《操作系统》总复习

Eslzzyl

2022 年 12 月 6 日

目录

考试题型

- 主观题 50%：简答、计算、问答
- 客观题 50%：单选、填空、判断

本材料的内容排布原则：以课本章节编号为准，只列出 PPT 有的内容

一 操作系统引论

操作系统的**定义**：操作系统是一组管理和控制计算机软件和硬件资源，合理组织计算机系统工作流程，以及方便用户使用的程序的集合。

1.1 操作系统的目标和作用

1.1.1 操作系统的目标

- 方便性：使计算机更易使用
- 有效性：使资源利用更高效
- 可扩充性
- 开放性

1.1.2 操作系统的作用

- 作为用户与计算机硬件系统之间的接口
- 作为计算机系统资源的管理者
- 实现对计算机资源的抽象

1.2 操作系统的发展过程

推动 OS 发展的主要动力：

- 不断提高计算机资源利用率
- 方便用户
- 器件的不断更新换代
- 计算机体系结构的发展
- 不断提出新的应用需求

1.2.1 无操作系统时代

对计算机的所有操作采用人工操作方式完成。人工操作方式的缺点：

- 用户独占全机
- CPU 等待人工操作

由于以上两点，昂贵的机器资源在大部分时间内处于空闲状态，非常浪费。
优点：交互性好，即用户可以得到机器的立即响应。

1.2.2 单道批处理系统

批处理系统：在计算机上加载一个专门监控软件，在其控制下，计算机能够自动地、成批地处理一个或多个用户的一批作业。

批处理系统的特点：

- 系统吞吐量大
- 资源利用率高
- 平均周转时间短
- 无交互能力（缺点）

fd **单道批处理系统**：先把一批作业输入到磁带上，在监控程序的控制下使这批作业一个接一个地处理。内存中始终只保持一道作业。

代表系统：IBM 的 FMS（Fortran Monitoring System），1960s

一定程度上改善了资源浪费，但失去了交互性。单个程序独占系统，程序 IO 时 CPU 空等。

单道批处理系统的特点：自动性、顺序性、单道性

1.2.3 多道批处理系统

出现的动力是希望提高资源利用率和系统吞吐量。

代表系统（也是第一个）：IBM 的 OS/360，1960s

由作业调度程序按照一定的算法，从后备队列中选择若干个作业调入内存，使它们共享 CPU 资源。

相比单道批处理系统，多道批处理系统的资源利用率更高，但仍然没有交互性。

特点：多道性、无序性、调度性

1.2.4 分时系统

出现的动力是改善交互性。

分时系统：在一台主机上连接若干个终端，允许多个用户通过终端以交互方式共享主机资源。

代表系统：CTSS(1962)、Multics(1964)

特点：

- 多路性：众多联机用户可以同时使用同一台计算机
- 独立性：各终端用户感觉到自己独占了计算机
- 及时性：用户的请求能在很短时间内得到响应
- 交互性：用户与计算机之间可进行“会话”

1.2.5 实时系统

实时计算：系统的正确性不仅由计算的逻辑结果来确定，而且还取决于产生结果的时间。

常见的实时系统：

- 工业控制系统、武器控制系统
- 信息查询系统
- 多媒体系统
- 嵌入式系统

代表系统：WinCE，嵌入式 Linux，ucOSII，VxWorks 等

特点：实时性、高安全性、高可靠性（效率放第二位）、整体性强、会话能力要求不高

1.3 操作系统的基本特性

四大基本特征

- 并发（Concurrence）
- 共享（Sharing）
- 虚拟（Virtual）
- 异步（Asynchronism）

1.3.1 并发

- 并行性：多个事件在同一时刻发生
- 并发性：多个事件在同一时间间隔内发生

单处理机可以实现并发（宏观上并发，微观上分时运行），多处理机可以实现并行。

1.3.2 共享

即：系统中的资源可供内存中多个并发执行的进程共同使用。
资源共享方式：

- 互斥共享方式：如打印机、磁带机
- 同时访问方式：如磁盘、内存

并发和共享是多用户 OS 的最基本特征。

1.3.3 虚拟

指通过某种技术把一个物理实体变为若干个逻辑上的对物。

1.3.4 异步

进程的异步性：进程是以人们不可预知的速度向前推进的。进程的执行通常都不是“一气呵成”的，而是“停停走走”

1.4 操作系统的主要功能

- 处理机管理功能
  - ◇ 进程控制
  - ◇ 进程同步
  - ◇ 进程通信
- 存储器管理功能
  - ◇ 内存分配
  - ◇ 内存保护
  - ◇ 地址映射
  - ◇ 内存扩充
- 设备管理功能
  - ◇ 缓冲管理
  - ◇ 设备分配
  - ◇ 设备处理
- 文件管理功能
  - ◇ 文件存储空间的管理
  - ◇ 目录管理
  - ◇ 文件的读写管理和保护
- 操作系统接口
  - ◇ 用户接口
  - ◇ 程序接口

1.5 操作系统结构

1.5.1 无结构 OS

基本上没有优点。

- 操作系统庞大杂乱, 缺乏清晰的程序结构
- 操作系统的编写、调试、维护复杂，保证操作系统程序的正确困难

1.5.2 模块化结构 OS

优点:

- 提高 OS 设计的正确性、可理解性和可维护性
- 增强 OS 的可移植性, 加速 OS 的开发过程

缺点:

- 结构划分和接口设计困难
- 模块之间调用关系复杂，牵一发而动全身

1.5.3 分层式结构 OS

原则：将功能模块分层，层内模块可以互相调用，层间模块只允许高层调用它下面紧邻着的那层，不允许反过来。

将最靠近硬件的低层 OS 部分称为操作系统内核，内核通常常驻主存。

优点：

- 容易保证系统的正确性。
- 容易扩充、维护和移植。

缺点：效率低下。

1.5.4 微内核 OS

将 OS 划分为两大部分：

- 内核，只提供客户-服务器通信机制和与硬件紧密相关的较基本的功能
- 提供各类服务的一组服务器进程

OS 采用 C/S 模式提供各类操作系统服务。

优点：

- 灵活、容易移植、容易扩展、可靠
- 支持分布式系统

缺点：还是效率低下。

二 进程管理

2.1 前趋图和程序执行

2.1.1 前趋图

前趋图：用于描述实体 (进程、程序段) 之间执行次序的 DAG。每个结点表示一个进程或程序段，乃至一条语句。结点之间的有向边表示两个结点之间存在偏序或前趋关系。

把没有前趋的结点成为初始结点，把没有后继的结点称为终止结点。

2.1.2 程序的顺序执行

特征：

- 顺序性：一个操作做完之后才进行下一个操作
- 封闭性：一旦开始执行，其结构不受外界因素影响
- (结果的) 可再现性

2.1.3 程序的并发执行

只有不存在前趋关系的程序之间才有可能并发执行。

特征：

- 间断性
- 失去封闭性
- 不可再现性

2.2 进程的描述

2.2.1 进程的定义和特征

(传统 OS 中) 进程的**定义**：进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位。

进程实体：程序段 + 相关的数据段 +PCB
进程的特征：

- 动态性：进程有生命周期，而程序只是一组有序指令的集合，是静态的。
- 并发性：指多个进程可以同时存在于内存中，且能在一段时间内同时运行。
- 独立性：进程实体是一个能独立运行、独立获得资源和独立接受调度的基本单位。
- 异步性：进程按照各自独立的、不可预知的速度向前推进。

