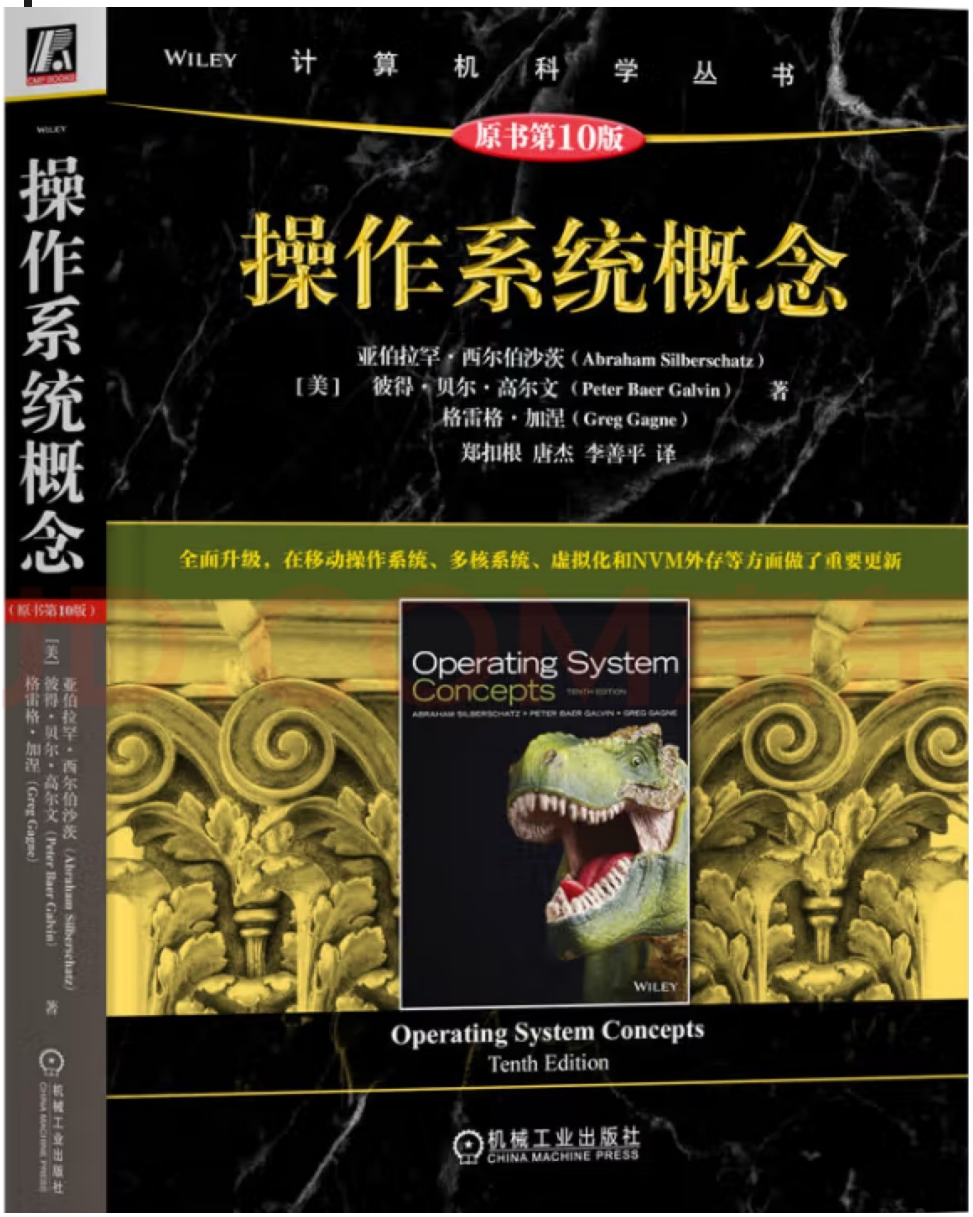
操作系统课程概述

- 操作系统课程内容
  - 操作系统的概念、演化历史和分类、操作系统的结构、内核功能及其实现原理、技术和方法
- 课程考试方式及成绩计算
  - 闭卷考试
  - 平时成绩40%
    - 课后作业10%、实验20%、期中考试10%
  - 期末考试成绩60%

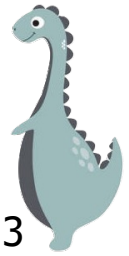
教材



- Abraham Silberschatz. Operating System Concepts (10th Edition), Wiley, 2018.
- 亚伯拉罕 希尔伯沙茨等著，郑扣根等译，操作系统概念（原书第10版），机械工业出版社，2023.9

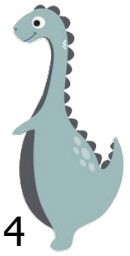
OS参考书

- Operating Systems: Three Easy Pieces https://pages.cs.wisc.edu/~remzi/OSTEP/
- 操作系统导论（Operating Systems: Three Easy Pieces），王海鹏译，人民邮电出版社，2019.6
- 《现代操作系统：原理与实现》，陈海波等，机械工业出版社，2024.3
- 《深入理解计算机系统（原书第3版）》， Randal E. Bryant etc.著，龚奕利，贺莲译. 机械工业出版社，2021.
- 《计算机操作系统》，郑鹏等，武汉大学出版社，2022.3。第3版
- 《现代操作系统》英文影印版第4版，Andrew S. Tanenbaum著，机械工业出版社，2017.10
- 《操作系统：精髓与设计原理》英文版第9版，William Stallings，电子工业出版社，2020.6



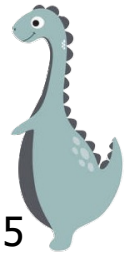
第1章 操作系统基础知识

- 本章内容
  - 1.操作系统的定义、作用、主要特征和核心功能；
  - 2.操作系统的历史演化，主要分类和不同的结构设计（操作系统的发展是由典型的软硬件技术的发展推动的，典型操作系统）；
  - 3.操作系统的引导方式及引导过程，操作系统的启动机制和原理，程序执行的典型流程。



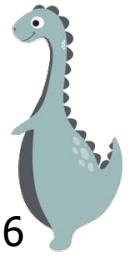
问题列表

- 什么是操作系统
- 操作系统的发展历程
- 操作系统面临的挑战
- 为什么要学习操作系统

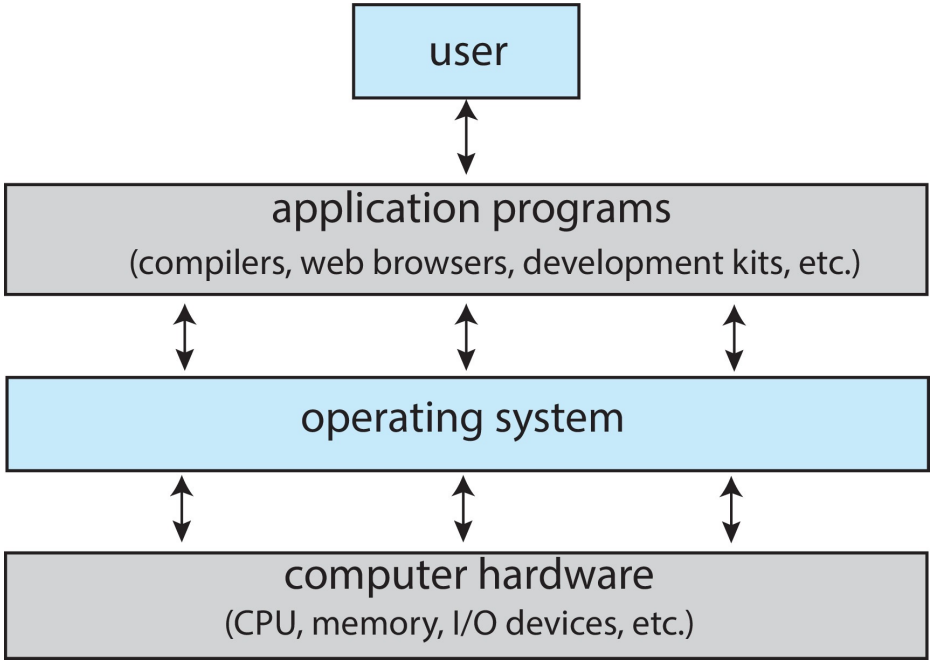


什么是操作系统？

- 以下哪些是操作系统？
  - Windows 10所包含的所有软件
  - Linux内核以及所有设备的驱动
  - 在Macbook上下载安装的第三方NTFS文件系统
  - 华为Mate 30出厂时所有的软件
  - 大疆无人机出厂时所有的软件
  - 火星车上运行的软件
- 如何定义操作系统？
  - 操作系统是**管理硬件资源**和为应用程序**提供基础服务**的核心**系统软件**



操作系统与各组件的关系



- 用户
  - 人、机器、其它计算机
- 应用程序
  - 浏览器、办公软件、数据库系统、游戏
- **操作系统**
  - **控制和协调不同的用户和应用程序使用硬件**
- 硬件
  - CPU、内存、I/O设备

操作系统是管理硬件资源、控制程序运行、改善人机界面和为应用软件提供支持的一种系统软件。

从hello world说起

```c
#include <stdio.h>

int main(){
  printf('Hello World!');
  return 0;
}
```

