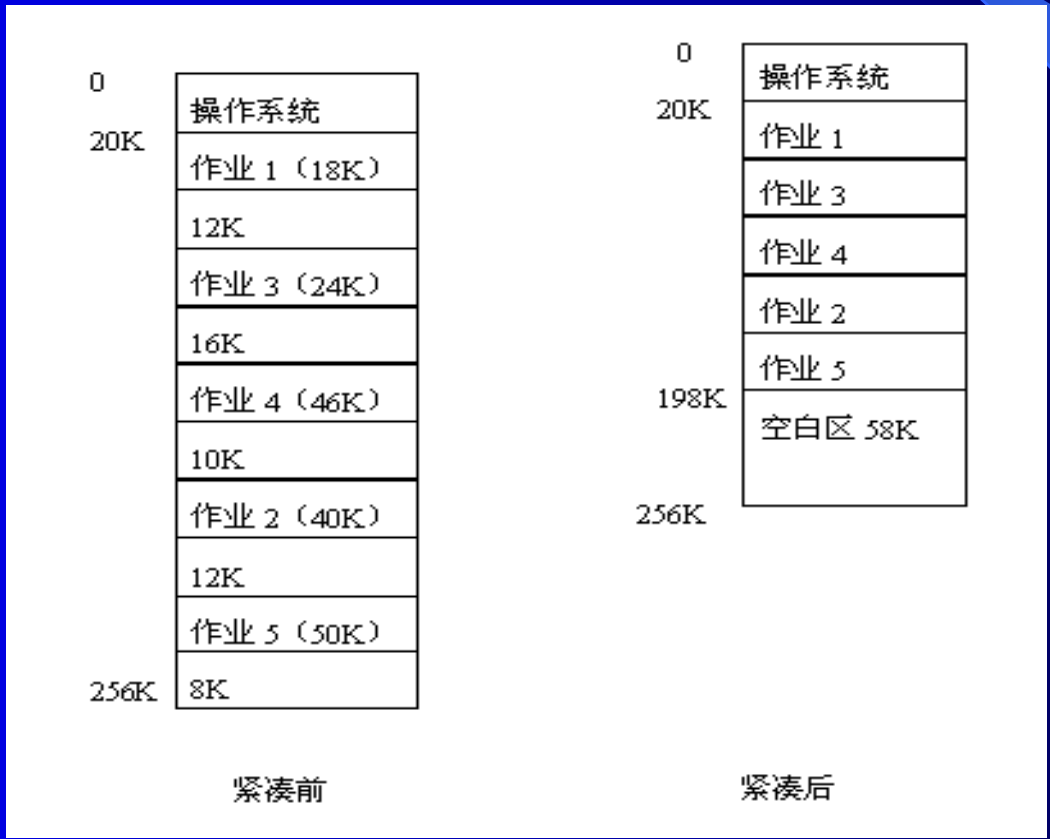
- 分区表
- 分区链表

可变式分区的分配策略：

- (1)最佳适应算法（Best Fit）：为一个作业选择分区时，总是寻找其大小最接近于作业所要求的存储区域。
- (2)最坏适应算法（Worst Fit）：为作业选择存储区域时，总是寻找最大的空白区。
- (3)首次适应算法（First Fit）：每个空白区按其在存储空间中地址递增的顺序连在一起，在为作业分配存储区域时，从这个空白区域链的始端开始查找，选择第一个足以满足请求的空白块。
- (4)下次适应算法（Next Fit）：把存储空间中

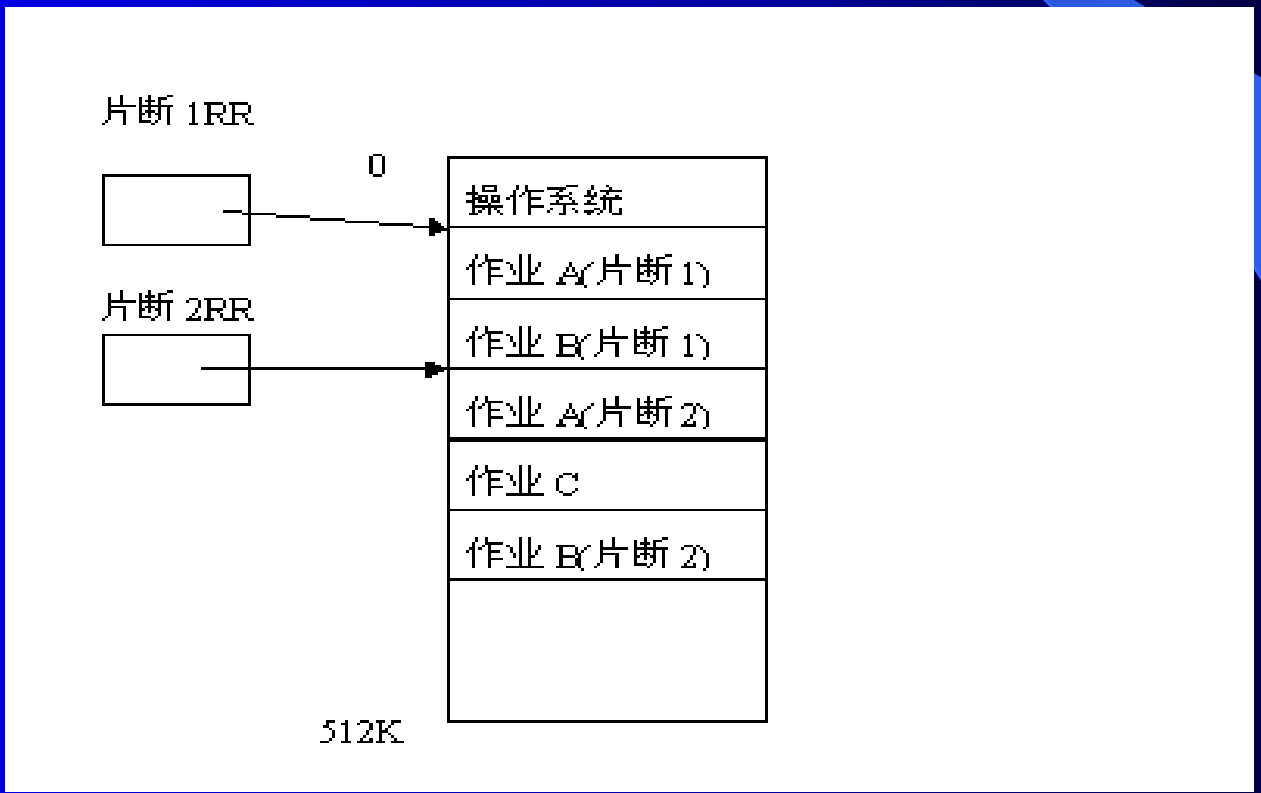
可重定位分区分配

- 可重定位分区分配：定时的或在内存紧张时，移动某些已分配区中的信息，把存储空间中所有的空白区合并为一个大的连续区。
-



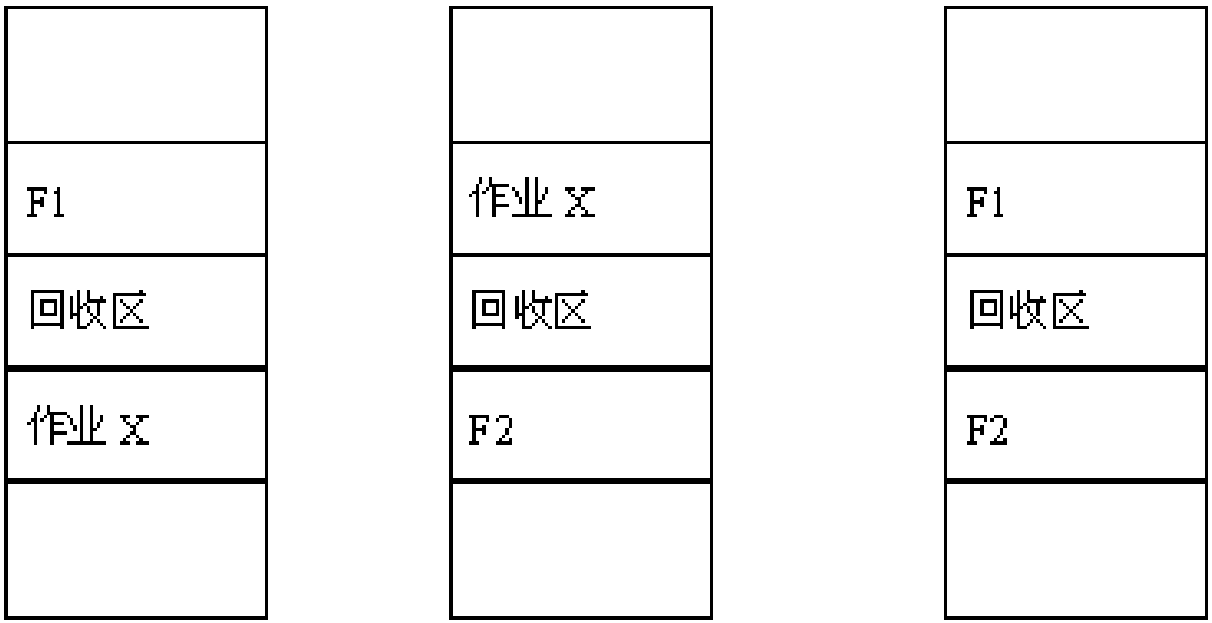
多重分区分配

- 多重分区分配：一个作业往往由相对独立的程序段和数据段组成，将这些片断分别装入到存储空间中不同的区域内的分配方式。



动态分区的操作和数据结构

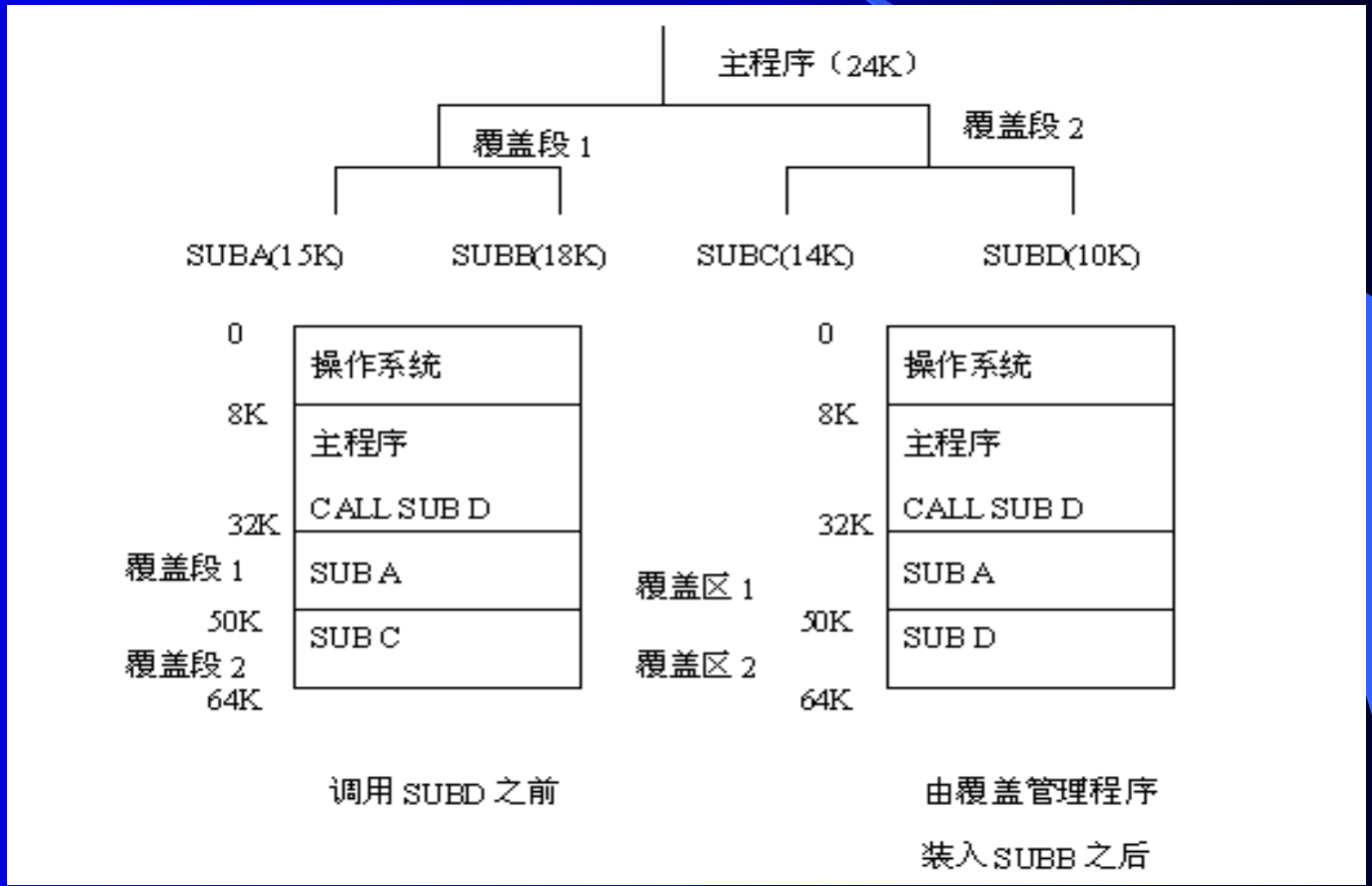
回收区域空白区邻接的三种情况



覆盖

- 覆盖：“覆盖”管理，就是把一个大的程序划分成一系列的覆盖，每个覆盖是一个相对独立的程序单位。把程序执行时并不要求同时装入主存的覆盖组成一组，称其为覆盖段，这个覆盖段被分配到同一个存储区域。这个存储区域称之为覆盖区，它与覆盖段一一对应。
- 缺点：编程时必须划分程序模块和确定程序模块之间的覆盖关系，增加编程复杂度。从外存装入覆盖文件，以时间延长来换取空间节省。