2.2.2 进程的基本状态及转换

三种基本状态：

- 就绪（Ready）状态：进程已经得到除了 CPU 之外的所有资源，一旦得到 CPU 就立即开始执行。
- 执行（Running）状态：单处理机系统中只有一个进程处于执行状态，多处理机系统中有多个进程处于执行状态。
- 阻塞（Blocking）状态：指正在执行的进程由于发生某事件为暂时无法继续执行的状态。此时会引发进程调度。

三种基本状态的转换：参见课本 P40 图 2-5

2.2.3 PCB

OS 管理的数据结构：

- 内存表
- 设备表
- 文件表
- 进程表

进程表就是 PCB。本节只介绍 PCB。
PCB 进程存在的唯一标志。

PCB 中的内容：

- 进程标识符
- 处理机状态
- 进程调度信息
- 进程控制信息

PCB 的组织方式：

- 线性方式
- 链接方式
- 索引方式

2.3 进程控制

待补

2.4 进程同步

2.4.1 进程同步的基本概念

两种形式的制约关系：

- 间接相互制约关系：共享临界资源引起
- 直接相互制约关系：直接依赖引起

临界资源：一次仅能为一个进程所使用的资源。
实例：

- 硬件资源：打印机
- 软件资源：所有**变**量、指针、数组等

临界区：每个进程中访问临界资源的那部分代码。
临界区访问控制模型：在临界区前面加入代码（进入区），在临界区后也加入代码（退出区）
同步机制应遵循的原则：

1. 空闲让进：当无进程处于临界区时，请求进入临界区的进程可立即进入
2. 忙则等待：当已有进程进入临界区时，其他试图进入临界区进程须等待

3. 有限等待：对要求访问临界资源进程，保证能在有限时间内进入临界区

4. 让权等待：当进程不能进入临界区时，应释放处理机

其中1和2是必须实现的，否则无法实现互斥；3和4不实现不会影响互斥，但会严重影响系统的性能。

解决同步问题的方法：

- 软件方法：如 Peterson 算法
- 硬件方法：
  - ◇ 关中断
  - ◇ 利用 Test-and-Set 指令
  - ◇ 利用 Swap 指令

早期的进程同步软件方法：

- 按需访问：违背忙则等待、有限等待、让权等待
- 轮询：违背让权等待、有限等待，同时严格限制了资源访问顺序
- 访前先看：违背空闲让进、有限等待、让权等待

2.4.2 信号量机制

1965 年荷兰学者 Dijkstra 提出。

整数型信号量：

- 定义整数信号量 S，表示可用资源数量
- 定义两个原子操作：wait/signal（P/V）
- 在 critical section 前后使用 wait(S) 和 signal(S)

```
1   VAR S:integer := 1;
2   function Wait ( var S: integer )
3   Begin
4       while ( S <= 0 );
5       S := S-1;
6   end
7
8   function Signal( var S : integer )
9   Begin
10      S := S+1;
11  end
```

记录型信号量:

- 定义 S 和 wait/signal 同上
- 进程无法访问临界资源时，阻塞自身；释放临界资源时，如果有进程还在阻塞，则唤醒该进程。

```
1   type semaphore = record
2       value: integer;      // 资源数量
3       L: list of process; // 阻塞进程队列
4   end
5
6   function wait (var S : Semaphore)
7   Begin
8       S.value := S.value-1;
9       if ( S.value < 0 ) then block (S.L);
10  end
11
12  function signal(var S : Semaphore)
13  Begin
14      S.value := S.value+1;
```

```
15          if ( S.value <= 0 ) then wakeup (S.L);
16      end
```

- S.value >= 0 时，表示资源数量。
- S.value < 0 时，其绝对值表示被阻塞进程的数量。

2.5 经典的进程同步问题

2.5.1 生产者-消费者问题

单生产者单消费者、多缓冲：

```
1       VAR mutex, empty, full: semaphore := 1, n, 0 ;
2           in,out: integer := 0, 0 ;
3           Buffer: array [0..n-1] of item ;
4       Parbegin
5           Producer:
6           begin
7               repeat
8                   produce an item in nextp ;
9                   wait( empty ) ;
10                  wait( mutex ) ;
11                  Buffer(in) := nextp ;
12                  in := (in + 1 ) mod n ;
13                  signal( mutex ) ;
14                  signal( full ) ;
15              until false
16          end
17          Consumer:
18          begin
19              repeat
20                  wait( full ) ;
```

```
21                  wait( mutex ) ;
22                  nextc = Buffer(out);
23                  out := (out + 1 ) mod n ;
24                  signal( mutex ) ;
25                  signal( empty ) ;
26                  consume the item nextc ;
27              until false
28          end
29      Parend
```

多生产者多消费者、单缓冲：删去上面代码的 mutex。

单生产者单消费者、单缓冲：删去上面代码的 mutex，将 empty 的初值改成 1。

允许生产者写时，消费者可读：

```
1       VAR mutexC, mutexP, empty, full: semaphore := 1,1, n, 0 ;
2           in,out: integer := 0, 0 ;
3           Buffer: array [0..n-1] of item ;
4       Parbegin
5           Producer:
6           begin
7               repeat
8                   produce an item in nextp ;
9                   wait( empty ) ;
10                  wait( mutexP ) ;
11                  Buffer(in) := nextp ;
12                  in := (in + 1 ) mod n ;
13                  signal( mutexP ) ;
14                  signal( full ) ;
15              until false
16          end
17          Consumer:
18          begin
```

```
            repeat
                wait( full ) ;
                wait( mutexC ) ;
                nextc = Buffer(out);
                out := (out + 1 ) mod n ;
                signal( mutexC ) ;
                signal( empty ) ;
                consume the item nextc ;
            until false
        end
    Parend
```