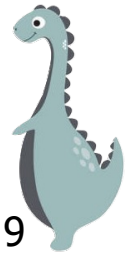
```
#编译Hello World程序
$gcc hello.c -o hello

#运行一个Hello World程序
$ ./hello

#同时启动两个Hello World程序
$ ./hello & ./hello

Hello World!
Hello World!
```

hello运行时发生了什么？

操作系统有什么作用？

从操作系统的角度思考问题

- hello可执行文件如何存储在计算机中？
- 如何根据字符串hello找到存储在计算机中的文件？
- hello可执行文件如何加载到内存并在CPU上运行？
- hello执行文件如何将“hello world”显示在屏幕上？
- 两个hello程序是如何同时在一个CPU上运行的？
- 。 。 。



操作系统的作用

- 让系统易于使用，且正确高效地运行
  - 让程序的运行更容易（允许运行多个程序）
  - 允许程序共享内存
  - 让程序能够与设备交互
- **研究操作系统的两个视角**
  - 从用户视角看：操作系统是用户与计算机硬件之间的接口。为用户使用计算机提供服务。
  - 从资源视角看：操作系统管理硬件资源且对硬件进行抽象。

如何让系统易于使用？

- 虚拟化（virtualization）技术
  - 将物理资源转换为更通用、更强大且易于使用的虚拟形式
  - 虚拟化CPU
    - 进程/线程，数量不受物理CPU的限制
  - 虚拟化内存
    - 虚拟内存，大小不受物理内存的限制
  - 虚拟化I/O
    - 将各种设备统一抽象为文件，提供统一接口

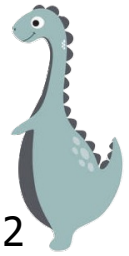
虚拟化CPU

```
1    #include <stdio.h>
2    #include <stdlib.h>
3    #include <sys/time.h>
4    #include <assert.h>
5    #include "common.h"
6
7    int
8    main(int argc, char *argv[])
9    {
10       if (argc != 2) {
11           fprintf(stderr, "usage: cpu <string>\n");
12           exit(1);
13       }
14       char *str = argv[1];
15       while (1) {
16           Spin(1);
17           printf("%s\n", str);
18       }
19       return 0;
20   }
```

$gcc –o cpu cpu.c –Wall
$./cpu A
A
A
A

$./cpu A &; ./cpu B &; ./cpu C &

A
B
C
A
C
B
…



虚拟化内存

```
1    #include <unistd.h>
2    #include <stdio.h>
3    #include <stdlib.h>
4    #include "common.h"
5
6    int
7    main(int argc, char *argv[])
8    {
9        int *p = malloc(sizeof(int));                    // a1
10       assert(p != NULL);
11       printf("(%d) memory address of p: %08x\n",
12              getpid(), (unsigned) p);                  // a2
13       *p = 0;                                          // a3
14       while (1) {
15           Spin(1);
16           *p = *p + 1;
17            printf("(%d) p: %d\n", getpid(), *p); // a4
18       }
19       return 0;
20   }
```

```
$./mem
(2134) Memory address of p: 00200000
(2134) p:1
(2134) p:2
(2134) p:3
(2134) p:4
```

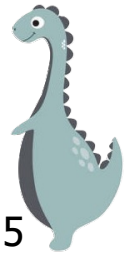
虚拟化内存

```
$./mem &; ./mem &
[1] 24113
[2] 24114
(24113) Memory address of p: 00200000
(24114) Memory address of p: 00200000
(24113) p:1
(24114) p:1
(24114) p:2
(24113) p:2
(24113) p:3
(24114) p:3
(24113) p:4
(24114) p:4
```

- 每个运行的程序都在相同的地址
- 每个运行的程序都独立更新了00200000地址处的值
- 每个运行的程序都有自己的私有内存，而并非共享物理内存

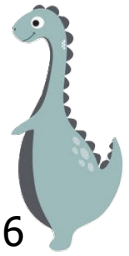
并发(Concurrency)

```
1     #include <stdio.h>
2     #include <stdlib.h>
3     #include "common.h"
4
5     volatile int counter = 0;
6     int loops;
7
8     void *worker(void *arg) {
9         int i;
10        for (i = 0; i < loops; i++) {
11            counter++;
12        }
13        return NULL;
14    }
15
```



并发(Concurrency)

```
16   int
17   main(int argc, char *argv[])
18   {
19       if (argc != 2) {
20           fprintf(stderr, "usage: threads <value>\n");
21           exit(1);
22       }
23       loops = atoi(argv[1]);
24       pthread_t p1, p2;
25       printf("Initial value : %d\n", counter);
26
27       Pthread_create(&p1, NULL, worker, NULL);
28       Pthread_create(&p2, NULL, worker, NULL);
29       Pthread_join(p1, NULL);
30       Pthread_join(p2, NULL);
31       printf("Final value    : %d\n", counter);
32       return 0;
33   }
```



并发(Concurrency)

```
$ gcc –o thread thread.c –Wall -pthread
$ ./thread 1000
Initial value:  0
Final value :  2000

$ ./thread 100000
Initial value:  0
Final value :  143012
$ ./thread 100000
Initial value:  0
Final value :  137128
$ ./thread 100000
Initial value:  0
Final value :  200000
```

- 当loops输入为N时，预计程序输出为2N
- 程序运行有时能得到正确结果，有时却不能
- 增加共享计时器的代码由3条指令构成，这3条指令不是以原子方式执行

持久性(Persistence)

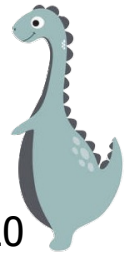
- 内存中的数据容易丢失(volatile)
- 需要通过硬件（硬盘）和软件（文件系统）来持久地存储数据
- 文件系统
  - 以可靠、高效的方式将用户创建的文件存储在系统磁盘上
    - 确定文件在磁盘上的存储位置
    - 维护管理文件涉及的数据结构
    - 提供一组系统调用帮助应用程序访问存储设备

操作系统的主要特征

- 1.并发：在同一时间间隔内发生两个或多个事件
- 2.共享：系统中的资源可供多个并发执行的进程共同使用
  - 互斥共享：一段时间只允许一个进程访问
  - 同时访问：一段时间允许多个进程访问
  - 时分复用、空分复用
- 并发与共享互为存在条件
  - 程序的并发执行带来资源共享问题；
  - 系统对资源共享的管理影响到程序并发执行的效率

操作系统的主要特征

- **3.虚拟**
  - 把一个物理上的实体变为若干个逻辑上的对应物
  - 分时技术、虚拟内存、虚拟设备
- **4.异步（不确定）**
  - 任务的执行顺序和每个任务的执行时间是不确定的

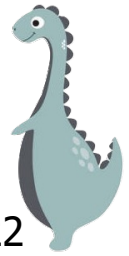


配置操作系统的主要目的

- 提供用户与计算机之间的接口，使计算机更易于使用；
- 有效地控制和管理计算机系统中的各种资源，使之得到更有效的利用；
- 合理地组织计算机系统的工作流程，以提高资源利用率并改善系统性能。
- **用户目标**：易用易学、可靠、安全、快速
- **系统目标**：易于设计、实现和维护，灵活、可靠、正确高效率

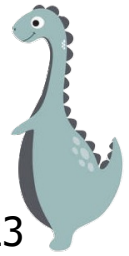
操作系统的设计目标

- 建立抽象(abstraction)，使系统易于使用
- 最小化操作系统的开销
- 为应用程序之间以及应用程序与操作系统之间提供保护（实现隔离）
- 提供高可靠(reliability)
- 能源效率(energy-efficiency)



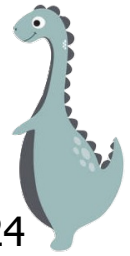
操作系统的功能

- 管理计算机系统的软硬件资源
  - 处理机 —— 进程管理
  - 存储器 —— 内存管理
  - 设备 ——设备管理
  - 文件 —— 文件系统管理
- 提供保护与安全
  - 控制进程或用户访问计算机系统资源
  - 防止系统不受外部或内部的攻击



进程管理

- 进程是正在**执行**的程序。
- 为了完成任务，进程需要一定的**资源**，包括CPU时间，存储器，文件，以及I/O设备
- 进程管理的主要任务是对CPU的使用实施有效的管理。
  - **进程控制**：负责进程的创建、撤消及状态转换
  - **进程同步**：对并发执行的进程进行协调。有同步与互斥
  - **进程通信**：负责完成进程间的信息交换
  - **调度**：决定哪个进程优先使用CPU



内存管理

- 内存可以看做是一个大数组，每个单元有自己的地址
- 内存可被CPU和I/O设备共享、可快速存取数据
- CPU执行的指令必须存放在内存中
- 内存管理功能
  - 按需**分配**和**收回**内存空间
  - **内存扩充**：逻辑上的扩充，虚拟内存
  - **内存保护**：彼此隔离

设备管理(I/O系统)

