操作系统

2014级种子班

电子信息与通信学院
钟国辉
zhonggh@hust.edu.cn

参考资料

- 《操作系统》
  - 屠祁 屠立德，清华大学出版社
- 《操作系统：设计与实现》
  - Tanenbaum等 电子工业出版社
- 《操作系统：精髓与设计原理》
  - William Stallings，机械工业出版社
- 《Unix环境高级编程》
  - APUE

授课方式

- 以自学为主
- 以课堂专题讨论为辅

与种子班其他课程的联系

- 微机原理
  - 操作系统是微机原理课程的后续课程
  - 在计算机系统当中处于高一层次位置
- Linux初步
  - Linux初步是操作系统课程的实践环节

回顾微机原理课程实验

- 软件模型的分类
  - 操作系统+应用程序
  - 无操作系统软件设计模型
- 无操作系统软件模型的特点
  - 应用程序即为独占全部资源的程序
    - 相当于单任务操作系统
  - 若有多重任务，需解决任务调度问题
  - 一般需直接与硬件接口打交道

回顾微机原理课程实验

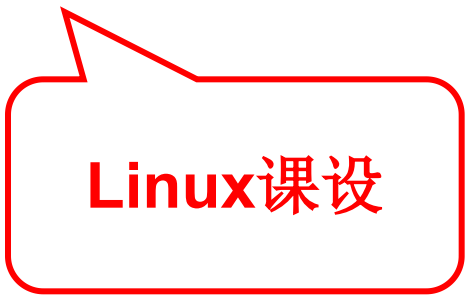
- 实验一：闪灯实验
  - 使用的模型？
- 实验二/三：运用定时器的闪灯实验
  - 使用的模型？
- 实验四：走马灯实验
  - 使用的模型？
- 实验七：串口实验
  - 使用的模型？

回顾微机原理课程实验

- 单道批处理操作系统
  - 装入程序、运行、打印结果、撤出、再重复
  - 用户把程序交给负责调度的操作员
  - 常驻监控程序自动地装入程序、运行、撤出作业
  - 单道批处理系统依然无法充分利用处理器资源
- 多道批处理操作系统/分时操作系统
  - 交互分时处理
  - 充分利用CPU资源
  - 有实时性要求
  - 有资源保护要求

操作系统

- 定义
  - 用以控制和管理系统资源，方便用户使用计算机的程序集合。
- 操作系统的主要设计目标
  - 方便用户使用
  - 扩大机器功能
  - 提高系统效率
  - 构筑开放环境



操作系统

- 操作系统的功能
  - **处理机管理**
  - **存储器管理**
  - 设备管理
  - 信息管理

操作系统的功能

- 处理机管理
  - 将系统中的各台处理机分给要求的用户作业使用。
  - 处理机管理的要点是**调度策略**和**调度算法**。
- 存储器管理
  - 按一定策略将主存空间分配给各作业
  - 记录存储器的使用情况，防止存贮破坏或损坏

操作系统

- 操作系统的特性
  - 并行性
  - 共享性
    - CPU的共享
    - 主存的共享
    - 外存储器的共享
    - 系统中数据或文件的共享
    - 外部设备的共享

操作系统的性能指标

- 系统的可靠性
  - 系统能发现、诊断和恢复硬件、软件故障的能力。
- 系统吞吐率
  - 系统在单位时间内所处理的信息量
- 系统响应时间
  - 从系统接收数据到输出结果的时间间隔
- 系统资源利用率
  - 系统中各部件、各种设备的使用程度
- 可移植性
  - 操作系统从一个硬件环境转移到另一个硬件环境仍能正常工作的能力

常见操作系统

- DOS
- Macintosh/ OS X
- Windows
- Unix
- Minix
- Linux
- MACH
- iOS
- Android

思考题/讨论题

- 操作系统作为一个程序，若要对应用程序进行有效调度，需要具备哪些条件？

操作系统

进程管理、处理机管理及其他

讨论题

- 旋转木马的管理问题
- 管理模型（管理要素）
  - 有一个定时机制，确保管理员可以获得控制权
    - 中断能力
  - 应当有权限控制机制
    - 特权级别
  - 管理员应当有策略和手段选择下一位游客
    - 进程控制、进程管理

