408 计算机操作系统冲刺背诵手册

北街学长倾力之作

作者：北街

时间：2025/08/17

版本：1.0

ElegantLaTeX Program

目录

第一章 操作系统概述

1.1 操作系统的基本概念

操作系统定义：操作系统是管理计算机硬件与软件资源的计算机程序，是计算机系统的内核与基石。

操作系统的目标：

1. **方便性**：为用户提供便利的使用环境
2. **有效性**：提高系统资源利用率
3. **可扩充性**：方便系统功能的扩展和修改
4. **开放性**：遵循标准，易于移植

操作系统的功能：

- **进程管理**：进程控制、调度、同步与通信
- **内存管理**：内存分配、地址映射、虚拟存储
- **文件管理**：文件存储、目录管理、存取控制
- **设备管理**：设备分配、驱动程序、中断处理
- **用户接口**：命令接口、程序接口、图形界面

1.2 操作系统的发展历程

发展阶段	主要特点	代表技术
手工操作阶段	程序员直接操作硬件	纸带、卡片输入
单道批处理	自动连续处理作业	监控程序、脱机 I/O
多道批处理	内存中多个程序并发	多道程序设计、中断
分时系统	多用户交互式使用	时间片轮转、虚拟存储
实时系统	对时间要求严格	优先级调度、快速响应
网络操作系统	支持网络功能	分布式处理、远程访问
分布式系统	多机协作处理	集群、负载均衡

1.3 程序运行环境

1.3.1 CPU 运行模式

内核模式（管态、核心态）：

- 可以执行所有机器指令

- 可以访问所有内存区域
- 可以直接访问硬件设备
- 操作系统内核运行在此模式

用户模式（目态、用户态）：

- 只能执行非特权指令
- 只能访问用户内存空间
- 不能直接访问硬件
- 用户程序运行在此模式

模式切换：

- **用户态 → 内核态**：中断、异常、系统调用
- **内核态 → 用户态**：执行特权指令修改 PSW

1.3.2 中断和异常的处理

中断分类：

类型	来源	特点
外部中断	I/O 设备、时钟等	异步、可屏蔽
内部中断	CPU 内部	同步、不可屏蔽
软中断	软件指令	系统调用、陷阱

中断处理过程：

1. 保存现场（PSW、PC 等寄存器）
2. 识别中断源
3. 转向相应的中断服务程序
4. 执行中断服务
5. 恢复现场
6. 返回被中断程序

1.3.3 系统调用

系统调用定义：用户程序请求操作系统服务的接口

系统调用类型：

- **进程控制**：fork、exec、exit、wait
- **文件操作**：open、read、write、close
- **设备管理**：ioctl、read、write
- **信息维护**：getpid、alarm、sleep
- **通信**：pipe、shmget、msgget

系统调用过程：

1. 用户程序调用库函数
2. 库函数将参数放入指定寄存器
3. 执行 trap 指令（软中断）

4. 切换到内核态
5. 执行系统调用服务程序
6. 返回用户态

1.3.4 程序的链接与装入

编译过程：

源程序 → 编译 → 目标模块 → 链接 → 装入模块 → 装入 → 内存映像

链接方式：

- **静态链接**：装入前链接，生成完整可执行文件
- **装入时动态链接**：装入时进行链接
- **运行时动态链接**：运行时需要时才链接

装入方式：

- **绝对装入**：编译时确定绝对地址
- **可重定位装入**：装入时修改地址
- **动态运行时装入**：运行时地址转换

1.3.5 程序运行时内存映像与地址空间

逻辑地址空间：

- **代码段**：程序指令
- **数据段**：全局变量、静态变量
- **堆段**：动态分配内存
- **栈段**：局部变量、函数调用

地址转换：

- **逻辑地址**：程序中的地址
- **物理地址**：内存中的实际地址
- **地址映射**：MMU 完成逻辑到物理地址转换

1.4 操作系统结构

分层结构：

- 系统分为若干层，每层只能调用更低层
- 优点：结构清晰，易于调试
- 缺点：效率低，层次间通信开销大

模块化结构：

- 将系统分解为若干模块
- 优点：可维护性好，可重用性强
- 缺点：模块间接口复杂

宏内核：

- 整个操作系统在内核态运行
- 优点：性能高，调用开销小
- 缺点：可靠性差，难以维护

微内核：

- 只有基本功能在内核态

- 其他功能在用户态作为服务器
- 优点：可靠性高，易于扩展
- 缺点：性能低，通信开销大

外核：

- 只提供硬件资源的安全复用
- 应用程序直接管理资源
- 优点：性能极高，灵活性好
- 缺点：应用编程复杂

1.5 操作系统引导

引导过程：

1. **开机自检**：POST 检测硬件
2. **主引导记录**：MBR 加载引导程序
3. **活动分区**：查找可引导分区
4. **引导程序**：加载操作系统内核
5. **内核初始化**：初始化系统数据结构
6. **启动进程**：创建第一个用户进程

1.6 虚拟机

虚拟机概念：通过软件模拟完整计算机系统的技术

虚拟机类型：

- **系统虚拟机**：虚拟完整的硬件平台
- **进程虚拟机**：虚拟单个程序的运行环境

虚拟化技术：

- **全虚拟化**：客户 OS 无需修改
- **半虚拟化**：客户 OS 需要修改
- **硬件辅助虚拟化**：CPU 提供虚拟化支持

第二章 进程管理

2.1 进程与线程

2.1.1 进程与线程的基本概念

进程定义：进程是程序在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基本单位。

进程特征：

- **动态性**：进程是程序的一次执行过程
- **并发性**：多个进程可同时存在
- **独立性**：进程是独立运行的基本单位
- **异步性**：进程按异步方式运行
- **结构性**：进程由程序、数据、PCB 组成

线程定义：线程是进程中的一个执行单元，是 CPU 调度和分派的基本单位。

进程 vs 线程：

对比项	进程	线程
调度单位	传统的调度单位	现代 OS 的调度单位
拥有资源	拥有资源的基本单位	不拥有资源
并发性	进程间可并发	同进程内线程可并发
系统开销	创建撤销开销大	创建撤销开销小
地址空间	独立的地址空间	共享进程地址空间
通信方式	IPC 机制	直接读写进程数据

