



操作系统

Operating System

北京理工大学计算机学院

马 锐

Email: mary@bit.edu.cn



版权声明

- 本内容版权归北京理工大学计算机学院
操作系统课程组马锐所有
- 使用者可以将全部或部分本内容免费用
于非商业用途
- 使用者在使用全部或部分本内容时请注
明来源
 - 内容来自：北京理工大学计算机学院 +
马锐 + 材料名字
- 对于不遵守此声明或其他违法使用本内
容者，将依法保留追究权

2

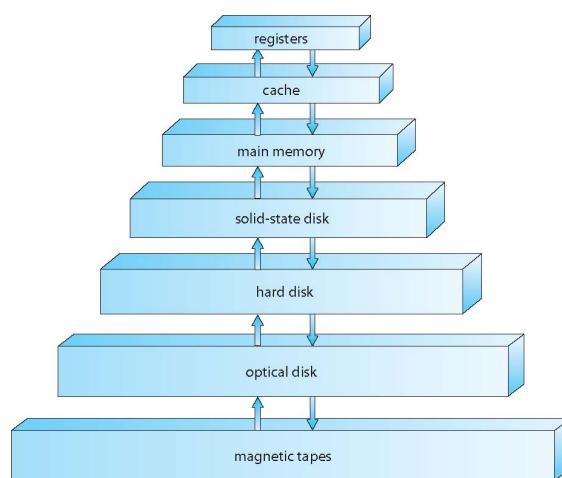
第4章 存储器管理

- 4.1 概述
- 4.2 程序的装入与链接
- 4.3 分区式存储管理
- 4.4 覆盖与交换技术
- 4.5 页式存储管理
- 4.6 段式存储管理
- 4.7 段页式存储管理
- 4.8 虚拟存储器
- 4.9 Linux存储器管理
- 4.10 Windows存储器管理

3

4.1 概述(1)

➤ 存储器体系示意图



4

4.1 概述(2)

➤ 存储器体系

- 快速的、昂贵的cache
- 中速的、价格中等的内存
 - ◆ 实存 (Real memory)
 - ◆ 虚存 (Virtual memory)
- 慢速、大容量、低价格的磁盘存储器

➤ 存储器管理的目的

- 方便用户使用
- 提高存储器利用率

5

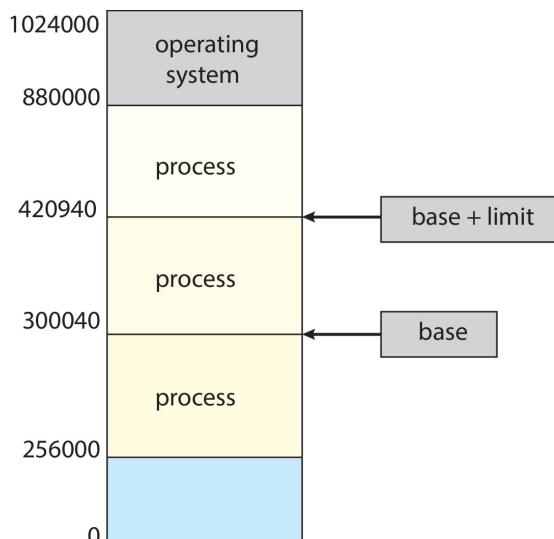
4.1 概述(3)

➤ 存储器管理的功能

- 存储器分配
 - ◆ 解决多道程序共享内存问题
- 地址转换
 - ◆ 研究地址变换方法及地址变换机构
- 存储器保护
 - ◆ 保证各道程序在内存中互不干涉的运行
- 存储器共享
 - ◆ 并发执行的进程共享程序和数据
- 存储器扩充
 - ◆ 实现虚拟存储器

6

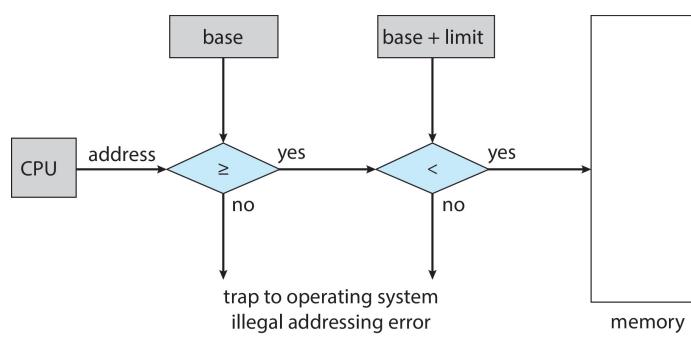
4.1 概述(4)



7

4.1 概述(5)