缓冲池无限大:

```
    VAR mutex, full: semaphore := 1, 0 ;
        in,out: integer := 0, 0 ;
        Buffer: array [0..n-1] of item ;
    Parbegin
        Producer:
        begin
            repeat
                produce an item in nextp ;
                wait( mutex ) ;
                Buffer(in) := nextp ;
                in := (in + 1 ) mod n ;
                signal( mutex ) ;
                signal( full ) ;
            until false
        end
        Consumer:
        begin
            repeat
                wait( full ) ;
```

```
20                  wait( mutex ) ;
21                  nextc = Buffer(out);
22                  out := (out + 1 ) mod n ;
23                  signal( mutex ) ;
24                  consume the item nextc ;
25              until flase
26          end
27      Parend
```

2.5.2 读者-写者问题

允许多个读者同时访问资源，写者写时不允许其他进程读或写。

```
1       VAR rmutex,wmutex: semaphore := 1, 1 ;
2           readcount: integer := 0 ;
3       Parbegin
4           Reader:
5           begin
6               repeat
7                   wait( rmutex ) ;
8                   if ( readcount = 0 ) wait( wmutex ) ;
9                   readcount := readcount + 1 ;
10                  signal( rmutex ) ;
11                  perform read operation ;
12                  wait( rmutex ) ;
13                  readcount := readcount - 1 ;
14                  if ( readcount = 0 ) signal( wmutex ) ;
15                  signal( rmutex ) ;
16              until false
17          end
18          Writer:
19          begin
```

```
20              repeat
21                  wait( wmutex ) ;
22                  perform write operation;
23                  signal( wmutex ) ;
24              until flase
25          end
26      Parend
```

2.5.3 哲学家进餐问题

```
1       VAR chopsticks: array [0..4] of semaphore := 1,1,1,1,1 ;
2       Parbegin
3           philosopher0-3:
4           begin
5               repeat
6                   wait( chopsticks[ i ] ) ;
7                   wait( chopsticks[ (i+1) mod 5 ] );
8                   eat ;
9                   signal( chopsticks[ i ] ) ;
10                  signal( chopsticks[ (i+1) mod 5 ] );
11                  think
12              until false
13          end
14          philosopher4:
15          begin
16              repeat
17                  wait( chopsticks[ (i+1) mod 5 ] );
18                  wait( chopsticks[ i ] ) ;
19                  eat ;
20                  signal( chopsticks[ i ] ) ;
21                  signal( chopsticks[ (i+1) mod 5 ] );
```

```
22                  think
23              until false
24          end
25      Parend
```

原则：按序申请资源

2.5.4 理发师问题

上课没讲，说是课后思考

2.6 进程通信

2.6.1 共享存储器系统

分类

- 基于共享数据结构的通信方式
- 基于共享存储区的通信方式

特点

- 适合大量的数据快速交换
- 程序员负担重

2.6.2 管道通信系统

通过一个固定大小的共享文件实现通信。UNIX 首创。

2.6.3 消息通信系统

当前应用最广泛。
分类：

- 直接通信方式
- 间接通信方式：通过信箱 (mailbox) 通信

- ◇ 公有信箱
- ◇ 私有信箱
- ◇ 共享信箱

特点:

- 操作系统隐藏通信细节
- 通信程序简单
- 可以跨网络传输
- 灵活

2.7 线程的基本概念

2.7.1 线程的引入

传统进程的两个基本属性

- 是拥有资源的独立单位
- 是调度和分派的基本单位

线程的特点:

- 是能独立运行的基本单位，且切换代价远小于进程。
- 线程不拥有系统资源，而是仅拥有一点必不可少的、能保证独立运行的资源。
- 多个线程可以共享一个进程的资源

2.8 线程的实现

2.8.1 内核支持线程 KST

在内核的支持下运行。调度以线程为单位进行。

2.8.2 用户级线程 ULT

无须内核支持，与内核无关，甚至内核不知道此类线程的存在。调度以进程为单位进行。

三 处理机调度与死锁

3.1 处理机调度的层次和调度算法的目标

3.1.1 处理机调度的层次

- 高级调度：又称长程调度、作业调度、接纳调度。
  - ◇ 调度对象是作业。
  - ◇ 根据某种算法，决定将外存上处于后备队列中的哪几个作业调入内存。
  - ◇ 设置在多道批处理系统中。分时系统和实时系统不设置。
  - ◇ 调度频率低，典型时间：几分钟。
  - ◇ 因为调度频率低，调度算法可以设计得很复杂。
- 低级调度：又称进程调度或短程调度。
  - ◇ 调度对象是就绪的进程或内核级线程。
  - ◇ 根据某种算法，决定就绪队列中的哪个进程获得处理机。
  - ◇ 这是最基本的调度方式，在各类 OS 中都必须设置。
  - ◇ 调度频率高，典型时间：几十毫秒。
  - ◇ 因为调度时间短，调度算法必须足够简单。
- 中级调度：又称内存调度。
  - ◇ 调度对象是就绪进程和阻塞进程。
  - ◇ 其实就是内存对换功能。引入的主要目的是提高内存利用率和系统吞吐量。

3.1.2 处理机调度算法的目标

- 面向用户的准则
  - ◇ 响应时间快
  - ◇ 截止时间有保证

- 面向系统的准则
  - ◇ 系统吞吐量高
  - ◇ 处理机利用率高
  - ◇ 资源利用均衡

例题 1. 单选：进程调度的准则不能是（ ）

A. 最大的内存利用率

B. 最短的周转时间

C. 最大的 CPU 利用率

D. 最短的等待时间

解答： A

3.1.3 周转时间

周转时间：从作业被提交给系统开始，到作业完成为止的时间。= 等待时间 + 执行时间

设 T_i 是第 i 个作业的周转时间，n 为作业总数，则系统的平均周转时间定义为

$$T = \frac{1}{n}\left[\sum_{i=1}^{n} T_i\right] \tag{1}$$

再设 T_{si} 是第 i 个作业的要求服务时间，则系统的平均带权周转时间定义为

$$W = \frac{1}{n}\left[\sum_{i=1}^{n} \frac{T_i}{T_{si}}\right] \tag{2}$$

3.2　作业与作业调度

3.2.1　先来先服务 (FCFS) 调度算法

FCFS(first-come, first-served) 是最简单的调度算法。即可用于作业调度，也可用于进程调度。

特点：

- 实现简单
- 对短作业不公平

3.2.2　短作业/短进程优先 (SJF/SPF) 调度算法

SJF(short job first)、SPF(short process first)

以作业要求的服务时间长短计算优先级。作业越短，优先级越高。

特点：

- 实现困难，因为难以估计作业的执行时间。
- 有利于短作业。
- 对长作业不公平。

3.2.3　优先级调度算法 (PSA)

PSA（priority-scheduling algorithm）

由外部赋予作业相应的优先级，根据该优先级调度。高优先级作业优先运行。

3.2.4　高响应比优先调度算法 (HRRN)