覆盖管理

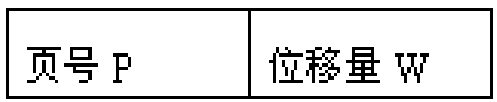

主程序（24K）
覆盖段 1
覆盖段 2
SUBA(15K)
SUBB(18K)
SUBC(14K)
SUBD(10K)
0
操作系统
8K
主程序
CALL SUB D
32K
覆盖段 1
SUB A
50K
SUB C
覆盖段 2
64K
调用 SUBD 之前
覆盖区 1
覆盖区 2
0
操作系统
8K
主程序
CALL SUB D
32K
SUB A
50K
SUB D
64K
由覆盖管理程序
装入 SUBB 之后

交换

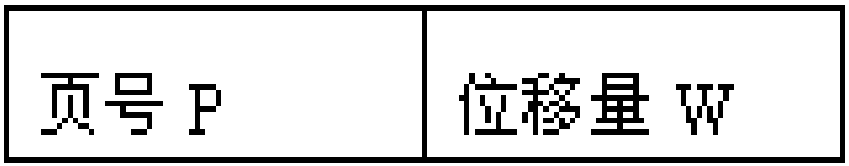
- 交换：广义的说，所谓交换就是把暂时不用的某个（或某些）程序及其数据的部分或全部从主存移到辅存中去，以便腾出必要的存储空间；接着把指定程序或数据从辅存读到相应的主存中，并将控制转给它，让其在系统上运行。
- 优点：增加并发运行的程序数目，并且给用户提供适当的响应时间；编写程序时不影响程序结构
- 缺点：对换入和换出的控制增加处理机开销；程序整个地址空间都进行传送，没有考虑执行过程中地址访问的统计特性。

分页式存储管理

- 碎片和紧凑问题
- 页：在分页存储管理系统中，把每个作业的地址空间分成一些大小相等的片，称之为页面或页。
- 存储块：在分页存储管理系统中，把主存的存储空间也分成与页面相同大小的片，这些片称为存储块



分页系统的地址结构



分页系统的地址结构

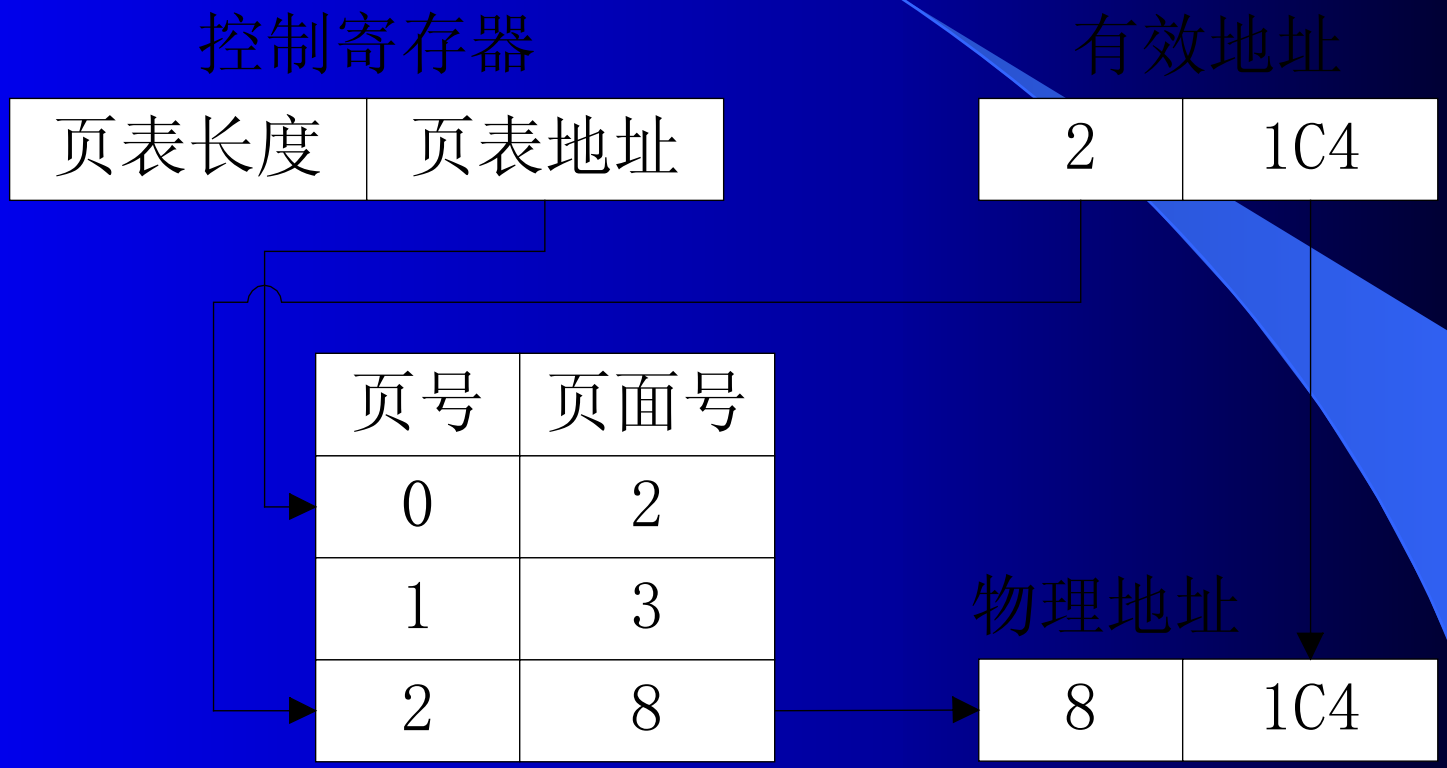
纯分页系统（Pure Paging System）

- 在调度一个作业时，必须把它的所有页一次装到主存的页框内；如果当时页框数不足，则该作业必须等待，系统再调度另外作业。
- 优点：
  - 没有外碎片，每个内碎片不超过页大小。
  - 一个程序不必连续存放。便于改变程序占用空间的大小（主要指随着程序运行而动态生成的数据增多，要求地址空间相应增长，通常由系统调用完成而不是操作系统自动完成）。
- 缺点：程序全部装入内存。