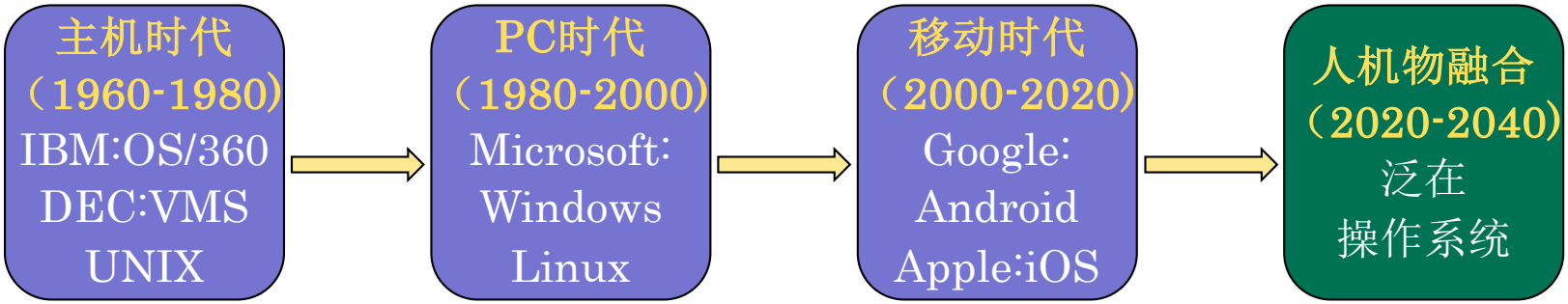
- 为用户隐藏具体硬件设备的特性
- 设备管理的功能
  - **设备分配**：根据用户的I/O请求，为之分配所需的设备，设备使用完成后还应回收
  - **缓冲管理**：对各类设备缓冲区进行有效管理
  - **设备驱动**：主要完成设备启动、I/O操作及中断处理
  - **设备独立性**：又称设备无关性，是指用户程序中的设备与实际使用的物理设备无关

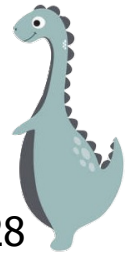
文件管理

- 文件是由其创建者定义的一组相关信息的集合
- 操作系统文件管理的功能
  - **文件存储空间的管理**：对文件存储空间进行管理，包括存储空间的分配与回收等功能。
  - **目录管理**：管理文件的数据结构，提供按名存取的功能。
  - **文件操作管理**：从外存读入数据或将数据写入外存
  - **文件保护**：防止未授权用户存取文件；防止授权用户以不正确方式存取文件

操作系统的历史演化

- 操作系统的发展过程是一个从无到有，从简单到复杂的过程。
- 最初计算机上无操作系统，20世纪50年代出现了简单批处理系统，60年代出现了多道批处理系统，不久又出现了分时系统及实时系统...



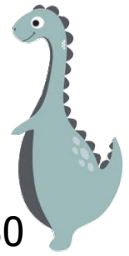


手工操作阶段

- 人工操作方式：46~50年代中后期，计算机系统上没有配置操作系统，人们使用计算机采用手工操作方式。
- 用户使用计算机的过程大致如下：
  - 先将程序纸带（或卡片）装入输入机
  - 然后启动输入机把程序和数据送入计算机
  - 接着通过控制台开关启动程序运行
  - 当程序运行完毕，由用户取走纸带和计算结果

手工操作方式的特点

- 手工操作方式的特点：
  - 用户独占计算机资源，资源利用率低
  - CPU等待人工操作
- 手工操作方式的不足：
  - 手工操作的慢速与CPU运算的高速之间的矛盾，此即人机矛盾。
  - CPU的快速与I/O设备慢速的矛盾。



早期批处理

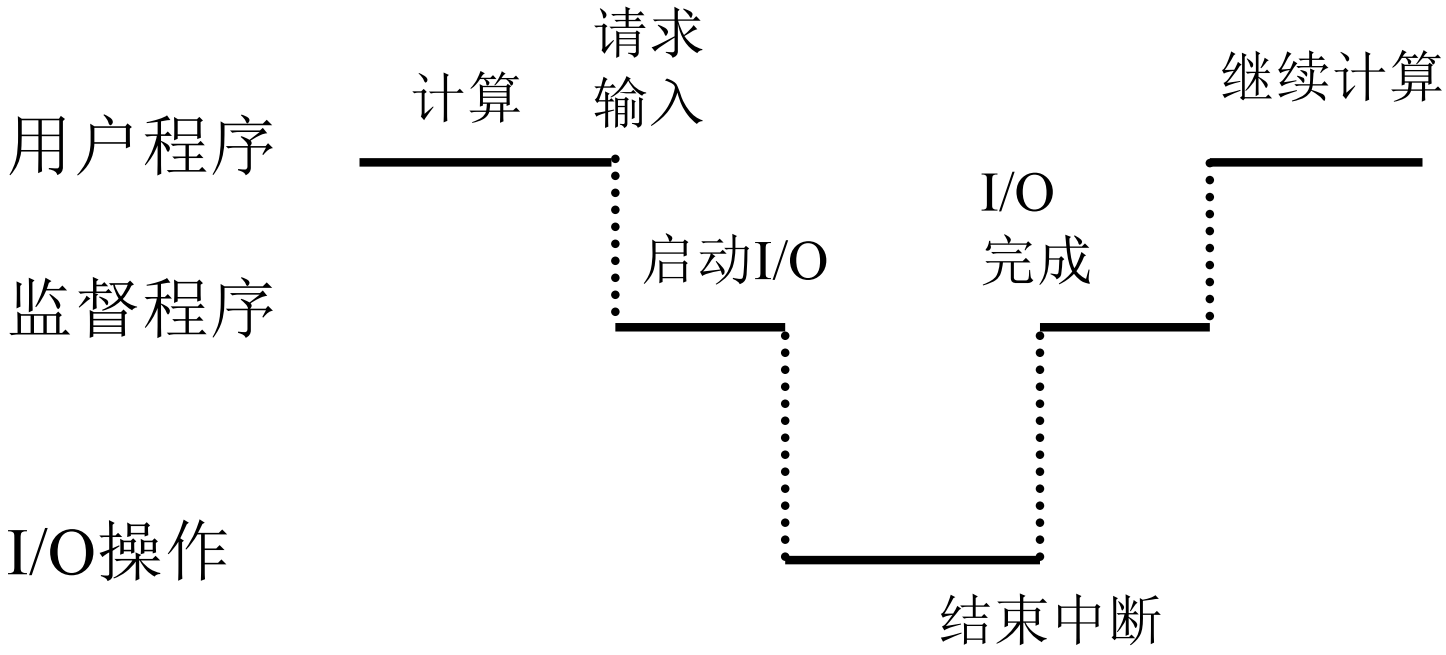
- 为解决人机矛盾，人们提出了从一个作业到下一个作业的自动过渡方式，从而出现了批处理技术。
- 监督程序是一个常驻内存的程序，它管理作业的运行，负责装入和运行各种系统程序来完成作业的自动过渡。
- 监督程序是最早的操作系统雏形。

批处理技术

- **批处理技术**是指计算机系统对一批作业自动进行处理的一种技术。
- 早期的批处理分为：
  - 联机批处理
    - 用户的输入和系统的处理是联机（在线）进行的，数据直接从终端或输入设备传输到计算机，并立即被处理的方式。
    - 适用于需要更快响应的场景
  - 脱机批处理
    - 在计算机系统不直接处理用户输入的情况下，通过外部设备（如卡片阅读器、磁带机）将数据输入到系统，然后再统一处理的方式。
    - 适合那些输入和输出操作较为耗时的环境