HRRN（Highest Response Ratio Next）

基本思想是动态优先级，作业等待时间越长，优先级越高。

$$\text{响应比}R_p = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}} = \frac{\text{响应时间}}{\text{要求服务时间}}$$

以响应比作为优先级。

特点：

- 既照顾了短作业，又考虑了作业到达的先后次序，不会使长作业长期得不到服务。
- 每次调度之前要计算响应比，系统开销加大。

例题 2. 5 个进程 A,B,C,D,E 分别于 0,1,2,3,4 时刻到达系统，要求服务时间分别为 4,3,5,2,4, 请计算：

(1). 采用 FCFS 时进程执行顺序和平均周转时间；

(2). 采用 SPF 时进程执行顺序和平均周转时间；

(3). 采用 HRRN 时进程执行顺序和平均周转时间。

解答：

(1). 进程执行顺序：A→B→C→D→E
平均周转时间：T = (4+6+10+11+14) / 5 = 9
平均带权周转时间：W=(4/4+6/3+10/5+11/2+14/4) / 5 = 2.8

(2). 进程执行顺序：A→D→B→E→C
平均周转时间：T= (4+8+16+3+9) / 5 = 8
平均带权周转时间：W=(4/4+8/3+16/5+3/2+9/4) / 5 ≈ 2.12

(3). 待补

3.3 进程调度

3.3.1 进程调度方式

- 非抢占方式：进程得到处理机后就一直运行，直到运行时间片结束。提前结束时间片的唯一可能是进程阻塞。

- 抢占方式：某些情况下暂停正在执行的进程，转而执行另一个进程。现代 OS 普遍采用抢占方式。

3.3.2　时间片轮转调度算法 (RR)

让就绪队列上的每个进程每次运行一个时间片长度。
时间片一般取略大于一次典型的交互所需要的时间，以保证及时响应。

- 时间片太长，则退化为 FCFS 算法
- 时间片太短，调度频繁，浪费资源

3.3.3　优先级调度算法

参考作业调度的3.2.3节。
可以抢占，也可以非抢占。

3.3.4　多级反馈队列 (MFQ) 调度算法

MFQ（multileved feedback queue）
有如下规则：

- 设置多个就绪队列，为每个队列赋予不同的优先级，第一个队列最高，第二个次之，以此类推。
- 优先级越高的队列，执行时间片越小。
- 每个队列都采用 FCFS 算法。一个新进程到来时，将其放到第一个队列的末尾。轮到它执行时，如果不能在一个时间片内完成，则放入第二队列末尾，以此类推。直到放入第 n 个队列，按照 RR 算法运行。
- 仅当第 1 队列空闲时，调度程序才调度第 2 队列中的进程运行；仅当第 1 ~ $(i-1)$ 队列均空时，才会调度第 i 队列中的进程运行。
- 若第 i 队列的进程正在执行，此时有新进程进入优先级更高的队列，则立即暂停当前进程并将其放入第 i 队列的末尾，转而执行新进程。

例题 3.　在分时系统中，假设就绪队列中有 10 个进程，系统将时间片设为 200ms，CPU 进行进程切换要花费 10ms。则系统开销所占的比率约为（5%）

解答：假设 10 个进程都运行过一个时间片，调度这些进程需要 10 次切换（调度到第一个进程需要一次切换），共 $10\times 10\text{ms}=100\text{ms}$。总时间为 $200\times 10+100\text{ms}=2100\text{ms}$，因此比率为 $100/2100\times 100\%=4.8\%$

例题 4.　下列选项中，满足短任务优先且不会发生饥饿现象的调度算法是（　）

A. 先来先服务

B. 高响应比优先

C. 时间片轮转

D. 非抢占式短任务优先

解答： B

3.4　实时调度

HRT：硬实时任务，必须在截止时间之前完成，否则有严重后果。

SRT：软实时任务，一般稍微错过截止时间也没关系。

假定系统中有 m 个周期性的硬实时任务，它们的处理时间可表示为 C_i，周期时间表示为 P_i，则必须满足下面的限制条件，系统才是可调度的：

$$\sum_{i=1}^{m}\frac{C_i}{P_i}\leq N$$

其中 N 是处理机数量。

3.4.1　最早截止时间优先（EDF）算法

EDF: Earliest Deadline First

根据任务的截止时间来确定任务的优先级，截止时间愈早，其优先级愈高。

3.4.2　最低松弛度优先（LLF）算法

LLF: Least Laxity First

定义

$$\text{松弛度} = \text{必须完成时间} - \text{本身运行时间} - \text{当前时间}$$

松弛度越小，说明任务越紧迫。调度程序总是优先执行松弛度最小的任务。

3.4.3　优先级倒置

优先级倒置问题的原因是高优先级进程（或线程）被低优先级进程（或线程）延迟或阻塞。

解决优先级倒置问题的方法：

- 规定进程进入临界区后处理机就不允许被抢占。这种办法在临界区较长时不可行。
- 规定高优先级进程试图使用临界资源而已有一个低优先级进程正在使用时，令低优先级进程继承高优先级进程的优先级。

3.5　死锁概述

3.5.1　死锁的定义

死锁的**定义**：如果一组进程中的每一个进程都在等待仅由该组进程中的其他进程才能引发的事件，那么该组进程是死锁的（Deadlock）。

3.5.2　产生死锁的原因

- 竞争非剥夺性资源
- 进程推进顺序不当

3.5.3 产生死锁的必要条件

必须同时具备才可能死锁

- 互斥条件：某一段时间内资源只能被一个进程使用，其他进程想用时必须等待。
- 请求和保持条件：进程已经得到一部分资源，还需要另一部分资源，但请求的资源已无空闲，因此进程等待，但不释放已经占有的资源。
- 不可抢占条件：进程占有资源时，资源不能抢占，只能等进程主动释放。
- 循环等待条件：死锁时必然有若干个进程形成了循环等待。

3.5.4 处理死锁的方法

- 预防死锁：部分或全部破坏死锁的四个条件。优点是简单，缺点是资源利用率低，效率低。
- 避免死锁：进程申请资源时，检查安全性，只有安全才分配资源。优点是效率高一些，缺点是实现困难。
- 检测和解除死锁：不做任何限制，定时或资源不足时检查系统中有无死锁，有则解除。优点是效率高。

3.6 预防死锁

预防死锁就是破坏死锁的四个必要条件中的至少一个。互斥条件是不能改变的，因此主要考虑其他三个条件。

3.6.1 破坏“请求和保持”条件

进程创建时，一次申请所有资源。成功则运行，否则阻塞。
优点：简单，容易实现，安全。
缺点：资源严重浪费，进程执行大大推迟。

3.6.2　破坏“不可抢占”条件

进程申请资源时，成功则运行，否则释放所有资源后阻塞。

没有优点。

缺点：资源严重浪费、代价太大、进程执行进度严重延迟。

3.6.3　破坏“循环等待条件”