页表

页面大小

地址变换



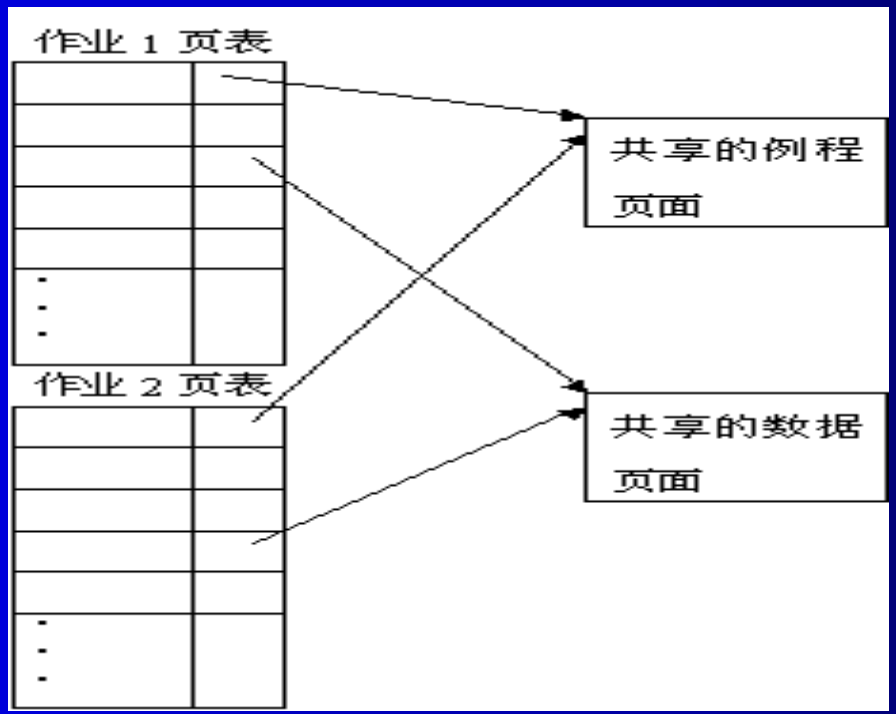
快表

两级页表

多级页表

反置页表

- 页面共享

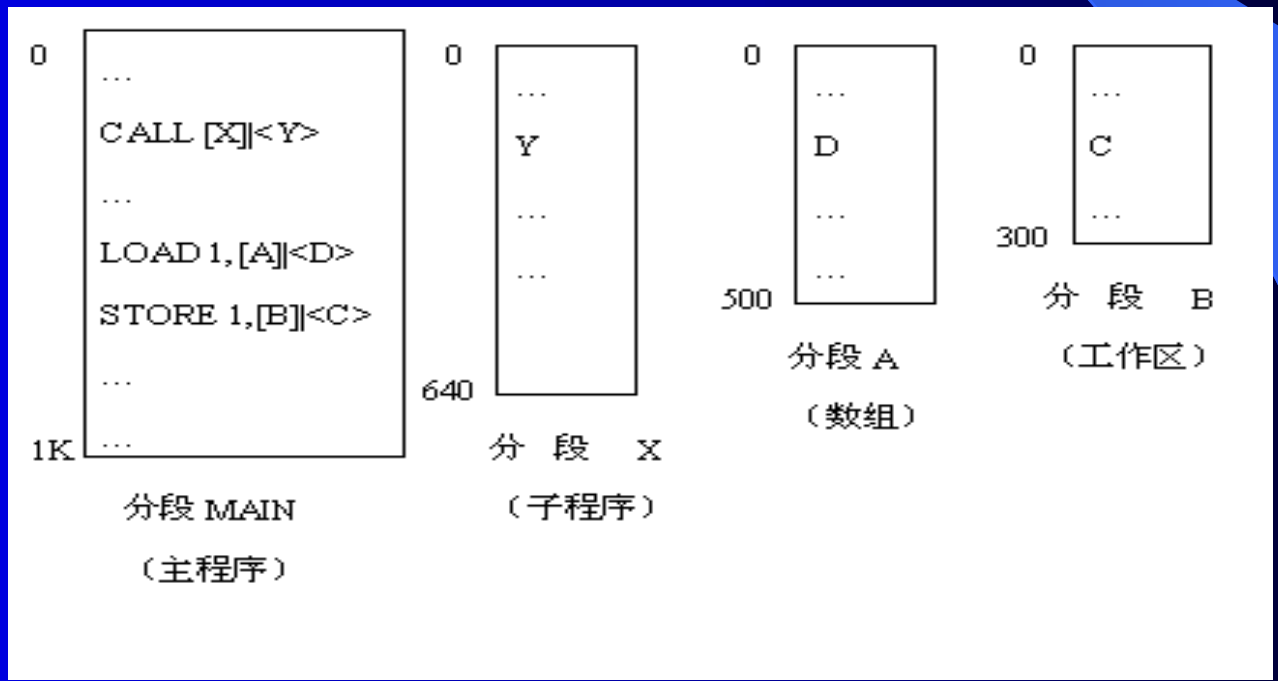


分段存储管理

- 方便编程
- 分段共享
- 分段保护
- 动态链接
- 动态保护
- 动态增长

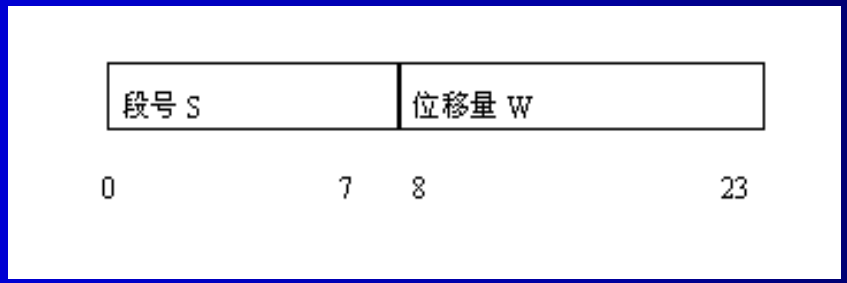
分段存储管理

- 分段地址空间
- 一个段可定义为一组逻辑信息，每个作业的地址空间是由一些分段构成的，每段都有自己的名字，且都是一段连续的地址空间。

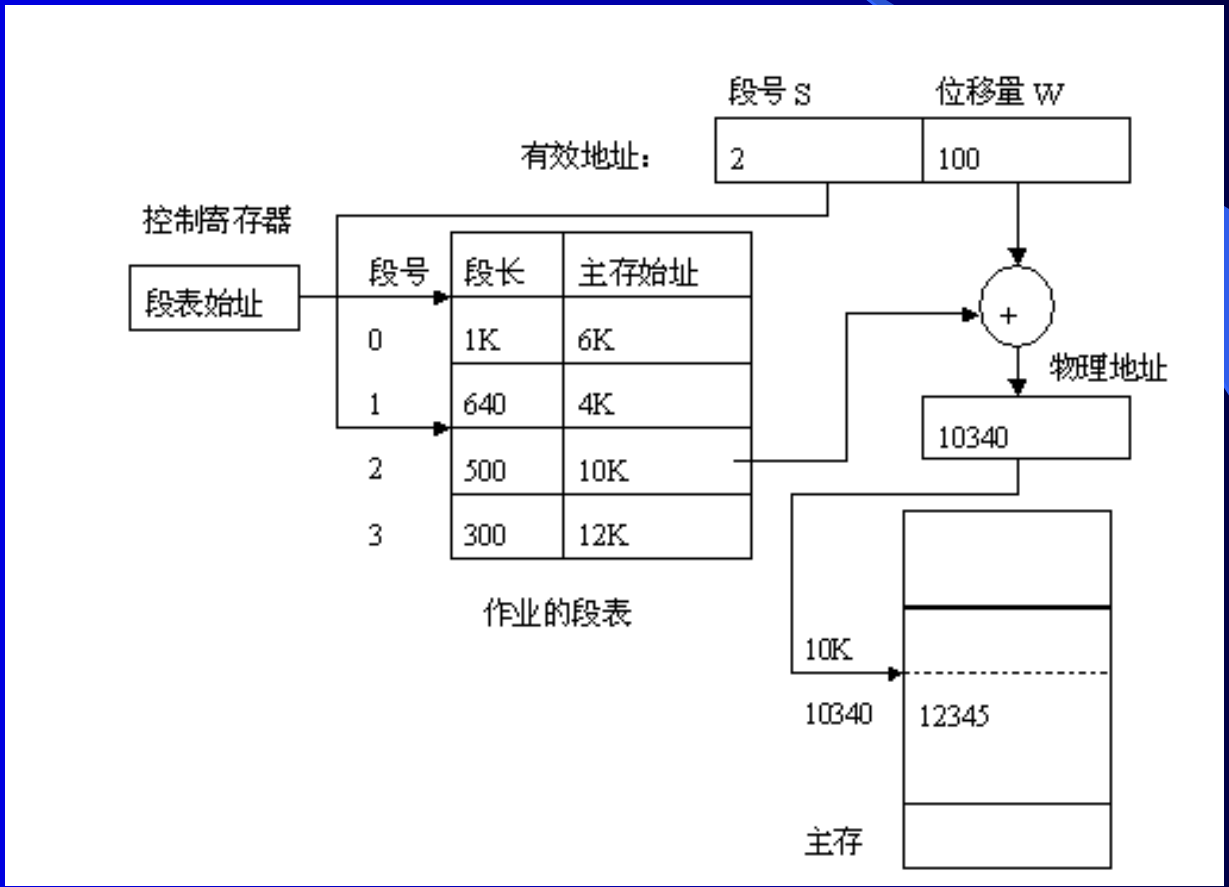


地址结构：

- 地址结构： 段号S + 位移量W
-

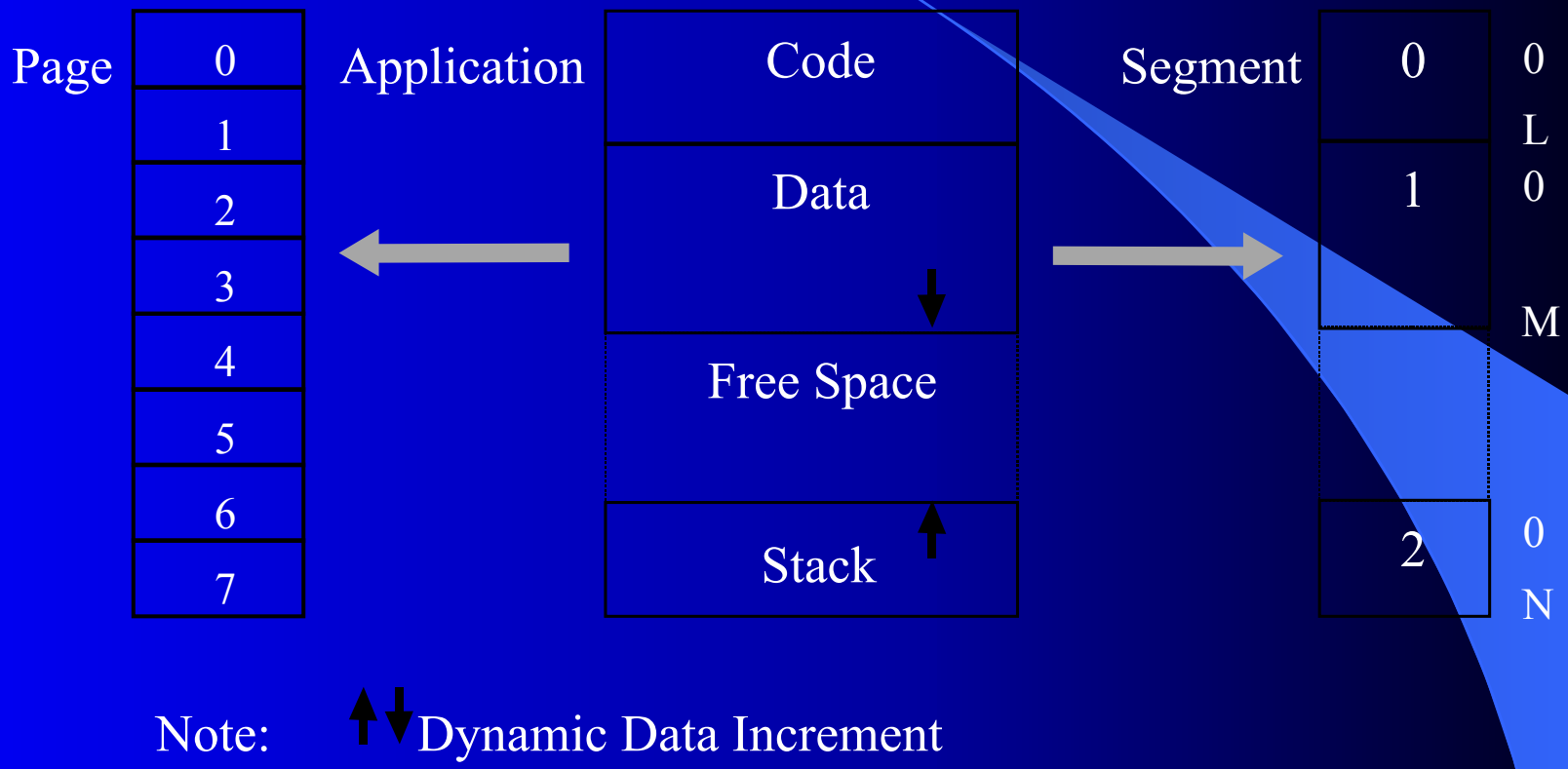


分段地址变换过程


段号 S
位移量 W
有效地址：
2
100
控制寄存器
段表始址
段号
段长
主存始址
0
1K
6K
1
640
4K
2
500
10K
3
300
12K
作业的段表
+
物理地址
10340
10K
10340
12345
主存

实现原理

- 分页与分段的比较：
- 1. 分页的作业的地址空间是单一的线性地址空间，分段作业的地址空间是二维的。
- 2. “页”是信息的“物理”单位，大小固定。“段”是信息的逻辑单位，即它是一组有意义的信息，其长度不定。
- 3. 分页活动用户是看不见的，而是系统对于主存的管理。分段是用户可见的（分段可以在用户编程时确定，也可以在编译程序对源程序编译时根据信息的性质来划分）。

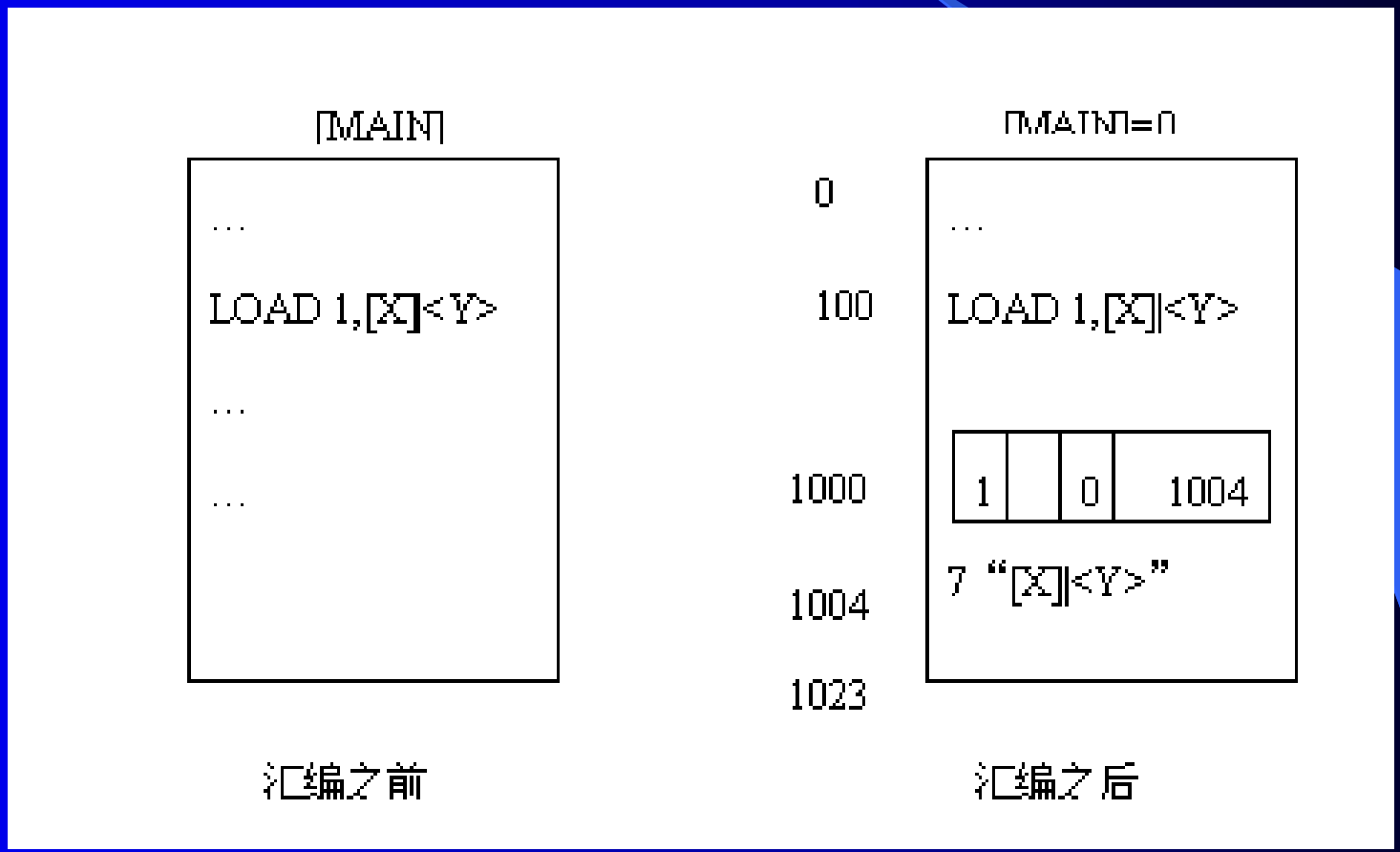


Note: Dynamic Data Increment

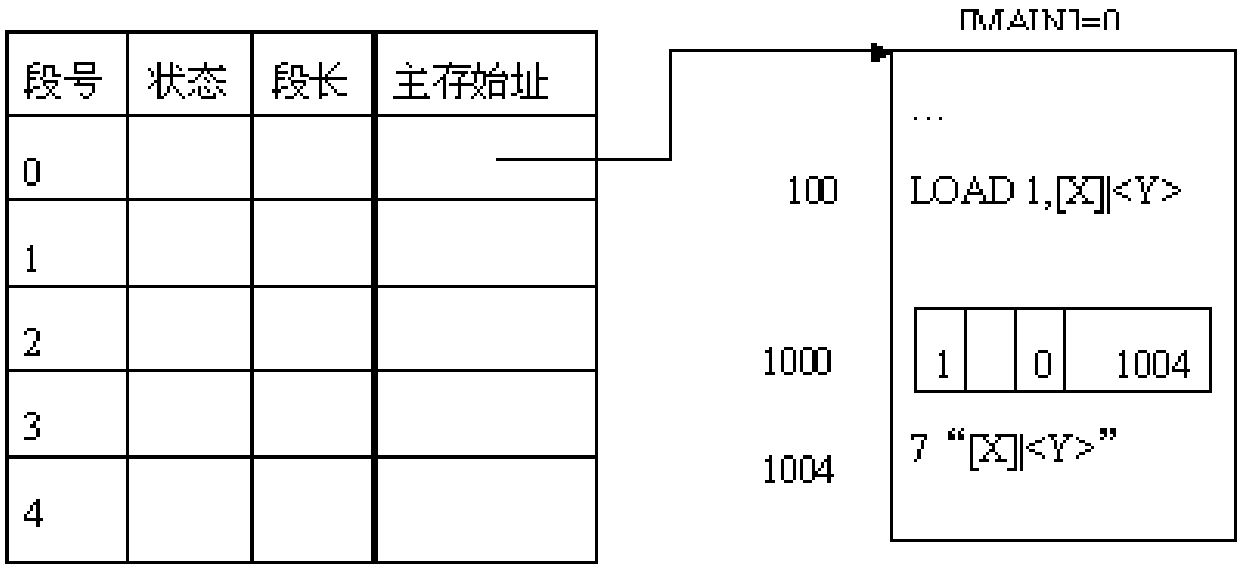
分段动态链接

- 静态链接：如果一个作业由若干个程序模块组成，在单一线性地址空间情况下，这些模块在执行之前由装配程序把它们链接和重定位。
- 动态链接：在程序运行过程中，到需要调用一程序模块时，再去链接它。