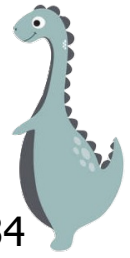
单道程序系统

- 单道批处理系统中内存仅一道程序，系统资源无法得到充分利用。下图是单道程序运行实例

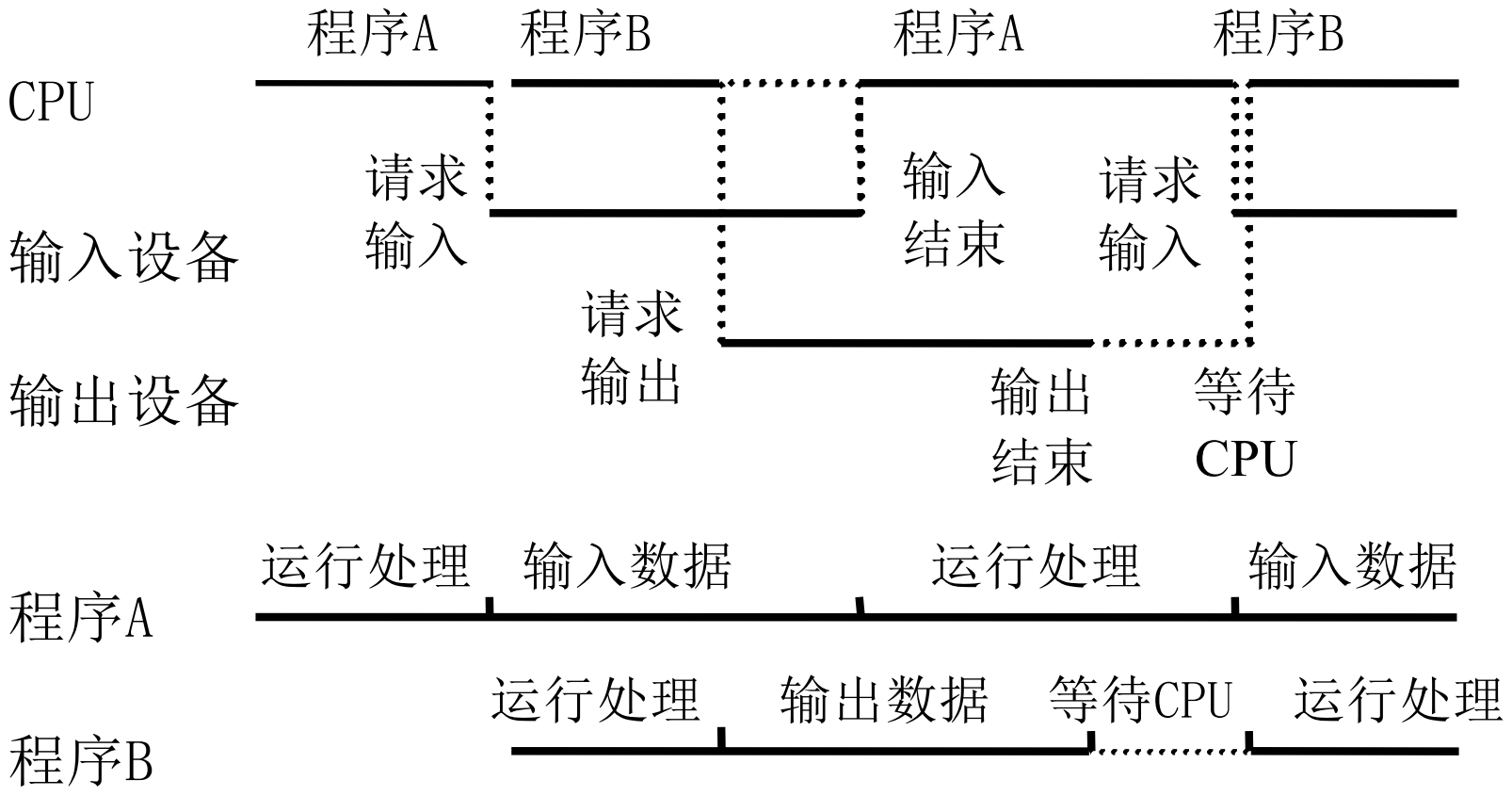


多道程序设计技术

- 传统的单道程序设计在一个程序需要等待I/O时只能让CPU处于空闲状态，这导致了大量的资源浪费
- 多道程序设计通过在内存中同时保存多道程序，让CPU在一个程序等待I/O时可以继续执行其他程序，从而显著提高了资源利用率
- 多个存储在主存中的程序，在管理程序的控制下交替运行，共享CPU和系统中的其他资源。

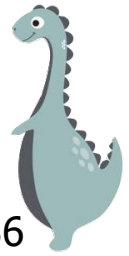


多道程序运行实例


程序A
程序B
程序A
程序B
CPU
请求
输入
输入
结束
请求
输入
输入设备
请求
输出
输出
结束
等待
CPU
输出设备
运行处理
输入数据
运行处理
输入数据
程序A
运行处理
输出数据
等待CPU
运行处理
程序B

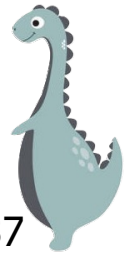
多道程序设计的特点

- 核心目标是提高CPU利用率
- 多个程序并发执行
- 允许资源共享
- 有内存管理能力防止程序间互相影响
- 高效的任务调度与上下文切换机制



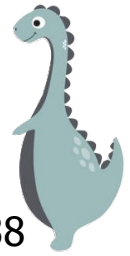
分时系统

- 它允许多个用户通过终端同时与计算机系统进行交互，每个用户都能以相对独立的方式使用系统资源。操作系统通过将CPU的时间分成细小的片段（称为时间片），并快速地在多个用户任务之间切换，给每个用户一种“独占”计算机资源的感觉，从而实现多用户的共享计算机资源。



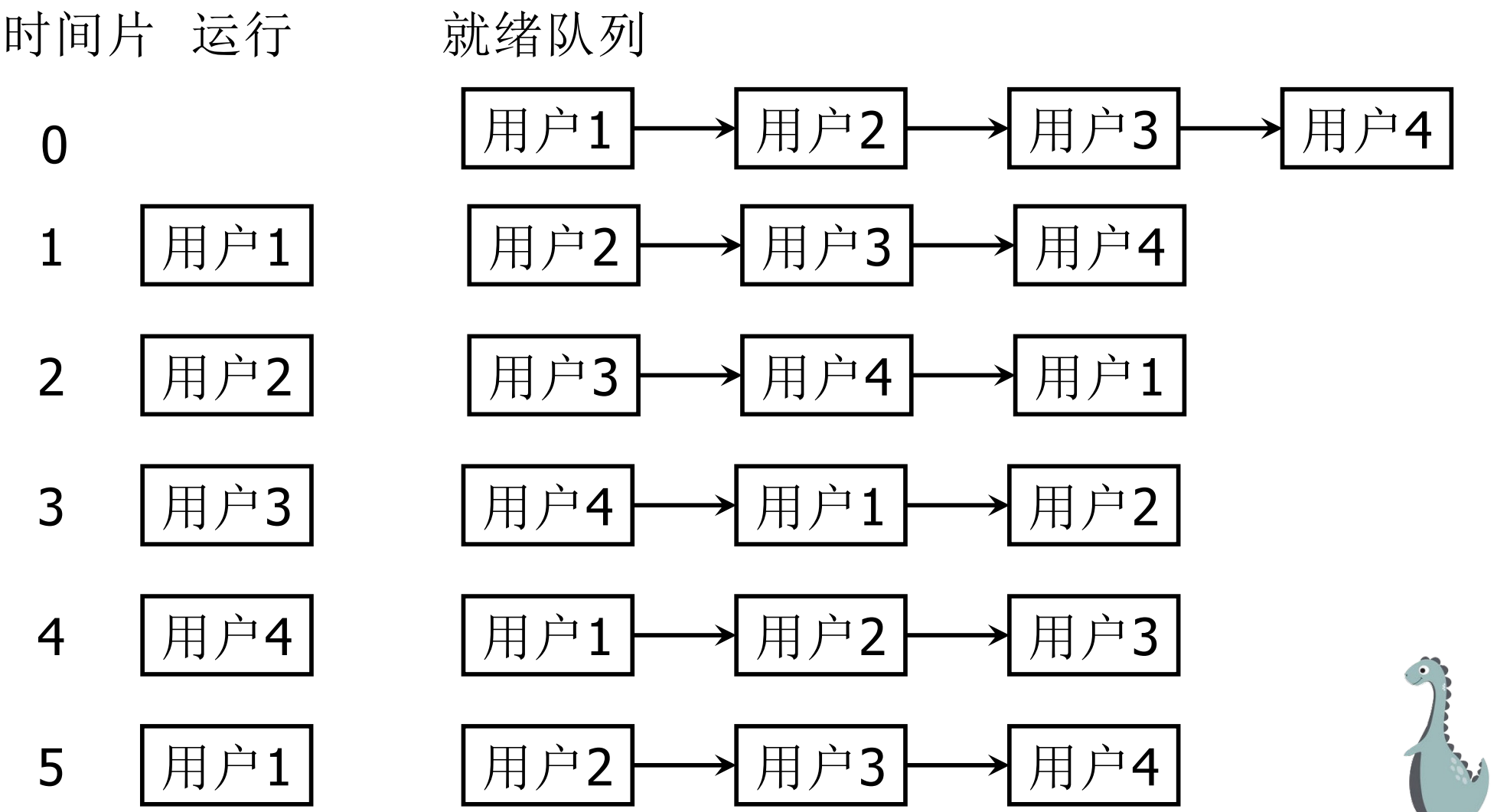
分时系统的特点

- 多用户通过终端与主机互联
- 具有实时性和交互型
- 通过时间片轮转的方式使用CPU，短时间内可以获得响应
- 提供资源的共享与保护



分时技术示意图

- 假设系统中有4个联机用户，按1、2、3、4的顺序排列：



实时系统

- **实时系统**（Real-Time System）是一种计算机系统，它在**特定的时间约束下**执行任务，对外部事件作出**快速响应**。实时系统的核心特征在于其时间敏感性，即系统必须在规定的时间内完成某项任务或对某个事件作出响应，否则将导致系统功能失效或严重后果。实时系统通常用于那些对时间要求严格的应用场景，如工业控制、航空航天、军事系统、医疗设备和通信系统等。

实时系统的特点

- 时间约束（Time Constraints）
  - 硬实时系统（Hard Real-Time System）
  - 软实时系统（Soft Real-Time System）
- 确定性（Determinism）
  - 系统在相同条件下应始终给出相同的输出和行为
- 高可靠性和稳定性：
  - 系统必须设计得非常健壮，能够承受各种故障并具有快速恢复能力。
- 任务优先级（Priority Scheduling）
  - 确保关键任务能够及时完成
- 快速上下文切换（Fast Context Switching）
  - 快速地进行上下文切换，以保证高优先级任务能够尽快执行