为所有资源编号，进程申请资源时必须按序申请资源。

优点：资源利用率、系统吞吐量显著提高。

缺点：还是有资源浪费。

3.7　避免死锁

3.7.1　系统安全状态

安全状态下，不会发生死锁；不安全状态下，可能发生死锁。

所谓安全状态，是指系统能按照某种进程推进顺序 $(P_1, P_2, \cdots, P_n)$ 为每个进程 P_i 分配其所需资源，直至满足每个进程对资源的最大需求，使每个进程都顺利完成。此时称 $(P_1, P_2, \cdots, P_n)$ 为安全序列。安全序列是不唯一的。

3.7.2　银行家算法

由 Dijkstra 提出。

在系统中设置四个数据结构：

- 可用资源向量 Available：长度为 m 的数组，其中每个元素代表一类可利用的资源数目。
- 最大需求矩阵 Max：n×m 的矩阵，表示系统中的 n 个进程每个对 m 类资源的最大需求。
- 分配矩阵 Allocation：n×m 的矩阵，表示当前已分配的资源情况。
- 需求矩阵 Need：n×m 的矩阵，表示每个进程还需要的资源数量。

上述矩阵存在如下关系：

$$\text{Need[i,j]} = \text{Max[i,j]} - \text{Allocation[i,j]}$$

银行家算法：设 Request_i 是进程 P_i 的请求向量。当 P_i 发出资源请求后，系统按照如下步骤进行检查：

(1). 若 $\text{Request}_i \leq$Need[i,j]，转 (2)；否则出错。

(2). 若 $\text{Request}_i \leq$Available[j]，转 (3)；否则表示资源不够，P_i 须等待。

(3). 系统试着把资源分给 P_i，并进行如下修改：

$$\text{Available[j]} -= \text{Request}_i\text{[j]}$$
$$\text{Allocation[i,j]} += \text{Request}_i\text{[j]}$$
$$\text{Need[i,j]} -= \text{Request}_i\text{[j]}$$

(4). 系统检查当前是否处于安全状态。若是，则正式分配资源；若否，则不分配，令 P_i 等待。

安全性算法：

(1). 设置两个向量：

- Work 向量，长度 m, 表示系统可以提供给进程的资源数目。算法开始时 Work=Available。
- Finish 向量，长度 m，表示系统是否有足够资源分给进程，使之运行完成。开始时 Finish[i]=false。

(2). 找到一个能满足下述条件的进程：

- Finish[i]=false;
- Need[i,j]$\leq$Work[j];

若找到，转 (3)，否则转 (4)。

(3). 进程 P_i 获得资源后，可执行至结束，并释放资源。因此执行：

$$\text{Work[j]} += \text{Allocation[i,j]}$$
$$\text{Finish[i]} = \text{True}$$

然后转 (2)。

(4). 若所有进程的 Finish[i] 都为 true，则表示系统处于安全状态，否则为不安全状态。

3.8 死锁的检测与解除

待补。

四 内存管理

4.1 程序的装入和链接

4.1.1 程序的装入

- **绝对装入方式**：编译时直接确定装入的内存位置，装入后无需再修改程序和数据的地址。用于单道程序环境。
- **可重定位装入方式**：用于多道程序环境。装入时对逻辑地址修改得到物理地址，又称**静态（地址）重定位**。程序在内存中不能移动。
- **动态运行时的装入方式**：相对于上面的装入时进行重定位，本方式在程序执行时才进行重定位。又称**动态（地址）重定位**。需要重定位寄存器的支持。

4.1.2 程序的链接

- 静态链接：编程时，链接所有模块
- 装入时动态链接：装入时，链接所有模块
- 运行时动态链接：运行时，根据需要链接模块

4.2 连续分配存储管理方式

连续分配是最早出现的一种存储器分配方式，该方式为用户分配一个连续的内存空间。

内存的零头：

- 内零头：分配给进程，而进程未用到的内存部分；
- 外零头：未分配给进程，但因为太小而无进程能用；

4.2.1 单一连续分配

主要用于单用户 OS。在一段时间内，只有一个进程在内存。

基本思想：内存分为两个区域：一个供操作系统使用，一个供用户使用。

三种典型的布局方式：

- 用户空间在低地址，系统空间在高地址。
- 和上面反过来。
- 系统空间在两边，把用户空间夹起来。

实现：设置基址寄存器、限界寄存器，要求逻辑地址映射为物理地址后，物理地址必须在两者之间。

特点：简单，适用于单任务 OS

4.2.2 固定分区分配

用于多道程序系统。把内存划分为固定的几个部分，维护一个分区使用表。

表 1: 分区说明表示例

区号	分区长度（KB）	起始地址	状态
1	12	20	已分配
2	32	32	已分配
3	64	64	未分配
4	128	128	已分配

特点：

- 最早的支持多道程序的内存管理方式。
- 简单，现在仍在嵌入式系统中应用。
- 内零头严重。

4.2.3 动态分区分配

基本思想：程序运行时划分大小合适的分区，程序结束后收回分区
配套数据结构：

- 空闲分区表：和固定分区一样
- 空闲分区链：在分区头部放入必要的说明信息和前向指针，分区尾部放入后向指针。

动态分区分配算法：分为分配算法和回收算法两部分。分配算法又分基于顺序搜索的算法和基于索引搜索的算法两部分。

回收算法比较简单，在回收内存之后要做：

- 上下两个相邻区都是空闲区时，合并三个区
- 上空闲下不空闲时，合并上面的区
- 下空闲上不空闲时，合并下面的区
- 上下都不空闲时，不合并。

基于顺序搜索的分配算法：用于小型系统

- 首次适应算法
  - ◇ 低地址端被快速分配，碎片迅速出现
  - ◇ 高地址端可能出现大块空闲区
- 循环首次适应算法
  - ◇ 需要设置一个查找指针
  - ◇ 本质是 CLOCK 算法
  - ◇ 碎片分布均匀
- 最佳适应算法
  - ◇ 需要将空闲区按照从小到大的顺序形成一个链，第一次找到的能放下的分区一定是最佳的。

◇ 碎片会迅速出现

- 最坏适应算法
  - ◇ 仍然形成空闲分区链，但顺序是从大到小，每次只检查第一个（最大的）分区能否满足要求。
  - ◇ 碎片出现最慢

基于索引搜索的分配算法：用于大中型系统

- 快速适应算法
- 伙伴系统
- 哈希算法

4.2.4 可重定位分区分配

在动态分区分配的基础上引入”紧凑“机制，消除外零头。使用动态地址重定位。

4.3 分页存储管理方式

与连续分配相对，有离散分配方式，分为以下三类：

- 分页存储管理方式
- 分段存储管理方式
- 段页式存储管理方式

例题 5. 某计算机主存按字节编址，逻辑地址和物理地址都是 32 位，其中页内地址用 12 位表示，请问：

(1). 页的大小是多少字节?

