操作系统

• 赵俊峰 zhaojf@pku.edu.cn 62751794(O) 办公室
北大信息科学技术学院 软件研究所
刘少钦(助教) liu_shaoqin@pku.edu.cn

• course.pku.edu.cn 课程网站（北大教学网）

教材：以讲义为主
Ref：操作系统概念…
操作系统教程…
modern operating system | cover-ing 涵盖了操作系统的各种概念

• 按先全力，课后要复习
• 阅读linux源码 {进程 / 文件 管理} → 结合课堂上的内容，从全局观念看 {模块化流程 / 用了哪些数据结构} → 期末考查吗
实习参考书 ~ Linux 内核完全注释 …
• 课程：主讲+讨论+作业
考试：笔试
成绩：作业+源码分析 40%，期末考试 60%
平时作业 20%（word或pdf提交）
源码分析 20%

OS 属于系统软件
- 实践性强 → 必须读源码
- 涉及面广 算法、体系结构、程序设计语言、软件工程等
- 错综复杂（纵横交叉）

为什么学操作系统
人类伟大软件排行 {创造性、实用性 / 他们的价值 / 广泛应用或巨大影响的}

12. The Morris Worm
蠕虫病毒（缓冲区溢出）→ 威力巨大！
email list 寻找地址 email list
从此人们意识到互联网的快速传播特点和巨大威力

11. Google 网络网页搜索及排名算法 Search Rank
创始人 Larry Page, Sergey Brin
彻底改变了人们获取信息的方式

10. Apollo Guidance System 阿波罗飞船的导航系统
内存只有8K 解决了冗余性和容错性
开创了编程风格

9. Excel (spreadsheet)

8. Macintosh OS 第一个图形化的操作系统，第一次引入鼠标，第一次面向对象的思想

7. Sabre system 航班查询系统（订票）
并发的问题如何处理

6. Mosaic Browser 第一个网络浏览器

5. Java Language 跨平台、网络、分布式的支持

4. IBM System 360 OS
第一个商品化软件，应用于大型机的OS（人月神话 软工经典著作）
由此开始了"软件工程"

3. Institute for Genomic Research
美国基因组研究所 基因测序软件。
巨大的数据的存储分析。
实现了对人类基因的[illegible]

2. IBM System R → IBM的R系统
是数据库系统的前身

1. Unix操作系统 [illegible]的第一
同时可在服务器、PC端运行.
开发、软件的良好环境.
高效、可移植性好

与操作系统有关的有3个：1. 4. 8.

- 内容介绍

  操作系统概述
  进程管理
  存储管理 — 最重要、最难
  文件系统
  设备管理
  用户接口与作业管理
  死锁
  多处理机概述（这学期可能不讲了）
  操作系统实例 Unix、Win、Linux
  实习 — 源码阅读.

```
#include <stdio.h>
int main
    printf("hello world");
```

操作系统做了什么？
用户 ←
程序装载
检查首部 <文本, 数据>
文件系统找到第一个存储块
22页. CPU 设置上下文
第一条指令执行，失败，缺页中断
OS 分配内存，读入代码，开始执行
更多的缺页中断，读入更多页面
⋮

程序设计语言
高级语言程序.
编译、汇编.
机器语言.

运行程序的步骤

OS 加载器、文件、内存
→ 进程
→ OS CPU调度、进程管理
→ CPU执行 —— 程序语言结构、指令集结构 的区别
→ 文件、IO
→ 结果

OS的基本概念、基本结构、运行环境

- 课程的目的
  - OS的原理、设计方法（如线程、调度）实现技术 ← 在其他领域中也可借鉴、应用
  - OS的设计理念、发展趋势、新技术、新思想
  - [illegible]OS、培养分析、解决问题的能力

·学习方法：

what. why. how ← 每天学习之后 问自己（主动）

OS是什么. 为什么要OS. OS如何工作.
做什么. 如何使用.

多维思考：以用户程序为主程序，来思考
{以OS程序为主程序，来思考.

·操作系统是人造科学

不精确：不同的人就不一样的设计，没有标准答案	自然科学
对从人类活动的现象导出	精确，绝对
科学的研究（队列，栈等）	不依赖人的意识和判断力 客观

·MIT课程借鉴 — 经济的培养

2.009 Product Engineering Processes

动用当值的老师

将学生分组，每组5000$预算，设计并制造产品.（15个学生）

结束时，向全体家长，工业界资助者展示

1. 操作系统概论

1> 什么是操作系统 { 为何要学习 / 有什么功能

- 操作系统的特征
- 考研究OS的观点
- OS的功能
- 发展历史
- 分类
- 设计
- 常用的os

OS的名称

Operating System

- Monitor
- Executive System (Program) 执行系统
- Control System Program
- Supervisor
- Kernel
- OS

OS的地位

Computer { 软件 { 应用 / 支撑 database… / 系统 ; 硬件/固件

系统：在所有硬件之上，在所有软件之下.

具有有效性（管理人员）管理、分配硬件、软件，合理组织工作流程

方便性（用户的观点）良好、一致的接口

可扩充性（开发的观点）

OS的定义：系统软件（程序 数据）

组织管理计算机软硬件资源

组织计算机工作流程，控制程序执行，向用户提供多种服务

{用户观点：一台虚拟机
Computer System观点：资源管理器
应用软件观点：软件开发和运行平台

OS的作用

资源种类：CPU、存储、外设、信息（数据、软件）

资源的内容：资源的当前状态、分配、回收、访问、相应的管理策略

管理技术：资源的复用（空分复用共享、时分复用共享）

资源的虚化（虚拟内存技术等）

资源抽象（处理系统复杂性，解决资源易用性）

· 是用户使用软硬件的接口

· OS是扩展机 / 虚拟机

2> OS的特征

并发性(concurrency)

1. 在计算机中有多个正在运行的程序。 并发性 → 注意：并发指的是在同一时间段内发生（宏观）
并行指的是同一时刻发生（微观）

eg. 用户program之间，用户pro和os之间并发

宏观上并发，微观上交替执行，多道程序。

2. 程序的静态实体是可执行文件，动态实体是进程（任务），并发性指的是进程

3. 多个进程共享机器的计算机资源

共享性　OS与并发，交替执行。

共享方式 { 同时共享 / 互斥共享

共享机制 { 透明资源（用户不可见） / 显式资源

4. 不确定性

OS对运行程序的行为和硬件的事件不能有任何事先的假定

也称异步性（进程的执行顺序，时间不确定） ~ 同步和互斥手段。停止正常执行。

5. 虚拟性

一个物理实体 — 多个逻辑实体（分时，分空间）

用的是为了提高资源利用率

3> 研究操作系统的观点 — 软件的观点（不能一开始就局限于细节）

尽管OS很复杂，但有一般软件结构的特点，也有特殊结构。

内在地 — 软件的结构 几部分？ { 每部分的功能 / 各部分的关系 ，是什么样的结构

外在地 — 命令集和它的接口（功能）

— 资源管理的观点。

操作系统 — 资源管理者 — 硬件资源（CPU，内存，外设） / 软件资源（文件，信息）

用什么数据结构记录资源。

{ 如何分配资源（动态，静态策略） / 具体完成分配 / 回收资源

— 进程的观点。

从OS运行的角度动态地观察OS。

进程 { 是一种有特定功能的prog. / 是一次执行过程 / 有生命，会消亡。

— 从虚拟机的观点

从OS的内部结构　分成若干层 / 每一层完成特定的功能，成为一个虚拟机，支持上层

Linux有1/3是汇编语言写成的，用汇编写成

从用户的角度看：操作系统为用户提供了功能强大的系统调用

应用程序
— 美丽的接口
OS
— 丑陋的接口
硬件

4> OS的功能 ～ 复习大纲

进程管理（最重要）
- CPU的管理
- 进程控制
- process 同步、互斥
- 进程间的通信（八种方式，各自优势）
- 调度
  - 处理器、线程的调度、进程调度、作业调度

内存管理（也重要）（四种方法 解决三种问题）
- 内存分配、回收
- 存储保护
- 内存的扩充（虚拟存储技术）

文件管理（easy 经常考）
- 对外存空间的管理（利用率，访问效率）
- 目录管理
- 文件系统的安全性

作业管理
- 高级、中级、低级的调度，更好地完成任务

设备管理
- CPU、IO设备、内存的交互
- 填补高速设备和低速设备之间的鸿沟

中断处理、错误处理

网络与通讯管理

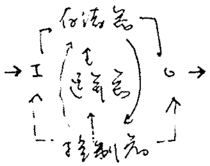
5> OS的发展历史

推动OS发展的主要动力 ← 器件技术的发展

算盘、差分机

信息技术的发展史 ≈ Computer发展史

Turing Machine 现代计算机的产生
→ ABC 第一台计算机 1937-1941
→ ENIAC → EDVAC 存储程序设计
现代计算机唯一的设计结构 →



OS的发展历史

1957 电子管 → 晶体管
1964 集成电路 → 1970 大规模集成电路

硬件的更新（摩尔定律） 与软件的修正
计算机体系结构的发展
└ 单机→多机
  网络技术的发展
随硬件更新发挥硬件软件的威力

提取各种软件的共性
并将之归入OS

soft / hard ⇒ soft / OS / hard | 人机接口 系统调用；资源管理

⇒ soft / 中间件 / OS / hard | 网络：远程调用、安全容错

更好地满足用户对易用性的需求
终端 → shell → 窗口 → 浏览器

1946-50年代（电子管时代，如ENIAC）
- 计算机资源非常昂贵 { 为了二战；美、英、德等国家

手工操作时代
- 工作方式：用户既是程序员，也是操作员
- 编程语言：机器语言
- IO介质：纸带/卡片

密歇根大学的执行系统 UMES：操作系统的萌芽
- 早期编译器的执行系统 { 可保存中间结果；进一步移植到Philco、Univac、CDC等系统；加入到fortran语言

50末-60中（单道批处理系统）
- 利用磁带将作业组织成执行序列：作业 { 用户程序；数据；执行序
- 高级语言(纸) → 成为卡片 → 执行
- 引入了通道和中断技术
  - 与CPU并行
  - IO与CPU并行执行，解决了两者的互相干扰拖慢
- FMS (Fortran Monitor System)
- IBMSYS

60年代中－70年代中（集成电路）

多道批处理系统　　非常复杂 { 系统复杂 / 完全使用汇编 / e.g. IBM 的 360 system，大量的 bug（人月神话）

问题 {
- 内存分配
- 内存保护（避免一个程序的 bug 影响其他程序）
- CPU 调度
- 各个子系统之间交互

多道程序设计技术的出现 {
- 资源利用率高，作业吞吐量大
- 但只是批处理而已，没有交互

分时系统

(1959, MIT 正式提出 Multics 系统)

多个用户（多个程序）分时使用

前台程序和后台程序 → 交互性好，可共享主机

Unix 操作系统

Ken Thompson 和 Dennis M. Ritchie.

