ch9 内存管理

引言

主要介绍了程序执行前链接和装入的内容

程序执行之前

编译，链接，装入（装入内存）

地址表现形式：

1. 符号 源程序
2. 可重定位的地址 目标模块
3. 相对地址 装入模块
4. 绝对地址 内存映像

程序链接技术

(深入理解计算机系统) bss段，data段、text段、堆(heap)和栈(stack)

程序链接技术（根据链接时间不同）：

1. 静态链接
2. 动态链接
    - 装入时
    - 运行时

链接解决的问题：

1. 对**相对地址**进行修改
2. 变换**外部调用符号**

程序装入技术

1. 地址空间（逻辑地址）和存储空间（物理地址）

   引入地址空间的原因： 安全性，并发度

   名空间：程序中符号名组成的空间

   逻辑地址空间（地址空间）：逻辑地址的集合

   物理存储空间（存储空间）：物理地址的结合

   进程间地址空间是相互独立的

2. 常用程序装入技术

   > 背景技术：地址再定位技术
   >
   > 逻辑地址和物理地址对应起来的过程

静态再定位由装入程序完成（对应第二种装入技术），动态再定位在程序执行时进行（对应第三种装入技术）

- 绝对装入技术（固定地址再定位）

  在**编译链接**时直接指定程序在执行时访问的实际存储器地址。

  程序中的逻辑地址和实际内存地址完全相同。操作系统把程序装入内存时，不需要对程序和数据的地址进行修改。

  优点：装入过程简单

  缺点：过于依赖硬件结构，不适于多道程序系统，**只适用于单道程序环境**

- 可重定位装入技术

  装入时由装入程序进行再定位

  可执行文件中，列出各个需要重定位的地址单元和相对地址值，**装入时**再根据所定位的内存地址去修改每个重定位地址项，添加相应偏移量

  - 静态再定位（静态映射）

    程序**执行之前**进行地址再定位

    优点：易实现，无需硬件支持

    缺点：程序**重定位后不能移动**，所以不能重新分配内存，不利于内存有效利用；程序在存储空间中**只能连续分配**

- 动态运行时装入技术
  - 动态再定位（动态映射）

    装入内存时不修改逻辑地址，**访问物理内存之前**再实时的将逻辑地址转换成物理地址

    优点：程序执行过程中可移动，利于内存充分利用；程序在内存中不必连续存放，只需增加基址或界限寄存器

    缺点：需要附加硬件支持，实现存储管理的软件算法较复杂

地址访问——地址保护：

1. 界限寄存器
   - 定位寄存器+界限寄存器：先利用定位寄存器将优先地址转换为物理地址，在把物理地址和界限寄存器比较
   - 上下限寄存器
2. 保护键

实例（ppt中的三个图）：

1. 静态链接，静态装入

   浪费硬盘空间，浪费内存空间

2. 静态链接，动态装入

   内存效率提高，硬盘没有得到最佳利用；内存效率仍有提升空间

3. 动态链接，动态装入

   桩(stub)

存储管理思想

存储组织

存储器功能：保存数据

存储组织的功能：在存储技术和cpu寻址技术许可范围内，组织合理的存储结构

依据：速度、容量、价格

存储层次结构：寄存器register->快速缓存cache->主存->外存

存储管理目的

- 充分利用内存
- 尽可能**方便**用户使用
- 存储保护与**安全**
- **共享与通信**
- 实现的性能和代价
- 解决程序空间比实际内存空间大的问题

存储管理任务

1. 分配和回收
2. 共享
3. 保护
4. 扩充

存储管理方案

1. 连续分配方式

    - 单一连续存储管理
    - 分区存储管理
2. 离散分配方式
    - 分页存储管理
    - 段式存储管理
    - 段页式存储管理
3. 虚拟存储器

连续分配方式

基本思想

内存分为两个区域：

1. 用于驻留操作系统的低内存（中断向量也位于低内存）
2. 驻留用户进程的高内存

单一连续存储管理

整个内存空间分成系统区和用户区，系统区给操作系统使用，用户区给用户使用。

适用于**单用户单任务**

分区存储管理

- **基本原理**

  把内存分为一些大小相等或不等的分区，每个应用进程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**，支持程序并发