- 存储器保护
 - 防止地址越界
 - 存取方式检查



Hardware address protection with base and limit registers

8

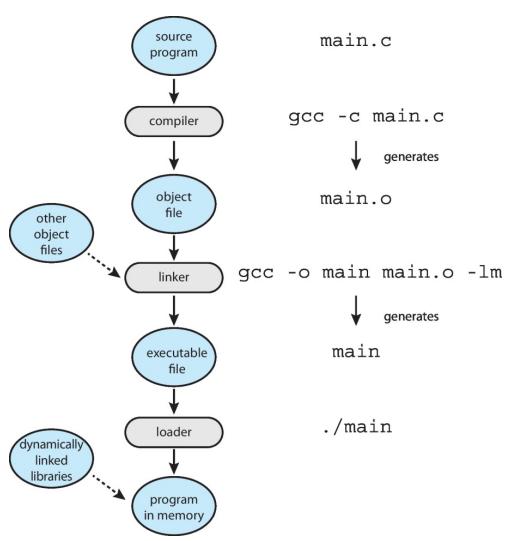
4.1 概述(6)

➤ 存储器共享

- 允许多个进程共享同一个主存区
- 既可以是数据区，也可以是程序区
- 被共享的程序叫可重入程序，其代码无论执行多少遍，都保持不变。具有这种性质的程序又叫纯代码

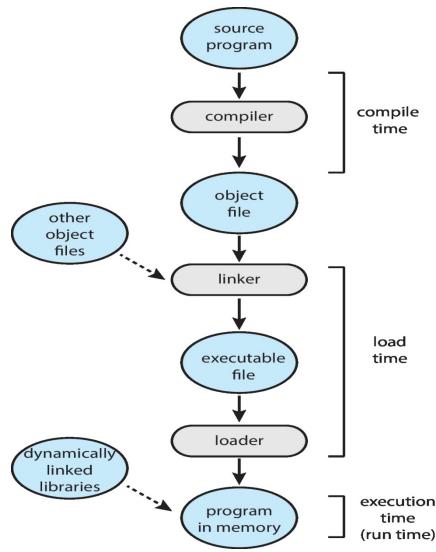
9

4.2 程序的装入与链接(1)



10

4.2 程序的装入与链接(2)



4.2 程序的装入与链接(3)

➤ 程序名空间

- 程序中各种符号名的集合所限定的空间

➤ 地址空间

- **地址空间**: 经编译或链接后目标代码所限定的地址域

- **相对地址** (逻辑地址, 虚地址) : 地址空间中的各个地址

➤ 存储空间

- **存储空间**: 物理存储器中全部物理单元的集合所限定的空间

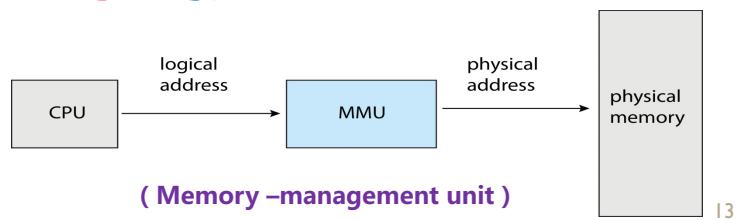
- **绝对地址** (物理地址, 实地址) : 存储空间中的各个地址

12

4.2 程序的装入与链接(4)

➤ 地址重定位 (地址映射, 地址变换)

- 将地址空间的逻辑地址转换为存储空间的物理地址
- 定位方式: 地址变换时间与技术不同
 - ◆ 静态重定位
 - ◆ 动态重定位



13

4.2 程序的装入与链接(5)

➤ 程序的链接

- 链接方式 (根据链接时间划分)
 - ◆ 静态链接
 - ◆ 装入时动态链接
 - ◆ 运行时动态链接
- 静态链接
 - ◆ 在程序运行之前, 将所有编译生成的目标模块和所需库函数链接成一个完整的装配模块, 以后不再拆开

14

4.2 程序的装入与链接(6)

- ◆ 需解决的问题
 - 对相对地址进行修改
 - 变换外部调用符号
- 装入时动态链接
 - ◆ 编译后的目标模块，在装入内存时边装入边链接
 - ◆ 优点
 - 各目标模块分开存放，便于修改和更新
 - 便于实现对目标模块的共享

15

4.2 程序的装入与链接(7)

- 运行时动态链接
 - ◆ 将对某些模块的链接推迟到执行时才进行
 - ◆ 执行时未调用的目标模块不会被装入
- 程序的装入
 - 装入模块的装入方式
 - ◆ 绝对装入方式
 - ◆ 可重定位装入方式
 - ◆ 动态运行时装入方式

16

4.2 程序的装入与链接(8)

- 绝对装入方式

- ◆ 编译时，若知道程序驻留在内存中的位置，编译程序将产生绝对地址的目标代码
- ◆ 绝对装入程序按照装入模块中的地址，将程序和数据装入内存
- ◆ **逻辑地址=实际内存地址**
- ◆ 绝对地址的获取
 - 程序员直接赋予
 - 编译或汇编时给出

17

4.2 程序的装入与链接(9)