在 PDP-7 上开发了游戏，为此开发了完备的系统

1970 诞生，取名 UNIX

成功的原因：不企图设计满足所有用户的所有需求的软件

简洁设计，为广大用户提供良好的编程和运行环境的 OS。

《The Art of Unix Programming》 UNIX 哲学

也是现代程序设计的哲学

个人计算机操作系统

MSDOS

⋮

OS 的发展方向 {
- 分布式 OS　云 OS
- 机群 OS
- 嵌入式 OS

操作系统　　赵俊峰教授　　PKU三教105
20181008

6> 操作系统的分类

计算机软件 = 程序 + 文档

硬件（提供计算能力）= 一组原本指令的集合

软件发展的三个阶段

- 1946~1956 { 机器语言 / 汇编语言 }　从第一个程序到高级实用语言出现前
- 1956~1968　从实用高级语言到软件工程出现前
  - 语言越来越类似英语表达
- 1968 ~ 现在　软件工程出现后
  - 软件开发有了系统性的工具和方法

软件的分类
- 系统软件 → 支持应用软件运行
- 支撑软件 } 基础软件 → 支持各层次的软件开发
- 应用软件

操作系统分类

① 批处理操作系统
- 作业录入 → 作业控制说明书
- 调度（优先级）
- 作用运行

缺点：不允许交互，调试难
优点：效率/吞吐高

单道批处理 和 多道批处理
(uniprogramming)　　→ 多个作业交替运行（而非多个处理器）

Spooling技术　利用磁盘缓冲区（将I/O与计算异步了）　虚拟化技术的第一次成功的应用

② 分时操作系统　一台主机接多个终端
- 计算　/　端I/O能力

优点：
- 多路性
- 交互性
- 独占性
- 及时性（及时响应）

CPU划分为时间片
串行处理各终端指令，为各用户服务（会话语言）
时间片的划分规则是需要trade-off的，非常讲究（轮转算法）

影响响应时间的因素
- 机器处理能力
- 请求服务时间长短
- 连接的终端的数目
- 服务请求分布
- 调度算法

通用操作系统 ↗ 分时与批处理结合

③ 实时操作系统（Real Time OS）
- 以足够快的速度在规定时间内快速地作出处理.
- 与普通应用机不太一样，工业上用得多（导弹制导，数控机床加工）功能专一
  - 实时过程控制
  - 信息查询系统
  - 事务处理系统

特点：对外部请求在严格的时间范围内作出响应　实时时钟管理（压力测试与过载防护 e.g. 铁路售票系统，12306系统）
有高可靠性，完整性

④ 网络操作系统

原有操作系统 + 网络通信与网络服务功能
（WIN-NT开始有）

特点与功能　目的是相互通讯和资源共享
- 通常的OS功能
- 网络通信
- 网络资源管理
- 网络服务
- 网络管理
- 互操作

主要的工作模式
- 客户机/服务器（Client/Server）
- 对等（P2P）模式 { 非中心化 / 可扩展性 / … } → 信息上分布，处理上不分布
  - 各计算机机器OS可以不同，只要遵从约定协议；资源可以共享；有主从关系

⑤ 分布式操作系统

与单机集中式OS的差别
- 资源管理
- 进程通信
- 系统结构

→ 处理上分布 / 更高级的形式，整体上调度所有的计算能力　无主从关系
- 资源为所有用户共享
- 机器必须使用同一个OS：掌握了
- 可选择可以分散运行
- 更强的容错能力

↓
对外部用户，用户没有主从之分，只有虚拟机

特征：透明性（用户不知道资源的分布）
自治性

⑥ 个人计算机OS
- 开放性
- 通用性
- 高性能
- ⋮

⑦ 嵌入式OS　依赖特定运行环境和应用的功能，与硬件结合紧密

传感器 → A/D转换（模拟）→ CPU → D/A转换 → 执行机构

特点：
- 微型化
- 可定制
- 实时性
- 可靠性
- 开发环境与普通OS不一样

⑧ 并行OS

面向繁重的计算任务（e.g. 天气预报）

常见类型：
- 阵列处理机
- 流水线处理机
- 多处理机

OS的发展趋势
- 模块化
- 动态可配置
- 网络化
- ⋮

新型的高效的操作系统

7) 操作系统的设计
- 困难
- 过程
- 目标
- 结构
- 体系结构

OS设计的困难

- 复杂程度高（程序长，接口信息多，动态性强）
- 正确性难以保证（没有数学证明保证正确性，只能通过良好的结构，先进的开发技术，工程化的管理，高效的工具）
- 研制周期长（开发周期5年，维护期几十年）

OS设计过程
- 功能设计：目标和使用要求；有哪些功能
- 算法设计
- 结构设计

OS设计目标
- 可靠性：正确性；健壮性
- 高效性
  - 计算机系统运行状态：
    - 目态（TU时间）——为用户服务 / 用户程序运行时间
    - 管态（TSU，TGM时间）——为用户服务；管理
  - $\eta = \frac{T_U + T_{SU}}{T_U + T_{SU} + T_{GM}} \times 100\%$
  - 只有TGM可以控制，使用高效的算法的用处在此
- 易维护性（对OS非常重要！）
  - └ 容易修改
    - 纠错性维护
    - 适应性维护（适应新的运行环境）
    - 完善性维护（增加新功能）
- 可移植性
  - 指能迁移到其他的操作系统并正确运行
- 安全性
- 简明性
- 可扩展性
- 性能

OS的结构设计

└ OS的各部分程序的存在方式及相互关系

OS结构设计目标
- 系统模块化：一组数据结构以及定义在这组数据结构上的操作
- 模块标准化：按层设计，模块规划统一，相同的构造准则
- 通讯规范化：模块之间的接口应该清晰划一，模块联系方式定统一

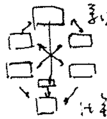
OS结构设计
- OS结构理论
  - 内核：为可信软件提供支持并发执行的基本功能和基本操作
    - └ 分类：单内核（整体式，层次式结构）；微内核
    - └ 内核功能
      - 资源抽象（屏蔽底层物理细节，统一接口）
        - └ 机制与策略分离（机制提供什么功能，策略如何使用功能）UNIX首创这个的最好实践
      - 资源分配
      - 资源共享
    - └ 执行属性
      - 由中断驱动
      - 执行连续，不可打断与
      - 可用特权指令
  - …
- …

OS的体系结构

整体式结构

基于模块化程序设计思想（结构化程序设计）

系统按功能分成模块化结构，是这里第一个庞大的软件系统



"杂乱无章"

各模块之间互相调用 效率高
- 灵活性高
- 但模块之间独立性不高，不利于修改

示例：Linux 单内核结构（宏内核）

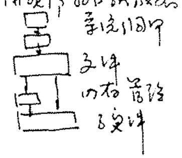
层次式结构

系统分为内核和若干模块

模块间按功能的调用次序排列成若干层

被调用的功能，应该在调用功能的下层

分为
- 全序：各层间单向依赖，层内独立
- 半序：层内也可以相互调用



"意大利面"

优点：
- 功能明确
- 高层错误不会影响到低层 —— 把硬件依赖的部分，增加了可适应性和可移植性，便于计算机在不同体系方面迁移

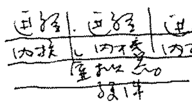
缺点是效率低下，通讯代价大

示例：E.W.Dijkstra 的THE系统（1968） 中转核

虚拟机结构

一个多任务系统在操作系统以下的一些硬件 多道程序

一个比裸机更方便，易于扩展的系统



"呼叫呼叫"

基本的原理：用一类设备模拟另一类设备

CPU调度程序，虚存管理，SPOOLING技术

使得每台虚拟机可以执行裸机的全部功能

微内核结构（最流行）

将OS分离出来，只留一个最小的kernel，其他功能用通讯机制实现

→ 分离程序的差别：和不同的功能全部分开

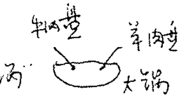
必须是关键的：硬件相关、内存管理、进程管理、进程通讯

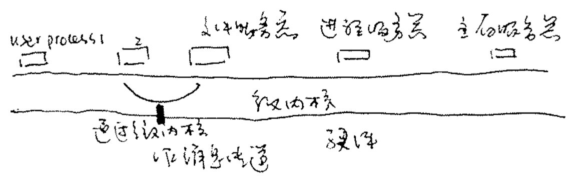
OS分为两个部分

用户态运行 以客户/服务器方式运行

核心态运行 将客户请求传给服务器，[illegible]

"蒸菜似的大锅"





优点：一致性接口
- 可扩充性
- 可移植性
- 可靠性（系统功能与微内核代码，易于测试，统一的API接口）
- 支持分布式系统
- 支持面向对象的操作系统（将系统内核分为产品与用户方面对象技术）

缺点：消息传递 比直接调用开销大

解决方案
- 扩充微内核的功能
- 微内核更小，但结构[illegible]

OS 操作系统的运行模型

（OS本身运行模式）

独立运行的内核模型

对应整体式，层次式结构

OS代码在内核模式运行，内核本身有自己的空间

OS功能在用户进程内执行的模型

对应虚拟机结构

小型机和工作站常用

OS的功能在独立的进程执行

对应微内核结构

更灵活，可扩展性更好

8) OS应提供的功能

- 创建程序
- 执行程序
- 文件
- 通信
- 错误检测与处理
- 资源分配
- 统计
- 保护

OS通过程序接口和操作接口向用户提供服务与功能

- 由一组系统调用组成 (system call)
- 用一组控制命令和作业控制语言组成 (如鼠标点击等)

好的接口应功能、性能完善、方便用户 (是对硬件能力进行封装, 提供更方便的接口)

是用户程序和其他程序获得操作系统服务的唯一途径

(e.g. POSIX 1003.1 基于unix的可移植接口)

(只是接口)

区别:
- API是一个函数定义, 说明如何获得给定的服务, 而不使用系统调用
- 库函数是一种API, 一般对应到系统调用
  - fprintf() 函数 → write 函数 → sys-write 系统调用 (前两个为 C语言函数)
- 系统调用: 比较底层

用户
编程系统程序
标准库函数
系统调用

Linux 最多有1000个系统调用 { 核心函数 / 接口函数 lib.a

系统调用 → asm 汇编完成的地址表 → 完成调用

OS标准化: 各OS不兼容

所以不同标准 { POSIX / GJB/Z-22-91 军方的标准 / iTron 日本最广泛的嵌入式操作系统

所以统一的标准有利于运行不同的应用程序, 生态系统更繁荣

9) 常用的操作系统

MS DOS 分层结构

命令处理程序
DOS核心
BIOS → Basic I/O System 硬件驱动

81年研发 ~98年基本放弃

→ 基于1973年的CP/M操作系统

特点: { "单任务" / FAT文件系统, 8+3文件名.