(2). 页表最大占用多少字节?（页表项为 4 个字节）

解答：

(1). 页内地址有 12 位，则页大小为 $2^{12} = 4K$ 个字节。

(2). 页号长度为 $32 - 12 = 20$ 位，于是有 $2^{20} = 1M$ 个页表项。每个页表项占 4 字节，于是有 4M 个字节。

页面不宜太大，也不宜太小，太大内存利用率低，太小又浪费内存。一般为 2 的整数次方，1KB-8KB。

4.3.1 页表

为每个进程设置一个页表。作用是实现从页号到物理块号的地址映射。

- 记录程序各页面所在的页框位置；
- 支持进行地址重定位；
- 实现页面访问控制；
- 存储保护：限制程序在操作系统指定的内存区域内运行。

由于页表一般放在内存中，因此通过页表访问一次内存变量需要两次访存，影响性能。

为此引入快表（TLB，Translation lookaside buffers）

例题 6. 一个分页存储系统，设访问一次内存需要 200ns, 访问一次快表需要 10ns。那么：

(1). 如果页表存放在内存，那么访问一个内存单元需要多少时间?

(2). 如果系统引入联想寄存器，90% 的页表项可以在快表中命中，则访问一个内存单元需要平均时间是多少?

解答：

(1). $2 \times 200 = 400\text{ns}$

(2). $200 + [210 \times 10\% + 10 \times 90\%] = 230\text{ns}$

注意到 TLB 未命中时，访问页表时间为 10(访问快表)+200(访问内存)=210

不知道怎么写，跳过了

五 虚拟存储器

5.1 虚拟存储器概述

存储管理的两个基本问题：

- 有的作业很大，不能完整装入内存。
- 作业很多，不能全部放入内存，但它们又要求同时运行。

5.1.1 程序运行的局部性原理

- 时间局部性：刚执行过的指令在不久后很可能再次被执行，或程序刚访问过的数据很可能再次被访问。典型原因是程序中的循环。
- 空间局部性：程序刚访问过某存储单元，则附近的存储单元也很可能被访问。典型原因是程序的顺序执行。

5.1.2 虚拟存储器的定义和特征

定义：虚拟存储器是指具有请求调入功能和置换功能，能从逻辑上对内存容量加以扩充的一种存储器系统。其逻辑容量由内存容量和外存容量之和决定，运行速度接近内存速度，而每位的成本由接近外存。

特征：

- 多次性。程序无须一次性全部装入内存，而是可以分多次装入。
- 对换性。程序和数据无须常驻内存，在必要时可以换入换出。
- 虚拟性。能从逻辑上扩展内存容量，使用户看到的容量远大于实际容量。

5.1.3 虚拟存储器的实现方法

- 分页请求系统
- 请求分段系统

5.2 请求分页存储管理方式

待补

5.2.1 页面调入策略

设访问页面成功的次数为 S，访问页面失败（即缺页）的次数为 F，则缺页率为

$$f = \frac{F}{S+F} \tag{3}$$

处理缺页中断进行内存交换时，没有被修改过的页可以直接丢弃，被修改过的页必须写回外存。设被置换的页面被修改的概率是 β，其缺页中断处理时间是 t_a，页面未被修改时的缺页中断时间是 t_b，则缺页中断的平均时间是

$$y = \beta \times t_a + (1-\beta) \times t_b \tag{4}$$

5.3 页面置换算法

5.3.1 最佳置换算法

理论上最好的算法，实际无法实现，因为页面的实际访问顺序无法预知。每次都选择未来不会再用到或者最长时间不再用到的页换出。

5.3.2 先进先出（FIFO）置换算法

换页时，选择最早进入内存的页面换出。由于程序不是一直顺序执行，这种方法效果不好。

5.3.3 最近最久未使用（LRU）算法

基于局部性原理。换页时选择最近最久未使用的页换出。注意 FIFO 看的是页面调入内存的时间，而 LRU 看的是页面最近一次被使用的时间。

实现：移位寄存器、栈

5.3.4 Clock 置换算法

LRU 需要较多硬件才能实现，实际中常常使用 LRU 的近似算法，如 Clock 算法。

PPT 没有，疑似不考?

5.4 ”抖动“与工作集

5.4.1 抖动

所谓抖动（Thrashing）是指刚刚换出的页很快又要用到，于是又换入，此时需要再选一页换出，但很快被换出的页又要被用到，如此往复，严重影响效率。

发生抖动的根本原因是系统中同时运行的进程太多，分配给每个进程的物理块太少，导致进程运行时频繁出现缺页。

5.4.2 工作集

所谓工作集，是指在某段时间间隔 Δ 里，进程实际所要访问页面的集合。但未来的访问情况是无法预知的，于是使用过去一段时间的行为来作为未来行为的近似。

具体地说，把某进程在时间 t 的工作集记为 $w(t,\Delta)$，其中的 Δ 称为工作集的窗口尺寸。由此可将工作集定义为：进程在时间间隔 $(t-\Delta,t)$ 中引用页面的集合。

5.4.3 抖动的预防方法

- 采用局部置换策略
- 把工作集算法融入到处理机调度中
- 利用“L=S”准则调节缺页率
- 选择暂停的进程

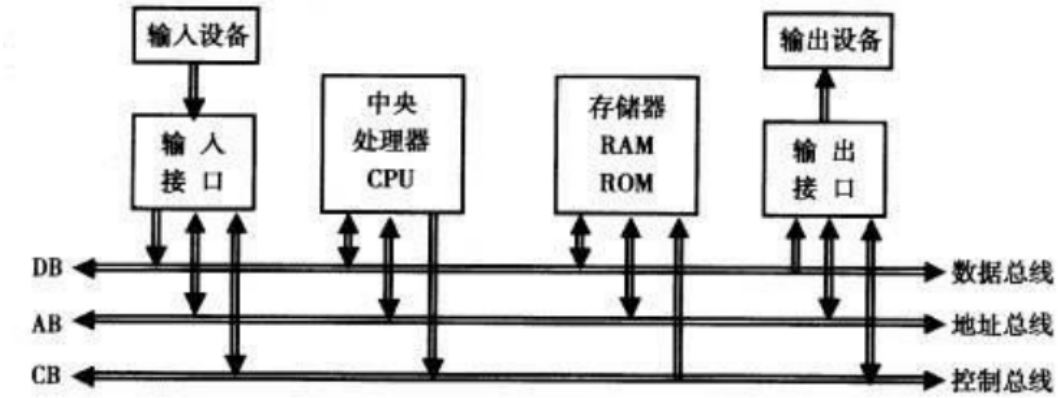
六 I/O 系统

本章内容以 PPT 顺序为准。

6.1 I/O 系统介绍

6.1.1 I/O 系统的基本结构

(1). 硬件基本结构



(2). I/O 系统组成成分

- I/O 设备：具体完成数据 I/O 的设备
- 设备控制器：负责连接 I/O 设备和数据总线，完成设备控制和数据格式转换。
- I/O 通道：目的是使原来由 CPU 处理的 I/O 任务转由通道来承担，从而把 CPU 从繁杂的 I/O 任务中解脱出来。