- ◆ 问题

- ◆ 多道环境下编译程序不可能预知所编译模块在内存中的位置

- ◆ 只适合单道程序设计

- 可重定位装入方式

- ◆ 多道程序环境下，目标模块的起始地址通常由0开始，程序中的其他地址均相对于该起始地址进行计算

- ◆ 采用可重定位装入方式

18

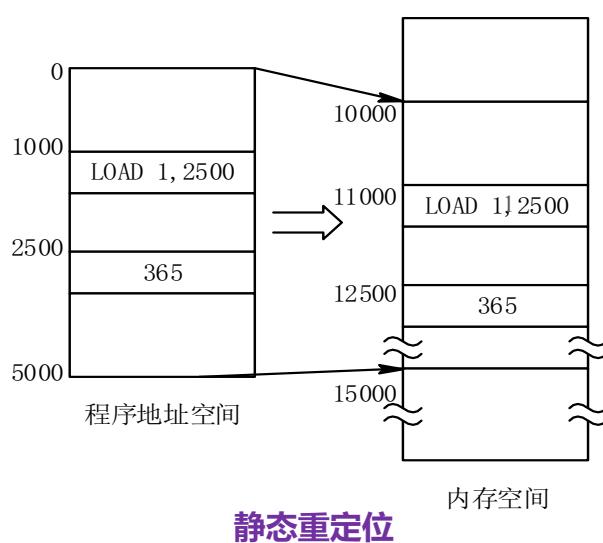
4.2 程序的装入与链接(10)

◆ 静态重定位

- 在程序装入时，装入程序把用户程序地址空间中的指令和数据的相对地址全部转换成存储空间的绝对地址
- 转换工作在程序执行前一次完成，不能更改和移动
- ◆ 优点
 - 无需硬件支持，易于实现
 - 为每个进程分配一个连续的存储区

19

4.2 程序的装入与链接(11)



静态重定位

20

4.2 程序的装入与链接(12)

- ◆ 缺点

- 进程占有连续的存储区
- 程序执行期间不允许在主存移动
- 不能共享存储器中的程序

- 动态运行时装入方式

- 在将装入模块装入内存后，并不立即把装入模块中的相对地址转换为绝对地址，而是将地址转换推迟到程序真正要执行时才进行
- 装入内存后所有地址都仍是相对地址

21

4.2 程序的装入与链接(13)

- 动态重定位

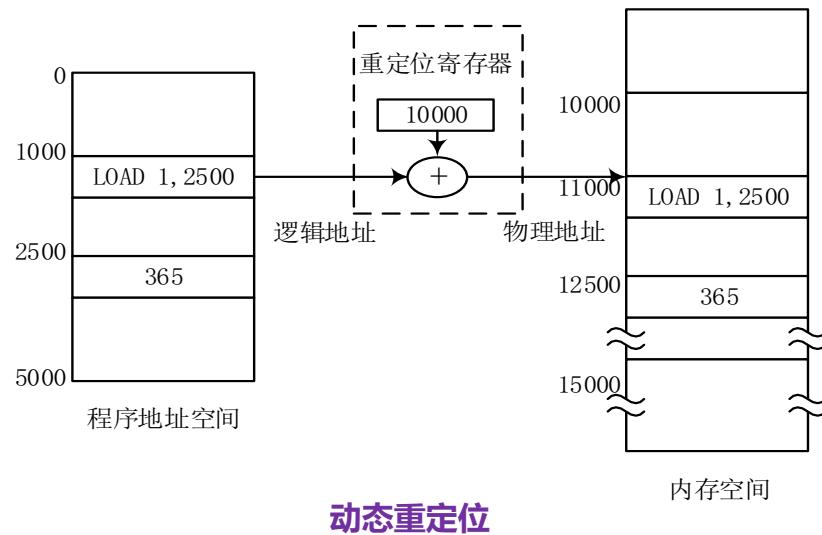
- 由硬件地址转换机构——重定位寄存器实现
- 转换工作在程序执行过程中进行

- ◆ 优点

- 执行期间可移动用户程序，移动后只需修改重定位寄存器即可
- 进程不必占用连续的存储空间
- 便于多用户共享存储器中的进程
- 主存利用率高

22

4.2 程序的装入与链接(14)



23

4.3 分区式存储管理

- 4.3.1 固定式分区
- 4.3.2 可变式分区
- 4.3.3 分区管理的地址重定位
- 4.3.4 分区管理的存储器保护
- 4.3.5 分区管理的优缺点

24

4.3.1 固定式分区(1)

- 适合多道程序的最简单的存储管理方式
- 将主存预先划分成几个分区
 - 大小相等
 - 大小不等
- 进程到达时选择一个适合程序要求的空闲区
- 若没有可用的空闲分区，进程在分区队列中等待
 - 每个分区设置独立等待队列
 - ◆ 进程按大小排入各队列等待，易造成某个队列拥挤
 - 全部分区设置一个等待队列