操作系统分类维度概览

按应用场景分类

- 桌面系统 | 服务器系统 | 移动系统 | 嵌入式系统

按架构特征分类

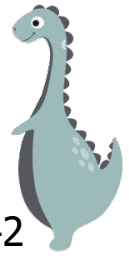
- 单用户 vs 多用户 | 单任务 vs 多任务 | 批处理 vs 交互式

按内核结构分类

- 宏内核 | 微内核 | 混合内核 | 外内核

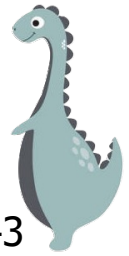
按处理方式分类

- 实时系统 | 分时系统 | 批处理系统 | 分布式系统



按应用场景分类

- 桌面操作系统
  - 代表：Windows、macOS、Ubuntu
  - 特点：用户友好、图形界面、多媒体支持
  - 优化：响应速度、易用性、兼容性
- 服务器操作系统
  - 代表：Linux Server、Windows Server、Unix
  - 特点：高稳定性、多用户、网络服务
  - 优化：吞吐量、可靠性、安全性
- 移动操作系统
  - 代表：Android、iOS、HarmonyOS
  - 特点：触控界面、省电、传感器支持
  - 优化：电池寿命、用户体验、应用生态
- 嵌入式操作系统
  - 代表：FreeRTOS、VxWorks、嵌入式Linux
  - 特点：实时性、资源受限、专用功能
  - 优化：实时响应、内存占用、功耗控制



按用户和任务特征分类

- 用户数量维度
  - 单用户系统
    - 早期PC系统 (DOS)
    - 个人设备系统
    - 简化管理，提高性能
  - 多用户系统
    - Unix/Linux服务器
    - 大型机系统
    - 用户隔离，资源共享
- 任务处理维度
  - 单任务系统
    - 早期微机系统
    - 简单嵌入式系统
    - 结构简单，实时性好
  - 多任务系统
    - 现代桌面系统
    - 服务器系统
    - 并发处理，资源利用率高

按内核架构分类

宏内核 (Monolithic Kernel)

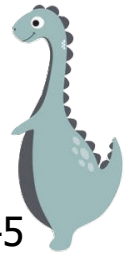
- 代表：Linux、传统Unix
- 特点：所有服务在内核空间运行
- 优势：性能高、通信效率好
- 劣势：稳定性风险、难以维护

微内核 (Microkernel)

- 代表：Minix、QNX、L4
- 特点：最小内核+用户态服务
- 优势：稳定性好、模块化强
- 劣势：性能开销、通信复杂

混合内核 (Hybrid Kernel)

- 代表：Windows NT、macOS
- 特点：关键服务在内核，其他在用户态
- 优势：平衡性能与稳定性
- 劣势：设计复杂度高

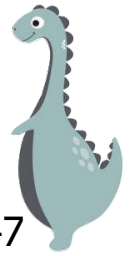


操作系统分类总结

- 🎯 分类的本质
  - 不同应用场景需要不同的系统设计权衡
- 📊 四个分类维度
  - 应用场景 | 用户任务特征 | 内核架构 | 处理方式
- 💡 设计智慧
  - 没有万能的操作系统
  - 每种设计都有其适用场景
  - 理解需求是选择的关键
- 🚀 实践价值
  - 为项目选择合适的操作系统
  - 理解系统设计的核心原则
  - 把握技术发展的趋势方向
- 💭 思考题
  - 你认为未来操作系统发展的最大挑战是什么？

操作系统面临的挑战

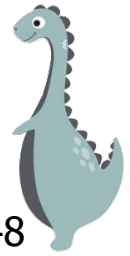
- 安全性和隐私保护
- 多核和异构计算支持
- 虚拟化与云计算优化
- 物联网和边缘计算支持
- 能效与电源管理
- 快速适应硬件与软件变化
- 用户体验和界面的改进
- 集成人工智能与机器学习技术
- 可靠性与可用性保障
- 数据与存储管理的优化



操作系统的结构

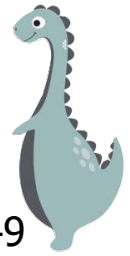
- OS是一个复杂的大型软件
- 如何控制OS设计的复杂度？
  - 策略和机制相分离，
  - 策略：做什么？
  - 机制：怎么做？

	策略	机制
登录	什么用户，以什么权限登录等	输入处理、策略文件管理、桌面启动加载等
调度	先到先服务、时间片轮转等	调度队列的设计、调度实体的表示、调度的中断处理等



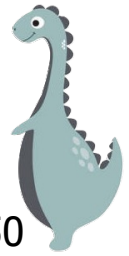
操作系统复杂度管理方法

- 管理复杂系统的重要方法：M.A.L.H
  - **模块化**：将复杂系统分解为一系列模块，通过明确定义的接口进行交互，高内聚、低耦合
  - **抽象**：接口与内部实现分离，模块通过抽象的接口进行交互，而无需关心模块的内部实现
  - **分层**：将模块按一定的原则进行层次划分，约束模块只能同层或与相邻层交互
  - **层级**：功能相近的模块组成具有清晰接口的自包含子系统，子系统再递归地组成有清晰接口的更大子系统



