Ch 06 基本存储管理

地址重定位

- **程序装载到内存才可以运行**：程序以可执行文件的格式保存在磁盘上
- **多道程序设计模型**：允许多个程序同时进入内存
- 每个进程有自己**独立的地址空间**

内存管理需要进行的工作

- 需要支持**地址重定位**
    - 程序中的地址不一定是最后的物理地址
    - 在程序运行前无法计算出物理地址

      不能确定程序被加载到内存的什么地方
- 需要支持**地址保护**
    - 进程间地址空间不能相互访问

逻辑地址（相对地址，虚拟地址）

- 用户程序经过编译、汇编后形成目标代码，目标代码通常采用相对地址的形式，**首地址**为 0，其余指令中的地址都相对于首地址而编址
- 不能用逻辑地址在内存中读取信息

物理地址（绝对地址，实地址）

- 内存中存储单元的地址，可直接选址

地址重定位：将用户程序中的逻辑地址转换为运行时可由机器直接寻址的物理地址的过程，目的是保证 *CPU* 执行指令时可以正确访问内存单元

- 静态地址重定位：当用户程序加载到内存时，**一次性**实现逻辑地址到物理地址的转换；一般由**软件**完成

- 动态地址重定位：在进程执行过程中进行地址变换，即**逐条指令执行时**完成地址映射；需要**硬件**支持

地址保护：确保每个进程可访问的**合法地址**的范围，确保进程只访问合法地址

基本内存管理方案

1. 单一连续区

一段时间内只有一个进程在内存中；简单，**内存利用率低**

2. 固定分区

- 把内存空间分割成若干区域，称为**分区**
- **分区大小可以相同也可以不同**
- 分区大小**固定不变**
- 每个分区装一个且只能装一个进程

3. 可变分区

- 根据进程的需要，把内存空闲空间分割出一个分区，分配给该进程
- 剩余部分成为新的**空闲区**
- 会产生很多小的、不易利用的空闲区（**碎片**）

碎片：很小的、不易利用的空闲区；导致内存利用率下降

- 解决方案：**紧缩技术**
- 在内存移动程序，将所有小的空闲区合并为较大的空闲区
- 紧缩时需要考虑：系统开销、移动时机

4. 页式存储管理

- 设计思想：用户程序地址划分为**大小相等**的区域——**页**
- 内存空间按页大小划分为大小相等的区域，称为**内存块**
- 内存分配规则：以**页**为单位进行分配，并按进程需要的页数来分配；逻辑上相邻的页，物理上不一定相邻
- 典型的页面尺寸：4K 或者 4M

页表

- **页表项**：记录了**逻辑页号**与**页框号**的对应关系
- 每个进程一个页表，存放在**内存**中
- 页表起始地址

地址转换：硬件支持

- CPU 取到逻辑地址，自动划分为页号和页内地址
- 用页号查页表，得到页框号
- 与页内地址（页内偏移）拼接成物理地址

5. 段式存储管理

设计思想：

- 用户程序地址空间：按程序自身的逻辑关系划分为若干个程序段，每个程序段都有一个段名
- 内存空间被动态的划分为若干个长度**不相同**的区域，称为物理段，每个物理段由起始地址和长度确定
- 内存分配规则：以段为单位进行分配，每一个段在内存中占据**连续空间**，但各段之间可以不相邻

逻辑地址：段号 + 段内地址

段表

- 记录了段号、段首地址、段长度之间的关系
- 每个进程一个段表，存放在内存
- 段表起始地址

地址转换：硬件支持

- CPU 取到逻辑地址
- 用段号查段表，得到段的起始地址
- 与段内偏移地址计算出物理地址

6. 段页式存储管理方案

产生背景：段页式存储管理方案结合了段式优点，克服二者的缺点

设计思想：用户进程划分：先按段划分，每一段再按页面划分

内存划分：按页式存储管理方案

内存分配：以页为单位进行分配

段表：记录每一段的页表起始地址和页表长度

页表：记录逻辑页号与内存块号的对应关系（每段对应一个页表）

物理内存管理方案

数据结构

- 位图

  每个分配单元对应位图中的一位，0 表示空闲，1 表示占用
- 空闲区表、已分配区表

  表中每一项记录了空闲区的起始地址、长度、标志
- 空闲块链表

- 首次适配 first fit

  在空闲区表中找到第一个满足进程要求的空闲区
- 下次适配 next fit

  从上次找到的空闲区处接着查找

- 最佳适配 best fit

  查找整个空闲区表，找到能够满足进程要求的最小空闲区

- 最差适配 worstfit

  总是分配满足进程要求的最大空闲区

内存回收算法：当某一块归还后，前后空闲空间合并，修改内存空闲区表

- 四种情况：上相邻、下相邻、上下都相邻、上下都不相邻

伙伴系统

Linux 底层内存管理采用的经典内存分配方案，一种特殊的“分离适配”算法

主要思想：将内存按 2 的幂进行划分，组成若干空闲块链表；查找该链表找到能满足进程需求的最佳匹配块

内存不足时的管理方案

- 内存紧凑（可变分区）
- 覆盖技术
- 交换技术
- 虚存技术

覆盖技术

- 应用场景：程序大小超过物理内存的总和
- 在程序执行过程中，程序的不同部分在内存中相互替代，按照自身的逻辑结构将那些不会同时执行的程序段共享同一块内存区域
- 要求程序各模块之间有明确的**调用结构**
- 程序员声明**覆盖结构**，操作系统完成自动覆盖

覆盖技术的不足：

- 增加编程困难
    - 需程序员划分功能模块，并确定模块间的覆盖关系
    - 增加了编程的复杂度
- 增加执行时间
    - 从外存装入覆盖模块
    - 时间换空间

交换技术

设计思想：内存空间紧张时，系统将内存中某些进程暂时移到**外存**，把**外存**中某些进程换进**内存**，占据前者所占用的区域（**进程在内存与外存之间的动态调度**）

交换区：一般系统会指定一块特殊的磁盘区域作为**交换空间**，包含连续的磁道，操作系统可以使用底层的磁盘读写操作对其高效访问

进行交换的时机：

1. 进程不再使用或很少使用
2. 内存空间不够或有不够的危险时换出
3. 与调度器结合使用

不应换出正处于等待 I/O 状态的进程

换出后又换入的进程不一定回到原处（采用动态重定位）