  问题：可能存在内碎片和外碎片；难共享

  数据结构：分区表/分区链表

  分区表分为：空闲分区表，占用分区表

- **固定分区**

  内存划分为**固定大小**（但不是所有分区大小一定相同）的**连续分区**

    - 分区大小相同
    - 分区大小不同：多个小分区，适量中等分区，少量大分区

  有内碎片

- **动态分区**

  在**装入**程序时按其初始要求分配，**或在其执行**过程中通过系统调用进行分配或改变分区大小。

  没有内碎片 有外碎片

- **分区分配/释放问题**
  - 分区分配算法
    - 如果空闲分区大于程序要求，则分割
  - 分区释放算法
    - 与相邻的空闲分区合并成一个
- **评价**

常用分区算法

4个最

最先适配算法

按分区先后次序，**从头**查找，找到符合要求的第一个分区。

循环最先适配算法

按分区先后次序，**从上次分配的分区**起查找（到最后分区时再回到开头），找到符合要求的第一个分区。

最佳适配算法

在所有大于或者等于要求分配长度的空闲区中挑选一个最小的分区，即对该分区所要求分配的大小来说，是最合适的。分配后，所剩余的块会最小。

空闲分区表从小到大排序

释放时由于涉及到与相邻区的合并，基于链表做这种操作很麻烦

最坏适配算法

分区时取所有空闲区中最大的一块，把剩余的块再变成一个新的小一点的空闲区。

空闲分区表从大到小排序

分区算法存在的问题

- 碎片问题
    - 外碎片->紧凑技术：将小的空闲区组合->但是紧凑技术系统开销大，所以引出离散分配方式
- 分区保护

内存扩充技术

其中的交换技术中的部分交换是虚拟存储系统的基础

问题提出

- 情况一：一个作业程序地址空间大于内存可用空间->作业不能装入运行
- 情况二：并发运行作业程序地址空间总和大于内存可用空间->多道程序设计遇到困难

内存扩充

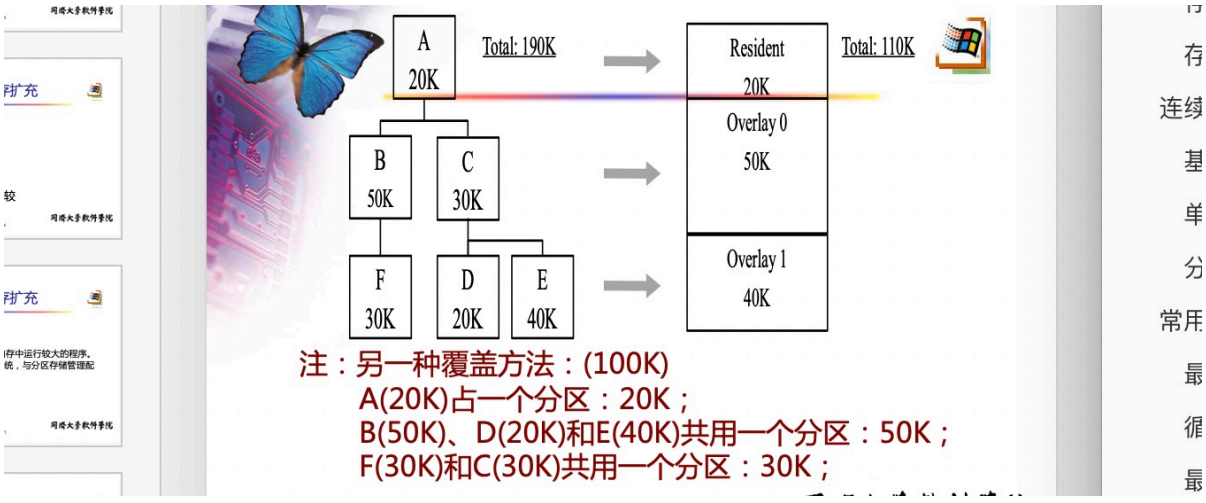
借助大容量辅存在逻辑上实现内存扩充，来解决内存容量不足的问题。

扩充技术

- 覆盖技术

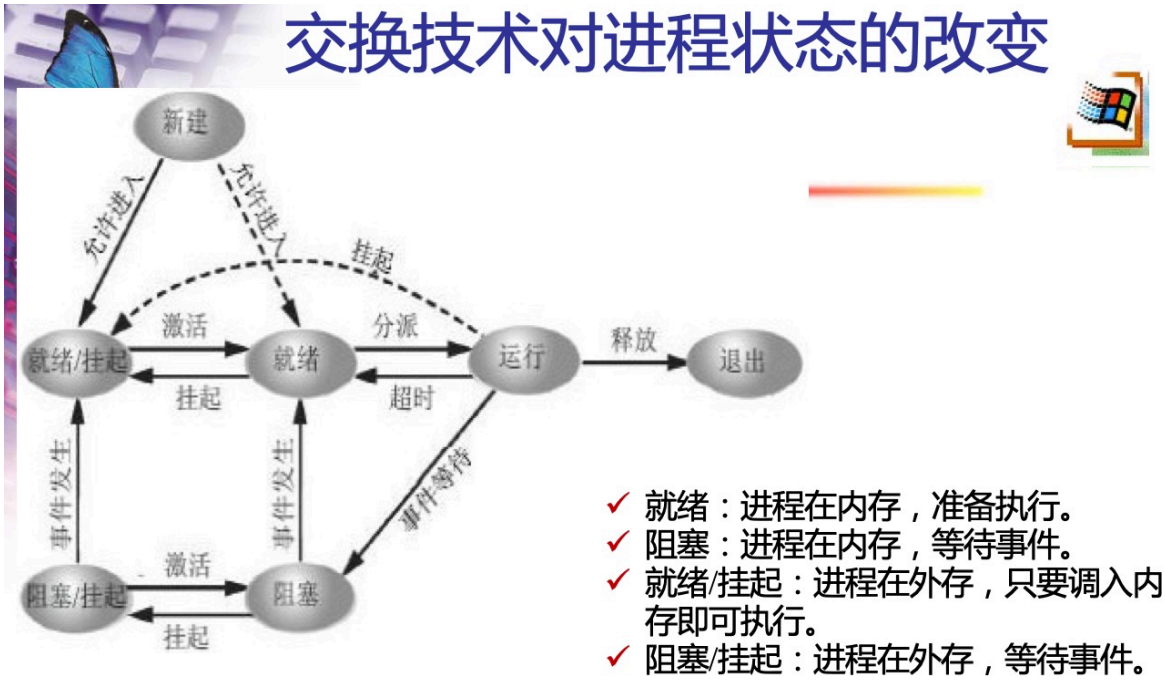