中断

- 指CPU对系统中或系统外发生异步事件的响应
- 中断处理是操作系统的一个重要组成部分
- 操作系统就是由中断驱动的
- 中断是现代计算机系统中基本设施之一，它起着通讯联络作用，协调系统对各种外部事件的响应和处理
- 中断是实现多道程序的必要条件

中断的类型

- 强迫性中断
  - 输入/输出(I/O)中断：主要来自外部设备通道
  - 程序性中断：运行程序中本身的中断
  - (如溢出，缺页中断，缺段中断，地址越界)
  - 时钟中断
  - 控制台中断
  - 硬件故障
- 自愿性中断
  - 执行I/O，创建进程，分配内存
  - 信号量操作，发送/接收消息

微机当中的中断

中断屏蔽通过CPU标志位设定

- 可屏蔽中断（IO中断）
- 不可屏蔽中断（机器内部故障、掉电中断）
- 程序错误中断（溢出、除法错等中断）
- 软件中断（Trap指令或中断指令INT）

权限控制机制

- 特权级别
  - ring0～ring3
- 特权指令和特权寄存器
  - 仅允许在ring 0级别运行的指令
  - 仅允许在ring 0级别访问的寄存器

进程控制、进程管理

- 进程
  - 进程是具有独立功能的程序关于某个数据集合上的一次**运行活动**，是系统**进行资源分配和调度的独立单位**
- 进程与程序的区别
  - 动态与静态的差异
  - 进程包括程序与数据
  - 一个程序可以对应多个进程
  - 一个进程可以涉及多个程序

进程的性质

- 结构性：包括数据集合和运行于其上的程序
- 共享性：同一程序同时运行于不同数据集合上时，构成不同的进程
- 动态性：进程是程序在数据集合上的一次执行过程，是动态概念；而程序是一组有序指令序列，是静态概念
- 独立性：是系统中资源分配和保护的基本单位，也是系统调度的独立单位(单线程进程)
- 制约性：并发进程之间存在着制约性，在进行的关键点上需要相互等待或互通消息
- 并发性：进程可以并发地执行