Macintosh (Mac os) 源于 Palo Alto 研究中心

- 第一台PC机
- 图形界面
- 用户鼠标
- 面向对象程序设计

1984年, 带图形界面和鼠标的 Macintosh 系统

Windows 1983年推出, 1985年正式上线

3.0 → 95 → 98 → ...

WIN-NT 体系结构, 第一个微内核的OS

...
内核
硬件抽象层

Mach 操作系统: 微内核结构
基于 BS-UNIX 优化

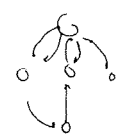
OS/390 — IBM 大型机的操作系统

UNIX → 传统的[illegible]整体式

hard 驱动
hard

LINUX

User Applications
O/S Service
Linux kernel →
Hard Control



VxWorks 最常用的嵌入式操作系统 (e.g. 火星探测器)

操作系统　赵海峰教授　PKU 3教105
20181022 Lecture 4

习题课(一)

问题：如果系统中有N个进程（不考虑内核）
- 运行进程最多1，最少0。← 系统调度或死锁
- 就绪进程最多N-1，最少0。
- 等待进程最多N，最少0。

不考虑调度的空隙

被调度选中 → 运行 → 等待事件发生
就绪 ← 等待（等待的事情发生）

是否可以 等待→运行？
　就绪→等待？
不行，需要获取资源，然后再转就绪，再运行
应该先运行，才能知道是不是缺少资源，才等待

问题：如果一个转换发生，是否另一个转换一定发生？
- 运行 → 等待：就绪，不一定
- 等待 → 就绪：不一定
- 就绪 → 运行：不一定（系统中仅一个进程）
- 运行 → 就绪：此时 $N_{运行}=0$，$N_{就绪}\ge 1$（若 $N_{就绪}=1$，则不一定）
　或者 $N_{就绪}\ge 1$，但因就绪进程们具有最高优先级，则不一定

进程概念的回顾
进程：并发性、动态性、独立性、交往性、异步性、结构性、制约性、共享性
进程即运行中的程序

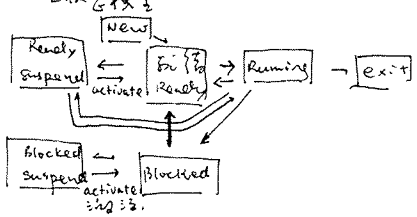
进程三状态的延伸：五状态 和七状态
↳ 即是挂起与内存相关等

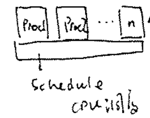
创建(新new)状态
OS完成创建进程的工作：进程符、资源表格
创建进程控制块，初始化相关寄存器，初始化内存，将程序代码载入内存。
将程序寄存器设为用户程序第一行代码
为什么第一行不一致？
OS的相关系统调用
UNIX: fork
windows: CreateProcess

终止(退出exit)状态
中断进入，不再有执行资格。

五状态模型
new → 就绪 → 运行 → exit (release)
　　　↑
　　　等待

Suspend → 或低优先级，或者时间调试（挂起调试进程）
- 内存不够，先把资源移入外存，当运行时再移入内存（可以提升内存利用率）
- 挂起后不能立即执行，等待事件与挂起条件都满足后才可执行。

七状态模型


进程在操作系统中的地位


如何实现OS内的进程
- 进程 = 程序 + 数据 + 进程控制块 → 进程映像 = 进程控制块
　{数据块, 用户堆栈} 数据
　PCB（灵魂）
　PCB块：最重要最重要的块
　（类似的有 TCB、FCB等控制块）
　　　　　　线程　文件
- PCB (Process Control Block)
　是系统进程的外部特征，描述其变化过程
　是OS感知Process的唯一方式，是进程存在的唯一标识
　进程Id（进程名、进程号）
　　State
　　Control Info → 调度信息：进程Name, Id；存储Info；优先级；当前状态；资源清单；消息队列指针；进程队列指针；当前打开文件
　　User Stack
　　⋮
　现场信息
　　可能被其他进程改变的
　　寄存器状态变化时，需要把它们寄存器
　　（用于保存和恢复现场）
　进程上下文（用于切换进程）
　　用户级上下文 (User-level context)
　　系统级 (System)
　　寄存器上下文 (Register)
- 进程的 创建 / 终止 / 控制 = PCB的创建 / PCB的回收 / 通过对PCB的组织管理实现

- 两个进程的状态转换

每次进程切换，都要把上下文写入PCB

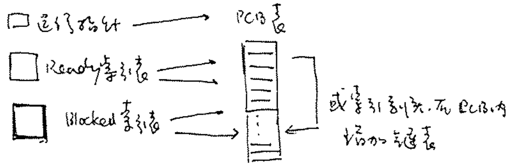
把PCB的上下文写入内存执行
Register.

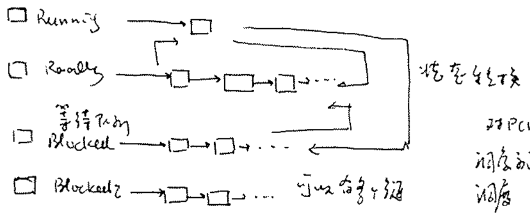
- PCB的组织

线性：简单，无需额外开销

索引方式：

(同多状多)





对PCB的管理，
调度就是进程的管理
调度

- 进程控制（对进程在生命周期的状态转换的控制）

原语：连续的指令列
{ 是操作系统的核心
连续不可分割（不可中断）被系统进程调用 —被调用→ 系统调用
必须在管态下运行，常驻内存 （可中断，被用户调用）

进程控制原语
{ 创建原语
撤销原语（必须先撤销子孙进程）
阻塞原语（运行队列 → Blocked队列）
唤醒原语（Blocked队列 → Ready队列）

3. 进程间的相互作用

{ 引入
同步与互斥
P.V操作
同步的实例
管程

引入进程的目的 → 并发、多道程序
→ 提高资源利用率

问题
{ 进程的独立 ⟶ 多进程访问相同的资源
进程的异步

无关进程：没有共享变量，互不影响
{ 相关进程：逻辑上依赖的并发进程
→ 可能引发问题

① 2个进程

ProcA	ProcB 异步
n=0	print(n)
n=n+1	n=0

② 飞机场航班售票
～一票多卖的情况

③ 多个进程

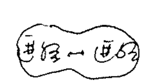
get copy put
f s t g

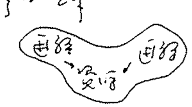
有严格的顺序.

相关概念：互斥，死锁，饥饿 —一个进程一直得不到资源.
争用同个资源 多进程互不相让
如解决，不用担心争用问题

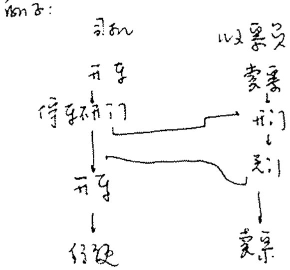
进程竞争（争用资源）→ 导致 { 死锁(deadlock) → 互相依赖
饥饿(starvation) → 总是因为优先级 } → 互斥

进程的协作：协同点
进程的同步：协调合作进程 } → 同步





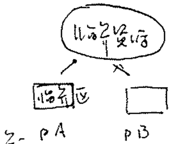
例子：



互斥：

临界资源：一次只允许一个进程访问的资源

临界区：分散在多个程序中，使用同一资源的代码



实现互斥访问的四条件（共享）
{ 任何两个进程不能同时进入临界区
不事先假定CPU的个数、运行速度
一个程序在临界区外运行时，不能阻碍其他进程进入临界区
一个程序想进入临界区，必须在有限时间内得到满足

实现原则 ←
有空让进
无空等待
多中选一
有限等待（避免饿死，死锁）
让权等待

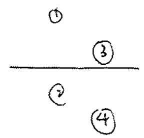
实现方式 ←
硬件，软件
（屏蔽中断，不太合适）

解决互斥问题的方案:

例子: 喂宠问题

方案1:

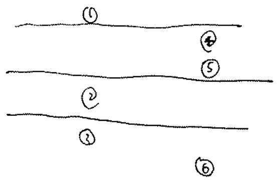
```
     Proc A (Joey)              Proc B (Tom)
①    if (no feed){           ③ if (no feed){
②        feed;               ④      feed;
     }                          }
```



]→ 存在问题

方案2:

```
①  if (no feed){              ④ if (no feed){
②      leave note;            ⑤      leave note;
③      feed;                  ⑥      feed;
   }                             }
```



] 存在问题

方案3:

```
leave note;                      leave note;
while (note Tom){
    (doing nothing)              if (no note Joey)
}                                      if (no feed)
if (no feed) feed;                         feed
Remove Note;                     Remove note;
```

- Joey 浪费时间循环.
- 程序不对称. 实现困难. 对于多个程序的正确性.
- 优先级倒转

方案4: lock

```
lock();  ~ 事实上是包含语句的原子;若锁已存在,等待其被释放,然后锁;若开锁,打开时互斥关闭
if (no feed)                     lock()
    feed                         if (no feed)
                                     feed
Unlock();                        Unlock();
```

(闭锁)

上节的锁仅具备的状态:

- 初始时打开;
- 进入临界区闭锁;
- 退出临界区开锁; ~ 每个锁都存在哪些问题
- 若发现闭锁而不停,就等待;

↓

都基于忙等待, 浪费资源

解决办法是P.V操作. 即有资源时发送通知 ~ 《浮尔实操作系统》

信号量与 P.V操作

同步: 并发程序之间在执行顺序上的相互制约关系.