  必要代码和数据常驻内存；可选部分平时在外存中，用到时调入内存；不存在调用关系的模块不必同时装入内存，可互相覆盖

  不存在调用关系的模块之间相互覆盖



- 交换技术

  把内存中暂时不能运行或暂不能用的程序和数据，调出到外存上，以便腾出足够内存空间给其它进程或程序使用。



- 整体交换（进程交换）：交换以整个进程为单位
- 部分交换（页面交换、分段交换）：**分页、分段交换的基础**，为了**支持虚拟存储系统**

两种技术比较

覆盖发生在同一进程或作业内，交换发生在进程或作业之间

覆盖对调用依赖关系有要求，交换没有

离散分配方式

页式存储管理

- 基本原理
  - 用户空间划分
    - **虚页**
  - 内存空间划分
    - **内存块**（物理页面，页框、**实页**）：长度相同
  - 内存分配
    - 逻辑上相邻的页，物理上不一定相邻
- 存储管理
  - 进程页表
  - 物理页面表

        - 位示图（使用固定分区）/链表（使用动态分区）
    - 请求表
    - 管理过程
- 硬件支持
    - 系统设置一对寄存器
        - 页表始址寄存器
        - 页表长度寄存器
    - 联想寄存器——快表（TLB）
- 页表结构
- 共享
- 评价
    - 优点：解决了碎片问题，便于管理
    - 缺点：不易实现程序共享，不便于动态链接

段式存储管理

- 引入目的
    - 为了用户
- 基本原理
    - 用户空间划分
        - 程序段
    - 内存空间划分
        - 物理段：长度不相同
    - 内存分配
- 存储管理
    - 进程段表
    - 系统段表
    - 空闲段表
    - 内存分配算法：首先适配；最佳适配；最坏适配
- 硬件支持
- 评价
- **页式和段式的比较**

段页式管理

- 产生背景
- 基本思想
    - 用户程序按段式，内存空间采用页式存储
    - 内存分配以页为单位
- 存储管理
    - 段表
    - 页表
    - 空块管理同页式管理
    - 分配同页式管理
- 硬件支持
- 举例

虚拟存储器