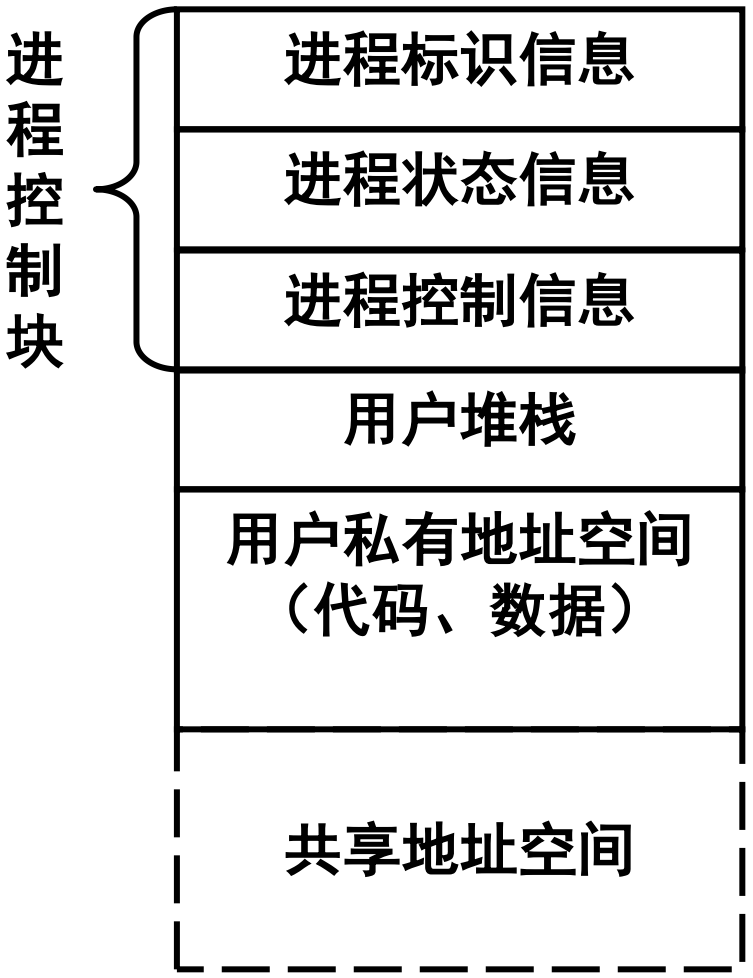
进程的状态

- 运行态（Running）
  - 进程占有CPU，并在CPU上运行
- 就绪态（Ready）
  - 一个进程已经具备运行条件，但由于无CPU暂时不能运行的状态（当调度给其CPU时，立即可以运行）
- 等待态（Blocked）
  - 指进程因等待某种事件的发生而暂时不能运行的状态（即使CPU空闲，该进程也不可运行）

操作系统应当管理（记录）进程的状态

进程的描述

- 进程程序块
- 进程数据块
- 系统/用户堆栈
- 进程控制块
  - 系统为了管理进程设置的一个专门的数据结构，用它来记录进程的外部特征，描述进程的运动变化过程



进程控制块

- 进程标志信息
  - PID、PPID、user
- 处理器状态信息
  - 用户使用的寄存器
  - 控制和状态寄存器:
    - 包括程序计数器PC和条件寄存器（或程序状态字PSW）.
  - 堆栈指针
- 进程控制信息

进程控制信息

- 调度和状态信息：
  - 进程的状态，进程的调度优先级，与调度有关的信息
- 进程在有关队列中的链接指针
- 进程间的通信信息：
  - 包括标志位、信号或信号量、消息队列等
- 主存使用信息：
  - 包括分给进程的主存大小和位置
- 进程使用的其他资源信息
- 进程得到有关服务的优先级

进程控制

- 创建、撤消进程以及完成进程各状态之间的转换。由具有特定功能的原语完成。
  - 建立进程原语
  - 撤销进程原语
  - 阻塞进程原语
  - 唤醒进程原语
  - 挂起进程原语
  - 解除挂起进程原语
  - 改变优先数原语
  - 调度进程原**语**

创建进程

- 创建一个PCB
- 赋予一个统一进程标识符
- 为进程映象分配空间
- 初始化进程控制块
- 许多默认值 (如: 状态为 New，无I/O设备或文件...)
- 设置相应的链接
  - 如: 把新进程加到就绪队列的链表中

进程切换

- 保存被中断进程的处理器现场信息
- 修改被中断进程的进程控制块的有关信息，如进程状态等
- 把被中断进程的PCB加入有关队列
- 选择下一个占有处理器运行的进程
- 修改被选中进程的PCB的有关信息
- 根据被选中进程设置操作系统用到的地址转换和存储保护信息
- 根据被选中进程恢复处理器现场

撤消进程

- 根据撤销进程标识号，从相应队列中找到它的PCB
- 将该进程拥有的资源归还给父进程或操作系统
- 若该进程拥有子进程，应先撤销它的所有子孙进程，以防它们脱离控制
- 被撤销进程出队，将它的PCB归还到PCB池

改变优先级数原语

- 进程的优先级数是表示进程的重要性及运行的优先级，进程调度程序以此来确定优先调用哪一个进程到处理机上运行。
- 为防止一些进程因优先数太低而长期不能运行，许多系统采用动态优先数。
- 影响优先数的因素
  - 作业开始时的静态优先数
  - 过程的类型
  - 过程所使用的资源量
  - 在系统中的等待时间

思考题/讨论题

- 在多处理器的时代，进程是当前操作系统调度的最小单位吗？
- 如何引入新机制使得操作系统能更好的利用资源
- 如何确保资源访问的共享性和可靠性

操作系统

线程、并发、通信问题

线程的引入

- 进程的优点
  - 将程序和运行时的数据合集进行统一管理
  - 为操作系统管理多用户、多任务环境提供了基础
- 进程的不足
  - 各进程的相对独立性为数据共享带来的限制
    - 可通过IPC手段予以解决（但效率并非最佳）
    - 进程间的切换开销太大
  - 进程不能很好的利用多处理器资源
    - 单个进程只能在一个处理器上运行