(3). I/O 系统结构

- 单通路 I/O 系统：无冗余设备，容错性差
- 多通路 I/O 系统：有冗余设备，容错性好

I/O 设备的分类

- 按速率分类

- ◇ 低速设备 (键盘)
- ◇ 中速设备 (打印机)
- ◇ 高速设备 (磁盘)

- 按信息交换单位分类
  - ◇ 字符设备 (键盘)
  - ◇ 块设备 (磁盘)

设备控制器的功能

- 接收和识别命令
- 数据交换
- 标志和报告设备状态
- 地址识别
- 数据缓冲
- 差错控制

6.1.2 I/O 控制方式

主要有四种

- 程序 I/O 方式
- 中断驱动 I/O 控制方式
- DMA 方式
- 通道控制方式

6.2 缓冲管理

6.2.1 缓冲的引入

引入缓冲的原因

- 缓和 CPU 和 I/O 设备的矛盾;
- 减少 CPU 中断的频率;
- 解决数据粒度不匹配的问题;
- 提高 CPU 和 I/O 设备的并行性。

6.2.2 单缓冲

引入如下几个时间:

- 从磁盘把数据输入到缓冲区的时间为 T
- OS 从缓冲区将数据传送到用户区的时间为 M
- CPU 处理这块数据的时间为 C

T 和 C 是可以并行的，故单缓冲区条件下，系统对每一块数据的处理时间为 Max(C,T)+M。

6.2.3 双缓冲

双缓冲时，系统处理一块数据的时间可以粗略认为是 Max(C,T)。

6.2.4 循环缓冲

略

6.2.5 缓冲池

略

6.3 设备分配

6.3.1 设备分配的基本概念

略

6.3.2 设备独立性

定义：用户程序独立于具体使用的物理设备。
实现：引入逻辑设备和物理设备

- 逻辑设备：是对一组具备相同功能物理设备的抽象；
- 物理设备：应用程序实际执行时，使用的特定类型设备。

基本做法：编写程序时，使用逻辑设备名称访问设备；程序执行时，使用具体的物理设备完成实际数据操作。

为每个用户设置一张逻辑设备表（LUT），完成逻辑设备到物理设备的映射。

设备独立性的优点

- 设备分配时的灵活性
- 容易实现 I/O 重定向

6.3.3 SPOOLing 技术

SPOOLing: Simultaneaus Periphernal Operating On-Line，中译名“假脱机”

为了缓和 CPU 的高速性和 I/O 设备低速性之间的矛盾，引入脱机输入/输出技术。多道程序环境下，可以利用其中的一道程序模拟脱机技术中的外围控制机，实现假脱机效果。

典型应用：共享打印机

SPOOLing 系统的基本组成

- 输入井和输出井
- 输入缓冲区和输出缓冲区
- 输入进程和输出进程

- 井管理程序

SPOOLing 特点

- 提高 I/O 速度
- 改造独占设备为共享设备
- 实现了虚拟设备功能

6.4 磁盘存储器的性能和调度

6.4.1 磁盘性能概述

磁盘访问时间

$$T = T_s + T_r + T_t \tag{5}$$

其中

- T_s：寻道时间
- T_r：旋转延迟时间
- T_t：传输时间

6.4.2 早期的磁盘调度算法

- 先来先服务（FCFS）算法：根据进程请求到来的次序进行服务。公平、简单，不会出现长时间得不到相应的情况。但平均寻道距离较大，不适合进程较多的情况。
- 最短寻道时间优先（SSTF）算法：访问距离最近的请求磁道，从而最小化寻道时间。有“磁臂粘着”现象。

6.4.3 基于扫描的磁盘调度算法

- 扫描（SCAN）算法：访问距离最近且在当前磁头移动方向上的请求磁道。磁头到达边缘后改变方向折返。有效解决“磁臂粘着”。

- 循环扫描（CSCAN）算法：即单向的 SCAN，磁头移动到最外磁道后，不是访问最近的请求磁道，而是立即返回最内侧磁道，再向外移动。

七 文件管理

7.1 文件和文件系统

文件的定义：文件是指由创建者所定义的、具有文件名的若干相关元素的集合。

7.1.1 数据项、记录和文件

在文件系统中，数据项是最低级的数据组织形式。也称字段。可分为

- 基本数据项：单个数据项
- 组合数据项：由若干个基本数据项组成

记录是一组相关数据项的集合，用于描述一个对象在某方面的属性。可分为

- 定长记录：记录长度固定
- 变长记录：记录长度不定

为了唯一标识一个记录，必须从记录的各个数据项中确定一个或多个数据项作为关键字（Key）。

7.1.2 文件系统的层次结构

文件管理系统管理的对象：

- 文件
- 目录
- 磁盘存储空间

文件系统接口：

- 命令接口
- 程序接口

7.1.3 文件操作

最基本的文件操作：

- 创建文件
- 删除文件
- 读文件
- 写文件
- 设置文件的读/写位置

7.2 文件的逻辑结构

所有文件都有如下两种形式的结构：

- 逻辑结构，这是从用户观点看到的结构，也叫文件组织。
- 物理结构，用户不可见。

7.2.1 文件逻辑结构的分类

按照是否有结构来分：

- 有结构文件（记录式文件）
  - ◇ 定长记录：检索容易
  - ◇ 变长记录：灵活
- 无结构文件（流式文件）

按照文件的组织方式，将有结构文件分为以下三类：

- 顺序文件
- 索引文件
- 索引顺序文件

正规 OS：顺序文件 + 暴露读写指针

7.2.2 顺序文件

顺序文件的记录是定长记录或可变长记录。

- 最佳应用场合：对文件中的记录进行批量存取时。
- 若共有 N 个记录，则查找一个记录平均需要查 $N/2$ 次。即查找性能不好。
- 此外添加或删除一个记录也很困难。可以引入一个事务文件，保存一段时间内的新增和删除记录，每隔一段时间进行实际写入。