P.V操作 (1965. Dijkstra 提出. P = proberen = test
V = increment = verhogen)

优点: 高效的方法.

缺点: 复杂. 不直观

引入类型的变量, semaphore (信号量) { value: int (init ≥0 初始 → 只是初始化一次; <0 表示没有空闲资源) Queue: Pointers to PCB ~ 只能作 P.V操作

P操作与V操作 (原语. 不能打断)

P操作: S.value = S.value - 1 (申请一个资源)
If S.value < 0 then:
进程等待 (阻塞)
该进程插入PCB的S信号量等待队列末尾. 等待其他的Proc的V操作

V操作: S.value = S.value + 1
If S.value ≤ 0 then:
释放队列中的一个进程, 插入ready状态者
执行本操作的进程继续执行.

讨论: S>0 表示S个资源可用
S=0 表示无资源可用
S<0 表示等待进程个数为 |S|

P(S) 申请资源
V(S) 释放资源.
信号量初值应 ≥0.

] 有P则必须有V操作.

- 若为互斥操作时. 同处于一个进程 ~ S仅仅相当于一个标志量
- 同步操作时. 处于不同进程 ~ 即所剩余资源数, 而在同步进程交替作用
- 若 P(S1) P(S2) 在一起. 则顺序至关重要. 同步信号的P操作在互斥P操作之前 ← 分开
- 两个V操作无关紧要 (只影响效率)

若多了几次mutex的检查操作, 问题不大

用 P.V 操作实现互斥：A.B 进入临界区

PA	PB
p(S)	p(S)
临界区操作	临界区操作
v(S)	v(S)

S 的初值为 1

PV 要成对出现，遗漏 P 不能保证互斥，遗漏 V 不能在使用资源后释放给其他进程

同步：设两个 S_1，S_2 → 缓冲区信息是否被取走

存一个缓冲区 初值均为 0

是否有信息

PA	PB
repeat	repeat
	…
送入缓冲区；	p(S1)；
v(S1)；	取缓冲区；
p(S2)；	v(S2)；
until false	until false

通过描述：用信号量描述前驱关系

P_1	P_2
c_1	$p(S_{12})$
↓	↓
$v(S_{12})$ →	c_2

解题步骤：

分析问题（有哪些进程，进程之间的关系？ 同步、互斥）

写出信号量（是信号量的 p.v 要成对使用，初值，是否要辅助变量？）

写出伪代码描述（避免饥饿、死锁）

└ 注意多个 p 操作的问题

举例：

repeat	repeat
p(empty)	p(full)
生产	取缓冲
送入缓冲	消费
v(full)	v(empty)
until false	until false

"生产—消费模型"

举例：三进程的并发读写

Proc1 get：f (in) → S (buffer)

Proc2 Copy：S (buffer) → T (buffer)

Proc3 put：T (buffer) → g (Out)

每次循环，次序必须保证 buffer 取完后，才能移入

使用信号量 $s_1=1$（buffer1 是否为空），$s_2=0$（buffer1 是否为满），$s_3=0$（buffer2 是否为满），$s_4=1$（buffer2 是否为空）

Proc1	Proc2	Proc3
repeat	repeat	repeat
p(s1)	p(s2)	p(s3)
get to buffer1	p(s4)	put to out
v(s2)	copy to buffer2	v(s4)
until false	v(s1)	until false
	v(s3)	
	until false	

举例：多个生产者、多个消费者问题

环形缓冲池：大小为 K（可放 K 个产品）

生产者放一个产品入缓冲池，消费者取一个

（非满）　　（非空）

互斥：每且仅有一个生产者放入或有且仅有一个消费者取

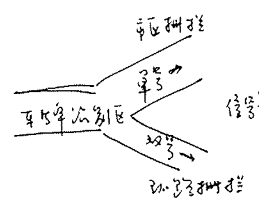
同步和互斥问题如何考虑？

$P_1 \cdots P_n$	$Q_1 \cdots Q_n$
i=0	j=0
repeat	repeat
p(empty) → 同步操作	p(full)
p(mutex) → 互斥操作	p(mutex)
放入 i	取出 j
i:=(i+1) mod k	j:=(j+1) mod K
v(mutex)	v(mutex)
v(full)	v(empty)
	消费产品
until false	until false

→ 能否交换两个 P 的次序？（不能！）

假设 $Q_1 \sim Q_n$ 的 p(full) 与 p(mutex) 互换，则 Q 可能停在 p(full)，而 P 停在 p(mutex) 死锁。

举例：道路单双号管制问题



车牌单双分别区只能有一个车。

信号量 { check=1. 可否通过检测；odd 车号为奇；even 车号是否为偶 }

车到

p(check)

通过检测

if n 为奇

v(odd)

else v(even)

市区 p(odd) v(check)

郊区 p(even) v(check)

案例5：读者、写者问题

同一时间：不允许多写，一读一写，允许多读。

读者优先：必须所有读者读完才能写

读者

```
Repeat
  p(mutex)
  readcount +=1        临界区
  if readcount =1
     p(w)
     v(mutex)
      读
     p(mutex);
  readcount -=1
     if readcount =0
        v(w)
        v(mutex)
until false
```

写者

```
Repeat
   p(w)
    写
   v(w)
until false
```

写者优先（更难情况 不要）

总结：P.V 原语和信号量

P.V 的使用可以归纳为三种情况，目的是为了解决多线程的使用。

- 实际为“锁”，允许加锁
- 可以为剩余资源数的计数
- 可以为同步用的

作业：用P.V操作实现售票员司机问题、理发师问题、集装箱问题、第二类读者写者问题（写者等待时优先 更加复杂）（扩展到辆号的识别问题）

操作系统习题课（二）

20181029 赵俊峰

人数 = 105

中断的调用过程（非常重要，因为操作系统是中断驱动的）

程序 → 中断响应 → 程序

↙ ↗ 中断处理

PSW 改变，由中断向量指向中断处理程序，恢复并返回

保护现场

中断与异常

自陷中断与硬中断。

进程 ～ 信号量与P.V操作（semaphore）

是操作系统管理的有效机制

有Block和唤醒机制，避免循环等待。

semaphore { value；队列：是指向PCB的指针队列 }

题1. 售票员司机问题 < 先关门后开车；先停车后开门

司机

```
Repeat
  P(S_Door)   =0
  启动
  驾车
  停车
  V(S_Stop)
Until false
```

车票售票员

```
Repeat
  关门
  V(S_Door)   设为1
  售票
  P(S-Stop)   =0
  开门
Until false
```

题2. 理发师问题 < N把椅子，理发师在有顾客时剪发，否则睡觉；顾客坐椅子上等，唤醒理发师

注意：临界资源是空闲椅子数N。

各类指针要规范

理发师叫醒顾客，顾客也要叫醒理发师，都需要semaphore.

问题要先排好, 理发

等待人数
若一个顾客, 叫醒理发师
或等待叫
理发
离开

判断有没有空椅
唤醒
叫人
理发

waiting ⇒ 等人数, 和 S_mutex

S_customer, S_barber, 两个同步量, 就分成两个信号量

>0, 没人等　　有空, 可理发

题3. 物流管理问题

入货和出货 都要同步

两个生产者, 两个消费者问题: 也有场地看作为 site

总共四个进程, site 为四个进程共享, 既有互斥作用, 也有同步作用
(初值为1)

题4. 第二类读者与写者问题 (写者优先)

{ 共享读
写互斥, 读写互斥
唤醒时优先考虑写者 }

不断地等待;
如何要有等待队列计数
不要有死锁问题;

题5. 猴子过桥问题

需要计数 CA, CB 之间

另外, 需要一个互斥锁 mutex, 只有计数为0后才释放.

题6: 多个读者多个写者问题, 但与之类似

不同的读者读不同的东西
或不同的消费者消费不同的东西

进程管理: 管程和进程通讯
└ 只知道概念既可,
与进程有什么区别? 实现什么东西?

信号量同步的优点: flexible, 强大

缺点: 同步操作分散, 有死锁 (P.V 重叠, 次序颠倒, 遗漏) 等问题
易读性差.
不利于修改和维护. 正确性难以保证.
└ 各部分都要检查.

- 1974年 Hoare and Hanson 提出管程, 把信号量与原语 装在一个对象内
  └ 关于共享资源的数据和操作过程, 集中在一个模块内
  是语言级的东西, 由编译器提供. (如 Java 的 Synchronized thread)

相当于一个"秘书", 将管理和控制条件 (正确性由编译器保证), 易读性提升.

- 管程的特点 { 局部数据变量
  抽象的数据类型
  信息封装 }

有共享性, 安全性, 互斥性

和进程的异同
管程是公用的数据结构　进程的私有的
是为了并发性而引入的, 是并行的实体
是为共享而建立
被进程调用
由程序员创建和编译器负责创建和消亡

管程中有多个进程进入 { 限制管程的互斥数
使其中一个进程等待, 或退出等. }

有互斥的入口等待队列.
有紧急等待队列, 优先级高于入口等待队列.
有条件变量, 可以 wait, signal. (条件变量不等于数值)
　　wait: 让其他进程进入　　signal: 唤醒

- 基于管程的生产者消费者问题

好处是: 几乎完全屏蔽了 P.V 操作, 使其封装在管程里面

进程通信

↳ 是进程间的直接相互作用

同步可以是一种通讯（但只传递少量数据）

通信会有大量的信息交换（IPC，Interprocess Communication）

- 低级通信：只传递状态和整数值
- 高级通信：可以传递任意量的数据。共享存储区 …
  - 有通信链路
  - 有数据格式
  - 有收发的同步机制

- 直接通信：e.g. 管道
- 间接通信：e.g. 发送到缓存区

· 进程间的管道通信 → 是连接两个进程间的一个打开的共享文件，用于进程间的数据通讯。
（通过文件 inode 文件系统进行读写 — 生产者消费者模型
双方必须要知道对方存在，一方退出时，必须通知另一方。）