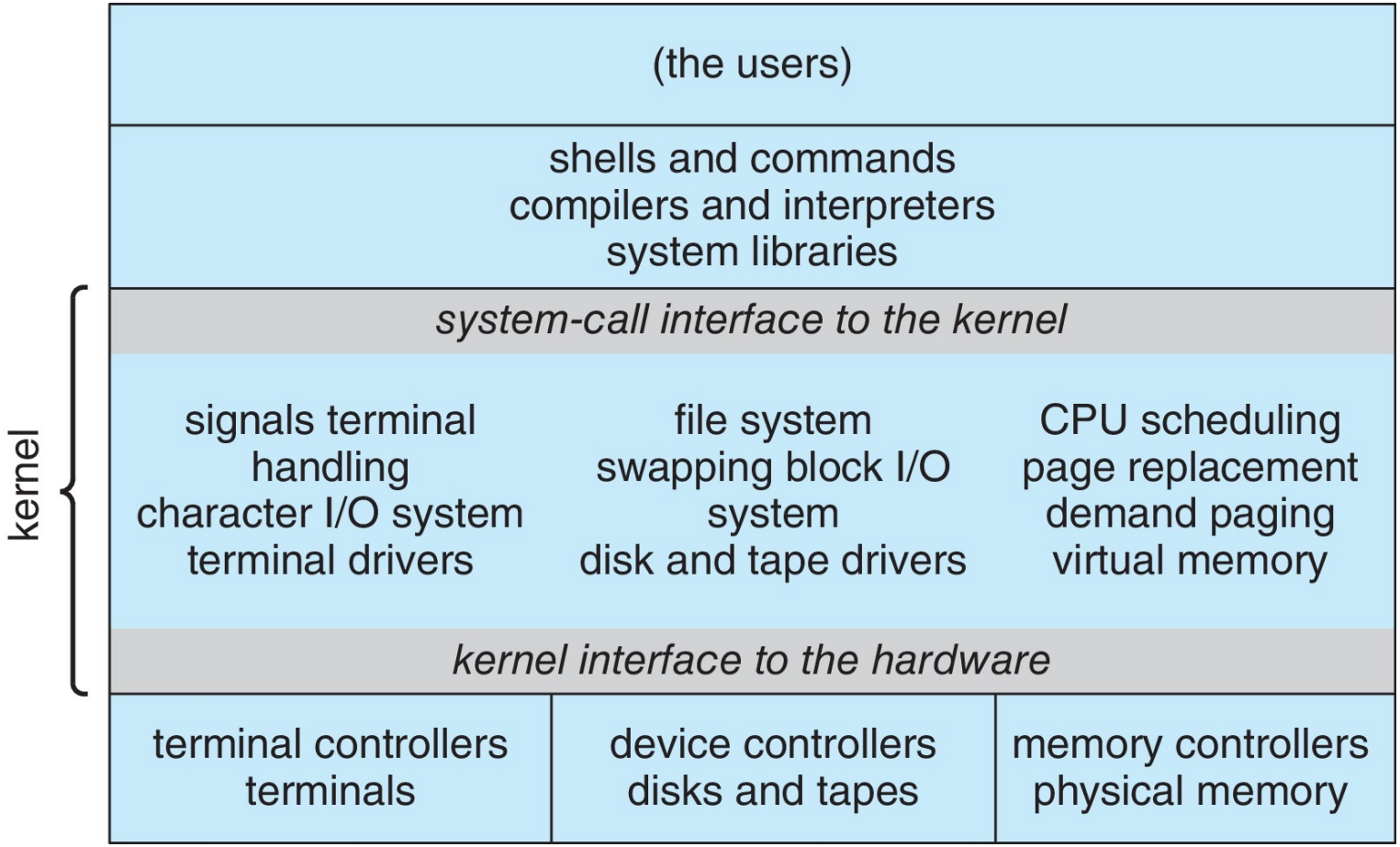
操作系统内核结构

- 简单结构 —— MS-DOS
- 宏内核（单内核）—— UNIX/Linux
- 模块化结构
- 微内核 —— Mach



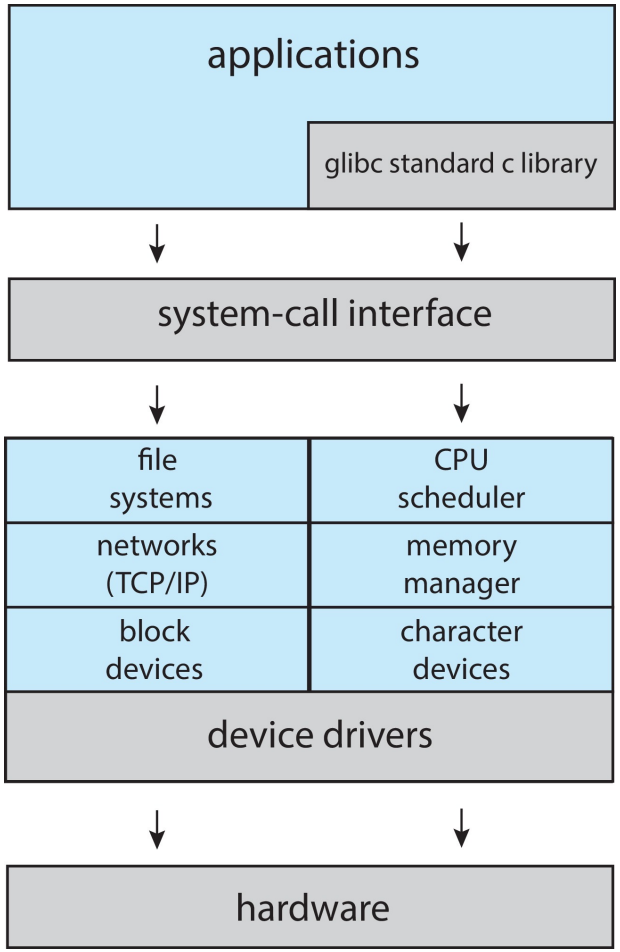
宏内核结构——初始的Unix

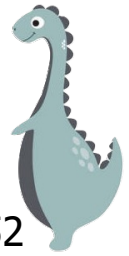
- Unix由系统程序和内核构成
  - 系统调用之下和物理硬件之上的均为内核
  - 内核通过系统调用提供文件系统、CPU 调度、内存管理和其它功能



Linux系统结构

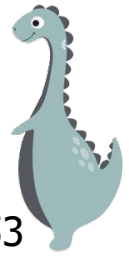
- 宏内核+模块化设计





宏内核的特点

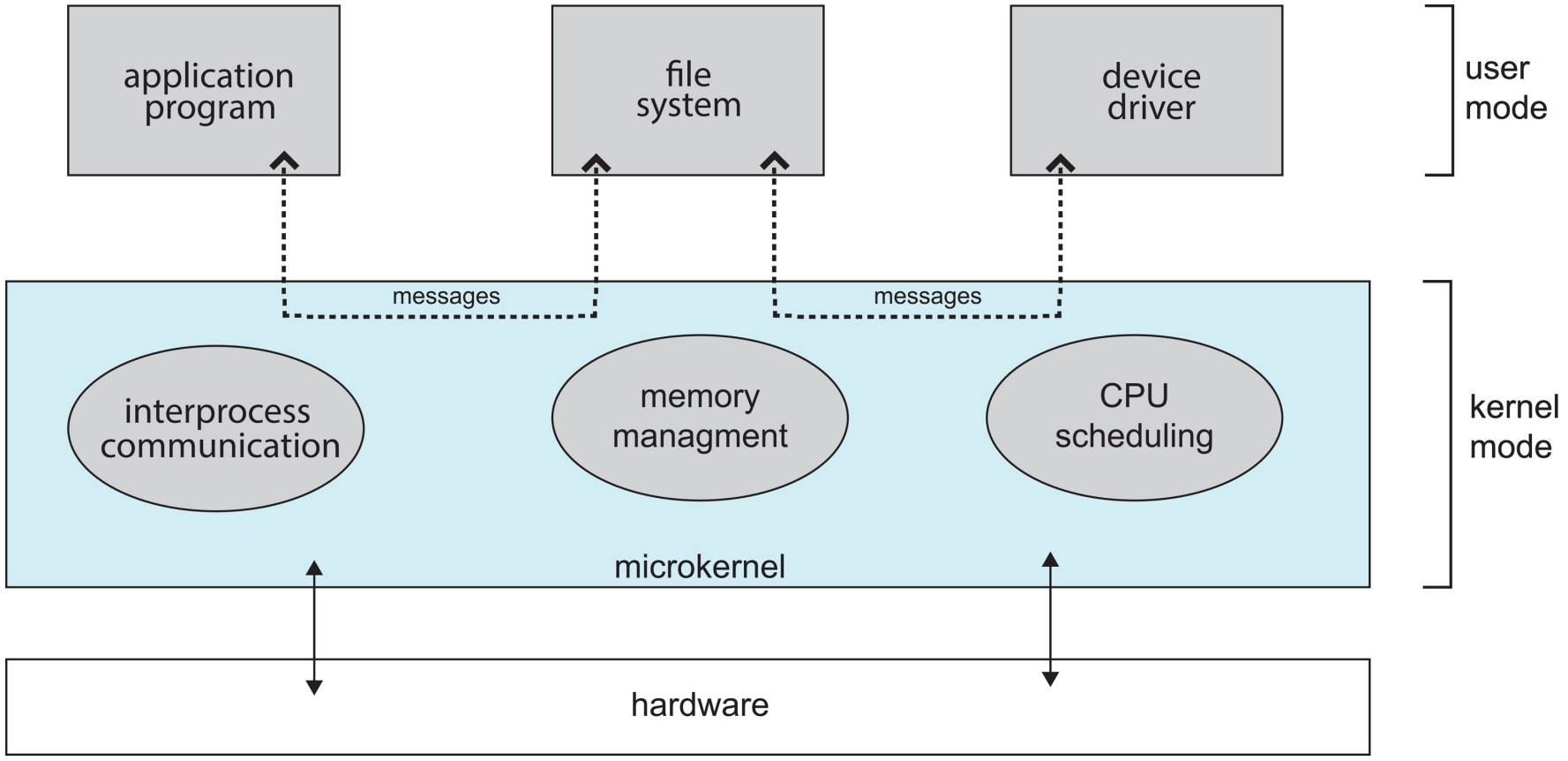
- 优点：运行效率高
- 缺点：结构难以理解、难以维护，随着复杂性的增加，可靠性和安全性难以保障



微内核结构

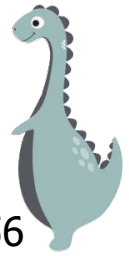
- 微内核结构基于客户/服务器模型，由微内核和核外用户空间的服务器进程构成
- 微内核保留进程管理、内存管理和通信功能，
- 通过消息传递使客户程序与服务程序进行通信
- 优点
  - 易于扩展
  - 易于从一个硬件架构移植到另一个
  - 更可靠（内核代码更少）
  - 更安全
- 缺点
  - 用户空间到内核空间的通信增加了系统开销，早期性能不及宏内核结构

微内核系统的结构


application program
file system
device driver
user mode
messages
messages
interprocess communication
memory managment
CPU scheduling
kernel mode
microkernel
hardware

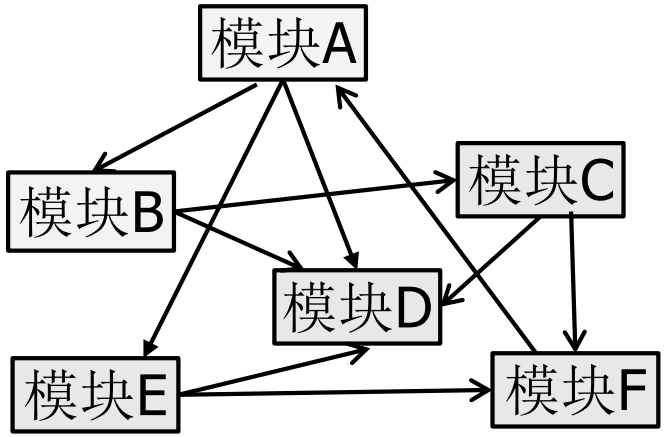
混合结构

- 现代OS往往采用的不是单一模型
  - 混合模型结合多种方法来满足性能、安全性和可用性需求
  - Linux和Solaris是单内核结构，但它们也是模块化的，可以动态向内核添加新功能
  - Windows是单内核结构，但也保留了微内核的模式来支持子系统，也提供可加载的内核模块
  - MacOS是混合、分层的系统，采用Mach微内核与部分BSD Unix构成内核，且有可动态加载的模块