2.1.2 进程/线程的状态与转换

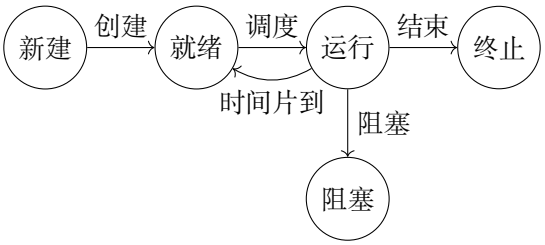
进程的三态模型：

- **就绪态 (Ready)**：已获得除 CPU 外的所有资源
- **运行态 (Running)**：正在 CPU 上执行
- **阻塞态 (Blocked)**：等待某个事件发生

进程的五态模型：

- **新建态 (New)**：进程正在被创建
- **就绪态 (Ready)**：等待分配 CPU
- **运行态 (Running)**：正在运行
- **阻塞态 (Waiting)**：等待事件
- **终止态 (Exit)**：进程结束

状态转换：



2.1.3 进程控制块 PCB

PCB 作用：

- 作为进程存在的唯一标志
- 记录进程的各种信息
- 供 OS 掌握进程情况

PCB 内容：

- **进程标识信息**：PID、用户 ID、组 ID
- **处理机状态信息**：通用寄存器、PC、PSW、栈指针
- **进程调度信息**：优先级、状态、队列指针
- **进程控制信息**：程序入口、代码数据地址、资源清单

PCB 组织方式：

- **链接方式**：同状态进程的 PCB 链成队列
- **索引方式**：建立各种状态的索引表

2.1.4 线程的实现方式

用户级线程：

- 由用户线程库管理
- 内核不知道线程存在
- 优点：切换快、可移植
- 缺点：一个线程阻塞全进程阻塞

内核级线程：

- 由内核管理
- 内核为每个线程建立 TCB
- 优点：一个线程阻塞不影响其他线程
- 缺点：切换开销大

混合模式：

- 用户级线程映射到内核级线程
- n:m 映射关系
- 结合两种方式的优点

2.2 处理机调度

2.2.1 调度的基本概念

调度层次：

- **高级调度 (作业调度)**：决定哪个作业进入内存
- **中级调度 (内存调度)**：决定哪个进程换入换出

- **低级调度 (进程调度)**：决定哪个就绪进程获得 CPU

调度时机：

- 当前运行进程主动放弃 CPU
- 当前运行进程因异常终止
- 当前运行进程被阻塞
- 当前运行进程时间片用完
- 有更高优先级进程就绪

调度方式：

- **非抢占式**：进程主动放弃 CPU
- **抢占式**：可强制进程放弃 CPU

2.2.2 调度算法

先来先服务 (FCFS)：

- 按到达时间排队
- 非抢占式
- 对长作业有利，对短作业不利
- 不考虑作业运行时间

短作业优先 (SJF)：

- 优先选择运行时间短的作业
- 平均等待时间最短
- 对长作业不利，可能饥饿
- 难以预知作业运行时间

高响应比优先 (HRRN)：

- 响应比 = (等待时间 + 服务时间) / 服务时间
- 综合考虑等待时间和服务时间
- 既考虑短作业又防止长作业饥饿
- 非抢占式算法

时间片轮转 (RR)：

- 每个进程分配一个时间片
- 时间片用完则轮转到下一个进程
- 抢占式算法
- 时间片大小影响系统性能

优先级调度：

- 根据优先级分配 CPU
- 可以是抢占式或非抢占式
- 静态优先级：优先级不变
- 动态优先级：优先级可调整

多级反馈队列：

- 设置多个就绪队列，优先级递减
- 新进程进入第 1 级队列
- 时间片用完降到下一级队列
- 兼顾短作业和长作业

调度算法比较：

算法	抢占	饥饿	适用环境	特点
FCFS	非抢占	无	批处理	简单，利于长作业
SJF	非抢占	可能	批处理	平均等待时间短
HRRN	非抢占	无	批处理	综合考虑多因素
RR	抢占	无	分时	响应时间好
优先级	两种	可能	实时	满足实时要求
多级反馈	抢占	无	分时	综合性能好

2.3 进程同步

2.3.1 同步与互斥的基本概念

临界资源：一次只允许一个进程使用的资源
临界区：访问临界资源的代码段
同步：协调多个进程的执行顺序
互斥：多个进程不能同时使用临界资源
临界区使用原则：

- **空闲让进**：临界区空闲时应允许进程进入
- **忙则等待**：已有进程在临界区时其他进程等待
- **有限等待**：等待进程应在有限时间内进入
- **让权等待**：等待时应释放 CPU

2.3.2 实现临界区互斥的基本方法

软件实现方法：
单标志法：

```
int turn = 0; // 0表示P0可进入，1表示P1可进入
// P0进程
while(turn != 0); // 等待
critical_section(); // 临界区
turn = 1; // 让对方进入
remainder_section();

// P1进程
while(turn != 1); // 等待
critical_section(); // 临界区
turn = 0; // 让对方进入
```

```
remainder_section();
```

问题：必须交替进入，违反空闲让进

双标志先检查：

```
bool flag[2] = {false, false};
// Pi进程
while(flag[j]); // 等待对方退出
flag[i] = true; // 表示自己要进入
critical_section();
flag[i] = false; // 表示自己退出
```

问题：可能两个进程都进入临界区

双标志后检查：

```
bool flag[2] = {false, false};
// Pi进程
flag[i] = true; // 表示自己要进入
while(flag[j]); // 等待对方退出
critical_section();
flag[i] = false; // 表示自己退出
```

问题：可能都无法进入临界区

Peterson 算法：

```
bool flag[2] = {false, false};
int turn = 0;
// Pi进程
flag[i] = true; // 表示自己要进入
turn = j; // 谦让对方
while(flag[j] && turn == j); // 等待
critical_section();
flag[i] = false;
```