线程

- 定义
  - 进程中的**一条执行路径**，是系统进行处理器调度的基本单位，同一个进程中的**所有线程共享进程获得的主存空间和资源**
- 线程独立拥有的资源
  - 线程执行状态
  - 受保护的线程上下文
  - 独立的程序指令计数器
  - 执行堆栈
  - 容纳局部变量的静态存储器

```
int GlobalInt;

void thread_func(void){
        int LocalInt;
        while(1);
}

main(){
    createthread(thread_func);
    createthread(thread_func);
    createthread(thread_func);
    while(1);
}
```

线程

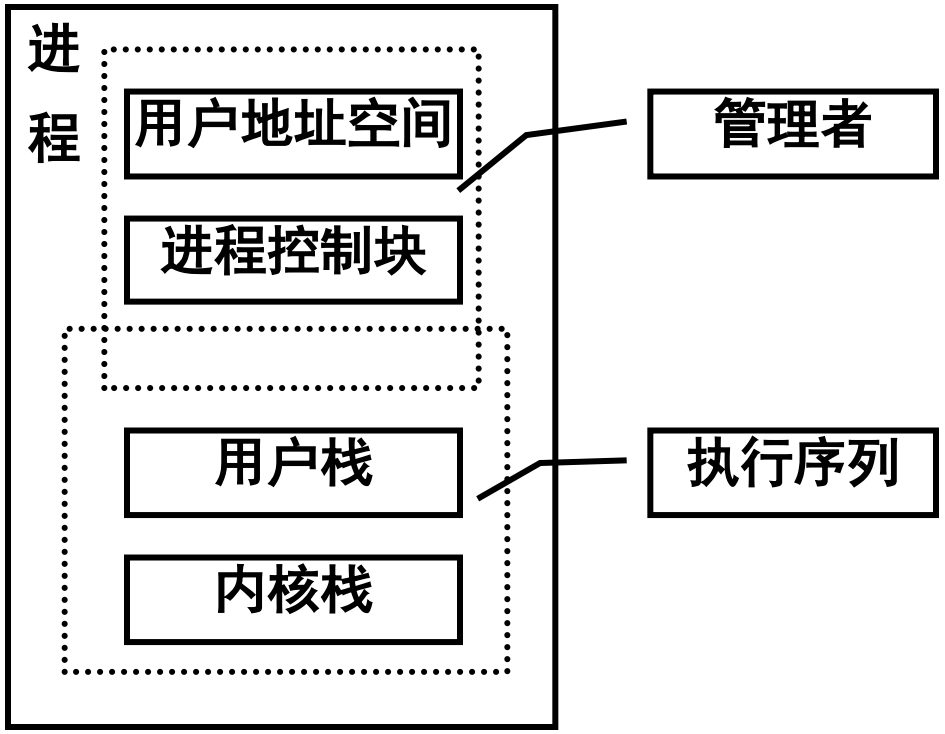
- 特点
  - 并行性
    - 同一进程的多个线程可在一个或多个处理器上并发运行
  - 共享性
    - 一个进程下的线程共享进程获得的主存空间和一切资源
  - 动态性
    - 线程也是程序在相应数据集上的一次执行，由创建而产生，至撤销而消亡，有其生命周期

线程

- 线程的优点
  - 创建一个新线程花费时间少（结束亦如此）
  - 两个线程的切换花费时间少
  - 因为同一进程内的线程共享内存和文件，因此它们之间相互通信无须调用内核
  - 适合多处理机系统