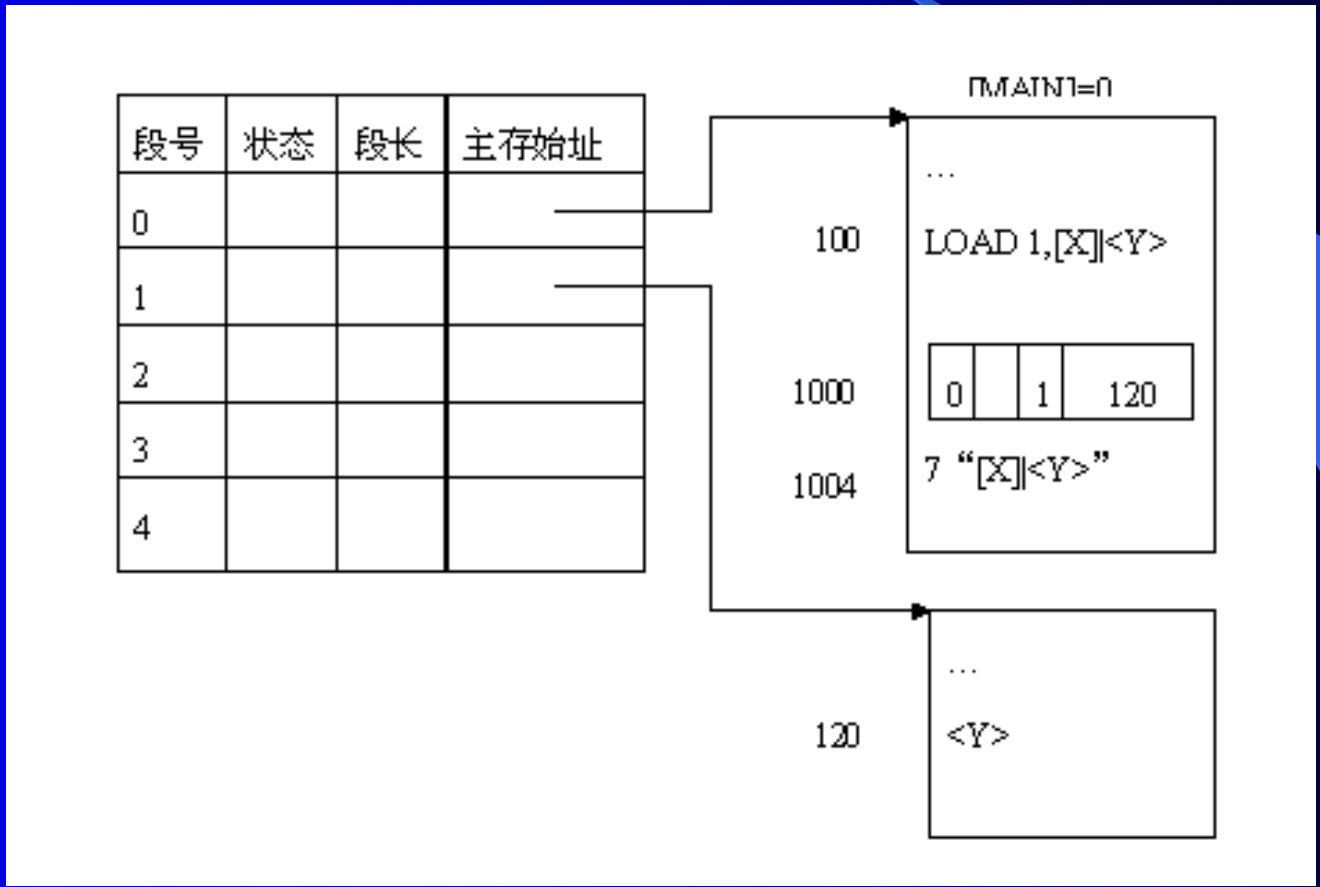
对段外访问的处理


[MAIN]
...
LOAD 1,[X]<Y>
...
...
汇编之前
[MAIN]=0
0
100
1000
1004
1023
...
LOAD 1,[X]|<Y>
1
0
1004
7 “[X]|<Y>”
汇编之后

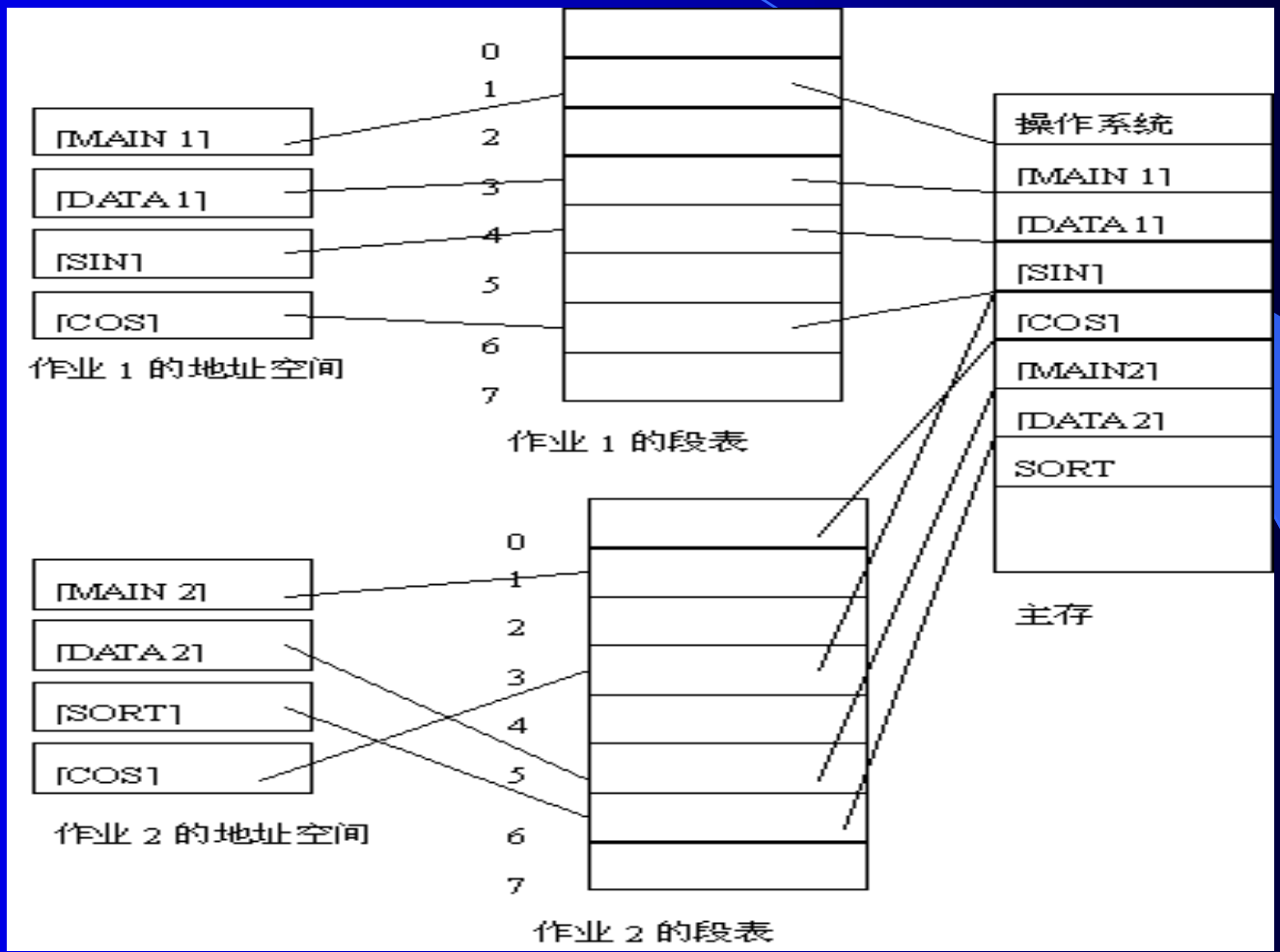
分段链接之前的情况



分段链接之后的情况


段号
状态
段长
主存始址
0
1
2
3
4
[MAIN]=0
…
100
LOAD 1,[X]|<Y>
1000
0
1
120
1004
7 “[X]|<Y>”
…
120
<Y>

分段的共享

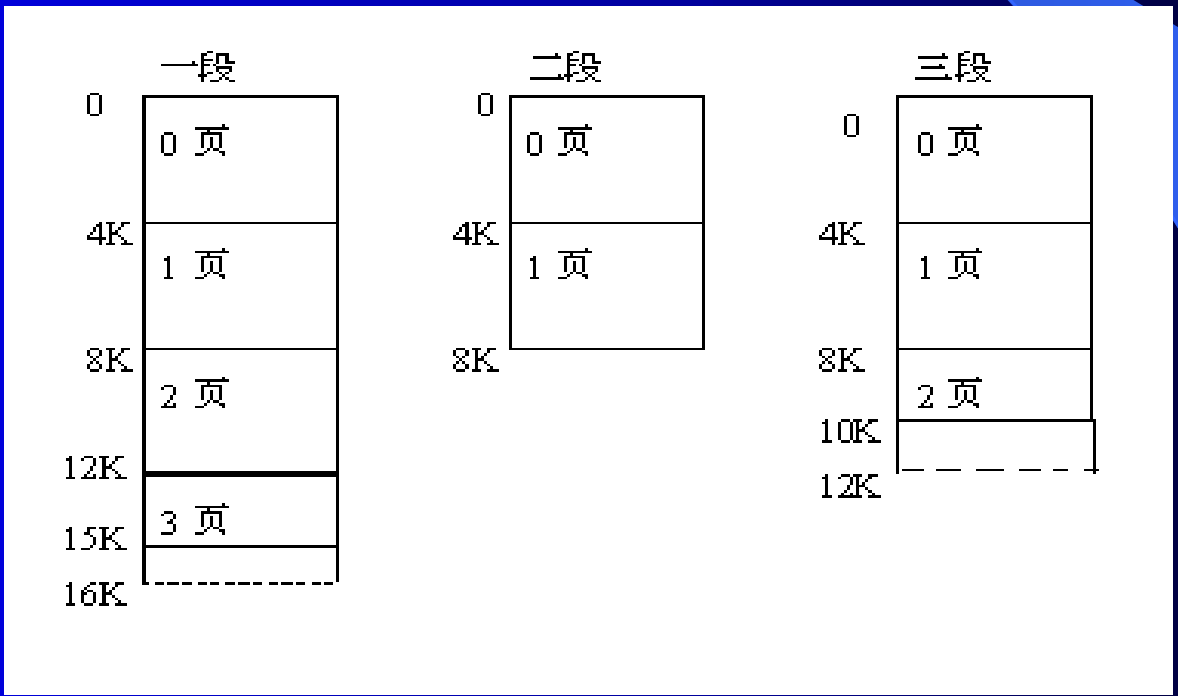

0
1
2
3
4
5
6
7
[MAIN 1]
[DATA 1]
[SIN]
[COS]
作业 1 的地址空间
作业 1 的段表
操作系统
[MAIN 1]
[DATA 1]
[SIN]
[COS]
[MAIN2]
[DATA 2]
SORT
主存
[MAIN 2]
[DATA 2]
[SORT]
[COS]
作业 2 的地址空间
作业 2 的段表

分段管理的优缺点

- 缺点：
- 1. 处理机要为地址变换花费时间；要为表格提供附加的存储空间。
- 2. 为满足分段的动态增长和减少外零头，要采用拼接手段。
- 3. 在辅存中管理不定长度的分段困难较多。
- 4. 分段的最大尺寸受到主存可用空间的限制。

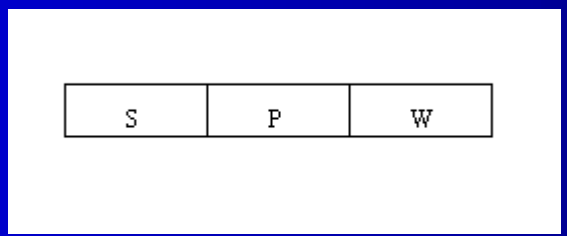
段页式存储管理——另一种段式虚拟存储器

- 基本思想：用分段方法来分配和管理虚拟存储器，而用分页方法来分配和管理实存储器。
-



实现原理

- 一个程序首先被分成若干程序段，每一段赋予不同的分段标识符，然后，对每一分段又分成若干个固定大小的页面。
- 地址结构： S + P + W
- S：段号；P：页；W：页内位移量



虚拟存储器

北京航空航天大学计算机科学与工程系　王雷

局部性原理

- 程序在执行时，大部分是顺序执行的指令，少部分是转移和过程调用指令。
- 过程调用的嵌套深度一般不超过5，因此执行的范围不超过这组嵌套的过程。
- 程序中存在相当多的循环结构，它们由少量指令组成，而被多次执行。
- 程序中存在相当多对一定数据结构的操作，如数组操作，往往局限在较小范围内。

局部性原理

- 指程序在执行过程中的一个较短时期，所执行的指令地址和指令的操作数地址，分别局限于一定区域。还可以表现为：
  - 时间局部性，即一条指令的一次执行和下次执行，一个数据的一次访问和下次访问都集中在一个较短时期内；
  - 空间局部性，即当前指令和邻近的几条指令，当前访问的数据和邻近的数据都集中在一个较小区域内。

虚拟存储的基本原理

- 在程序装入时，不必将其全部读入到内存，而只需将当前需要执行的部分页或段读入到内存，就可让程序开始执行。
- 在程序执行过程中，如果需执行的指令或访问的数据尚未在内存（称为缺页或缺段），则由处理器通知操作系统将相应的页或段调入到内存，然后继续执行程序。
- 另一方面，操作系统将内存中暂时不使用的页或段调出保存在外存上，从而腾出空间存放将要装入的程序以及将要调入的页或段——具有请求调入和置换功能，只需程序的一部分在内存就可执行，对于动态链接库也可以请求调入

优点

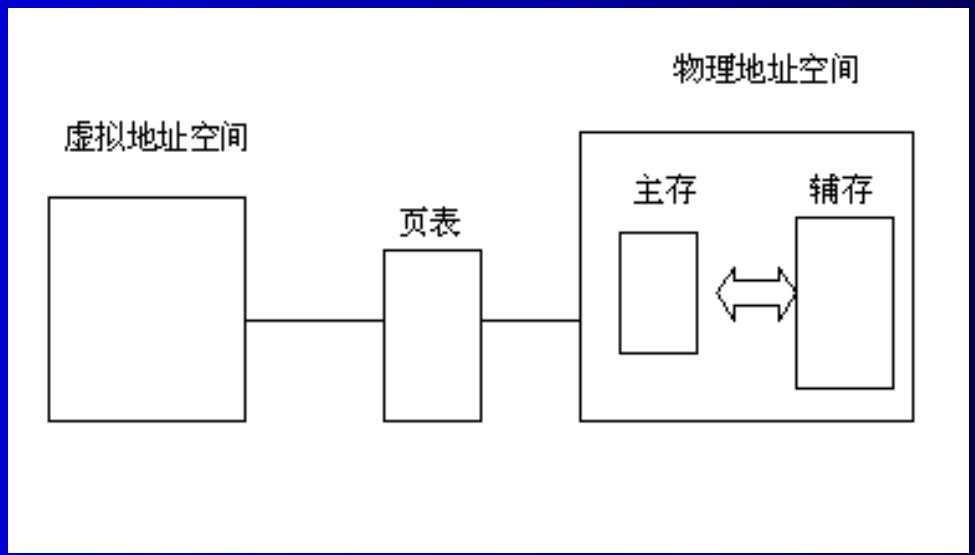
- 可在较小的可用内存中执行较大的用户程序；
- 可在内存中容纳更多程序并发执行；
- 不必影响编程时的程序结构（与覆盖技术比较）
- 提供给用户可用的虚拟内存空间通常大于物理内存(**real memory**)