- 单向传输，一方发送，一方接收
- 通过共享的存储空间（可以是外存，可以是内存）来传输，最早是利用文件系统

缺点：单向，速度慢，不方便（无名或有名管道）

消息队列 必须在内存

UNIX中，存在无名管道 write，read
{ 有名管道

· "信号" 通信机制

类似于电报，发送，但是对方要立即阅读（不需要事先建立连接）

↳ 使用管道，连接字不划算

信号是一种"软中断"，传递短消息，通知进程某个异常事件发生。

e.g. Ctrl+C 中断

内核向 proc 发送错误（e.g. 溢出溢出）

kill 给另一个进程发送信号。Linux 有64个信号。…

信号有
- 数据结构
- 产生与发送

信号的检测与处理。

～ 每个进程的 task-struct 存放接收到的信号。
（相当于 CPU 中断，消息传递存在）
（有多个不同信号，每次处理优先级最高的）
（sigaction() 设置信号处理机制）

中断与信号：概念一致 → 软中断
都是异步的（在异常处理程序或时间片某程处理）

硬中断（外部设备，或执行指令不正确）

· 信号量的通讯机制 — 进程通信（如 P.V 操作）

只传递状态

· 共享存储通信机制

管理复杂，不安全。
但最简单

· 消息传递机制 — 信件发送

无固定的读写进程
只在内存中实现 } ⇒ 与管道不同

利用公共缓存区实现

send() → 将信息复制到 buffer → 有锁
自陷时中断。 copy。
挂载到接收进程的消息链。
Receive()

发送原语 Send() 和接收原语 Receive()
（等待有空再发送）
↳ 会等待直到收到
Linux有相应原语（为什么不用 MPI）

· 信箱通讯

· 套接字（Socket），是用户与客户服务器传信的机制。
"出词"
UNIX BSD 首创，windows 有 winsock
是 TCP/IP 的基本构件
可双向传递 { 创立 socket，在客户和服务器建立点对点的通讯
e.g. windows 剪切板。

进程管理 { 进程调度（时机、算法） / 线程 / ...

2018.11.12　　操作系统概论　赵俊峰教授　PKU 3教105

进程的调度

- 进程调度 → 决定处理机时间分配的关键　负责3件事：选择、选择好后执行

　单道批处理 → easy，按序执行

　多道批处理/分时系统 → 复杂

　调度分为三个层次 { 高级 / 中级 / 低级

　— 作业调度/宏观调度（在作业系统介绍），时间尺度（分钟、小时、天），决定哪个作业进入内存

　控制多道程序的道数（一次调入一个或多个作业，以保证CPU的利用率）

　对于分时系统，则控制终端是否接入

（在本节详介绍）低级调度：直接针对CPU资源，从Ready队列调入CPU执行，时间尺度（毫秒），直接影响系统的性能（核心代码常驻内存，要求Robust & 高效）

中级作业（中程）：在内、外存之间交换（Block-Suspend Swap），作用是平衡负载，决定进程数

- 进程的行为 { CPU-bound（CPU密集）— 计算Big Bang的程序 / I/O-bound（IO密集）— 人机交互、游戏 / 平衡型程序

　← 与外部因素（工作量）/ 与外部因素有关（工作时间）

　好的调度程序：能同时掌握 IO-bound & CPU-bound 的程序

何时调度：新进程创建 / 进程完成 / 进程被block / 时间中断 / 被其他因素中断 —→ 进程的上下文切换：保存现场寄存器状态 { 修改状态 / 使进程移到队列 / ... }

{ 不可抢占方式（时间片用完，或自己阻塞自己）（阻塞原语 ↙ 或PV阻塞 / 或IO阻塞）

　可抢占方式（Ready队列有更高优先级的task）

- 调度算法的类别 ～ 交互式、实时系统、批处理系统的要求不同 ～ 一个机器有不同的调度算法组合（系统）

　为保证用户体验 / 使用可抢占方式；一次性抢占；不可抢占或长时间才抢占

(执行) 调度算法的目标

| 公平：不同进程无差别
| 均衡：使系统各部分都忙起来（IO/CPU） → 性能指标：极小化响应时间 / 极大化吞吐 / 保障各部分忙、公平

对不同类型的算法，要求要高

- 调度算法的评价指标

　吞吐量（throughput）　是否存在系统的设计目标？

　周转时间（执行 + block + 等待时间）

　CPU利用率

　(交互) 响应时间（交互或实时系统），等待时间

　确定性

　可预测性（实时多媒体系统：音频播放必须连续，在给定时间内完成）

- 就绪队列的管理

　优先级的大小决定进程的顺序，从队首选择。...

　① 先来先服务算法（FCFS - first come first served / FIFO - first in first out）

　　先进先出

　　不可抢占

　② 时间片轮转算法

　　时间片完了后，放到Ready末尾（时间片设置(固定)不能太大，否则执行完一个程序；合适：不能太小，否则切换开销太大）

　　{ 固定时间片（均分）/ 动态时间片（与优先级有关）

　③ 基于优先级的调度（HPF）

　　{ 静态优先级 — 先定优先级，后不能改变 / 动态优先级 ～ 等待时间长，则提高优先级 } ⇒ 有可抢占的 / 有不可抢占的

　④ 多队列反馈调度算法

　　不同级别，时间片不同；（高级别的时间片短，低级别的时间片长 — 更公平）

　　不同级别，优先级不同；（同级别按先来先服务；高级别执行完，低级别才能执行）

　⑤ 保证调度 — 每进程公平

⑥ 彩票调度：发彩票～抽奖。
高优先级则多发彩票

⑦ 用户公平调度：按用户使用时间，进程多的用户时间片就短

线程

· 进程的缺陷：只能干一件事。

一个进程阻塞，则整个任务阻塞（不能提高系统效率，多加设备计算机的效率）

80年代，只有进程概念
80年代中期：提出更小的独立运行单位 → thread

举例：MP3播放软件 { 从音频文件读取 / 解压 / 播放

单进程时：Read() Decompress() play() | 不能连续

多进程时：数据通信共享复杂，进程切换与创建的开销大

文件服务器：有多个服务请求，共享磁盘buffer，但查询是独立的。

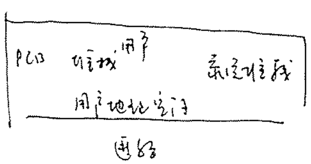
讨论：共同特性：多任务共享资源

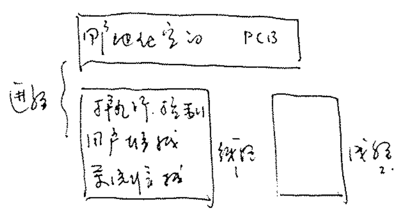
进程间切换的问题：→ 进程调度切换，进程的状态信息也非常多。
进程创建开销大，通信代价大，并发粒度（切换）粗，不适合并行计算和分布式并行计算的要求（大量通信，降低并行度）
也不适合C/S（客户/服务器）计算的要求
进程只能分开，大量计算

· 新的实体 → 线程（Thread）
{ 并发执行
共享地址空间

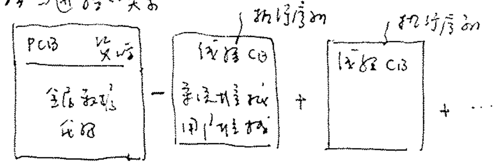
线程是同一进程，是CPU调度与分派的单位，自己基本不拥有资源

进程 — 每个进程虚拟地址空间
{ 创建 / 撤消 / 切换
{ 资源 / 有状态（优先级，调度）] 仅有一部分私有，不需要频繁切换，提取出给线程。





· 线程与进程的关系



{ 资源分配、保护的基本单位 ← 进程
地址空间、文件 ⇒ 更好的共享资源
CPU调度单位 ← 线程 ⇒ 减少并发创建、切换、撤销的开销

有不可少的资源（TCB、寄存器上下文、栈）
可共享进程的存储空间、资源，但不拥有资源

所以进程 = 线程 + 资源平台
（CPU调度）→ 执行信息（寄存器、上下文、执行栈、局部变量与静态存储区...）有多个线程

提供地址空间、资源、打开文件等（file、部分Process management）= 程序 + 数据 + PCB } 两个角度
父进程、PPID

每个线程有唯一标识符，线程控制表（TCB）
有并发、共享、动态、结构性

e.g. 大厦的工程队（进程）{ 浇地 / 修路 / ⋮ } 都是同时进行

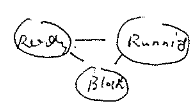
MS Word 中的多线程（IO、保存、显示...）

现代操作系统：多进程多线程

· 线程实现 ～ 线程控制块TCB
标志线程存在，保存线程全部信息

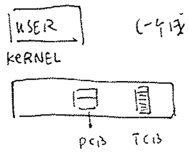
PCB存在于系统空间
而TCB可存在于用户空间或系统空间
（用户线程）（内核线程）
如果某资源不独享就会出错，就独享于线程

线程的基本状态



** 而挂起状态是否是线程独享的？

- 两类线程 { 内核线程 — 依赖于OS的内核，实现由内核完成



(一个线程发起系统调用而阻塞，不影响其他线程)
(时间片以线程为单位分配)

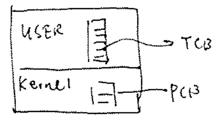
用户线程 → 不依赖OS核心

通过线程库来读写创建

用户态程序管理，完全在用户空间。

(一个线程若阻塞，整个进程就阻塞

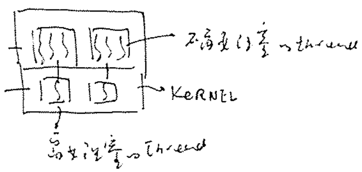
时间片以进程为单位分配，线程多不能多得时间片)



优点：线程切换不依赖于内核，完全在用户空间，因而快。(不需要切到管态)

缺点：但系统调用是阻塞的。

└ 解决方法：混合式线程



- 进程中线程的组织 { 调度者一员工
  组模式
  流水线模式

系统内核

常用核心 → 功能 { 中断处理
进程管理(队列管理)
存储、文件管理
时钟管理(读时钟、定时器)

外部的管理

特点(特权指令、中断屏蔽)

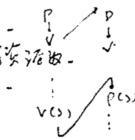
中断是调用核心功能的唯一方法

操作系统的习题课4.