满足所有要求，但忙等浪费 CPU

硬件实现方法：

中断屏蔽：

```
关中断;
critical_section();
开中断;
```

简单有效，但不适用于多处理机

TestAndSet 指令：

```
bool TestAndSet(bool *lock) {
    bool old = *lock;
    *lock = true;
    return old;
}
```

```
// 使用
bool lock = false;
while(TestAndSet(&lock));
critical_section();
lock = false;
```

Swap 指令：

```
void Swap(bool *a, bool *b) {
    bool temp = *a;
    *a = *b;
    *b = temp;
}
// 使用
bool lock = false;
bool key = true;
while(key) Swap(&lock, &key);
critical_section();
lock = false;
```

2.3.3 信号量

信号量定义：信号量是一个整型变量，除初始化外只能通过 P、V 操作访问。

P 操作 (wait)：

```
P(S) {
    S--;
    if(S < 0) {
        将进程加入等待队列;
        阻塞进程;
    }
}
```

V 操作 (signal)：

```
V(S) {
    S++;
    if(S <= 0) {
        从等待队列取出一个进程;
        唤醒该进程;
    }
}
```

信号量类型：

- **整型信号量**：可为任意整数
- **记录型信号量**：包含 value 和队列指针
- **二元信号量**：只能取 0 或 1，类似互斥锁

信号量应用：

实现互斥：

```
semaphore mutex = 1; // 互斥信号量
P(mutex);
critical_section();
V(mutex);
```

实现同步：

```
semaphore S = 0; // 同步信号量
// P1进程
statement1;
V(S);

// P2进程
P(S);
statement2; // 必须在statement1之后执行
```

2.3.4 管程

管程定义：管程是一种高级同步机制，由一组数据和定义在数据上的操作组成。

管程特性：

- 局部数据只能被管程内部过程访问
- 进程通过调用管程过程进入管程
- 同一时刻只有一个进程在管程内执行

条件变量：

- wait 操作：阻塞调用进程
- signal 操作：唤醒等待进程
- 只能在管程内使用

2.4 经典同步问题

2.4.1 生产者-消费者问题

问题描述：一组生产者进程和一组消费者进程共享一个有界缓冲区。

同步关系：

- 缓冲区满时生产者等待
- 缓冲区空时消费者等待
- 互斥访问缓冲区

信号量解法：

```
semaphore empty = n; // 空缓冲区数量
semaphore full = 0;  // 满缓冲区数量
semaphore mutex = 1; // 互斥访问

// 生产者
```

```
while(true) {
    produce_item();
    P(empty);
    P(mutex);
    put_item();
    V(mutex);
    V(full);
}

// 消费者
while(true) {
    P(full);
    P(mutex);
    get_item();
    V(mutex);
    V(empty);
    consume_item();
}
```

2.4.2 读者-写者问题

问题描述：多个读者和写者共享一个文件，读者只读文件，写者只写文件。

约束条件：

- 允许多个读者同时读
- 不允许读者和写者同时访问
- 不允许多个写者同时写

读者优先解法：

```
semaphore rw = 1;    // 读写互斥
semaphore count_mutex = 1; // 读者计数互斥
int read_count = 0;  // 读者数量

// 读者
P(count_mutex);
read_count++;
if(read_count == 1) P(rw); // 第一个读者申请
V(count_mutex);

reading();

P(count_mutex);
read_count--;
if(read_count == 0) V(rw); // 最后一个读者释放
V(count_mutex);

```

```
// 写者
P(rw);
writing();
V(rw);
```

2.4.3 哲学家进餐问题

问题描述： 5 个哲学家围坐圆桌，每人需要两把叉子才能进餐。

死锁解决方案：

方案 1：奇偶哲学家不同的取叉顺序

```
semaphore fork[5] = {1,1,1,1,1};

// 奇数号哲学家
P(fork[i]);
P(fork[(i+1)%5]);
eating();
V(fork[i]);
V(fork[(i+1)%5]);

// 偶数号哲学家
P(fork[(i+1)%5]);
P(fork[i]);
eating();
V(fork[(i+1)%5]);
V(fork[i]);
```

方案 2：最多允许 4 个哲学家同时拿叉子

```
semaphore fork[5] = {1,1,1,1,1};
semaphore count = 4; // 最多4个人

P(count);
P(fork[i]);
P(fork[(i+1)%5]);
eating();
V(fork[(i+1)%5]);
V(fork[i]);
V(count);
```

2.5 死锁

2.5.1 死锁的基本概念

死锁定义：在多道程序系统中，一组进程中的每个进程都在等待仅由该组中的其他进程才能引发的事件，则称该组进程发生了死锁。

死锁的必要条件：

1. **互斥条件**：资源不能被多个进程同时使用
2. **持有并等待条件**：进程已获得至少一个资源，又在申请其他资源
3. **不可剥夺条件**：已分配的资源不能被强制性地释放
4. **循环等待条件**：存在进程循环等待链

死锁相关概念：

- **饥饿**：进程长期得不到所需资源
- **死循环**：程序陷入无限循环
- **活锁**：进程没有阻塞但无法继续执行

2.5.2 死锁的处理策略

死锁预防：破坏死锁产生的四个必要条件之一

破坏互斥条件：

- 允许资源同时被多个进程使用
- 不适用于大多数资源（如打印机）
- 仅适用于可读共享资源

破坏持有并等待条件：

- **方案 1**：进程运行前一次性申请所有资源
- **方案 2**：进程申请新资源前必须释放已有资源
- 优点：简单，无死锁
- 缺点：资源利用率低，可能饥饿

破坏不可剥夺条件：

- 申请新资源失败时释放已有资源
- 适用于状态易保存的资源（CPU、内存）
- 不适用于打印机等资源

破坏循环等待条件：

- 对资源类型进行线性排序
- 进程必须按递增顺序申请资源
- 实现相对简单
- 限制用户编程，资源利用率低

死锁避免：

系统安全状态：

- 存在至少一个进程序列能顺利完成
- 安全状态一定无死锁
- 不安全状态可能发生死锁

银行家算法：用于多种资源的死锁避免

数据结构：

- **Available[m]**：可用资源向量
- **Max[n][m]**：最大需求矩阵
- **Allocation[n][m]**：分配矩阵
- **Need[n][m]**：需求矩阵，Need = Max - Allocation