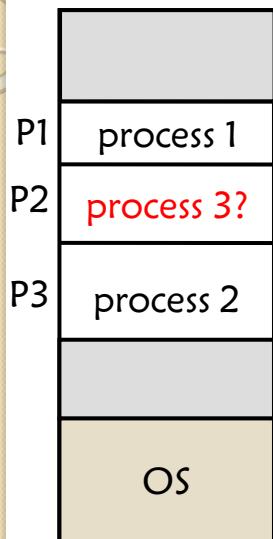
25

4.3.1 固定式分区(2)

- 实现
 - 设置主存使用情况表描述主存使用情况
 - 内存管理过程
- 优点
 - 简单，易于实现
- 缺点
 - 主存利用不充分

26

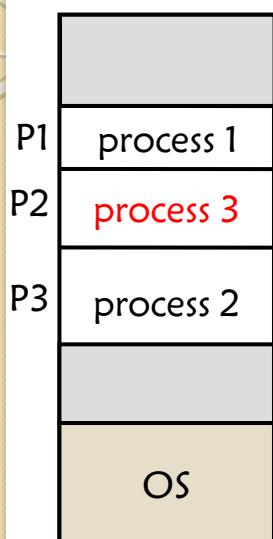
4.3.1 固定式分区(3)



No.	Size(K)	Starting address(K)	state
1	12	20	yes
2	32	32	no
3	64	64	yes
4	16	128	no
5	48	144	no
...

27

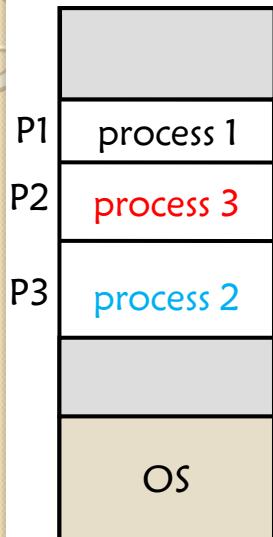
4.3.1 固定式分区(3)



No.	Size(K)	Starting address(K)	state
1	12	20	yes
2	32	32	yes
3	64	64	yes
4	16	128	no
5	48	144	no
...

28

4.3.1 固定式分区(3)

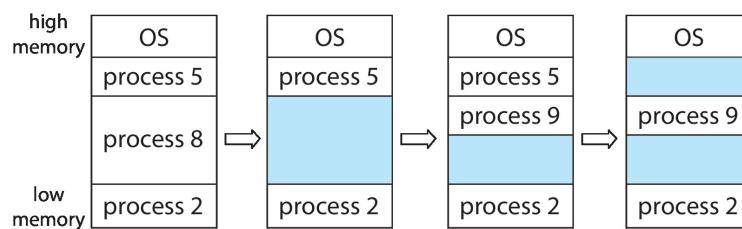


No.	Size(K)	Starting address(K)	state
1	12	20	yes
2	32	32	yes
3	64	64	no
4	16	128	no
5	48	144	no
...

29

4.3.2 可变式分区(1)

- 根据进程的大小动态地划分分区，使分区的大小正好等于进程大小
- 程序要求运行时，由系统从可用的空闲存储空间划分一大小正好等于进程大小的存储区分配给程序
- 提高了存储器的使用效率，分配与释放复杂



30

4.3.2 可变式分区(2)

➤ 问题

- 分区大小不定
- 分区数目不定

➤ 管理方式

- 分区说明表
- 空闲区链

➤ 分区说明表

- 已分配区表
- 未分配区说明表

31

4.3.2 可变式分区(3)

● 分配

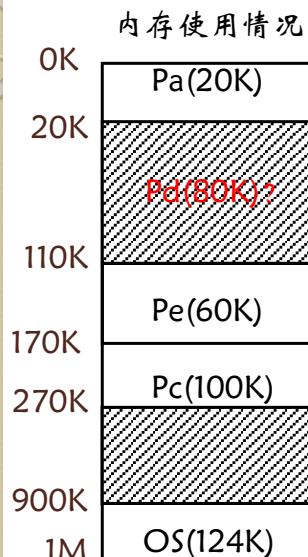
- ◆ 从空闲区表中找一个足以容纳该进程的空闲区，若该分区较大，则一分为二，一部分分配给进程，另一部分仍作为空闲区留在空闲区表中
- ◆ 在已分配区表中找一个空表目，填入新分配进程的信息

● 回收

- ◆ 回收分区登记在未分配区表中
- ◆ 若有相邻接的空闲区，**合并**后再登记
- ◆ 将该进程占用的已分配区表目置空

32

4.3.2 可变式分区(4)