P.V操作的一般步骤 { 进程有哪些？(确定有几个进程)
有同步、互斥关系吗？(确定进程间的关系)
如何设置Semaphore. (设置信号量)
如何使用P.V，避免死锁的现象 (用P.V来调整信号量)

P.V操作总结 { 信号量S = int + queue、初始值
P.V操作的实现与含义

P.V操作的应用：互斥 -
利用资源数 -
同步 -



存储管理

王建功

·背景:

程序的规模增速远超存储容量的增速

存储管理的重要性 { 任何程序，数据都要载入内存才能运行 / 存储管理的优劣直接影响系统的性能

(CPU直接存取 (与CPU交互))

· 存储体系 / 存储器的层次结构

Cache / 内存 / 外存(磁盘)

可以存储 — CPU存取

存储时间↓ 速度↑

存储成本↑ 存储容量↓

Intel { x86 CPU — 指令复杂

ARM CPU (简单) 没有cache 或很少 — 功耗低 [illegible]

OS的存储管理 { 记录空闲 / 分配和回收 / 交换

目的: 充分利用，使存储器保持……提高存取速度 (内存命中率高)

理想状态: 只有cache，没有内存与外存

实际状态: 就是不断地把数据从内存载入cache，保证命中率

CPU寄存

存储管理 ↓ 设备管理

存储器 = 内存 + 外存

由存储单元(字节或字)组成连续地址空间，存放指令与数据，由程序计数器PC取指令执行

{ 地址保护 ~ 防止地址越界(即非法访问存储区，不同地址空间之上、下界)
  地址转换 — 程序逻辑地址与物理地址的对应

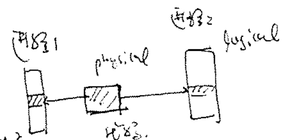
内存 { 系统区 (OS常驻内存的一部分)
       用户区 (分配给用户进程) ← 存储管理的对象

· 内存分配 { 静态 / 动态

内存共享: 进程共用某个内存段

代码段可以共享

数据需要考虑互斥共享



虚拟存储技术

★虚地址空间与内存大小没有关系 (swap文件)

与指针的宽度有关 2^{64}

64位系统和32位系统的最大差别是 虚地址空间的大小

历史沿革

— 虚地址空间是程序编写时的，认为无限

有进程的空间，64位系统就是 2^{64}；虚地址空间不一定映射到物理空间；只是程序的地址的潜力

— 物理空间被OS分割为段

物理空间

段式管理 { — 相同大小的段 segment / — / —

分区管理: 进程在物理空间连续(不允许进程无限伸)，一个进程占一个分区，分区大小固定，大小可以不等

页式管理: 进程在物理地址空间不连续 (提高内存使用效率)

如果一分区内产生内碎片，大小不等会产生外碎片？没有……

1M / 2M / 4M / 8M

— 虚拟存储技术的优点

虚拟地址空间与物理地址空间的映射

使得程序编写不受内存容量的限制

· 程序的处理 (汇编器) ~ 二进制目标代码

将"名字"放在虚地址空间

↓

链接 — 形成完整的装入模块

将"名字"转换为地址

↓

装入 — 将程序装入内存 = 构造PCB，形成进程

机器语言 ↓ 汇编 ↓ C语言 ↓ SQL语言

[illegible] 高级语言 ↓ 自然语言，用词不同，越来越抽象

替代解决方案 紧凑技术

分区算法:

最先匹配法(first-fit) — 从头，找第一个能用的

下次匹配法 — 从上一次匹配的位置，找第一个合适的

最佳匹配法 — 找大小最适合的，容易产生碎片

最坏匹配法 — 找最大的最不适合的 [illegible]

快速适应算法 — 将空闲区分为若干个链，找合适的

存储管理 { 分区管理
页式管理
段式管理
⋮

覆盖技术与交换技术

在较小的存储空间运行更大的程序
数据与程序较多无法存
当前需要执行的部分放在内存 > 内外存之间信息交换

早期：覆盖 — 存在的问题：不友好，需要了解内存管理细节（人机界面不好）。需要用户指定段之间的依赖
— 一个程序划分模块，相互调用的执行的模块划分开存；一段执行完调入另一段外存（调入时检查是否在内存，有什么要覆盖回去的）
— 如果部分常驻内存，可选部分模块外存，要自己依赖
小型分时系统：交换 → 导致虚拟技术的出现
— 内存空间紧张时，整个进程换出，整个进程换入（roll-in/roll-out）
多用户分时系统
— trick：选择问题：哪个进程换入换出？（从不被共享的进程）（按用户优先）
在哪交换？另外一个swap空间
换回时内存位置的改变：动态地址映射

优点：增加并发程度（多道数）
编程者必须考虑如何覆盖时间
编写程序不能自由结构

举例：C语言的汇编语言
只有地址，指令，没有类型；而C/C++抽象出函数/类的概念，更友好的人机界面

举例：进程的地址空间转换 — 进程之间用socket通信
虚地址空间都是0～0xffff...
物理地址空间不同
使得之间分开独立地址空间

对比覆盖与交换 → 不用手动分配

虚拟存储技术 { 局部性原理
基本思想
虚拟存储技术的种类
缺页中断异常
虚拟存储管理的性能

换出：程序大于内存
程序暂时不执行或运行完毕后立即退出内存

局部性原理：时间局部性与空间局部性（sinet一样）
(priciple of locality)
同一指令执行与数据访问都集中在短时间内
访问的与邻近的数据，指令都集中在一个小区域内

{ 程序运行前不需要全部装到内存
程序不必在内存空间连续

虚存：结合内存、外存，形成"大内存"，就是虚存

虚拟存储管理

虚拟页式 { 程序开始时，装入部分页
后面用到再加载页
内存不够时，换出一部分页

主存管理单元～MMU（硬件支持）
物理地址 ← 虚拟地址
内容 { 页表基址寄存器（访问页表）
— 页面可以是放在内存：页面很小，直接放内存；页面很大，存放到外存
存储单元地址（页面号+页内偏移）
管理快表TLB（在cache内）

虚拟段式
代码，数据等性质不同，每个（不可写）（可写）
可另一段，形成独立的地址空间
段式变成有问题，加锁也更容易
缺页检查
缺页中断处理：利用段是否可扩充，可扩充则扩充
否则引发段错误。

局部性原理保证
与之内存页表中查找时间的选择，
当出现常态，会出现找不到物理地址（TLB命中）
不命中才去查页表（页表命中）
如果页表中没有对应，就出现缺页中断，调页，装入，改表。
e.g. 一个指令swap(A,B) 如果swap指令在两个页表之间，A B 一个在外存，一个在内存，则要6次交换。

缺页中断处理：
查内存是否有空的空间，有则装入
无则置换
再则淘汰一些页

其余的内容以后看

虚拟段页式管理

有缺段与缺页中断.

虚拟存储管理的性能

调入策略
- 请求调页 (demand paging)
- 预调页 (prepaging)

清除策略

衡量置换算法的好坏 — 缺页中断率

与
- 主存页框数
- 页面大小 (大则中断少些, 但置换一次代价高)
- 页面置换算法
- 程序特性

有关

程序特性 ~ 程序编制方法不同

对缺页中断次数有很大影响.

程序的局部性要好

e.g. for j=1 to 128

for i=1 to 128

每次所访都引发缺页中断.

不断的换入换出页面

按行读取更好

可以在 Linux 系统上测试

但编译器可能会优化

置换算法 ([illegible])

OPT (最佳算法)

- FIFO
- 随机页面置换
- 最近最久未使用.
- ...

存储管理

虚拟存储管理

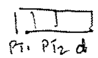
虚拟存储的置换算法

发生P. 调入页. 置换页.

缺页率 $f = \frac{F}{F+S}$（F：失败访问页面；S：成功访问页面）→ 影响因素 { 分配页框数；页面大小；页面置换算法；程序大小 }

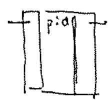
页面小：内碎片少，但页表增大，IO开销大

页表结构：多级页表（缺点：多级的寻址时间比单级的长页表寻址时间更长）

逻辑地址 

反置页表（IPT）（优点：解决虚拟地址空间太大，导致页表太大的问题）

按物理页面号组织页表

页帧号 →  → 逻辑页面号，特征位

可以用hash定位，有重复的产生碰撞链。每次访问也需要在页表中寻找

页面置换算法分类 { 全局算法：所有页面一视同仁；局部置换算法 }

随机页面置换算法 → 简单、低效（一般不采用）

最佳算法（难以实现）OPT → 将来最远的被置换

先进先出算法 → 性能较差

页框数增加时，缺页率可能增加

（Belady 现象）（问题：置换算法与访问内存的使用不适应）

先进先出的改进 → { 第二次机会算法；页面缓冲算法（page buffering）未修改的页面 / 已修改的页面 ⇒ 积累到一定数目后，一次性写出；优先被置换. }

最近最久未使用（LRU, Least-recent-used）

适用一般的算法，淘汰最长时间未被使用的。

记录使用时间：开销较大 { 链表法，使用时移到队首；淘汰队尾；栈法，使用时上到栈顶；淘汰栈底 }

最不常用算法（LFU, least frequently used）

淘汰访问次数最少的（特定时间间隔内）

记录使用次数的计数器；已发生置换时计数归零，同时周期性地清空重新计数

时钟算法（clock，时钟算法）

use bit（访问时设1）

缺页置换时，指针向前推进，将遇到的1都变为0，找第一个是0的。

（第二次机会页面，使用链表）置换（这里是用的指针，并且是循环查） } 共同缺点：速度不快！没有考虑使用效率

最近未使用算法（NRU, NOT Recently used）

结合LRU、FIFO，并结合引用位 r 与修改位 m

扫描循环队列，第一轮找 m=0 & r=0 页面 ← Macintosh 采用此策略；优点：选择很优；缺点：大量扫描

若没有找，r=0 & m=1；扫描时找r都置0。

若没有，找 r=0 & m=0

- 分配，还要提高
- 分配，进程数太多后，会发生抖动现象，使系统全崩溃；导致CPU使用率骤降

负载控制：e.g. 基于时钟算法，若指针转动太快，则表明系统负载过高

常驻集和工作集策略 ~ 给每个进程分多少页面，以及如何动态调整

前面的算法都是基于局部性原理

常驻集：分配给进程的物理页帧数 { 太大：对缺页率下降效果不明显；太小：缺页多，但并发度也高

- 大小的确定：固定分配：平均/比例分配，但固定
  - 可变分配：根据缺页率动态调整
- 置换范围：局部/全局

固定分配+局部置换：初始分配数就决定缺页率

可变分配+局部置换：缺页率高，多分一点

可变分配+全局置换：每个进程有预分的页面数，OS也有空闲页。
缺页时，先从OS空闲区拿，再从其他进程拿

工作集：1968年Denning提出，目的是根据进程访问页面来调整常驻集的大小。

工作集 = 在某段时间间隔内，进程实际访问的页面集合。
W(t, Δ)　　Δ ← 工作集窗口

working set strategy
工作集策略：对存储空间的管理 ~ 要保证常驻集总是包含工作集 → 保证了工作集，高出大量的开销

工作集太大（Δ太大），缺页少，并发度低
Δ太小，抖动严重，也降低吞吐量 } → Δ很关键

工作集是进程运行的动态特征

缺页率算法：缺页间隔 t < F（阈值）=> 增加常驻集
　　　　　　　　　　　> F => 减小常驻集

问题：在locality过渡，常驻集会膨胀。
不用页面在F后才淘汰

工作集页面置换算法：记录页面最后使用时间 t，若 $t_{curr} - T > t$ 则不在工作集内
要替换页面时，访问位是1，清零
访问位是0，不在工作集内，淘汰
在工作集内，则在扫过的页面中找最早的淘汰