安全性算法：

```
// 1. 初始化工作向量
Work = Available;
```

```
Finish[i] = false; // 对所有进程

// 2. 查找满足条件的进程
for(i = 0; i < n; i++) {
    if(Finish[i] == false && Need[i] <= Work) {
        // 3. 模拟分配资源
        Work = Work + Allocation[i];
        Finish[i] = true;
        goto step2; // 重新查找
    }
}

// 4. 检查结果
if(all Finish[i] == true)
    系统处于安全状态;
else
    系统处于不安全状态;
```

资源请求算法：

```
// 进程Pi请求资源Request[i]
// 1. 检查请求是否超过需求
if(Request[i] > Need[i])
    出错返回;

// 2. 检查请求是否超过可用资源
if(Request[i] > Available)
    等待;

// 3. 试分配资源
Available = Available - Request[i];
Allocation[i] = Allocation[i] + Request[i];
Need[i] = Need[i] - Request[i];

// 4. 执行安全性算法
if(安全状态)
    正式分配;
else {
    恢复原状态;
    等待;
}
```

银行家算法示例：

进程	Max	Allocation	Need	Available
P0	(7,5,3)	(0,1,0)	(7,4,3)	
P1	(3,2,2)	(2,0,0)	(1,2,2)	(3,3,2)
P2	(9,0,2)	(3,0,2)	(6,0,0)	
P3	(2,2,2)	(2,1,1)	(0,1,1)	
P4	(4,3,3)	(0,0,2)	(4,3,1)	

安全序列：P1 → P3 → P4 → P2 → P0

死锁检测：

资源分配图：

- 用有向图表示资源分配关系
- 进程 → 资源：请求边
- 资源 → 进程：分配边
- 图中有环则可能死锁

死锁检测算法：

```
// 类似银行家算法的安全性检测
Work = Available;
Finish[i] = false; // 初始化

for(i = 0; i < n; i++) {
    if(Finish[i] == false && Request[i] <= Work) {
        Work = Work + Allocation[i];
        Finish[i] = true;
    }
}

// 检查是否有死锁
if(存在 Finish[i] == false)
    系统发生死锁;
```

死锁解除：

进程终止：

- **终止所有死锁进程**：代价大但简单
- **逐个终止死锁进程**：每次终止一个直到死锁解除

资源抢占：

- 选择牺牲者：选择代价最小的进程
- 回滚：回滚到安全状态
- 饥饿：避免总是选择同一进程

死锁处理策略比较：

策略	实现难度	资源利用率	适用场景
鸵鸟策略	最简单	高	死锁很少发生的系统
死锁预防	简单	低	小型系统
死锁避免	复杂	中等	资源数量固定的系统
死锁检测	较复杂	高	大型系统

2.6 进程通信

2.6.1 进程通信的基本概念

进程通信 (IPC)：进程间传送信息的活动

通信分类：

- 按通信方式：直接通信、间接通信
- 按通信介质：共享存储、消息传递、管道通信

2.6.2 共享存储

基于共享数据结构：

- 进程共享某些数据结构
- 只适用于传递格式化数据
- 速度慢，限制多

基于共享存储区：

- 在内存中划出共享存储区
- 进程可直接访问
- 速度快，但需要同步机制

2.6.3 消息传递

消息传递特点：

- 无需共享变量
- 适用于分布式系统
- 通过 send/receive 原语通信

直接通信：

- 发送方明确指定接收方进程名
- 接收方明确指定发送方进程名
- 建立一对一通信链路

间接通信：

- 通过邮箱或端口通信

- 多个进程可共享一个邮箱
- 建立一对多或多对多链路

消息传递方式：

- **阻塞发送**：发送方等待接收方接收
- **非阻塞发送**：发送方不等待立即返回
- **阻塞接收**：接收方等待消息到达
- **非阻塞接收**：接收方立即返回

2.6.4 管道通信

匿名管道：

- 半双工通信
- 只能在父子进程间使用
- 存在于内存中
- 读一个写一个，边读边清空

命名管道 (FIFO)：

- 可在无关进程间通信
- 存在于文件系统中
- 有文件名，可以打开
- 先进先出

2.7 线程

2.7.1 线程的概念和多线程模型

多线程的优点：

- **响应性好**：一个线程阻塞其他线程可继续
- **资源共享**：线程共享进程资源
- **经济性**：创建切换开销小
- **可扩展性**：可利用多处理器

多线程模型：

多对一模型：

- 多个用户线程映射到一个内核线程
- 线程管理在用户空间进行
- 一个线程阻塞导致所有线程阻塞

一对一模型：

- 每个用户线程映射到一个内核线程
- 提供更好的并发性
- 创建用户线程需要创建内核线程

多对多模型：

- 多个用户线程映射到多个内核线程
- 结合前两种模型优点
- 实现复杂

第三章 内存管理

3.1 内存管理基础

3.1.1 内存管理基本概念

内存管理功能：

- **内存分配与回收**：为进程分配和回收内存
- **地址转换**：逻辑地址到物理地址的转换
- **内存保护**：防止进程越界访问
- **内存扩充**：虚拟存储技术扩充内存