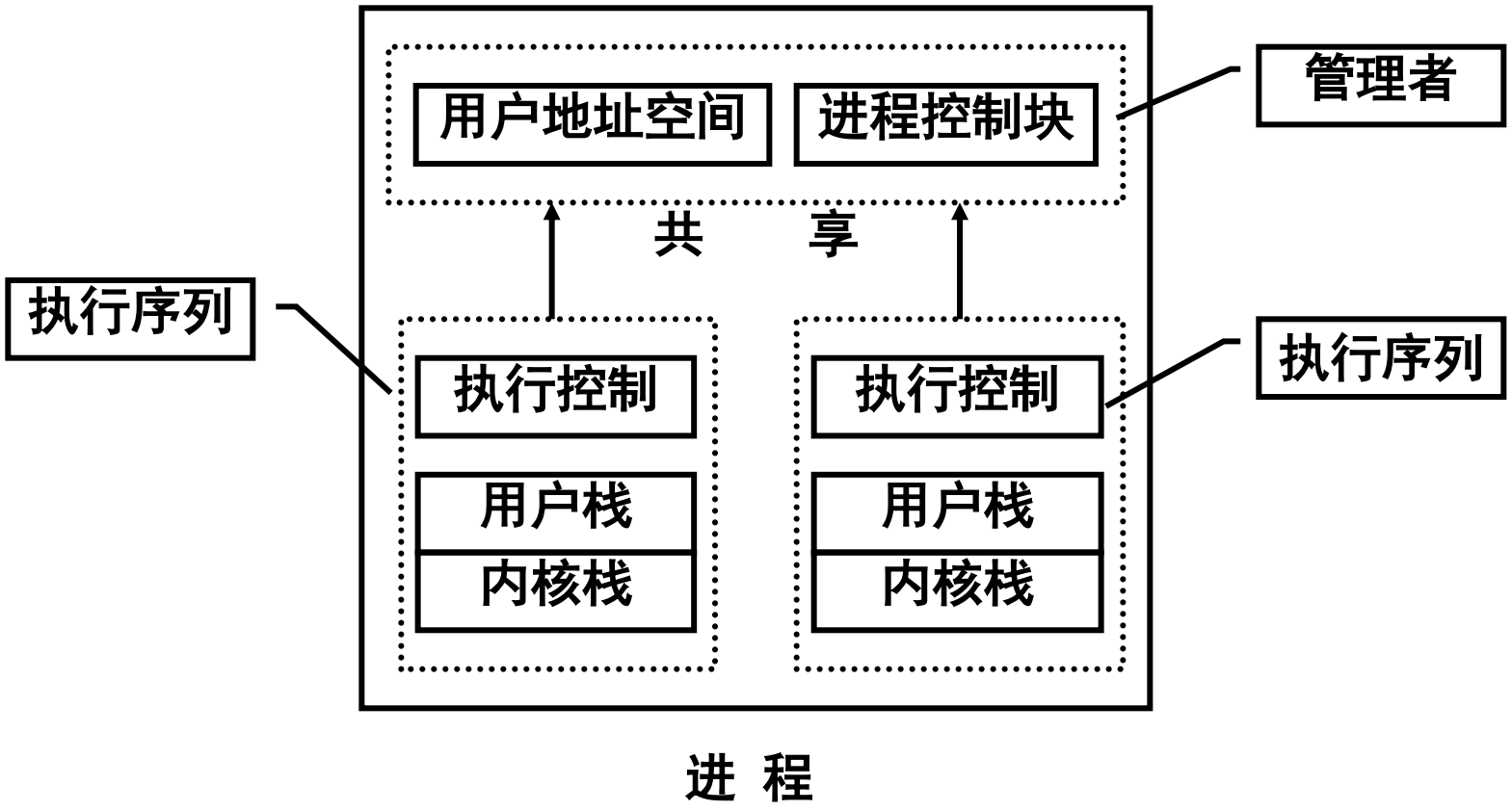
线程

- 进程和线程的模型
  - 进程（单线程进程）



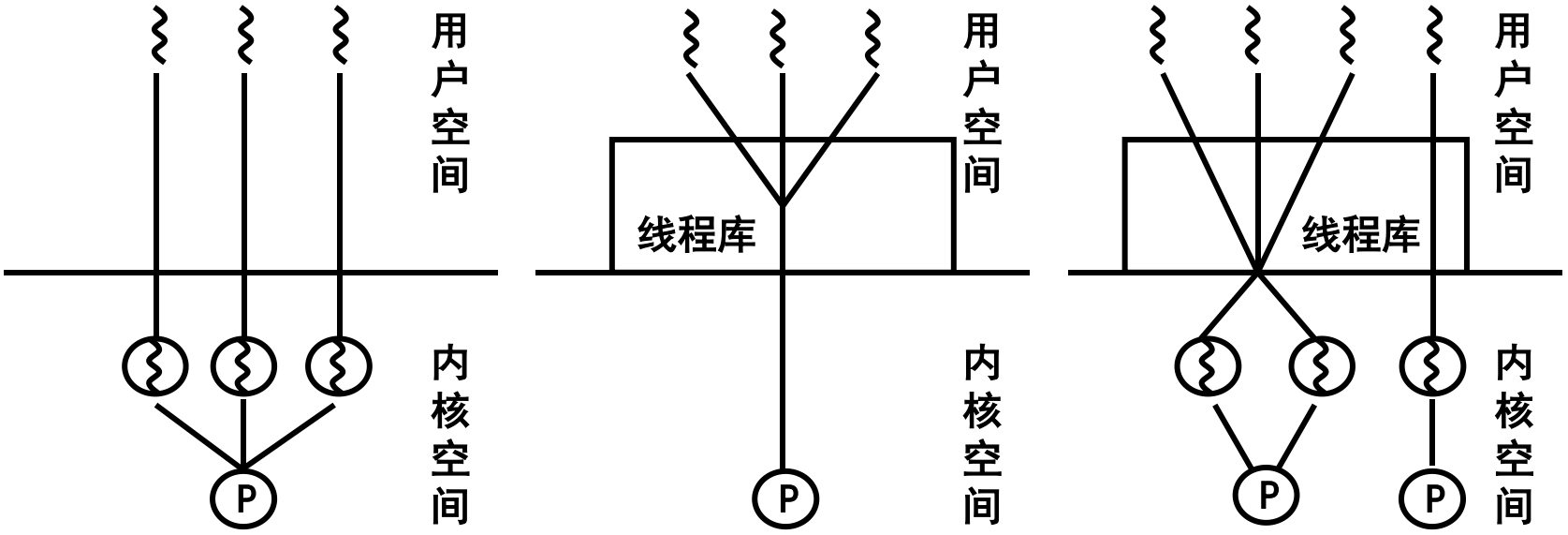
线程

- 进程和线程的模型
  - 进程（单线程进程）
  - 多线程进程



线程

- 线程的实现
  - 用户级线程（User Level Thread）
  - 内核级线程（Kernel Level Thread）
  - 两者结合的混合级线程



同步/互斥

- 多道/分时操作系统的目标之一
  - 任务的并行性
- 任务并行性的要求
  - 并行程序设计
- 并行程序设计的特点
  - 互斥性
    - 例：售票系统
  - 同步性
    - 例：流水计算工作

同步/互斥

- 进程间的相互制约关系:
  - **同步关系**（直接制约）：多个进程共同完成一个任务，它们之间必须协同动作，互相配合，相互交换信息--进程通信
  - **互斥关系**（间接制约）：多个进程共享资源，互斥资源的使用具有排它性，因此进程间往往需要互相竞争，以使用这些互斥的资源。也可看成一种特殊的同步关系!

同步/互斥

- 临界段定义