虚拟存储技术的特征

- 离散性：物理内存分配的不连续，虚拟地址空间使用的不连续（数据段和栈段之间的空闲空间，共享段和动态链接库占用的空间）
- 多次性
- 对换性：与交换的比较：调入和调出是对部分虚拟地址空间进行
- 虚拟性：通过物理内存和快速外存相结合，提供大范围的虚拟地址空间
  - 范围大，但占用容量不超过物理内存和外存交换区容量之和
  - 占用容量包括：进程地址空间中的各个段，操作系统代码

请求式分页系统

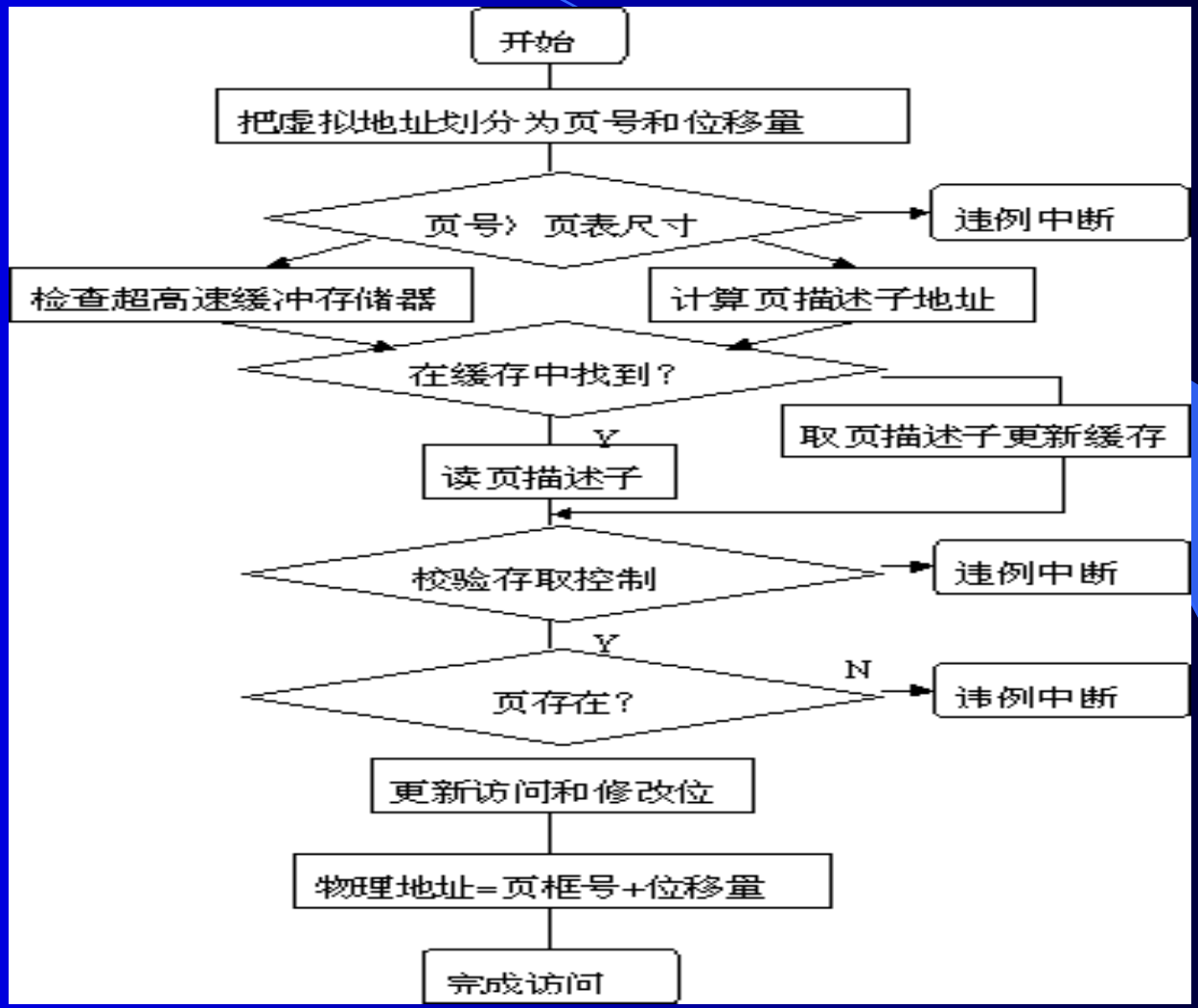
- 在运行作业之前，只要求把当前需要的一部分页面装入主存。当需要其它的页时，可自动的选择一些页交换倒辅存去，同时把所需的页调入主存。
- 虚拟存储系统：控制自动页面交换而用户作业意识不到的那个机构，成为虚拟存储系统。



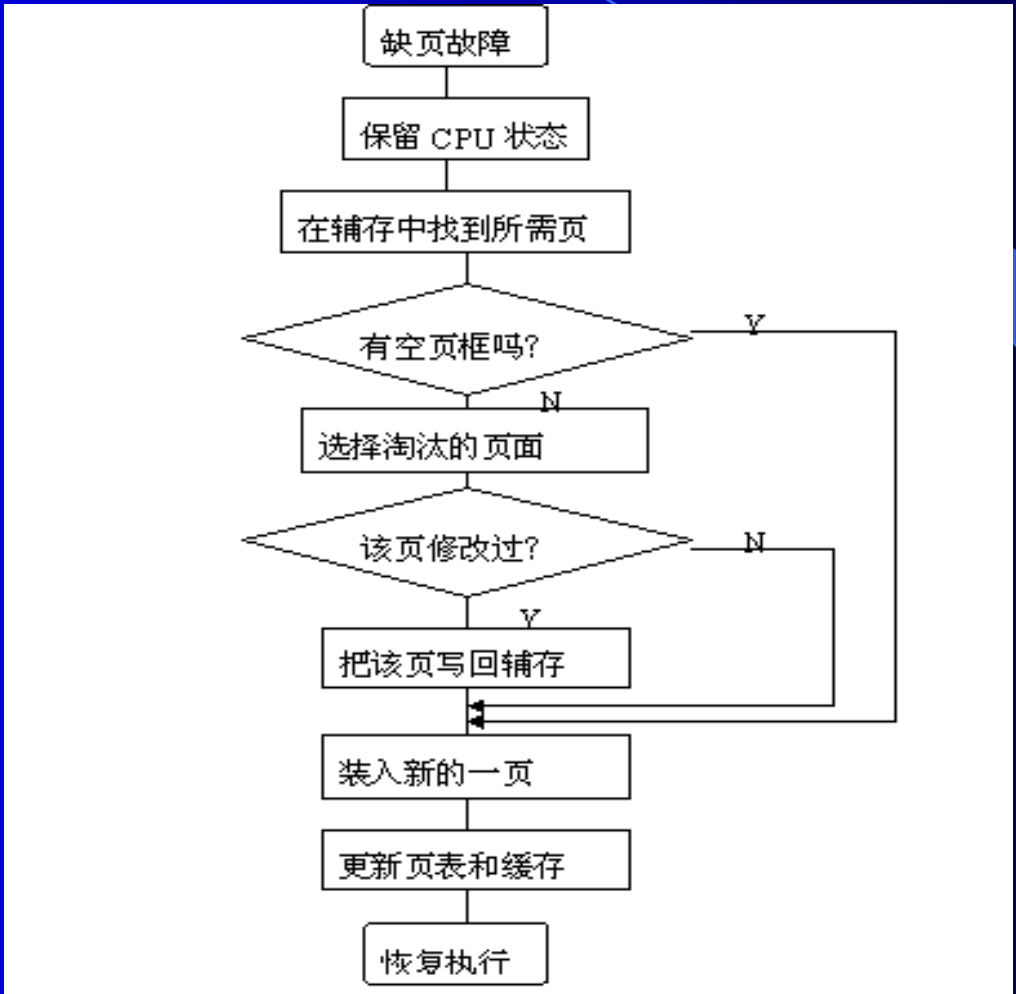
页表机制

- 需要在进程页表中添加若干项
  标志位：存在位（**present bit**，内存页和外存页），修改位**(modified bit)**
  访问统计：在近期内被访问的次数，或最近一次访问到现在的时间间隔外存地址

地址变换


开始
把虚拟地址划分为页号和位移量
页号〉 页表尺寸
违例中断
检查超高速缓冲存储器
计算页描述子地址
在缓存中找到？
取页描述子更新缓存
Y
读页描述子
校验存取控制
违例中断
Y
N
页存在？
违例中断
更新访问和修改位
物理地址=页框号+位移量
完成访问

缺页故障处理


缺页故障
保留 CPU 状态
在辅存中找到所需页
有空页框吗?
Y
N
选择淘汰的页面
该页修改过?
N
Y
把该页写回辅存
装入新的一页
更新页表和缓存
恢复执行

虚拟存储器的管理

- 最小物理块数问题
- 分配问题
- 置换问题

分配和置换策略

- 固定分配局部置换
- 可变分配全局置换
- 可变分配局部置换

页面调入策略

- 1. 请求式提取：仅当需要时才提取页面的策略。
- 2. 预先调页：事先提取页面的策略。

- 系统对换区
- 从文件调入

页面置换策略（Replacement Strategies）

- 一，一种最佳策略:从主存中移出永远不再需要的页面，如无这样的页面存在，则应选择最长时间不需要访问的页面。
- 二，先进先出算法（First-in, First-out）:总选择作业中在主存驻留时间最长的一页淘汰。
- 三，最近最久不用的页面置换算法（Least Recently Used Replacement）:当需要置换一页面时，选择在最近一段时间内最久不用的页面予以淘汰。

Clock算法

- 也称最近未使用算法(NRU, Not Recently Used)，它是LRU和FIFO的折衷。
  - 每页有一个使用标志位(use bit)，若该页被访问则置user bit=1。
  - 置换时采用一个指针，从当前指针位置开始按地址先后检查各页，寻找use bit=0的页面作为被置换页。
  - 指针经过的user bit=1的页都修改user bit=0，最后指针停留在被置换页的下一个页。

最不常用算法(LFU, Least Frequently Used)

- 选择到当前时间为止被访问次数最少的页面被置换；
- 每页设置访问计数器，每当页面被访问时，该页面的访问计数器加1；
- 发生缺页中断时，淘汰计数值最小的页面，并将所有计数清零；

页面缓冲算法(page buffering)

- 它是对FIFO算法的发展，通过被置换页面的缓冲，有机会找回刚被置换的页面；
- 被置换页面的选择和处理：用FIFO算法选择被置换页，把被置换的页面放入两个链表之一。即：如果页面未被修改，就将其归入到空闲页面链表的末尾，否则将其归入到已修改页面链表。

工作集策略(working set strategy)

- 引入工作集的目的是依据进程在过去的一段时间内访问的页面来调整常驻集大小。
  - 工作集的定义：工作集是一个进程执行过程中所访问页面的集合，可用一个二元函数**W(t, Δ)**表示，其中：**t**是执行时刻；**Δ**是窗口尺寸**(window size)**；工作集是在**[t - Δ, t]**时间段内所访问的页面的集合，**| W(t, Δ) |** 指工作集大小即页面数目；
- 工作集大小的变化：进程开始执行后，随着访问新页面逐步建立较稳定的工作集。当内存访问的局部性区域的位置大致稳定时，工作集大小也大致稳定；局部性区域的位置改变时，工作集快速扩张和收缩过渡到下一个稳定值。

困难：

- 工作集的过去变化未必能够预示工作集的将来大小或组成页面的变化；
- 记录工作集变化要求开销太大；
- 对工作集窗口大小Δ的取值难以优化，而且通常该值是不断变化的；

改善时间性能的途径

- 降低缺页率：缺页率越低，虚拟存储器的平均访问时间延长得越小；
- 提高外存的访问速度：外存和内存的访问时间比值越大，则达到同样的时间延长比例，所要求的缺页率就越低；

抖动问题(thrashing)

- 随着驻留内存的进程数目增加，或者说进程并发水平(multiprogramming level)的上升，处理器利用率先是上升，然后下降。这里处理器利用率下降的原因通常称为虚拟存储器发生“抖动”，也就是：每个进程的常驻集不断减小，缺页率不断上升，频繁调页使得调页开销增大。OS要选择一个适当的进程数目，以在并发水平和缺页率之间达到一个平衡。

抖动的预防

- 局部置换策略
- 在CPU调度程序中引入工作集算法
- L=S准则
- 挂起若干进程

请求分段式

- 在简单段式存储管理的基础上，增加请求调段和段置换功能。
- 地址变换和缺段中断：指令和操作数必定不会跨越在段边界上

保护

- 界限保护（上下界界限地址寄存器）：所有访问地址必须在上下界之间；
- 存取控制检查
- 环保护：处理器状态分为多个环(ring)，分别具有不同的存储访问特权级别(privilege)，通常是级别高的在内环，编号小（如0环）级别最高；可访问同环或更低级别环的数据；可调用同环或更高级别环的服务。