3.1.2 程序装入与链接

编译链接过程：

源程序 → 编译 → 目标模块 → 链接 → 装入模块 → 装入 → 运行

程序的链接：

- **静态链接**：运行前链接成完整程序
- **装入时动态链接**：装入内存时进行链接
- **运行时动态链接**：执行时需要才链接

程序的装入：

- **绝对装入**：编译时产生绝对地址
- **可重定位装入**：装入时调整地址
- **动态运行时装入**：运行时进行地址转换

3.1.3 逻辑地址与物理地址

地址类型：

- **逻辑地址**：程序中的地址，相对地址
- **物理地址**：内存中的实际地址，绝对地址
- **虚拟地址**：虚拟存储器中的地址

地址转换：

逻辑地址 → MMU → 物理地址

3.2 连续分配管理方式

3.2.1 单一连续分配

特点：

- 内存分为系统区和用户区
- 用户区只有一个进程
- 无需内存保护
- 内存利用率低

3.2.2 固定分区分配

分区大小固定：

- 将用户区分为若干固定分区
- 每个分区装入一个进程
- 产生内部碎片
- 分区数量限制并发度

分区分配表：

分区号	大小	起始地址	状态
1	8K	64K	空闲
2	16K	72K	已分配
3	32K	88K	空闲

3.2.3 动态分区分配

基本思想：

- 分区大小不固定
- 根据进程需要动态分割
- 产生外部碎片
- 需要内存紧凑技术

分配算法：

首次适应 (First Fit)：

- 从头开始找第一个满足的空闲区
- 优点：简单，开销小
- 缺点：产生小碎片

最佳适应 (Best Fit)：

- 选择大小最接近的空闲区
- 优点：内存利用率高
- 缺点：产生很多小碎片，速度慢

最坏适应 (Worst Fit)：

- 选择最大的空闲区
- 优点：剩余空闲区较大
- 缺点：大空闲区很快用完

邻近适应 (Next Fit)：

- 从上次分配位置开始查找
- 优点：分配均匀
- 缺点：缺乏大的空闲区

3.3 分页管理方式

3.3.1 基本分页存储管理

基本概念：

- **页面**：逻辑地址空间分成固定大小的块
- **页框**：物理地址空间分成固定大小的块
- **页号**：页面的编号
- **页内偏移**：页面内的位移量

地址结构：

$$\text{逻辑地址} = \text{页号} + \text{页内偏移}$$

页号占 m 位，页内偏移占 n 位，则：

- 页面大小 = 2^n 字节
- 页面数量 ≤ 2^m 个

页表：

- 记录页号到页框号的映射
- 存放在内存中
- 由页表寄存器指示页表位置

地址转换过程：

1. 从逻辑地址中提取页号和页内偏移
2. 查页表得到页框号
3. 物理地址 = 页框号 + 页内偏移

3.3.2 基本地址变换机构

页表寄存器 (PTR)：

- 存放页表在内存中的起始地址
- 存放页表长度

地址变换步骤：

1. 检查页号是否越界
2. 查页表：页表始址 + 页号 × 页表项长度
3. 检查页框号是否合法
4. 计算物理地址：页框号 × 页面大小 + 页内偏移

3.3.3 具有快表的地址变换机构

快表 (TLB)：

- 高速缓冲存储器
- 存放常用的页表项
- 由相联存储器组成
- 按内容并行查找

地址变换过程：

1. 首先查快表
2. 若命中，直接得到页框号
3. 若未命中，查内存中的页表

4. 将页表项写入快表

有效访问时间：设快表命中率为 α，快表访问时间为 t，内存访问时间为 m，则：

$$\text{有效访问时间} = (t+m) + (1-)(t+2m) = t+2m-m$$

3.3.4 两级页表

两级页表结构：

- 页目录表：一级页表
- 页表：二级页表
- 减少页表占用的内存空间

地址结构：

$$\text{逻辑地址} = \text{页目录号} + \text{页表号} + \text{页内偏移}$$

地址转换：

1. 根据页目录号查页目录表
2. 得到页表的物理地址
3. 根据页表号查页表
4. 得到页框号
5. 计算物理地址

3.4 分段管理方式

3.4.1 基本分段存储管理

段的概念：

- 按程序逻辑结构划分
- 段长度可变
- 段内地址连续，段间可不连续
- 便于编程、修改、保护、共享

逻辑地址结构：

$$\text{逻辑地址} = \text{段号} + \text{段内偏移}$$

段表：

段号	段长	基址
0	1000	8000
1	2000	10000
2	1500	15000

地址转换：

1. 检查段号是否越界

2. 查段表得到段长和基址
3. 检查段内偏移是否越界
4. 物理地址 = 基址 + 段内偏移

3.4.2 段页式管理方式

基本思想：

- 结合分段和分页的优点
- 段内分页
- 地址转换需要段表和页表

地址结构：

$$\text{逻辑地址} = \text{段号} + \text{页号} + \text{页内偏移}$$

段表项：

- 页表长度
- 页表起始地址

地址转换：

1. 根据段号查段表
2. 得到页表起始地址和长度
3. 检查页号是否越界
4. 根据页号查页表
5. 得到页框号
6. 计算物理地址

3.5 虚拟内存管理

3.5.1 虚拟内存的基本概念

虚拟内存特征：

- **多次性**：作业分多次调入内存
- **对换性**：允许作业换入换出
- **虚拟性**：逻辑上扩充内存容量

实现技术：

- 请求分页系统
- 请求分段系统
- 请求段页式系统

3.5.2 请求分页管理方式

页表项结构：

页框号	状态位	访问字段	修改位	外存地址
frame	P	A	M	addr

缺页中断处理：

1. 查页表发现缺页
2. 产生缺页中断
3. 选择调入页面的物理块
4. 从外存调入页面
5. 修改页表

3.5.3 页面置换算法

页面置换时机：

- 发生缺页中断
- 内存已满需要调入新页面
- 选择被置换的页面

最佳置换算法 (OPT)：

- 置换将来最长时间不访问的页面
- 缺页中断次数最少
- 无法实现，仅作为性能评价标准

先进先出置换算法 (FIFO)：

- 置换最先进入内存的页面
- 实现简单，用队列管理
- 性能差，可能产生 Belady 异常
- Belady 异常：分配页框数增加，缺页率反而增加

最近最久未使用算法 (LRU)：

- 置换最近最长时间未访问的页面
- 性能接近 OPT
- 实现复杂，开销大

LRU 实现方式：

寄存器实现：

- 为每个页面配置移位寄存器
- 访问时最高位置 1，定期右移
- 寄存器值最小的页面是 LRU 页面

栈实现：

- 维护一个页面栈
- 访问页面时将其移到栈顶
- 栈底页面是 LRU 页面

时钟置换算法 (Clock)：

- 近似 LRU 算法
- 为每页设置使用位
- 访问时使用位置 1
- 置换时扫描，使用位为 0 则选中，为 1 则清 0

改进的时钟算法：

- 考虑使用位和修改位
- (0,0)：最近未使用且未修改，首选
- (0,1)：最近未使用但已修改，次选
- (1,0)：最近使用但未修改

- (1,1)：最近使用且已修改，最后选择

页面置换算法比较：

算法	实现难度	性能	特点
OPT	无法实现	最优	理论标准
FIFO	简单	差	可能 Belady 异常
LRU	复杂	好	开销大
Clock	简单	较好	近似 LRU