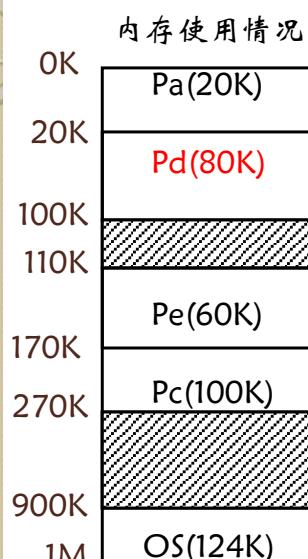


始址	长度	标志
20K	90K	未分配
270K	630K	未分配
		空
		空

始址	长度	标志
0K	20K	Pa
110K	60K	Pe
170K	100K	Pc
		空
		空

空闲区表
已分配区表
33

4.3.2 可变式分区(4)

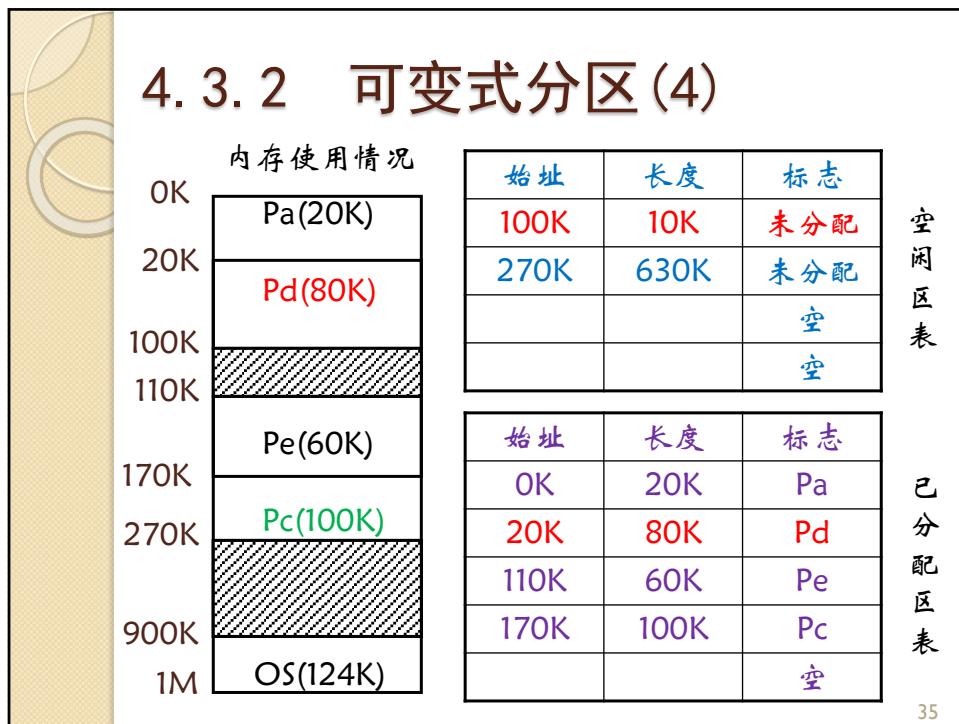


始址	长度	标志
100K	10K	未分配
270K	630K	未分配
		空
		空

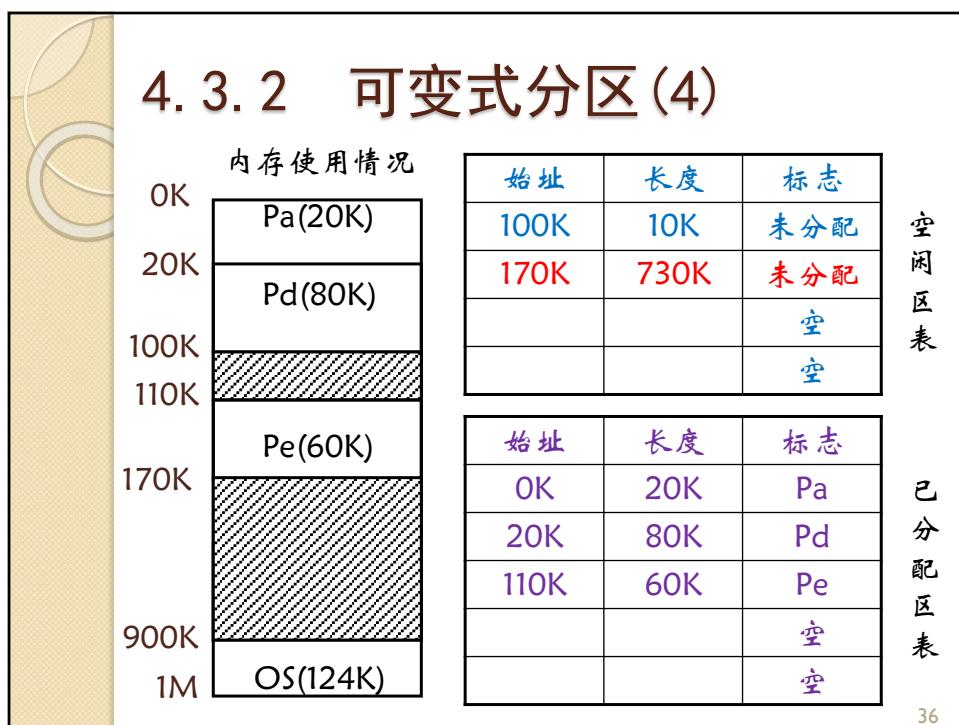
始址	长度	标志
0K	20K	Pa
20K	80K	Pd
110K	60K	Pe
170K	100K	Pc
		空