工作集时钟页面置换算法

不仅查看访问位、使用时间，还要修改位。

存储管理实例：

WINDOWS NT
- 页面4KB (2^{12})
- 地址空间 2^{32}，4GB
  - 系统存储区，2GB { 固定页面区 / 页交换区 / 直接映射区
  - 用户区
- 地址转换机制：二级页表 [页目录，页表号，页内偏移]
- 页面调度策略：取页策略/提前取页
  - 淘汰法：局部的FIFO
  - 工作集策略：Δ在进程创建时确定；内存页面不大时，虚存管理程序可以动态调整常驻集

文件管理 { 目录与实现 / 分配与实现 / 管理与保护 / 硬盘和闪块的存储

1 概述

文件是用来信息存储与检索(访问)的

数据 信息 { 组织 — 数据结构 / 处理 — 数据库 / 存储 — 文件 / 传输 — 网络 } → { 结构 / 命名 (e.g. 长文件名) / 存取 / 保护 / ⋮ }

各级使信息持久地存在于disk

对用户屏蔽细节，提供统一的接口

支持高效的数据控制与保护

支持高效地提高效率

作业：用语言来实现文件系统的管理，模拟出一个小型文件系统

基本概念：文件 = 文件名 + 数据项

= 文件体 + 文件说明 (内部标识、文件名、存储地址)

各信息以次序排列，有读指针与写指针.

UNIX的文件 { 普通文件 / 目录文件 / 特殊文件(设备) { 块设备 / 字符设备

2 文件的结构

{ 逻辑结构 / 物理结构 } 映射机制. / 文件的存储方式 { 连续的 / 随机的

文件系统的功能 { 文件操作 / … / … / …

文件系统的modules.

对象及其属性 (底层)

操纵和管理软件 (核心部分)

文件系统的接口.

• 文件控制块(FCB, File Control Block). 存储文件的所有管理信息 { 类型、长度 / 所有者、访问权限.

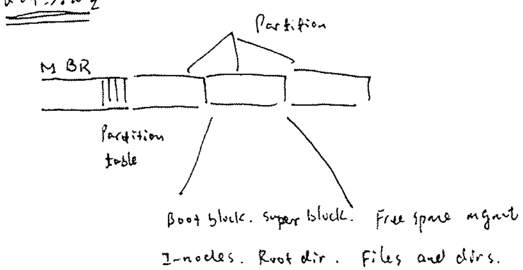
FCB在 在Unix中 称为 I-node (i节点)

• 文件的逻辑结构是字节流 → 也可以有结构(定长记录，或可变长记录) ← 可以实现更多处理 / 目录树 — 逻辑结构 / 字节流

{ 复杂结构 (树形结构)

现在主流的OS都是无结构的字节流，只对应的解释方式不同而对应不同结构

文件系统是将其保存在磁盘上 → 文件的存储



主板上的BIOS启动，读入MBR，找到主分区的分区表.

找到活动分区的Boot block 并读入内存.

连续 无结构文件 → 字节流 stream

如何 随机 / 顺序 / 按key 访问.

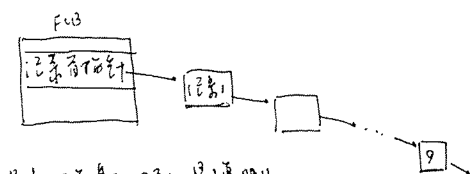
• 物理结构

簇(物理块的集合)，是文件的基本物理单位

连续结构 { 简单、易实现 / IO速度快，寻道时间/次数很少 } ⇒ 现在不常用 e.g. 只读CD等可以用

缺点是：不利于增删

链接结构



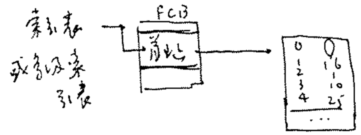
提高了磁盘利用率，易增删 (无碎片)

缺点是：寻道时间很长，降低了效率

可靠性更不好

访问速度更慢

索引结构



磁盘寻道更慢

优点是可随机访问 缺点占空间

WIN — FAT表 (File Allocation Table)

链与数据分离，FAT表保存链信息，可以存在内存中方便更快的索引

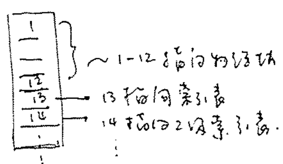
是WINDOWS下常用的文件格式

保持了链的优点

也加快了IO速度

UNIX

i节点的方法： i节点是OS管理文件的FCB



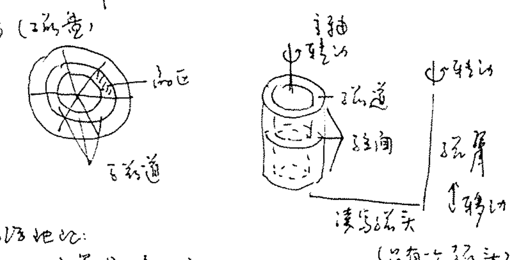
将大文件与小文件统一起来

文件存储介质

要求：大容量，快速度

顺序存储设备 (e.g. 磁带)

随机存储设备 (磁盘)



物理地址：

磁头号， 磁道号，扇区号.

柱面号.

一扇区长度 4KB

时间 = 寻道 + 旋转 + 数据传输

3 目录管理

目录 directory / 文件夹 folder { 组织 / 命名 / 分组功能

区别

文件目录 (FCB的集合)

{ 目录项就是FCB
是文件名到物理位置的映射

目录文件：是文件目录的存在形式，保存在外存

文件控制块 { 基本信息 (名，物理位置，第一块号) 有一些用户可获取的
存取控制信息 有一些是透明的
使用信息 (创建时间，日期)

索引结点：将文件名与文件信息分开，避免一次调入过多

磁盘索引结点：每个文件唯一

内存索引结点：open时把磁盘索引结点读入.

但还有更多信息 { 状态 / 访问计数 / ⋮

目录的组织方式 (结构)

一级目录：线性结构 { 检索时间长 / 重名问题 (单用户使用) / 不能分组 / 系统启动时一次调入索引表

二级目录：根目录 → 用户目录 → 线性结构 { 解决重名 / 但仍不具备统一分组能力

多级目录 (树形目录) tree-like

适合大型的文件系统

目录名，目录树，路径，...

绝对路径名，当前路径名，相对路径名

{ 层次清楚 / 不重名 / 查找速度快

改进的多级目录

符号目录项 (符号名，对应物理地址) ~ 载入内存

基本目录项 (内容，找其他部分) ~ 需要时载入

文件目录检索

文件查找 (与物理/逻辑结构有关)

目录项：直接法（MS-DOS），文件名+FCB
（条目）间接法（Unix，inode），文件名+FCB地址

长文件名问题：固定255字符→浪费空间

- 可变长的文件名长度→空间长度不确定，不紧凑（或太小）
- 目录项中的长度是固定的，给出指向可变长部分名字的指针
  （可变与不变的结合）

目录的搜索方法：
- 线性搜索（大目录时，时间太长）
- Hash表（将文件名映射为0~N-1的值）
  - 但需解决冲突
  - 新的通配符方法的Hash，只能线性检查

4. 文件系统的实现

如何管理磁盘空间（分配、回收）

创建文件时的存储空间 {静态分配 / 动态分配

磁盘分区→文件卷→格式化（文件系统的建立与初始化）

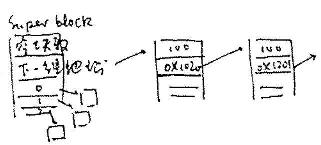
扩展文件卷　　IDE、SATA }硬盘类型

磁盘交叉存储（一个卷对应多个磁盘）

文件存储单位：簇（cluster）
- 由若干连续的扇区组成，太大太小都不好
- 簇大小一般不可变（有些也有可变的）

文件存储分配
- 连续分配
- 链接分配
- 索引分配

空闲空间管理
- 位示图
- 空闲块表（适合连续分配的）
- 空闲块链表（存储在空闲块中的）
- 空闲空间索引法（比空闲块链表高效）
- 成组链接（UNIX的方法）



大型的文件系统都适用
但也需要算法

实现文件系统的表目
- 系统打开文件表：放在内存，共享计数等→可以共享
- 用户打开文件表：每个进程一个表 {fd，打开方式，读写ptr，系统表的入口 ～私有信息

记录的成组与分解
- 减少对设备的启动次数，提高存取速度
- 读写时加入buffer

文件的操作
- create：建立fcb
- open：把fcb读入内存，与系统表关联，计数+1，返回fd
- read、write：读、写，seek

实现文件操作有关的数据结构
- 外存 {目录结构，fcb
- 内存 {系统打开文件表，进程（用户）打开文件表

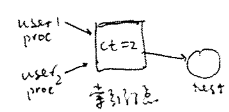
open（文件名，打开方式）系统调用
- 目录查找
- 检验访问合法性
- 查找系统表或载入系统表，共享计数+1
- 载入进程表，存指针，访问方式
- 返回一个fd，指向进程表的表项

close 系统调用
- 删除进程表项
- 系统表项减1，可能要删除（计数）

文件的保护和安全
- 文件可以共享 {多用户→权限，连接共享；多进程→共享或私有读写指针
- 时间上 {可以同时读→读者写者问题；不可以同时读
  - 文件锁 lock
- e.g. Win2000的登录和注销，finger user@…，UNIX的漏洞