3.5.4 页面分配策略

驻留集：

- 进程在内存中的页面集合
- 驻留集太小：缺页频繁
- 驻留集太大：内存利用率低

分配策略：

- **固定分配**：进程创建时确定页框数
- **可变分配**：运行时可调整页框数

置换范围：

- **局部置换**：在本进程的页框中选择
- **全局置换**：在所有页框中选择

工作集：

- 进程在某时间窗口内访问的页面集合
- 工作集模型：$W(t,)$
- t：当前时刻，Δ：工作集窗口大小
- 基于程序局部性原理

缺页率置换：

- 缺页率 = 缺页次数 / 内存访问次数
- 缺页率过高：增加页框
- 缺页率过低：减少页框

抖动：

- 系统频繁进行页面置换
- CPU 利用率急剧下降
- 原因：并发进程太多或驻留集太小
- 解决：暂停部分进程或增加内存

第四章 文件管理

4.1 文件系统基础

4.1.1 文件的基本概念

文件定义：文件是具有文件名的相关信息的集合，是用户进行信息存储的基本单位。

文件属性：

- **文件名**：用户可见的文件标识
- **标识符**：系统内部文件标识
- **类型**：不同类型文件的分类
- **位置**：文件在存储设备上的位置
- **大小**：当前文件大小
- **保护**：访问权限控制
- **时间**：创建、修改、访问时间

文件类型：

- **按用途分类**：系统文件、用户文件、库文件
- **按保护级别**：只读文件、读写文件、可执行文件
- **按组织形式**：普通文件、目录文件、特殊文件

文件操作：

- **创建**：分配存储空间，建立目录项
- **删除**：回收存储空间，删除目录项
- **打开**：将文件控制块调入内存
- **关闭**：将文件控制块写回外存
- **读写**：按当前指针位置读写数据
- **重定位**：修改当前文件指针位置

4.1.2 文件的逻辑结构

无结构文件（流式文件）：

- 文件内容是字符流
- 没有任何结构
- 如文本文件、二进制文件

有结构文件（记录式文件）：

顺序文件：

- 记录按某种顺序排列
- 支持顺序存取和随机存取
- 查找效率：O(n)

索引文件：

- 为文件建立索引表
- 索引表包含关键字和记录地址
- 查找效率：O(log n)

索引顺序文件：

- 结合顺序和索引的优点

- 记录分组，每组建索引
- 组内顺序排列，组间索引查找

直接文件（散列文件）：

- 通过散列函数计算记录地址
- 查找效率：O(1)
- 可能产生冲突

4.1.3 目录结构

文件控制块 (FCB)：

- 存储文件管理信息的数据结构
- 包含文件属性、存储位置等信息
- UNIX 中称为 inode

目录结构类型：

单级目录：

- 所有文件在同一目录下
- 简单，但文件名易冲突
- 不适合多用户系统

两级目录：

- 主文件目录 (MFD) + 用户文件目录 (UFD)
- 解决文件名冲突
- 不同用户可有同名文件

多级目录（树形目录）：

- 目录形成树形结构
- 支持子目录嵌套
- 绝对路径：从根目录开始
- 相对路径：从当前目录开始

无环图目录：

- 允许文件有多个父目录
- 支持文件共享
- 通过链接实现
- 删除时需检查链接数

4.2 文件存储空间的管理

4.2.1 存储空间的划分与初始化

存储空间划分：

- **文件区**：存放文件数据
- **目录区**：存放文件目录

文件卷：

- 逻辑存储单位
- 可以是整个磁盘或磁盘分区
- 有独立的文件系统

4.2.2 存储空间管理

空闲表法：

- 为每个空闲区建立表项
- 记录起始地址和长度
- 适用于连续分配

空闲链表法：

- **空闲盘块链**：链接所有空闲盘块
- **空闲盘区链**：链接空闲盘区
- 分配时从链头摘下盘块

位示图法：

- 用位图表示盘块使用情况
- 0：空闲，1：已分配
- 查找效率高
- 适用于小型文件系统

成组链接法：

- 空闲盘块分组管理
- 第一个盘块存放下一组信息
- UNIX 系统采用此方法

4.3 文件存储结构

4.3.1 文件分配方式

连续分配：

- 文件占用连续的盘块
- 目录项记录起始地址和长度
- 优点：支持顺序和随机访问，读取效率高
- 缺点：产生外部碎片，文件长度难以确定

链接分配：

隐式链接：

- 每个盘块有指针指向下一盘块
- 目录项记录首末盘块地址
- 优点：无外部碎片
- 缺点：只支持顺序访问，可靠性差

显式链接（文件分配表 FAT）：

- 将指针集中存放在 FAT 中
- FAT 表项记录下一盘块号
- 优点：支持随机访问，可靠性好
- 缺点：FAT 占用内存空间

索引分配：

- 为每个文件建立索引表
- 索引表记录文件各盘块地址
- 目录项指向索引表
- 优点：支持随机访问，无外部碎片

- 缺点：索引表占用存储空间

多级索引：

- 建立多级索引结构
- 适用于大文件
- UNIX 的 inode 结构

混合索引：

- 直接地址：直接指向数据盘块
- 一级间接地址：指向一级索引表
- 二级间接地址：指向二级索引表
- 三级间接地址：指向三级索引表

4.3.2 文件存储结构比较

分配方式	访问方式	外部碎片	适用场景
连续分配	顺序 + 随机	有	文件大小已知且很少修改
链接分配	顺序	无	文件大小变化频繁
FAT	顺序 + 随机	无	小型文件系统
索引分配	随机	无	文件大小变化且需随机访问