  - 进程中访问共享变量的代码段
- 临界资源
  - 允许多个进程共享使用的资源

同步/互斥

- 临界段设计原则
  - 每次允许一个进程停留在临界段
  - 进程只能在临界段内逗留有限时间
  - 进程不能在无限期等待在临界段之外
  - 临界段外进程不可以阻止其他进程进入临界段
  - 不能预期和假定进程进展的相对速度以及可用的处理器的数目

同步/互斥

- 临界段的实现
  - 软件方法
    - 复杂，局限性大
  - 硬件方法
    - 简单实用，使用广泛
    - 中断屏蔽方法
    - 硬件指令方法

同步/互斥

- 中断屏蔽方法
  - 一个进程正使用处理器执行它的临界段代码，为防止其他进程进入它们的临界段的简单直接的方法是禁止一切的中断发生，即为中断屏蔽
- 缺点
  - 系统付出的代价很大
  - 在多处理机系统中不能保证系统的互斥

同步/互斥

- 硬件指令法
  - 用公共变量标识是否占用临界资源
  - 用一条不可打断的指令完成对公共变量的测试和修改
- 优点
  - 不但适用于单处理器情况，而且适用于共享主存的SMP多处理器情况（即对称多处理器）
  - 方法简单，行而有效
  - 可以被使用于多重临界段情况，每个临界段可以定义自己的共享变量