计算机操作系统

主讲教师：王　雷

磁盘储存器管理

- 分配空间
- 组织文件的存取方式
- 提高磁盘储存空间的利用率
- 提高I/O速度
- 保证文件系统的可靠性

提高I/O速度的主要途径

- 选择性能好的磁盘
- 采用适当的调度算法
- 设置磁盘高速缓冲区

磁盘的组织

磁盘的类型

- 固定头磁盘
- 移动头磁盘

磁盘访问时间

- 寻道时间
- 旋转延迟时间
- 传输时间

磁盘调度算法

- 先来先服务
- 最短寻道时间优先

- 扫描算法
- 循环扫描算法

- N-Step-SCAN
- FSCANS

文件物理组织的不同方式

1. 连续文件
2. 串联文件
3. 索引文件
4. **Hash文件**

1. 连续分配

- 连续分配(contiguous)：只需记录第一个簇的位置，适用于预分配方法。可以通过紧缩(compact)将外存空闲空间合并成连续的区域。

2. 链接分配

- 链接分配(chained)：在每个簇中有指向下一个簇的指针。可以通过合并(consolidation)将一个文件的各个簇连续存放，以提高I/O访问性能。

3. 索引分配

- 索引分配(indexed)：文件的第一个簇中记录了该文件的其他簇的位置。可以每处存放一个簇或连续多个簇（只需在索引中记录连续簇的数目）。

存储器存储空间的管理

- 空闲表法
- 空闲链表法
- 位视图
- 成组链接法

磁盘容错技术

- SFT-I
- SFT-II
- SFT-III

SFT-I

- 双份目录和双份文件分配表
- 热修复定向和写后读校验

SFT-II

- 磁盘镜像
- 磁盘双工

RAID

- 并行交叉存取
- **RAID**分级

RAID的优点

- 可靠性高
- 磁盘I/O速度高
- 性能价格比

后备系统

- 类型
  - 磁带机
  - 硬盘
  - 光盘
- 拷贝方法
  - 完全转储法
  - 增量转储法

高速缓存

- 磁盘高速缓存的形式
- 数据交付
- 置换算法
- 周期性写回

优化数据布局

- 优化物理块的分布
- 优化索引节点的分布

- 提前读
- 延迟写
- 虚拟盘

数据一致性控制

事务

- 事务定义
- 事务记录
- 恢复算法
  - **undo**
  - **redo**

检查点

- 检查点的作用
- 新的恢复算法

并发控制

- 利用互斥锁
- 利用互斥锁和共享锁

- 重复文件的一致性
- 盘块号一致性的检查
- 链接数一致性检查

处理机调度

- 调度的类型与模型
- 调度算法
- 实时系统中的调度
- 多处理机调度

问题

- 处理机管理的工作是对CPU资源进行合理的分配使用，以提高处理机利用率，并使各用户公平地得到处理机资源。这里的主要问题是处理机调度算法和调度算法特征分析。

调度的类型

- 高级调度
- 中级调度
- 低级调度

高级调度

- 高级调度：又称为“宏观调度”、“作业调度”。从用户工作流程的角度，一次提交的若干个作业，对每个作业进行调度。时间上通常是分钟、小时或天。

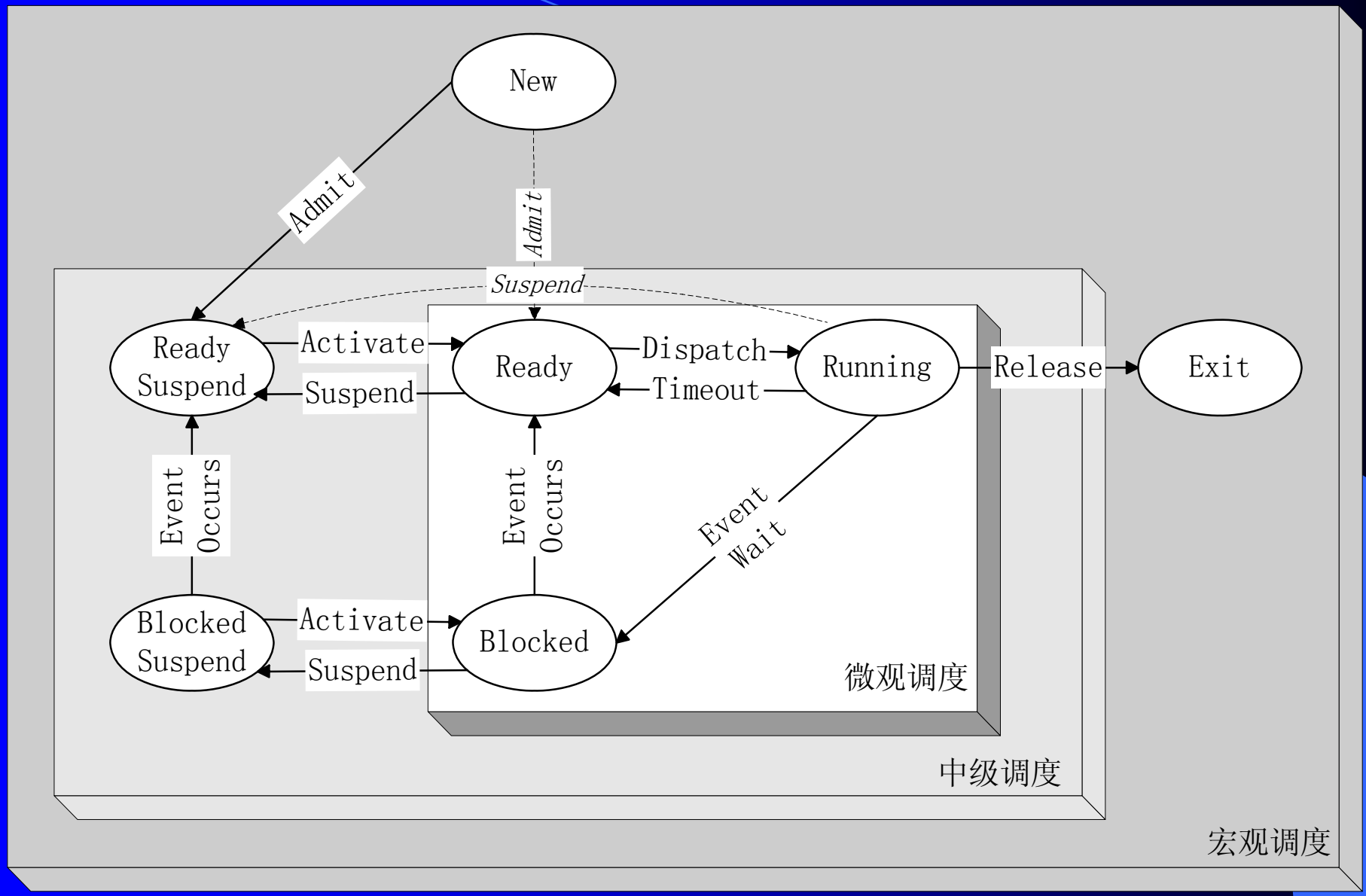
- 接纳多少个作业
- 接纳那些作业

中级调度

- 内外存交换：又称为“中级调度”。从存储器资源的角度。将进程的部分或全部换出到外存上，将当前所需部分换入到内存。指令和数据必须在内存里才能被CPU直接访问。

低级调度

- 低级调度：又称为“微观调度”、“进程或线程调度”。从CPU资源的角度，执行的单位。时间上通常是毫秒。因为执行频繁，要求在实现时达到高效率。
- 非抢占式
- 抢占式
  - 时间片原则
  - 优先权原则
  - 短作业（进程）优先


New
Admit
Admit
Suspend
Ready
Suspend
Activate
Suspend
Ready
Dispatch
Timeout
Running
Release
Exit
Event
Occurs
Event
Occurs
Event
Wait
Blocked
Suspend
Activate
Suspend
Blocked
微观调度
中级调度
宏观调度

调度的性能准则

- 从不同的角度来判断处理机调度算法的性能，如用户的角度、处理机的角度和算法实现的角度。实际的处理机调度算法选择是一个综合的判断结果。

面向用户的调度性能准则1

- 周转时间：作业从提交到完成（得到结果）所经历的时间。包括：在收容队列中等待，CPU上执行，就绪队列和阻塞队列中等待，结果输出等待——批处理系统
  - 外存等待时间、就绪等待时间、CPU执行时间、I/O操作时间
  - 平均周转时间、带权平均周转时间（T/Ts)
- 响应时间：用户输入一个请求（如击键）到系统给出首次响应（如屏幕显示）的时间——分时系统

面向用户的调度性能准则2

- 截止时间：开始截止时间和完成截止时间——实时系统，与周转时间有些相似。
- 优先级：可以使关键任务达到更好的指标。
- 公平性：不因作业或进程本身的特性而使上述指标过分恶化。如长作业等待很长时间。

面向系统的调度性能准则

- 吞吐量：单位时间内所完成的作业数，跟作业本身特性和调度算法都有关系——批处理系统
  - 平均周转时间不是吞吐量的倒数，因为并发执行的作业在时间上可以重叠。如：在2小时内完成4个作业，而每个周转时间是1小时，则吞吐量是2个作业/小时
- 处理机利用率：——大中型主机
- 各种资源的均衡利用：如CPU繁忙的作业和I/O繁忙（指次数多，每次时间短）的作业搭配——大中型主机

调度算法本身的调度性能准则

- 易于实现
- 执行开销比

调度算法

- 通常将作业或进程归入各种就绪或阻塞队列。有的算法适用于作业调度，有的算法适用于进程调度，有的两者都适应。

先来先服务(FCFS, First Come First Service)

- 这是最简单的调度算法，按先后顺序调度。
  - 按照作业提交或进程变为就绪状态的先后次序，分派CPU；
  - 当前作业或进程占用CPU，直到执行完或阻塞，才出让CPU（非抢占方式）。
  - 在作业或进程唤醒后（如I/O完成），并不立即恢复执行，通常等到当前作业或进程出让CPU。最简单的算法。
- FCFS的特点
  - 比较有利于长作业，而不利于短作业。
  - 有利于CPU繁忙的作业，不利于I/O繁忙的作业。

短作业优先(SJF, Shortest Job First)

- 又称为“短进程优先”SPN(Shortest Process Next)；这是对FCFS算法的改进，其目标是减少平均周转时间。
  - 对预计执行时间短的作业（进程）优先分派处理机。通常后来的短作业不抢先正在执行的作业。

SJF的特点

- 优点：
  - 比FCFS改善平均周转时间和平均带权周转时间，缩短作业的等待时间；
  - 提高系统的吞吐量；
- 缺点：
  - 对长作业非常不利，可能长时间得不到执行；
  - 未能依据作业的紧迫程度来划分执行的优先级；
  - 难以准确估计作业（进程）的执行时间，从而影响调度性能。

时间片轮转(Round Robin)算法

- 前两种算法主要用于宏观调度，本算法主要用于微观调度，设计目标是提高资源利用率。其基本思路是通过时间片轮转，提高进程并发性和响应时间特性，从而提高资源利用率；

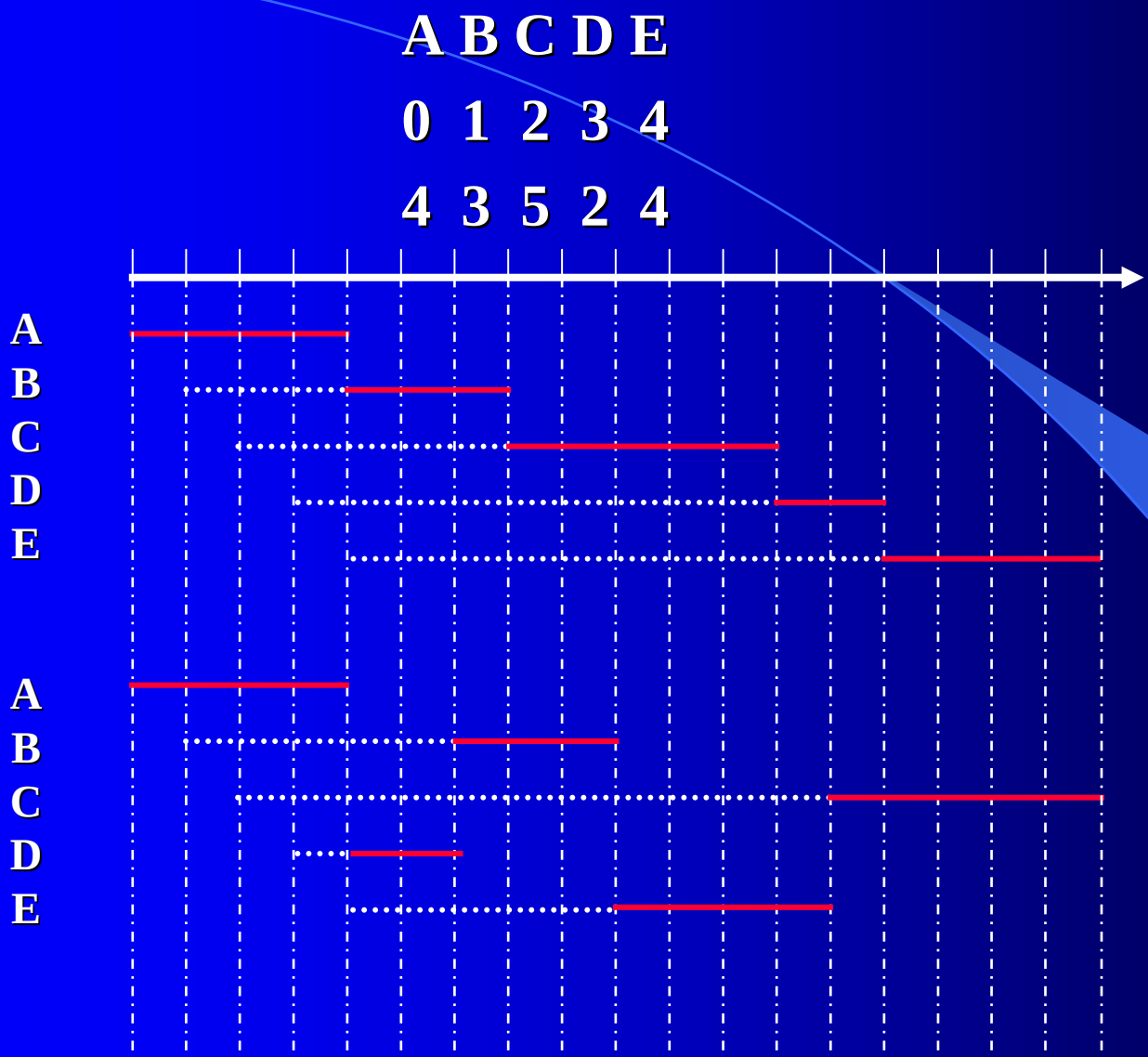
时间片轮转算法

- 将系统中所有的就绪进程按照FCFS原则，排成一个队列。
- 每次调度时将CPU分派给队首进程，让其执行一个时间片。时间片的长度从几个ms到几百ms。
- 在一个时间片结束时，发生时钟中断。
- 调度程序据此暂停当前进程的执行，将其送到就绪队列的末尾，并通过上下文切换执行当前的队首进程。
- 进程可以未使用完一个时间片，就出让CPU（如阻塞）。