4.4 磁盘组织与管理

4.4.1 磁盘的结构

磁盘物理结构：

- **盘片**：磁盘的存储介质
- **磁道**：盘片上的同心圆
- **扇区**：磁道的最小存储单位
- **柱面**：所有盘片同一位置磁道的集合
- **磁头**：读写磁性信息的部件

磁盘地址： (柱面号, 磁头号, 扇区号)

磁盘性能参数：

- **寻道时间**：磁头移动到指定磁道的时间
- **旋转延迟**：等待扇区转到磁头下的时间
- **传输时间**：数据传输的时间

磁盘访问时间 = 寻道时间 + 旋转延迟 + 传输时间

4.4.2 磁盘调度算法

先来先服务 (FCFS)：

- 按访问请求的先后顺序服务
- 公平，但平均寻道距离大

最短寻道时间优先 (SSTF)：

- 选择距当前磁头最近的请求
- 平均寻道时间短
- 可能导致饥饿现象

扫描算法 (SCAN)：

- 磁头在磁盘上来回扫描
- 边扫描边处理路径上的请求
- 又称电梯调度算法

循环扫描 (C-SCAN)：

- 磁头只向一个方向扫描
- 到达末端后直接回到起始端
- 提供更均匀的等待时间

LOOK 和 C-LOOK：

- 改进的 SCAN 和 C-SCAN
- 磁头到达最后一个请求就返回
- 不必到达磁盘端点

磁盘调度算法比较：

算法	性能	公平性	特点
FCFS	差	好	简单，无饥饿
SSTF	好	差	可能饥饿
SCAN	较好	较好	电梯算法
LOOK	较好	较好	仅在请求范围内来回扫描
C-SCAN	好	好	单向循环扫描等待更均匀
C-LOOK	好	好	单向扫描仅覆盖请求范围

第五章 I/O(输入/输出) 管理

5.1 I/O 管理概述

5.1.1 I/O 设备

I/O 设备分类：

按使用特性分类：

- **人机交互设备**：键盘、鼠标、显示器
- **存储设备**：磁盘、光盘、磁带
- **网络通信设备**：网卡、调制解调器

按传输速率分类：

- **低速设备**：键盘、鼠标（几字节/秒到几百字节/秒）
- **中速设备**：打印机、激光打印机（几千字节/秒）
- **高速设备**：磁盘、网卡（几兆字节/秒到几百兆字节/秒）

按信息交换单位分类：

- **块设备**：以数据块为单位，可随机访问（磁盘）
- **字符设备**：以字符为单位，只能顺序访问（键盘、打印机）

5.1.2 I/O 控制器

I/O 控制器功能：

- **接收和识别命令**：接收 CPU 发送的命令
- **报告设备状态**：向 CPU 报告设备状态
- **数据交换**：控制设备与内存间的数据传送
- **地址识别**：识别设备地址
- **数据缓冲**：缓解速度不匹配
- **差错控制**：检测和纠正传输错误

I/O 控制器组成：

- **设备控制逻辑**：控制设备工作
- **主机接口**：与主机通信
- **设备接口**：与设备通信
- **I/O 逻辑**：实现 I/O 操作

I/O 端口：

- **数据寄存器**：存放输入输出数据
- **状态寄存器**：反映设备当前状态
- **控制寄存器**：接收控制信号

5.1.3 I/O 控制方式

程序直接控制方式：

- CPU 直接控制 I/O 操作
- 采用程序查询方式
- 优点：硬件简单
- 缺点：CPU 利用率低，等待时间长

中断驱动方式：

- I/O 操作完成后发出中断信号
- CPU 响应中断处理 I/O
- 优点：CPU 利用率提高
- 缺点：每传送一个字符就中断一次

DMA 方式：

- 数据在内存和设备间直接传送
- 不需 CPU 干预每个数据传送
- 传送完成后产生中断
- 优点：CPU 利用率高，传输速度快
- 缺点：硬件复杂

通道控制方式：

- 通道是专用处理机
- 有自己的指令集
- 可控制多台设备
- CPU 只需启动通道

I/O 控制方式比较：

控制方式	CPU 参与度	数据传输	适用设备
程序直接控制	完全参与	逐字符	简单低速设备
中断驱动	部分参与	逐字符	中等速度设备
DMA	少量参与	成块	高速设备
通道控制	微量参与	成块	多设备系统

5.2 I/O 软件层次结构

5.2.1 I/O 软件层次

用户层 I/O 软件：

- 实现与用户交互的 I/O 软件
- 提供用户接口
- 如文本编辑器、图形界面

设备独立性软件：

- 实现设备独立的 I/O 功能
- 提供统一的 I/O 接口
- 功能：设备命名、设备保护、缓冲管理、错误处理

设备驱动程序：

- 与硬件相关的软件
- 直接控制 I/O 设备
- 将设备独立软件的命令转换为设备相关命令

中断处理程序：

- 处理 I/O 中断
- 唤醒被阻塞的进程
- 保存和恢复现场

硬件：

- I/O 设备和控制器
- 提供基本的 I/O 功能

5.2.2 设备独立性软件

设备独立性：

- 应用程序独立于具体的物理设备
- 通过逻辑设备名访问设备
- 由 OS 建立逻辑设备名与物理设备的映射

设备分配：

- **设备控制表 (DCT)**：描述设备特性
- **控制器控制表 (COCT)**：描述控制器状态
- **通道控制表 (CHCT)**：描述通道状态
- **系统设备表 (SDT)**：记录全部设备信息

设备分配策略：

- **静态分配**：进程运行前分配全部设备
- **动态分配**：需要时才分配设备

设备分配算法：

- 先请求先分配
- 优先级高者先分配
- 短任务优先分配

5.3 缓冲管理

5.3.1 缓冲的引入

缓冲作用：

- **缓解速度不匹配**：缓解 CPU 与 I/O 设备速度差异
- **减少中断频率**：积累数据批量处理
- **提高并行性**：CPU 和 I/O 并行工作
- **复制语义**：保证数据正确性