空闲区表
已分配区表
34

4.3.2 可变式分区(4)



4.3.2 可变式分区(4)



4.3.2 可变式分区(5)

- 优点

- ◆ 直观、简单

- 缺点

- ◆ 分区个数不定，难以确定分区表长度

- 空闲区链

- 将表格信息附加在分区中

- ◆ 状态信息：0表示空闲，1表示已分配

- ◆ 该区的大小(以字或块为单位)

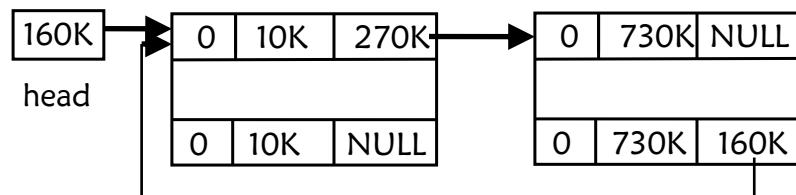
- ◆ 指针：分别指向其上/下一分区

37

4.3.2 可变式分区(6)

状态位	分区大小 (N+2)	前向指针
大小为N的已分配区或空闲区		
状态位	分区大小 (N+2)	后向指针

带有表格信息的分区格式



空闲区双向链表

38

4.3.2 可变式分区(7)

➤ 动态分区分配算法

- 首次适应(first fit)法

- ◆ 空闲区按照起始地址的大小从小到大排列

- ◆ 分配

- 从链首开始扫描空闲区链，直至找到一个足够大的空闲区

- 按照进程大小，从分区中划出满足要求的空间分配给请求者，剩余空间仍保留在空闲链中

39

4.3.2 可变式分区(8)

- 若无满足要求的分区则分配失败(等待)

- ◆ 优点

- 优先利用低址空闲区，保留高址大空闲区

- ◆ 缺点

- 低址部分不断被划分，留下许多难以利用的、很小的空闲分区

- 每次查找均从低址部分开始，增加了查找空闲分区的开销

- ◆ 改进

- 循环首次适应算法

40

4.3.2 可变式分区(9)

- 最佳适应(best fit)法
 - ◆ 分配
 - 扫描整个链表，将最接近进程需求量的空闲区分配给进程
 - ◆ 缺点
 - 每次查找整个链表，效率低
 - 浪费更多的存储空间，将主存划分为较多小的、无用的碎片
 - ◆ 改进
 - 空闲区按容量大小从小到大排列

41

4.3.2 可变式分区(10)

- 最坏适应(worst fit)法
 - ◆ 分配
 - 扫描整个链表，直到找到该链中能满足进程要求且为链中最大的空闲区为止
 - 把这个最大空闲区一分为二，一部分分给进程，另一部分仍留在链中

42

4.3.2 可变式分区(11)

➤ 要求分配20K, 10K和5K

分配前

已	未	已	未	已	未	已	未	已	未	已	未	已	未
10K	10K	20K	30K	10K	5K	30K	20K	10K	15K	20K	20K		

首次

已	未	已	未	已	未	已	未	已	未	已	未	已	未
10K	10K	40K	10K	10K	5K	30K	20K	10K	15K	20K	20K		

最佳

已	未	已	未	已	未			已		未	已	未	
10K	10K	20K	30K	10K	5K			60K		15K	20K	20K	

最坏

已	未	已	未	已	未	已	未	已	未	已	未	已	未
10K	10K	40K	10K	10K	5K	30K	20K	10K	15K	20K	20K		

43

4.3.2 可变式分区(12)

➤ 要求分配20K, 10K和5K

分配前

已	未	已	未	已	未	已	未	已	未	已	未	已	未
10K	10K	20K	30K	10K	5K	30K	20K	10K	15K	20K	20K		

首次

	已		未	已	未	已	未	已	未	已	未	已	未
	60K		10K	10K	5K	30K	20K	10K	15K	20K	20K		

最佳

	已		未	已	未			已		未	已	未	
	40K		30K	10K	5K			60K		15K	20K	20K	

最坏

已	未	已	未	已	未	已	未	已	未	已	未	已	未
10K	10K	40K	10K	10K	5K	40K	10K	10K	15K	20K	20K		

44

4.3.2 可变式分区(13)