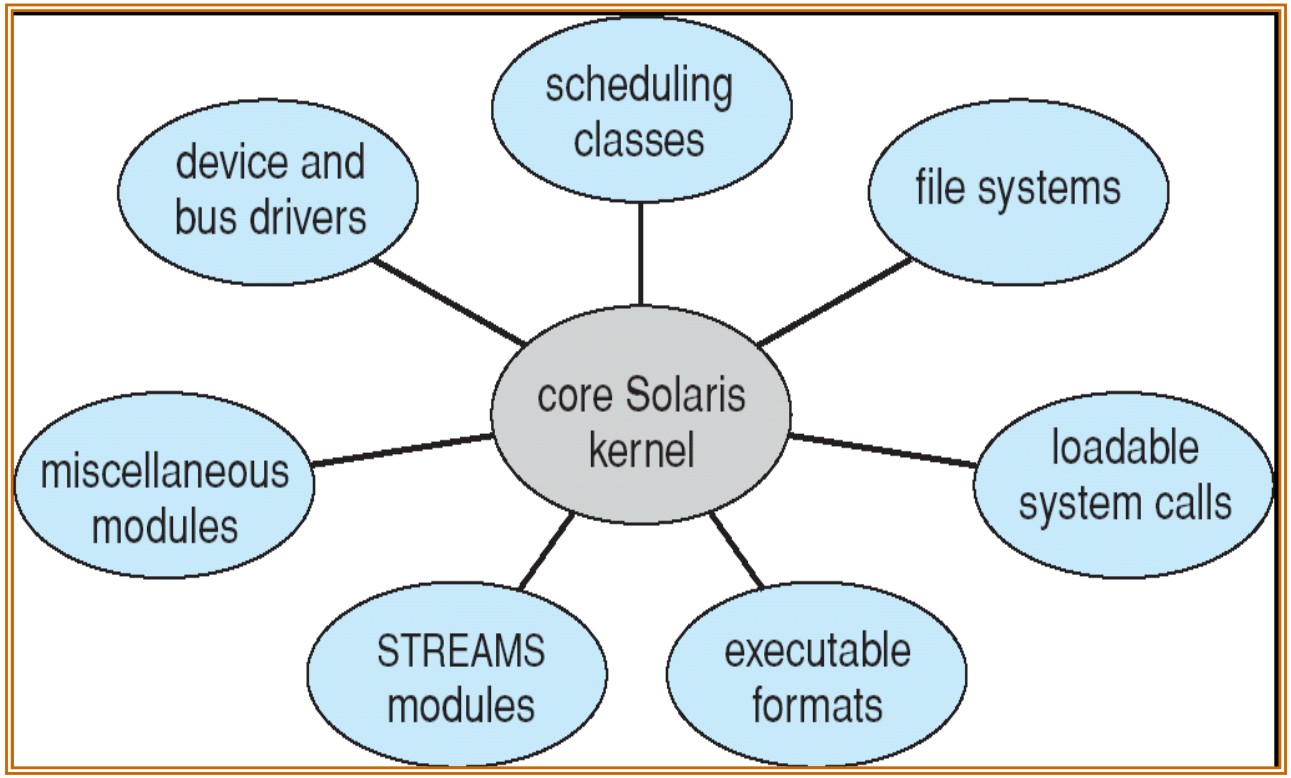
模块化结构

- 模块功能独立，且被封装
- 具有良好定义的接口
- 现代OS(如Unix、Linux、Windows等)均采用模块化的策略组织各功能
- 没有商业OS是纯粹采用模块化结构设计
- 优点：易于理解，效率高
- 缺点：全局函数使用多造成访问控制困难，结构不清晰会导致可理解性、可维护性及可移植性差，存在潜在的性能退化



可加载的Solaris模块

- Solaris组织成7个可加载的内核模块及一个核心内核所构成：



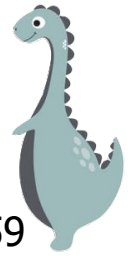
操作系统的启动流程

- 思考：计算机上电瞬间发生了什么？

🤔 问题场景

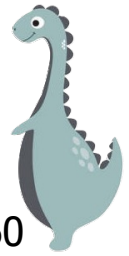
  - CPU刚上电，内存是空的
  - CPU不知道要执行什么程序
  - 硬件设备都未初始化
  - 操作系统还在磁盘上

❓ 核心问题 如何让一台"空白"的计算机运行起复杂的操作系统？



启动过程的本质问题

- CPU执行起点问题
  - CPU不知道从哪个地址开始执行
- 程序加载问题
  - 内存里什么都没有，如何加载程序？
- 硬件初始化问题
  - 设备未配置，如何访问外部存储？
- 复杂性管理问题
  - 操作系统太复杂，如何分步加载？



解决方案概览

- 分阶段启动策略
  - 阶段1: 硬件自举 (ROM/BIOS)
    - 🎯 让CPU有最基本的执行能力
  - 阶段2: 引导加载器 (Bootloader)
    - 🎯 准备运行环境，加载操作系统
  - 阶段3: 内核初始化
    - 🎯 建立完整的操作系统环境

整体启动流程图

[上电] → 硬件复位

↓

[ROM启动] → 固定地址开始执行

↓

[Bootloader] → 环境准备，硬件初始化

↓

[内核加载] → 程序加载，内存布局

↓

[内核初始化] → 系统建立，进程管理

↓

[用户空间] → 服务启动，用户登录

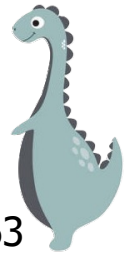
阶段1-硬件自举

- ROM启动阶段：解决“鸡蛋问题”

🥚 鸡蛋问题
  - CPU需要程序才能工作
  - 程序需要CPU来加载

💡 解决方案
  - 主板ROM固化启动程序
  - CPU硬件设定固定启动地址
  - RISC-V: 0x1000

🎯 核心任务
  - 初始化内存控制器
  - 初始化基本I/O设备
  - 查找下一阶段程序
  - 加载并跳转到Bootloader



为什么要分阶段

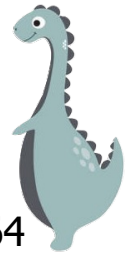
- 单阶段 vs 多阶段对比

❌ 如果把所有功能都放在ROM里：

- ROM容量有限 (通常只有几KB到几MB)
- ROM内容固化，无法灵活更新
- 不同设备需要不同操作系统
- 成本高昂

✅ 分阶段启动的优势：

- ROM只需很小空间，降低成本
- 后续阶段可灵活更换和升级
- 支持多操作系统启动选择
- 职责分离，便于维护和调试



阶段2-引导加载器

- Bootloader：搭建桥梁

🌉 为什么需要中间层？

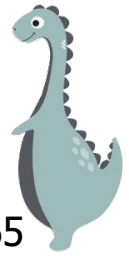
- ROM功能太简单 ←→ 操作系统太复杂

🎯 核心职责

- 1. 硬件抽象 - 为不同硬件提供统一接口
- 2. 环境准备 - 设置内存布局、栈空间等
- 3. 加载内核 - 从存储设备读取内核到内存
- 4. 参数传递 - 告诉内核硬件配置信息

🔧 典型功能

- 理解文件系统 (FAT32, ext4等)
- 解析内核格式 (ELF等)
- 多启动选项支持
- 硬件检测和配置



阶段3-内核初始化

- 内核初始化：建立秩序
- 🏗 从单程序到多任务系统的转变
- 🎯 初始化任务
  - 1. 建立管理体系
    - 内存管理子系统
    - 进程调度机制
    - 文件系统框架
  - 2. 硬件驱动加载
    - 设备驱动程序
    - 中断处理机制
    - I/O子系统
  - 3. 系统服务启动
    - 网络服务・安全机制・用户管理
  - 4. 用户环境准备
    - 启动init进程・加载shell・准备用户登录

RISCV启动实例

- RISC-V架构下的启动过程

⚡ 上电瞬间

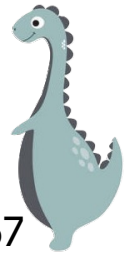
- 1. 电源稳定，时钟开始工作
- 2. PC寄存器硬件设置为 0x1000
- 3. CPU从此地址开始取指执行
- 4. 0x1000 映射到ROM区域

🎯 地址选择原理

- 0x0000: 异常向量表保留
- 0x1000: ROM区域起始
- 标准化保证兼容性

📍 内存布局示例

- 0x00000000 - 0x00000FFF: 异常向量
- 0x00001000 - 0x0001FFFF: ROM区域
- 0x80000000 - 0xFFFFFFFF: RAM区域

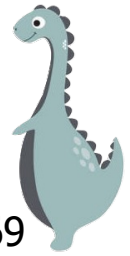


关键设计原则

- 渐进复杂性原则
  - ROM (几KB) → Bootloader (几百KB) → 内核 (几MB) 每阶段复杂度递增，避免跳跃式复杂化
- 职责分离原则
  - ROM: 基本硬件初始化
  - Bootloader: 加载和环境准备
  - 内核: 系统管理和服务
- 标准化原则
  - 明确的接口规范
  - 标准的参数传递格式
  - 统一的启动协议

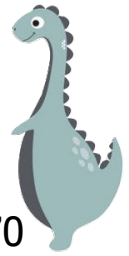
实际应用场景

- 💻 PC启动: BIOS/UEFI → GRUB → Linux内核
- 📱 手机启动: BootROM → U-Boot → Android内核
- 🔌 嵌入式: ROM → 自定义Bootloader → RTOS
- ☁️ 服务器: UEFI → 网络启动 → 容器化系统
- 🔧 启动优化考量
  - 启动速度优化
  - 安全启动验证
  - 故障恢复机制
  - 远程启动支持