（UNIX使用）（1）不同目录表项，指向同一个i结点。
父子进程共享系统表，指向同一i结点。

基于索引结点的共享方式



索引结点

利用符号链接实现文件共享
- 创建一个link的文件，只包含原文件的名字路径
- …
- 链接，只有一个属主，可以对文件进行直接删除
- 软链接

文件的保护方式
- 建立副本（简单）
- 定时转储
- 规定文件的存取权限 ～
  - 访问权限可以附在文件上
    - 访问控制表
  - 可以附在用户上
    - 用户权力表
  - 二级存取控制（UNIX）
    - owner / group / other + rwx　共需要9个位组成控制矩阵
- 隐藏、口令、密码

文件系统的实例

MS DOS的FAT文件系统
- 文件卷信息
- 目录信息
- 打开文件管理

WIN2000的文件系统NTFS
- NTFS引入卷集：多分区构成的单卷
  条带卷、镜像卷

UNIX
- 可执行和目标的文件

设备管理　　设备管理软件是分层管理的

1. 概述

设备 = 除CPU、内存外的所有设备

{ I/O设备是计算机系统
  是操作系统庞大的原因
  与其他系统、特别是文件系统关联

设备的类型（块、字符）
　　　　存储　交互式

设备管理的目标：{ 设备独立（接口一致）
　　　　　　　　　设备保护

任务：缓冲、接口、故障、设备与信息保护

2. IO硬件

设备管理结构

User Process

Logical Io（统一接口，屏蔽底层细节）
↓
Device Io（对用户指令转为硬件指令）
↓
调度、控制（Device Driver）

Hardware

IO硬件的组成
　　低级接口（连接与通信）
设备 ← 设备控制器
　　（适配器）
　　　在控制器上，有Register，借此与CPU交互。（控制、状态、数据）写入；独立端口、中断

设备接口：USB-2、HDMI等

CPU如何控制设备寄存器、读写与buffer通信

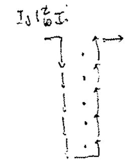
方法1. 给每个设备寄存器分配一个端口编号，用专门I/O指令操作 ⇒ 完全独立
方法2. 内存映像编址：
　　端口地址与内存编址统一；利用专门的I/O指令读写
方法3. 混合编址：部分独立（寄存器和其他的设备分区不同）
　　　　部分编入内存（寄存器不用）　} ⇒ 显主要的方式

3. IO软件 — 直接决定I/O管理的功能
　　　　　也是通用性（设备独立性）

基本思想：分层构造
　　　　硬件
　　　　中断处理程序
底层硬件
　　　　设备驱动程序 — 与设备类型有关，由厂商提供
接口标准　　　　　　　对用户的逻辑I/O翻译为物理I/O
　　　　设备独立的系统soft
统一接口
　　　　用户空间的I/O软件
库函数

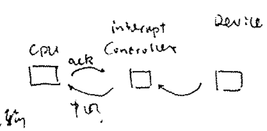
设备驱动程序的通用结构　　　　参数检查
（是个普通方法）　　　　　　　↓
　　　　　　　　　　　　　　转为物理参数
　　　　　　　　　　　　　　↓
　　　　　　　　　　　　　　查硬件是否空闲，否则加入等待队列
　　　　　　　　　　　　　　↓
　　　　　　　　　　　　　　向设备控制器发送指令
　　　　　　　　　　　　　　↓
　　　　　　　　　　　　　　检查出错情况

设备独立的系统soft { 统一的接口（统一命名，e.g. UNIX /dev/ttyoo 即多一个特殊文件的 inode。即主设备号、次设备号（硬件）（驱动）
　　　　　　　　　　统一的数据块大小　拼接或分割
　　　　　　　　　　缓冲技术 → 块设备，大的pool；字符设备，小的buffer region
　　　　　　　　　　出错报告
　　　　　　　　　　独占设备的分配与释放



与I/O有关的技术 Spooling 缓冲井，输入与输出与CPU独立
{ …
缓冲技术（单、多、环形buffering）

4　I/O控制方式．早期 — 忙等待 pooling/busy waiting
　　　　　　　　　　　　轮询
　　　　　　　．现在 — 中断驱动，每次读一个字节
　　　　　　　　　　　使CPU浪费
　　　　　　　．DMA方式（60~70年代）
　　　　　　　　内存与I/O设备直接成块传输
　　　　　　　　完成时才引发中断　　（DMA控制器可直接访问总线）



DMA方式与中断的区别
└ 数据块传输完成才中断　└ 控制器的buffer较大

DMA的问题：总线竞争　　好处：高速的、大量的传输
数据输入问题

· 通道方式（IO处理机）
└ 独立于CPU的，由指令和通道程序构成的专用控制器
{ 选择通道
　多路通道

现在，单独的通道已经不存在，但融入了芯片中。

与DMA不同的是，执行程序来传输
可以多设备多路传输，更灵活。

设备
5. I/O分配方式

设备控制表（DCT）～每个设备一张
系统设备表（SDT）～系统所有设备 { DCT指针 / 设备proc标识 / DCT的所有信息
控制器控制表（COCT）
通道控制表（CHCT）

独享设备、共享设备 ⟷ 静态、动态分配
↓ 可能死锁　↓ 安全死锁　↓ FCFS、基于优先级
P(sm)申请、使用、释放　设备的逐一申请、释放
控制器和通道同时分配　但要分配通路

6. 磁盘调度

扇区	标识 10byte 同步信息、标识	数据 512byte	尾区 ECC纠错、尾区

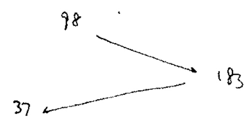
t_2 8.3ms　t_3 16.7ms

I/O时间：柱面定位 + 旋转延迟 + 数据传输
20ms t_1　　　　爆发速度　　持续速度

{ 若扇区是连续的，时间 $t_1 + (t_2 + t_3) \times n$
　若是随机的，时间 $(t_1 + t_2 + t_3) \times n$

对于随机分布的请求序列，可以有不同的算法来调度

FIFO算法.



来回寻道

优先级算法（不优化寻道时间）
后进先出（事务系统）
最短查找时间优先（SSTF）
扫描（SCAN）算法 → 电梯调度算法，优先沿一个方向进行SSTF
　　无饥饿（公平）
　　摇臂臂效应
循环扫描（C-SCAN）
N步扫描（N-step-SCAN）
双队列扫描（FSCAN）算法.

其他方法 { 磁盘高速缓存
　提前读
　延迟写
　优化物理布局
　虚拟盘.

7. 几种典型的I/O设备

8. 操作系统的例子

死锁

赵俊峰 zhaojf@pku.edu.cn 理科1号楼1542

属于进程管理的一部分

死锁的发生、检测与解决

多个进程陷入永久等待，无法运行

出于"资源竞争"目的，互相影响进程 (↗ 内存)

↳ 共享文件、I/O设备、CPU等

区别资源的类型和实例 (用于资源分配图)

↓ 完成任务

可分为可抢占的和不可抢占的。

原因是：每个进程都拥有部分资源但不释放，等待其他进程所持资源

e.g. PV操作的顺序，申请不同类资源

P(S1)	P(S2)
P(S2)	P(S1)

e.g. 页面请求时的死锁 — 同类资源 (内存区)

产生条件：互斥 + 请求保持 + 非剥夺 + 环路等待 → 不会同时发生

- 同时满足才能死锁。
- 破坏一个即打破死锁

互斥 → 不太好打破，通常与设备特性有关；方法：使用排队列来将独享设备模拟成共享设备

非剥夺 → 若被抢，则剥夺其他的资源

请求保持 → 若被抢，可释放自己的资源

应对策略：无为而治，假设不会发生死锁 (民用设备很常见)

- 死锁预防：破坏4个条件之一
- 死锁避免：精心设计资源分配流程
- 检测与解除

} 都有不小的代价

死锁预防与避免

破坏"互斥"：将互斥设备模拟为非互斥设备。e.g. 打印机等

破坏"非剥夺"：等待时，占用资源可以被全部剥夺。

破坏"请求和保持"：不允许进程占有资源时又申请资源，必须一次性申请。↳ 困难

破坏"环路等待"：有序资源使用法。

先申请小编号的，再申请大编号的 ⇒ 不可能形成环路，不可能死锁

Proc1	Proc2
P(S1)	P(S1)
P(S2)	P(S2)

e.g. 哲学家就餐问题：Dijkstra 提出

测试同步方法可能出现的试金石。见 'Modern Operating Systems'

思路1：多一根筷子，否则等两根；唤醒时注意唤醒左右邻居

思路2：有序资源

思路3：[illegible]

有没有可能完全避免死锁问题？算法 + 数据结构

判断系统状态是否安全，只保证安全状态通过

银行家算法

系统状态 { 不安全，不一定会死锁 { 可能死锁 / 可能不死锁 } ; 安全，一定不会死锁 }

银行家算法：可以避免出现死锁的结局，避免不安全状态 (Dijkstra 1965) ← 理论上重要，但不实用

分配不安全时，就不分配，并想办法腾出一部分资源；

要求进程事先声明最大所需资源，申请超过最大所需的资源，则拒绝

死锁的检测和解除 → 以最小代价恢复 { 选择牺牲进程 (以此或回退到之前保存点)；剥夺资源；要考虑代价计算 (高优先级？完成所需时间？不可剥夺资源？经常死锁的进程？) }

↓ 是否有循环等待：等待时检测或定时检测 (检测占有表和请求表是否成环)

UNIX，可以考虑运行时间，长时间不运行则 kill

解除方法：

剥夺与挂起（代价★★★）→ 最小花费资源剥夺

↑ 之后再恢复，避免饿死

撤消进程（代价★★★★★）～ UNIX可以直接杀死其他进程

↑ 尽可能选择以重新运行的进程

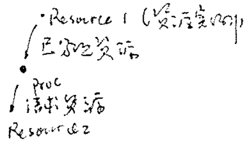
进程回退（代价★，但记录代价高，很少使用，非常复杂）

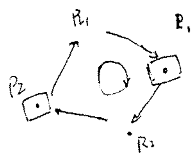
定期地记录进程状态到磁盘

（含内存映像和占用资源的状态）

当检测到死锁，按记录回退

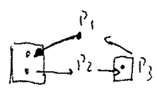
资源分配图：检测/解决死锁问题的一有力工具





成环则（可能）死锁（每类资源实例唯一时，成环 ⇔ 死锁

不唯一时 成环 ⇐ 死锁）



可能死锁时，尝试化简，若能消除，则不死锁

不能，死锁