➤ 要求分配20K, 10K和5K

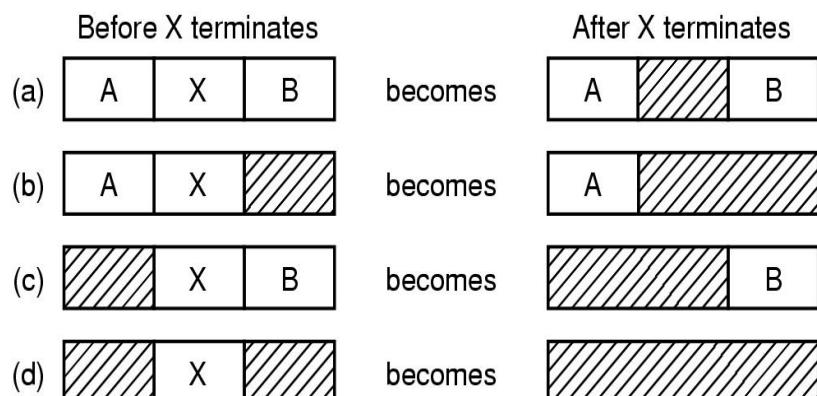
分配前	已	未	已	未	已	未	已	未	已	未	已	未
	10K	10K	20K	30K	10K	5K	30K	20K	10K	15K	20K	20K
首次		已		未	已	未	已	未	已	未	已	未
		65K		5K	10K	5K	30K	20K	10K	15K	20K	20K
最佳		已		未			已			未	已	未
		40K		30K			75K			15K	20K	20K
最坏	已	未	中	未	已	未	已	未	已	未	已	未
	10K	10K	40K	10K	10K	5K	40K	10K	10K	15K	25K	15K

45

4.3.2 可变式分区(14)

➤ 回收方法

● 若释放区与空闲区相邻接，要进行合并



进程X终止时邻接的存储区情况

46

4.3.3 分区管理的地址重定位

➤ 固定式分区

- 通常固定分区采用静态重定位，进程运行时使用主存绝对地址

➤ 可变式分区

- 采用动态重定位，进程运行时CPU给出的是相对地址

47

4.3.4 分区管理的存储器保护

➤ 上下限寄存器

- 用于静态重定位
- 下界寄存器内容 \leq 访问内存的地址 \leq 上界寄存器内容

➤ 基址+限长寄存器

- 用于动态重定位
- 基址（重定位）寄存器：分区首址
- 限长寄存器：分区长度

48

4.3.5 分区管理的优缺点(1)

➤ 优点

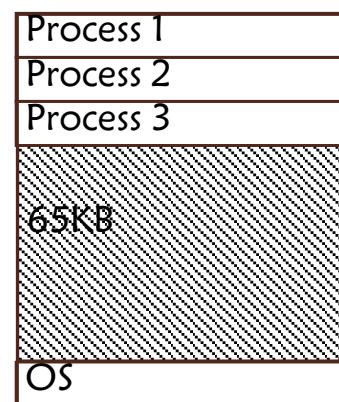
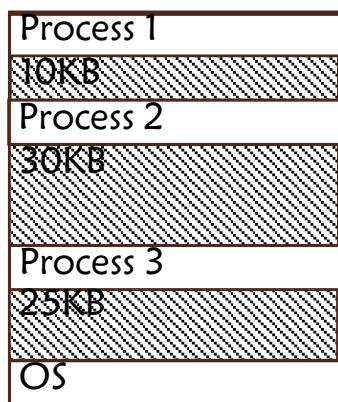
- 实现了多道程序共享主存
- 实现分区管理的系统设计相对简单，不需要更多的系统软硬件开销
- 实现存储保护的手段比较简单

➤ 缺点

- 主存利用不充分，存在零头(或碎片)
 - 内零头：存在于进程存储空间内部
 - 外零头：存在于整个内存中

49

4.3.5 分区管理的优缺点(3)



紧凑

50

4.3.5 分区管理的优缺点(2)

◆解决方法

- 紧凑

- 程序不再占用连续的主存空间

- 程序地址空间大于存储空间时，程序无法运行，即程序的地址空间受到实际存储空间的限制，无法对主存进行扩充

◆解决方法

- 覆盖与交换

- 虚拟存储器

51

4.4 覆盖与交换技术(1)

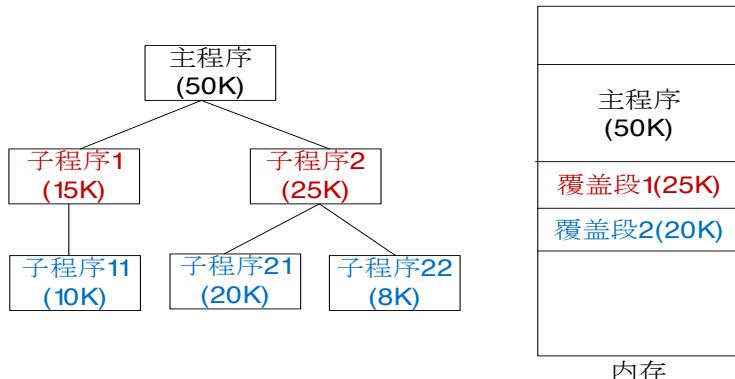
➤ 覆盖(Overlay)