5.3.2 缓冲实现

单缓冲：

- 在内存中设置一个缓冲区
- 设备 → 缓冲区 → 用户区
- 处理时间：max(C, T) + M
- C：计算时间，T：I/O 时间，M：传送时间

双缓冲：

- 设置两个缓冲区交替使用

- 一个接收数据，一个传送数据
- 处理时间：max(C+M, T)
- 提高系统效率

循环缓冲：

- 多个缓冲区组成环形队列
- 输入指针和输出指针
- 进一步提高并行度

缓冲池：

- 系统中设置公共缓冲池
- 包含空缓冲队列、输入队列、输出队列
- 动态分配缓冲区
- 提高缓冲区利用率

5.4 磁盘存储器的管理

5.4.1 磁盘初始化

低级格式化：

- 将磁盘分成扇区
- 每个扇区有头部、数据区、尾部
- 在工厂完成

分区：

- 将磁盘分成一个或多个分区
- 每个分区可以有独立的文件系统
- 分区表记录分区信息

高级格式化：

- 创建文件系统
- 建立文件分配表、目录结构等
- 用户执行的格式化

5.4.2 引导块

引导过程：

1. 开机时运行 ROM 中的引导程序
2. 读取磁盘第一个扇区（引导扇区）
3. 执行引导扇区中的引导程序
4. 加载操作系统内核

主引导记录 (MBR)：

- 位于磁盘第一个扇区
- 包含分区表和引导代码
- 大小为 512 字节

5.4.3 坏块处理

扇区备用：

- 为每个扇区设置备用扇区

- 发现坏块时自动替换
- 对软件透明

扇区滑动：

- 将坏扇区后面的扇区前移
- 将备用扇区加在磁道末尾
- 保持扇区编号连续

交替扇区：

- 在每个磁道后设置几个备用扇区
- 坏扇区用备用扇区替代
- 简单但性能略有下降

5.5 固态硬盘

5.5.1 SSD 基本原理

NAND 闪存：

- 基于浮栅晶体管
- 通过电荷存储信息
- 非易失性存储

存储单元类型：

- **SLC**：单层单元，存储 1 位
- **MLC**：多层单元，存储 2 位
- **TLC**：三层单元，存储 3 位
- **QLC**：四层单元，存储 4 位

5.5.2 SSD 特点

优点：

- 随机访问性能好
- 无机械运动，低功耗
- 抗震动，体积小
- 启动时间短

缺点：

- 成本高于机械硬盘
- 写入次数有限
- 写入性能不如读取

SSD vs HDD：

特性	SSD	HDD
随机访问	快（0.1ms）	慢（5-10ms）
顺序访问	快	较快
功耗	低	高

特性	SSD	HDD
噪音	无	有
容量	较小	大
成本	高	低
寿命	写入次数限制	机械磨损

磨损均衡：

- 均匀分布写操作
- 避免某些块过度磨损
- 延长 SSD 使用寿命

第六章 重点公式汇总

6.1 进程调度公式

平均周转时间：

$$T = \frac{1}{n}\sum_{i=1}^{n} T_i$$

其中 T_i 为第 i 个作业的周转时间

周转时间：

$$T_i = \text{完成时间} - \text{到达时间}$$

带权周转时间：

$$W_i = \frac{T_i}{\text{服务时间}}$$

响应比：

$$\text{响应比} = \frac{\text{等待时间} + \text{服务时间}}{\text{服务时间}}$$

6.2 存储管理公式

页面大小计算：逻辑地址为 32 位，页号 20 位，页内偏移 12 位

$$\text{页面大小} = 2^{12} = 4KB$$

页表项数：

$$\text{页表项数} = 2^{\text{页号位数}}$$

有效访问时间：

$$EAT = \alpha \times (t + m) + (1 - \alpha) \times (t + 2m)$$

其中 α 为快表命中率，t 为快表访问时间，m 为内存访问时间

6.3 磁盘调度公式

磁盘访问时间：

$$T_{\text{访问}} = T_{\text{寻道}} + T_{\text{旋转}} + T_{\text{传输}}$$

平均旋转延迟：

$$T_{\text{旋转}} = \frac{1}{2} \times \frac{60}{RPM}$$

传输时间：

$$T_{\text{传输}} = \frac{\text{传输字节数}}{\text{传输速率}}$$

第七章 核心概念对比

7.1 进程 vs 线程

对比项	进程	线程
定义	程序的一次执行实例	进程内的执行单元
资源拥有	拥有独立资源	共享进程资源
调度单位	传统调度单位	现代调度单位
并发性	进程间并发	同进程内线程并发
地址空间	独立地址空间	共享进程地址空间
通信开销	大（需 IPC）	小（直接访问）
创建开销	大	小
切换开销	大	小

7.2 分页 vs 分段 vs 段页式

对比项	分页	分段	段页式
地址结构	页号 + 页内偏移	段号 + 段内偏移	段号 + 页号 + 页内偏移
大小特点	页面大小固定	段长度可变	段长可变, 页面定长
逻辑单位	物理划分	逻辑划分	结合两者优点
内存利用	无外部碎片	有外部碎片	无外部碎片
共享保护	较难实现	容易实现	容易实现
访问速度	快	快	较慢（两次查表）

7.3 常见错误总结

进程状态转换错误：

- 就绪态不能直接转到阻塞态
- 阻塞态不能直接转到运行态
- 运行态转就绪态由时间片用完或高优先级进程抢占

PV 操作错误：

- P 操作在临界区前，V 操作在临界区后
- 实现同步时，V 操作在前，P 操作在后
- 多个信号量时注意操作顺序，避免死锁

死锁理解错误：

- 死锁四个条件必须同时满足
- 银行家算法是死锁避免，不是预防
- 安全状态不等于无死锁，不安全状态不等于有死锁
- 资源分配图有环不一定死锁（单资源类型才成立）

页面置换算法理解错误：

- OPT 是理想算法，无法实现
- FIFO 可能产生 Belady 异常
- LRU 性能好但实现复杂
- Clock 算法是 LRU 的近似