时间片长度的确定

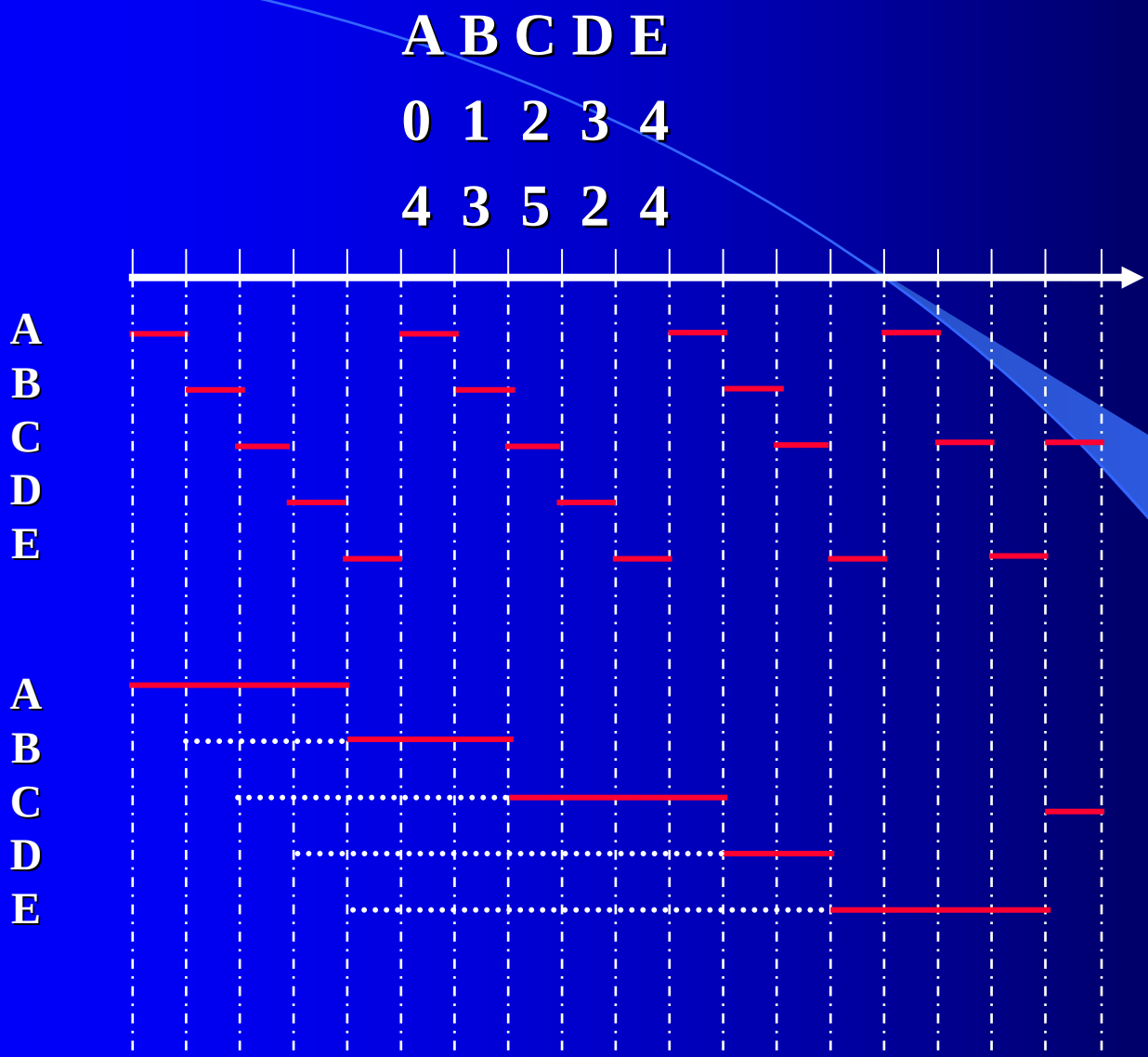
- 时间片长度变化的影响
  - 过长—>退化为FCFS算法，进程在一个时间片内都执行完，响应时间长。
  - 过短—>用户的一次请求需要多个时间片才能处理完，上下文切换次数增加，响应时间长。
- 对响应时间的要求：T(响应时间)=N(进程数目)*q(时间片)
- 就绪进程的数目：数目越多，时间片越小
- 系统的处理能力：应当使用户输入通常在一个时间片内能处理完，否则使响应时间，平均周转时间和平均带权周转时间延长。

- A 0 4
- B 1 3
- C 2 5
- D 3 2
- E 4 4



平均周转时间 ＝ 9
带权平均周转时间＝2.8

平均周转时间 ＝ 8
带权平均周转时间＝2.1


A B C D E
0 1 2 3 4
4 3 5 2 4
A
B
C
D
E
A
B
C
D
E

优先级算法(Priority Scheduling)

- 本算法是平衡各进程对响应时间的要求。适用于作业调度和进程调度，可分成抢先式和非抢先式；

静态优先级

- 创建进程时就确定，直到进程终止前都不改变。通常是一个整数。依据：
  - 进程类型（系统进程优先级较高）
  - 对资源的需求（对CPU和内存需求较少的进程，优先级较高）
  - 用户要求（紧迫程度和付费多少）

动态优先级

- 在创建进程时赋予的优先级，在进程运行过程中可以自动改变，以便获得更好的调度性能。如：
  - 在就绪队列中，等待时间延长则优先级提高，从而使优先级较低的进程在等待足够的时间后，其优先级提高到可被调度执行；
  - 进程每执行一个时间片，就降低其优先级，从而一个进程持续执行时，其优先级降低到出让CPU。

高响应比优先调度算法

- 响应比=(执行时间＋等待时间）/执行时间

- 等待时间相同，短作业优先
- 要求服务时间相同，优先权决定于等待时间，（FCFS)
- 长作业等待时间长，优先权提高

多级队列算法(Multiple-level Queue)

- 本算法引入多个就绪队列，通过各队列的区别对待，达到一个综合的调度目标；
  - 根据作业或进程的性质或类型的不同，将就绪队列再分为若干个子队列。
  - 每个作业固定归入一个队列。
  - 不同队列可有不同的优先级、时间片长度、调度策略等；在运行过程中还可改变进程所在队列。如：系统进程、用户交互进程、批处理进程等。

多级反馈队列算法(Round Robin with Multiple Feedback)

- 多级反馈队列算法时间片轮转算法和优先级算法的综合和发展。优点：
  - 为提高系统吞吐量和缩短平均周转时间而照顾短进程
  - 为获得较好的I/O设备利用率和缩短响应时间而照顾I/O型进程
  - 不必估计进程的执行时间，动态调节

多级反馈队列算法

- 设置多个就绪队列，分别赋予不同的优先级，如逐级降低，队列1的优先级最高。每个队列执行时间片的长度也不同，规定优先级越低则时间片越长，如逐级加倍
- 新进程进入内存后，先投入队列1的末尾，按FCFS算法调度；若按队列1一个时间片未能执行完，则降低投入到队列2的末尾，同样按FCFS算法调度；如此下去，降低到最后的队列，则按“时间片轮转”算法调度直到完成。
- 仅当较高优先级的队列为空，才调度较低优先级的队列中的进程执行。如果进程执行时有新进程进入较高优先级的队列，则抢先执行新进程，并把被抢先的进程投入原队列的末尾。

几点说明

- I/O型进程：让其进入最高优先级队列，以及时响应I/O交互。通常执行一个小时间片，要求可处理完一次I/O请求的数据，然后转入到阻塞队列。
- 计算型进程：每次都执行完时间片，进入更低级队列。最终采用最大时间片来执行，减少调度次数。
- I/O次数不多，而主要是CPU处理的进程。在I/O完成后，放回优先I/O请求时离开的队列，以免每次都回到最高优先级队列后再逐次下降。
- 为适应一个进程在不同时间段的运行特点，I/O完成时，提高优先级；时间片用完时，降低优先级；

调度算法性能分析

- 调度算法的性能通常是通过实验或计算得到的。
  - FCFS, Round Robin,响应比
  - 周转时间
    - 长作业时：T(FCFS) < T(响应比) < T(RR)（运行时间是主要因素）
    - 短作业时：T(RR) < T(响应比) < T(FCFS)（等待时间是主要因素）

实时调度

- 要求更详细的调度信息：如，就绪时间、开始或完成截止时间、处理时间、资源要求、绝对或相对优先级（硬实时或软实时）。
- 采用抢先式调度。
- 快速中断响应，在中断处理时（硬件）关中断的时间尽量短。
- 快速任务分派。相应地采用较小的调度单位（如线程）。

实时调度算法

- 时间片轮转
- Rate
- EDF
- 非抢占优先权
- 基于时钟中断抢占的优先权
- 立即抢占

多处理机调度

- 与单处理机调度的区别：
  - 注重整体运行效率（而不是个别处理机的利用率）
  - 更多样的调度算法
  - 多处理机访问OS数据结构时的互斥（对于享内存系统）
- 调度单位广泛采用线程

对称式多处理系统(SMP)

- 按控制方式，SMP调度算法可分为集中控制和分散控制。下面所述静态和动态调度都是集中控制，而自调度是分散控制。

非对称式多处理系统(ASMP)

- 主－从处理机系统，由主处理机管理一个公共就绪队列，并分派进程给从处理机执行。
- 各个处理机有固定分工，如执行OS的系统功能，I/O处理，应用程序。

- 静态分配(static assignment)：每个CPU设立一个就绪队列，进程从开始执行到完成，都在同一个CPU上。
  - 优点：调度算法开销小。
  - 缺点：容易出现忙闲不均。
- 动态分配(dynamic assignment)：各个CPU采用一个公共就绪队列，队首进程每次分派到当前空闲的CPU上执行。

- 自调度(self-scheduling)：各个CPU采用一个公共就绪队列，每个处理机都可以从队列中选择适当进程来执行。需要对就绪队列的数据结构进行互斥访问控制。是最常用的算法，实现时易于移植采用单处理机的调度技术。
  - 变型：Mach OS中局部和全局就绪队列相结合，其中局部就绪队列中的线程优先调度。

自调度的问题

- 瓶颈问题
- 低效问题
- 线程切换问题

成组调度(gang scheduling)

- 将一个进程中的一组线程，每次分派时同时到一组处理机上执行，在剥夺处理机时也同时对这一组线程进行。
- 优点
  - 通常这样的一组线程在应用逻辑上相互合作，成组调度提高了这些线程的执行并行度，有利于减少阻塞和加快推进速度，最终提高系统吞吐量。
  - 每次调度可以完成多个线程的分派，在系统内线程总数相同时能够减少调度次数，从而减少调度算法的开销

专用处理机调度(dedicated processor assignment)

- 为进程中的每个线程都固定分配一个CPU，直到该线程执行完成。
- 缺点：线程阻塞时，造成CPU的闲置。优点：线程执行时不需切换，相应的开销可以大大减小，推进速度更快。
- 适用场合：CPU数量众多的高度并行系统，单个CPU利用率已不太重要。

传统UNIX的进程调度

- 未设置作业调度，进程调度采用基于时间片的多级反馈队列算法，进程优先级分为核心优先级和用户优先级。

调度时机

- 调度由0号进程完成（始终在核心态执行）。
  时机：
  - 进程由核心态转入用户态时：在每次执行核心代码之后返回用户态之前，检查各就绪进程的优先级并进行调度。如中断——进程回到就绪队列
  - 进程主动放弃处理机时：进程申请系统资源而未得到满足（如read），或进行进程间同步而暂停（如wait或pause），或进程退出（如exit）——进程进入阻塞队列或exit状态。

调度标志

- UNIX System V中有三个与调度有关的标志：
  - runrun：表示要求进行调度，当发现有就绪进程优先级高于当前进程时，设置该标识。在wakeup, setrun, setpri（设置优先级）过程和时钟中断处理例程进行设置。
  - runin：表示内存中没有适当的进程可以换出或内存无足够空间换入一个外存就绪进程。
  - runout：表示外存交换区中没有适当的进程可以换入。

用户优先级

- 进程在用户态和核心态的优先级是不同的，这里说的是用户态进程的优先级。它是基于执行时间的动态优先级，进程优先级可为0~127之间的任一整数。优先数越大，优先级越低。0~49之间的优先级为系统内核保留，用户态下的进程优先级为50~127之间。

- 在UNIX System V中：进程优先数：P_pri = P_CPU / 2 + PUSER + P_nice + NZERO
  - 系统设置部分：PUSER和NZERO是基本用户优先数的阈值，分别为25和20
  - CPU使用时间部分：P_CPU表示该进程最近一次CPU使用时间。每次时钟中断则该值加1（最多可达80）。如果时钟中断的周期为16.6ms，则每秒钟过后将该值为60。
    - 新创建进程的P_CPU值为0，因而具有较高的优先级。
  - 用户设置部分：P_nice是用户可以通过系统调用设置的一个优先级偏移值。默认为20。超级用户可以设置其在0到39之间，而普通用户只能增大该值（即降低优先级）。

核心优先级

- 内核把进程阻塞事件与一个睡眠优先级（0~49）联系起来；当进程从阻塞中醒来时，可及时进行处理。核心优先级分为可中断和不可中断两类优先级。当一个软中断信号到达时，若进程正在可中断优先级上阻塞，则进程立即被唤醒；若正在不可中断优先级上，则继续阻塞。其中：
  - 不可中断优先级：对换，等待磁盘I/O，等待缓冲区，等待文件索引结点——关键操作，应该很快完成
  - 可中断优先级：等待tty（虚终端）I/O，等待子进程退出