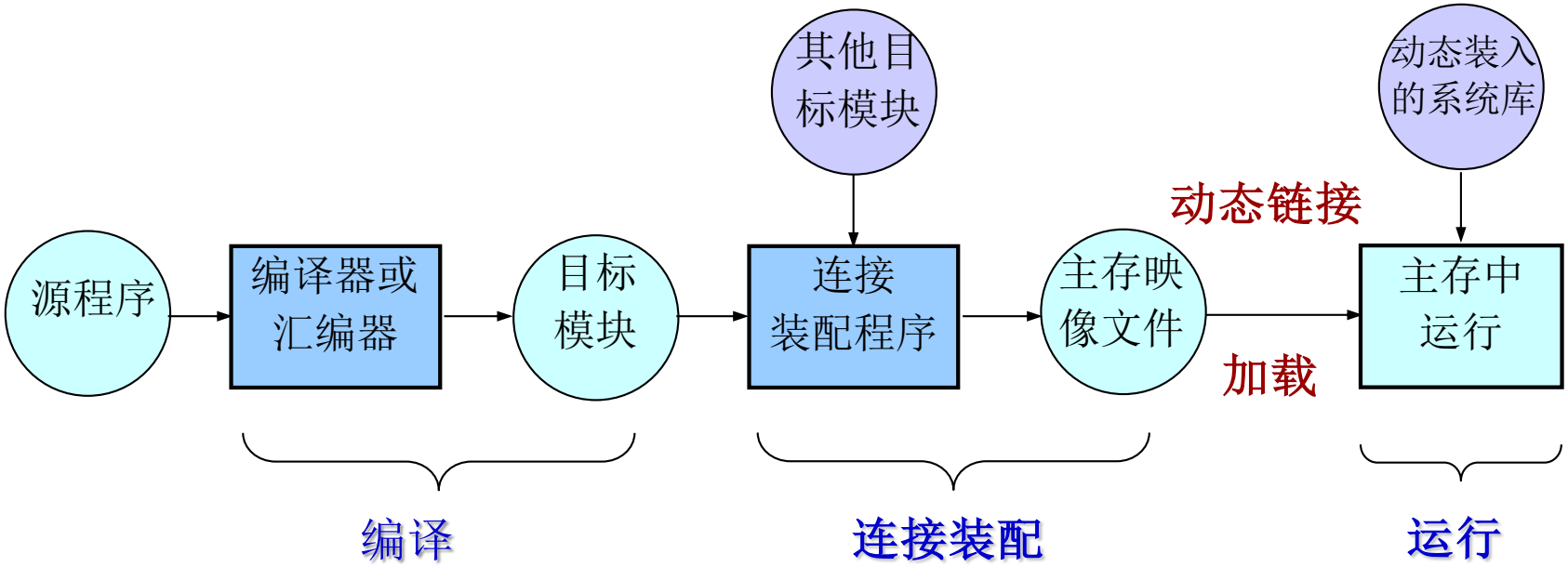


总结与思考

- 🎯 启动本质
  - 解决CPU执行起点和程序加载的根本问题
- 🔄 三阶段模式
  - ROM自举 → Bootloader桥接 → 内核建立
- 💡 设计智慧
  - 分而治之的复杂性管理
  - 职责分离的模块化设计
  - 标准化的接口规范
- 🚀 实践
  - 设计并实现一个简化的RISC-V启动程序



应用程序的处理步骤



应用程序处理步骤示意图

应用程序的处理步骤

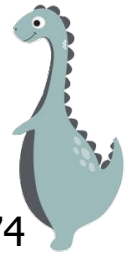
- 编辑
  - 建立一个新文件，或对已有的文件中的错误进行修改。
- 编译
  - 将源程序翻译成浮动的目标代码。
- 连接
  - 主程序和其他所需要的子程序和例行程序连接装配在一起，使之成为一个可执行的、完整的主存映像文件。
- 运行
  - 将主存映像文件调入主存，启动运行，得出计算结果。

一个程序的典型执行流程

1. **加载：** 当启动一个程序时，由OS的加载器将该程序的可执行文件加载到内存中。
2. **运行：** OS创建一个新的进程，并将CPU的控制权交给该进程。该新进程开始执行加载到内存中的程序代码。
3. **系统调用：** 当程序需要进行一些特权操作（如读写文件，发送网络数据等）时，它将发起系统调用。
4. **中断和信号：** 程序的执行可能会被中断和信号打断。中断通常是由硬件事件触发的，如I/O操作的完成，定时器的超时等。信号则是一种软件中断，可以由其他进程或者内核发送。
5. **退出：** 当程序执行完成或者由于某种原因需要停止时，它将执行退出操作，包括释放资源，关闭打开的文件，通知父进程等。

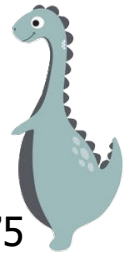
操作系统的运行(Operation)

- 引导程序→加载内核→启动系统守护进程
- 内核是中断驱动的（硬件和软件）
  - 由任一设备触发硬件中断
  - 软件中断（异常或陷阱）
    - 软件错误（例如，除数为零）
    - 请求操作系统服务 —— 系统调用
    - 其他进程问题包括无限循环，进程相互修改或修改操作系统



OS对程序执行的支持

- OS如何让程序能在计算机上顺利执行？
  - 装载并运行程序
  - 管理内存空间：为程序提供虚拟地址空间，实现逻辑地址到物理地址的转换
  - 支持输入/输出
  - 提供错误处理和保护

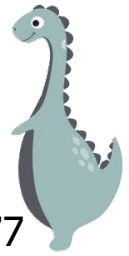


为什么要学习操作系统？

- 理解计算机的基本工作原理
- 提高编程和软件开发能力
- 适应现代科技和发展趋势
- 增强系统安全意识和技能
- 适应不同的职业需求
- 培养计算机科学思维和能力
- 适应科技发展的快速变化
- 。 。 。

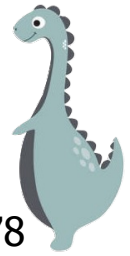
问答题

- 1.2 We have stressed the need for an operating system to make efficient use of the computing hardware. When is it appropriate for the operating system to forsake this principle and to "waste" resources? Why is such a system not really wasteful?
- 1.4 Keeping in mind the various definitions of operating system, consider whether the operating system should include applications such as web browsers and mail programs. Argue both that it should and that it should not, and support your answers.



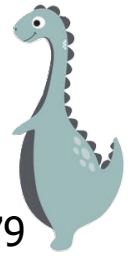
问答题

- 1.6 Which of the following instructions should be privileged?
  - a. Set value of timer.
  - b. Read the clock.
  - c. Clear memory.
  - d. Issue a trap instruction.
  - e. Turn off interrupts.
  - f. Modify entries in device-status table.
  - g. Switch from user to kernel mode.
  - h. Access I/O device.



问答题

- 请简述计算机从开机到操作系统完全启动的基本过程？
- 什么是引导扇区（Boot Sector）？它在操作系统的启动过程中起什么作用？
- 请描述一个程序从点击图标到完全运行的典型执行流程？



选择题1

- 在计算机系统中，操作系统是_____。

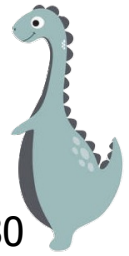
  A. 一般应用软件　　B. 核心系统软件

  C. 用户应用软件　　D. 硬件

- 操作系统中最基本的两个特征是_____。

  A. 虚拟和不确定　　B.共享和虚拟

  C. 并发和共享　　D.并发和不确定



选择题2

- 实时操作系统必须在_____内处理来自外部的事件。

  A.时间片　　B.周转时间

  C.被控制对象规定时间　　D.一个机器周期

- 在设计实时操作系统时，不重点考虑的是_____。

  A．及时响应，快速处理　　B．有高安全性

  C．提高系统资源的利用率　　D．有高可靠性

选择题3

- 下述关于并发性的叙述中正确的是_____。

  A.并发性是指若干事件在不同时间间隔内发生

  B.并发性是指若干事件在同一时间间隔内发生

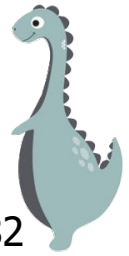
  C.并发性是指若干事件在不同时刻发生

  D.并发性是指若干事件在同一时刻发生

- 分时系统追求的目标是_____。

  A.快速响应用户　B.充分利用I/O设备

  C.充分利用内存　D.提供系统吞吐率



选择题4

- 一个多道批处理系统，提高了计算机系统的资源利用率，同时_____。

  A. 减少各个作业的执行时间

  B. 增加了单位时间内作业的吞吐量

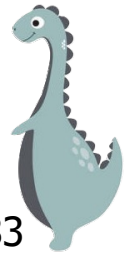
  C. 减少了部分作业的执行时间

  D. 减少单位时间内作业的吞吐量

- 批处理系统的主要缺点是_____。

  A. 无交互能力　　B. 系统吞吐量小

  C. 资源利用率低　　D. CPU利用率不高



选择题5

- 从用户的观点看，操作系统是_____。

  A.合理地组织计算机工作流程的软件

  B. 由若干层次的程序按一定的结构组成的有机体

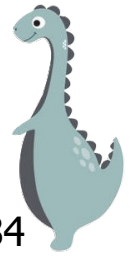
  C. 控制和管理计算机资源的软件

  D.用户与计算机之间的接口

- 所谓_____是指将一个以上的作业放入内存，并且同时处于运行状态，这些作业共享处理机的时间和外围设备等资源。

  A.多道程序设计　　B.多重处理

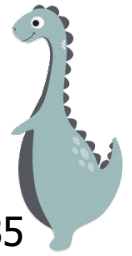
  C.共行执行　　D.实时处理



选择题6

- 关于操作系统的发展历史，下列说法错误的是：

A. 批处理系统是为了提高计算机资源利用率而提出的
B. 分时系统的核心思想是通过时间片轮转实现多用户共享计算机资源
C. 实实时系统是为实时任务设计的，通常要求高的响应速度和确定性
D. 微内核架构是最早的操作系统架构形式，后来被宏内核取代



选择题7

- 引导扇区上存放的代码是怎么来的？

A. 由BIOS写进去的

B. 安装操作系统时写进去的

C. 硬盘格式化时写进去的

D. 所有硬盘都固有的一段代码