同步/互斥

- 由此形成的同步/互斥的操作原语——PV操作
  - P(S)操作——将信号量S减1
  - V(S)操作——将信号量S加1
- 同步互斥的经典场景
  - 互斥——售票系统
    - P、V操作
  - 同步——生产者消费者问题
    - V、P操作

同步/互斥

- 当前操作系统当中的同步互斥手段
  - 信号量
  - 读写锁
- 其他的同步互斥概念
  - spinlock（自旋锁）
- 设计当中需要注意到问题
  - 死锁

思考/阅读/实践

- 在Linux环境下实验进程和线程编程
  - fork()函数调用
  - pthread_create()函数调用
- 阅读《Unix环境高级编程》当中
  - 进程章节
  - 线程章节
  - IPC通信章节
- 分别用多进程、多线程的方法实现累加和计算

操作系统

存储器管理

存储器管理

- 存储器管理的目的
  - 尽可能的充分利用存储器资源
  - 尽可能满足作业进程对存储器的需求

存储器管理

- 尽可能充分利用存储器资源
  - 对抗目标——内存碎片
- 存储器分配方案
  - 固定分区方案
    - 实现简单
    - 对不同大小作业进程的能力较弱，内存利用率低
  - 可变分区方案
  - 虚拟存储方案

存储器管理

- 可变分区方案
  - 在空闲内存当中为作业请求分配内存
  - 记录每一个作业任务的内存占用情况
- 存在的风险
  - **空闲内存碎片化**
- 改进措施
  - **碎片整理**
- 所需条件
  - 地址动态重定位的硬件、软件支持
    - Runtime Relocator

存储器管理

- 固定/可变分区方案的问题
  - 内存分区是唯一的
  - 无法实现数据共享
- 虚拟存储方案
  - 分页管理
  - 分段管理
  - 段页式管理

存储器管理

- 分页管理的出发点
  - 可变分区方案的延伸
- 实现方式
  - 作业进程申请的空间由多个（可不连续）分页构成
  - 由操作系统当中的页面映像表标识页面与实际地址的转换关系（操作时的地址转换由MMU硬件完成）
- 存在的问题
  - 代码和数据同时存在于页中，未作区分，共享时的数据保密不易达到

存储器管理

- 分段管理的出发点
  - 代码、数据分区存放，以便共享
- 实现方式
  - 作业进程根据存储对象不同在不同的段内存储
  - 由操作系统的段表和硬件的段表地址寄存器完成运行时的地址转换
- 存在的问题
  - 段的大小和分配方案限制同可变分区存储方案

存储器管理

- 段页式管理
  - 代码、数据分区的分段管理方式
  - 段内采用分页管理的方案
- 实现方式

如何尽可能满足作业进程对存储器的需求？

存储器管理

- 尽可能满足作业进程对存储器的需求
  - 进程的所有代码和数据是否需要全部加载到内存？
- 仅加载最近需要运行的部分
  - 若发现运行的部分不在内存当中？
    - 触发缺页中断，由操作系统负责加载到内存
  - 存储器管理中需标识作业空间中内容不在内存的部分
- 置换暂时不使用的部分
  - 置换策略
    - FIFO、LRU、NUR等
  - 置换范围
    - 全局置换、局部置换

存储器管理

- 尽可能共享存储器
  - 进程主动共享——作业明确要求的部分
  - 进程被动共享
    - 代码——相同的程序运行，代码段可共享
    - 数据——重复的进程，数据段共享
      - 仅在某一个进程改变数据时，复制一份予以修改
      - **Copy-on-Write**技术

存储器管理

- 其他的一些问题
  - 如何判定进程越界访问
    - 通过设定MMU当中的界限寄存器
    - 越界访问时触发中断（Linux下收到SIGSEGV信号）
  - 如何限定访问权限
    - MMU当中段页式管理的段权限描述
      - 硬件仅能区分Ring0~3的范围
      - 可区分内核权限和用户权限

小结

- 操作系统的核心问题
  - 高效管理
- 实现的核心
  - 处理机管理
    - 作业调度
      - 手段（中断）、策略（调度算法）
    - 作业共享、同步、互斥
  - 存储器管理
    - 充分利用存储器（段页式虚拟存储器管理）
    - 尽可能满足进程要求（动态加载，随时置换）
- 其他的管理——设备和文件（数据结构组织问题）