调度的实现

- 分三个阶段
  - 检查是否作上下文切换（runrun标志）和核心是否允许作上下文切换（对核心的各种数据结构的操作都已经完成，核心处于正确的状态）。如果允许作上下文切换，则保存当前进程的上下文。
  - 恢复0号进程的上下文，然后执行0号进程，寻找最高优先级的就绪进程，如果没有这样的进程存在，则执行idle过程。如果有这样的进程存在，则该进程作为当前进程分派处理机，保存0号进程的上下文。
  - 恢复当前进程的上下文，执行该进程。

Linux的调度

- Linux中实现了三种进程调度策略：
  - SCHED_OTHER。一般进程。
  - SCHED_FIFO。先进先出（First In First Out）的实时进程。
  - SCHED_RR。轮转（Round Robin）方式执行的实时进程。

- Linux并不为这三种调度策略的进程分别设置一个运行队列，而是通过权重的不同计算以及其他的一些队列操作，在一个运行队列中实现这三种不同的调度。发生进程调度时，调度程序要在运行队列中选择一个最值得运行的进程来执行，这个进程便是通过在运行队列中一一比较各个可运行进程的权重来选择的。权重越大的进程越优，而对于相同权重的进程，在运行队列中的位置越靠前越优。

- 调度策略为SCHED_RR的实时进程，在分配的时间片到期后，插入到运行队列的队尾。
- 调度策略为SCHED_FIFO的进程，在时间片到期后，调度程序并不改变该进程在运行队列中的位置。

- sys_sched_setscheduler ， sys_sched_setparam ， sys_sched_getscheduler ， sys_sched_getparam ， sys_sched_get_priority_max ， sys_sched_get_priority_min

优先级priority

- 进程的优先级反映了进程相对于其他进程的可选择度，其实就是系统每次允许进程运行的时间。子进程继承了父进程的优先级。**priority**也可以通过系统调用**sys_setpriority**（**sys_nice**已被**sys_setpriority**取代）设置。
  系统为每个进程预定的**priority**为**DEF_PRIORITY**（**include/linux/sched.h**），**200ms**。

相对优先级rt_priority

- 对于实时进程，除了用**priority**来反映其优先级（可执行时间）外，还有相对优先级用于同类进程之间的比较选择。实时进程的**rt_priority**取值**1 ~ 99**，一般进程的**rt_priority**值只能取**0**。进程的**rt_priority** 可通过**setscheduler**函数而改变。

计数器counter

- counter用以反映进程所剩余的可运行时间，在进程运行期间，每次发生时钟中断时，其值减1，直至0。由于时钟中断为快中断，在其底半处理过程中，才刷新当前进程的counter值。因此，也可能在发生了好几次时钟中断后才集中进行处理。

- 计数器counter是衡量一般进程权重的重要指标，主要因为如下几种事件而改变：
  - task 0的counter初值为DEF_PRIORITY，在执行sys_idle()时，将counter值置为 -100。
  - 在创建子进程时，父进程的counter变为原值的一半，并将该值赋予子进程。
  - 在进程运行期间，每次发生时钟中断时，counter值减1，直至为0。
  - 若所有的可运行进程的counter值都为0，则需要为所有的进程都重新赋counter值。

- 权重通过调用函数**goodness**来计算，对于实时进程，其权重为**1000+rt_priority**；否则，权重为**counter**。对于当前进程，可以得到比其他进程稍高的权重，为**counter+1**。这样处理是为了在某个进程与当前进程权重相同时可选择当前进程继续执行，以减少进程切换的开销。

小结

- 调度的类型（如调度单位的不同级别，时间周期，不同的OS），性能准则
- 调度算法：**FCFS, SJF, RR,** 多级队列，优先级，多级反馈队列
- 调度算法的性能分析：周转时间和作业长短的关系
- 实时调度：概述，调度算法
- 多处理机调度：自调度，成组调度，专用处理机调度

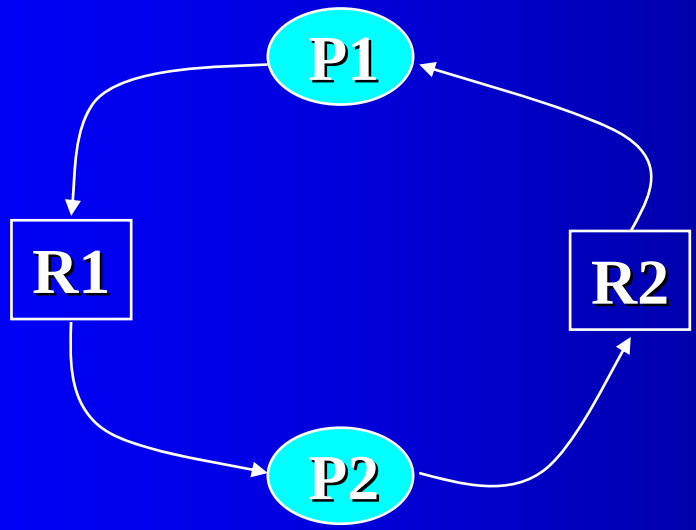
死锁问题(Deadlock)

- 死锁发生原因
  - 竞争资源
  - 并发执行的顺序不当

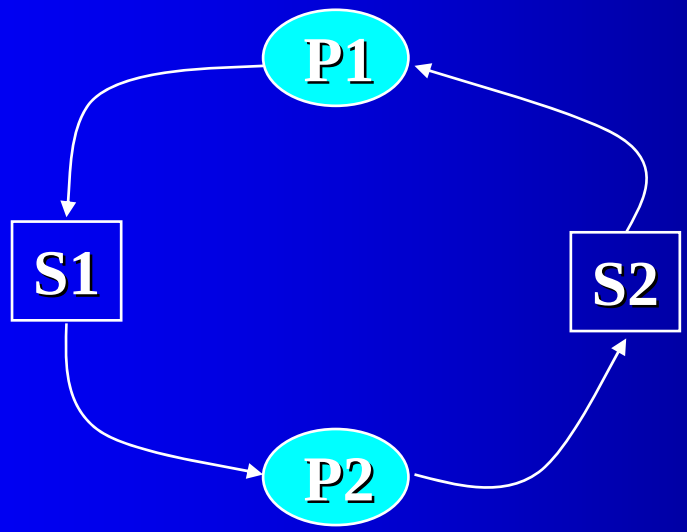
竞争资源引起死锁

- 可剥夺资源：CPU，内存；
- 非可剥夺资源：磁带机、打印机；
- 临时性资源：消息、中断；

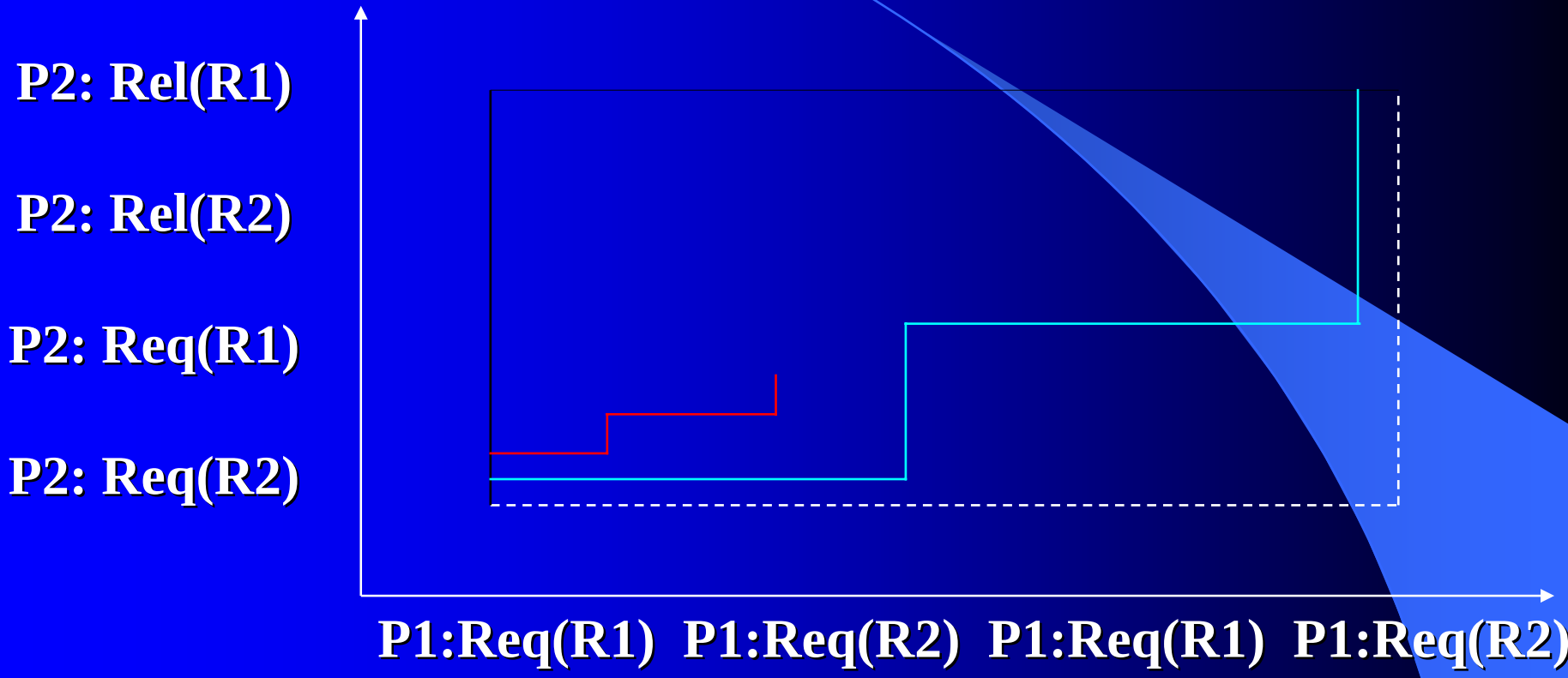
竞争非可剥夺资源



竞争临时性资源



进程推进顺序不当引起死锁



死锁发生条件

- 互斥：任一时刻只允许一个进程使用资源
- 请求和保持：进程在请求其余资源时，不主动释放已经占用的资源
- 不剥夺：进程已经占用的资源，不会被强制剥夺
- 环路等待：环路中的每一条边是进程在请求另一进程已经占有的资源。

处理死锁方法

- 预防死锁
- 避免死锁
- 检测死锁
- 解除死锁

预防死锁

- 互斥：设备特性
- 请求和保持：一次性申请所有资源
  - 资源浪费
  - 运行延迟
- 不剥夺：增加系统开销
- 环路等待：影响资源利用。

避免死锁

- 安全状态：系统存在一个序列<p1,p2,..pn>可顺利完成
- 不安全状态：

进程	最大需求	已分配	可用
P1	10	5	3
P2	4	2	
P3	9	2	

<P2，P1，P3>　　P3请求1

进程	最大需求	已分配	可用
P1	10	5	2
P2	4	2	
P3	9	3	

银行家算法

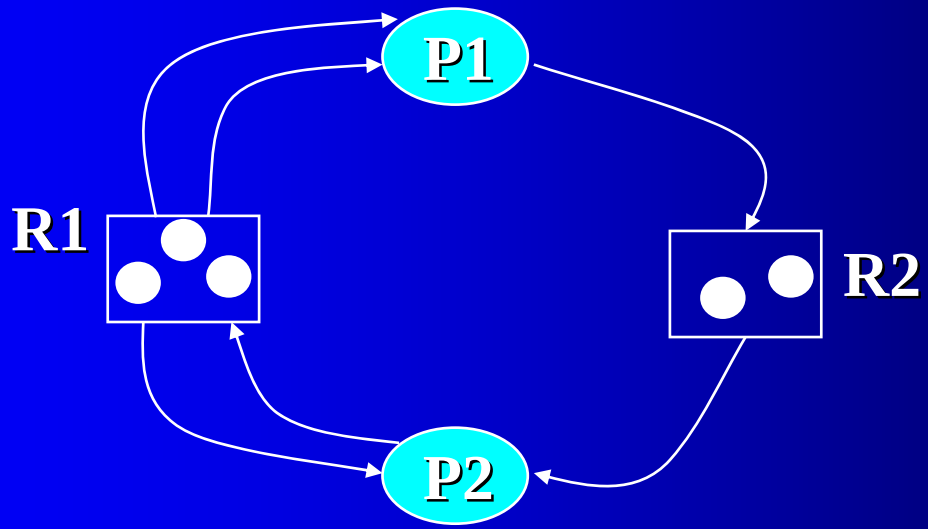
- 可利用资源向量Available
- 最大需求矩阵Max
- 分配矩阵Allocation
- 需求矩阵Need

检测死锁

- 保存资源的请求和分配信息，利用某种算法对这些信息加以检查，以判断是否存在死锁。死锁检测算法主要是检查是否有循环等待。

资源分配图(resource allocation graph)算法

- 有向图G的顶点为资源或进程，从资源R到进程P的边表示R已分配给P，从进程P到资源R的边表示P正因请求R而处于等待状态。有向图的循环表示死锁的存在。
  资源分配图的简化：
  - 删除不处于等待状态的进程（即没有从该进程出发的边）。
  - 依次删除当前的叶顶点。
- 不可简化（简化后还存在边）的资源分配图存在死锁，其中的有边进程为死锁进程。


P1
R1
R2
P2

死锁定理

- 不可简化的资源分配图=====>存在死锁

计算机操作系统

主讲教师：王　雷

计算机网络概述

- 网络组成
  - 通信子网
  - 资源子网

网络拓扑结构

- 星形网络
- 树形网络
- 总线型网络
- 环形网络
- 网状网络

交换网

- 交换方式的引入
- 线路交换
- 报文交换
- 分组交换

局域网

- 公用总线LAN
- 环形LAN

开放系统互连参考模型

北京航空航天大学计算机科学与工程系　王雷

OSI七层模型

- 客户/服务器模式
  - 形成
  - 结构
- 类型
  - 文件服务器
  - 数据库服务器

对等模式

网络操作系统的构成

- 工作站软件
- 网络环境软件
- 网络服务软件
- 网络管理软件

文件与打印服务

北京航空航天大学计算机科学与工程系　王雷