- 同一主存区可被不同的程序段重复使用
- 原理：一个程序由若干功能独立的程序段组成，程序运行时，一些程序段不会同时执行，可以共用同一主存区
- 覆盖区：可以共享的主存区
- 覆盖段：程序执行时不要求同时装入主存的程序段（覆盖）组成一组，叫做覆盖段，并分配同一个主存区（覆盖区）
- 覆盖结构由用户实现，无需OS特殊支持

52

4.4 覆盖与交换技术(2)

- 难点是如何设计覆盖结构
- 通常用于系统程序的主存管理中



53

4.4 覆盖与交换技术(3)

➤ 交换

- 系统根据需要把主存中暂时不运行的某个(或某些)进程部分或全部移到外存，而把外存中的某个(或某些)进程移到相应的主存区，并使其投入运行

➤ 交换的时机

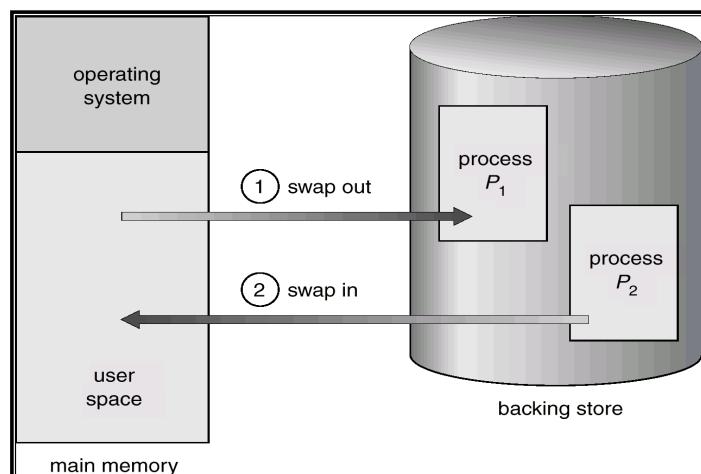
- 进程用完时间片或等待输入输出
- 进程要求扩充存储而得不到满足时

➤ 关键

- 在外存保留副本，每次仅修改变化部分
- (标准)交换发生在进程间，覆盖发生在进程内

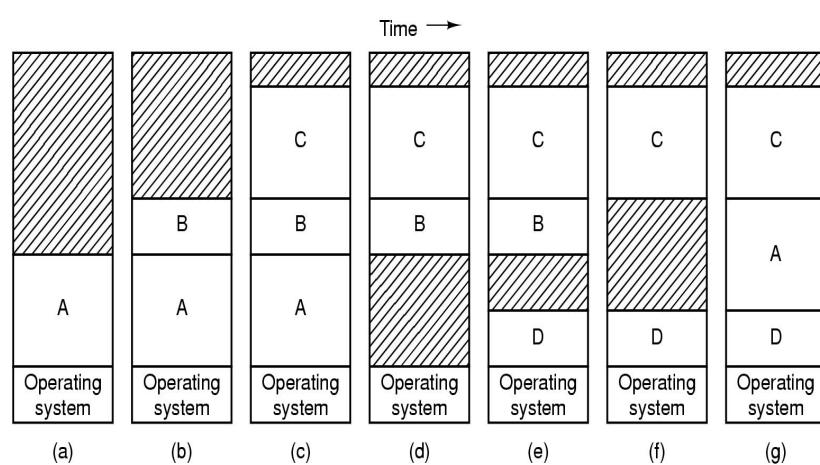
54

4.4 覆盖与交换技术(4)



55

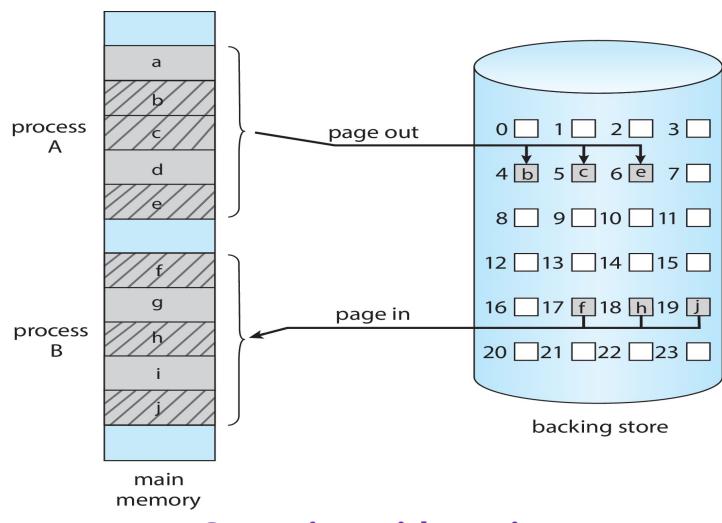
4.4 覆盖与交换技术(5)



四个进程的交换

56

4.4 覆盖与交换技术(6)



Swapping with paging.

57