为了访问顺序文件中的记录，必须首先找到记录的地址。有两种方法

- 隐式寻址：就是从头找起，无论记录是定长的还是不定长的。
- 显式寻址：仅能用于寻址定长记录文件，直接将基址加上偏移量得到，由于定长，偏移量很容易得到。

7.2.3 索引文件

主要用于解决变长记录文件的快速查找问题。

方法：建立一张索引表，其中保存有每个记录的“指针”，是定长的，这样就可以按照关键字查找索引表，然后再跳转到实际记录。

也可以建立多个索引表，从而支持按照多个属性进行索引。

缺点是索引表须占用额外存储空间，尤其当记录数量很大时。

7.2.4 索引顺序文件

顺序文件和索引文件的一种折中。

将记录分组，为每组记录在索引表中建立一个项。

分组方式：按关键字分组，或直接按记录号分组。

查找过程：首先查索引表，找到待查项所在的组，通过指针跳转，再在组中顺序查找。

可以引入多级索引。

7.2.5 直接文件和哈希文件

课本和 PPT 说法不太一样，略了。

八 磁盘存储器的管理

8.1 外存的组织方式

常见方式：

- 连续组织方式
- 链接组织方式
- 索引组织方式

8.1.1 连续组织方式

为每个文件分配一组相邻接的磁盘块，且文件的逻辑记录的顺序与所存储磁盘块的块号顺序一致。

优点：

- 顺序访问很容易。
- 顺序访问速度快。

缺点：

- 要求为文件分配连续的存储空间，很容易出现碎片。而外存的紧凑代价是难以想象的。
- 必须事先知道文件长度，同理，很难处理动态增长的文件。
- 不能快速插入和删除记录。

8.1.2 链接组织方式

为每个文件分配一组不相邻接的磁盘块，且文件的逻辑记录的顺序与所存储磁盘块的块号顺序可以不一致。

核心问题：如何设计数据结构，来存储文件的所有磁盘块的位置。

两种方法：

- 隐式链接法：每个磁盘块设置一个指针指向下一个磁盘块。

- 显式链接法：把所有指针存在一张链接表内，即所谓文件分配表 FAT（File Allocatin Table），使用时将 FAT 表载入内存，在内存中查找，从而大大加速。

8.1.3 FAT 技术

引入“卷”的概念，支持将一个物理磁盘分成 4 个卷（也称分区）。
一个扇区一般是 512KB 大小。
FAT12

- 每个分区保存两张相同的 FAT 表。
- 将文件的第一个盘块号放在自己的 FCB 中。
- 可以将相邻的若干个扇区在逻辑上合并为“簇”（cluster），进一步扩大最大容量。
- 支支持 8+3 格式的短文件名。

FAT16：FAT12 的简单改进版本
FAT32

- 分区容量不大于 8GB 时，每个簇都固定为 4KB。
- 分区为 8-16GB 时，每个簇为 8KB。
- 分区为 16-32GB 时，每个簇为 16KB
- 分区为 32GB 以上时，每个簇为 32KB
- 支持长文件名。
- 不支持小于 512MB 的分区，单文件长度不能大于 4GB。
- 不能向下兼容。

8.1.4 NTFS 技术

略

块大小	FAT12	FAT16	FAT32
0.5KB	2MB		
1KB	4MB		
2KB	8MB	128MB	
4KB	16MB	256MB	1TB
8KB		512MB	2TB
16KB		1024MB	2TB
32KB		2048MB	2TB

表 2: FAT 中簇的大小与最大分区的对应关系

8.1.5 索引组织方式

每个文件分配一个索引表，记录该文件的所有盘块。
优点：支持直接访问，不会产生外部碎片。
缺点：不适合小文件
也可使用多级索引。

8.2 文件存储空间的管理

8.2.1 空闲表法

系统为外存所有空闲区建立一张空闲表，每个空闲区对应一个空闲表项。

8.2.2 空闲链表法

把外存所有空闲区拉成一张空闲盘区（块）表。

- 空闲盘块链：结点代表一个空闲盘块
- 空闲盘区链：结点代表一个空闲盘区

8.2.3 位视图法

利用二进制的一位来表示磁盘中一个盘块的使用情况。0 表示空闲，1 表示占用。

- 当从 1 开始编号时，有：
  - ◇ 位视图第 i 行第 j 列对应的盘块号为

$$b = n(i-1) + j \tag{6}$$

其中 n 为每行的位数。

  - ◇ 给定盘块号 b，按照如下公式对应到位视图：

$$i = [(b-1)\text{整除}n] + 1 \tag{7}$$

$$j = (b-1)\text{MOD}n + 1 \tag{8}$$

- 当从 0 开始编号时，有：
  - ◇ 位视图第 i 行第 j 列对应的盘块号为

$$b = n \times i + j \tag{9}$$

其中 n 为每行的位数。

  - ◇ 给定盘块号 b，按照如下公式对应到位视图：

$$i = b\text{整除}n \tag{10}$$

$$j = b\text{MOD}n \tag{11}$$

例题 7. 在位视图法中，假如字长为 32，则 90 号磁盘块对应的行号为（ ），列号为（ ）。（行号、列号、磁盘块号均从 0 开始编号）

解答：

$$i = 90/32 = 2$$

$$j = 90 \text{ MOD } 32 = 26$$

九 操作系统接口

操作系统是人和计算机之间的接口。
操作系统接口是用户和操作系统之间的接口。
分类:

- 用户接口：提供给用户。
  - ◇ 联机命令接口：字符式。用户使用命令语言和 OS 交互。命令语言包括:
    - ∗ 命令行方式
    - ∗ 批命令方式
  - ◇（联机）图形用户接口 GUI
  - ◇ 脱机用户接口（用于批处理）
- 程序接口：提供给程序。用户程序取得 OS 服务的唯一途径。

9.1 系统调用

程序接口是由一组系统调用（system call）组成的。
系统调用是特殊的过程调用，它和一般的过程调用有如下区别:

- 运行在不同的系统状态。一般的过程调用的调用者和被调用者运行在同一个状态，而系统调用的调用者运行在用户态，而被调用者运行在系统态。
- 状态的转换。一般的过程调用不涉及状态转换，系统调用涉及从用户态到系统态的转换，这是通过软中断机制实现的。
- 嵌套调用，系统调用可以嵌套，但一般都有最大深度限制，而一般的过程调用则没有限制。

系统调用的实现:

- 在操作系统中，每个系统调用都对应一个事先给定的功能号，例如 0、1、2、3 等。

- 在陷入指令中必须包括对应系统调用的功能号。有些陷入指令中，还带有传给陷入处理机构和内部处理程序的有关参数。
- 为实现系统调用功能的子程序编造入口地址表，每个入口地址与相应的系统程序对应。

系统调用的过程:

- 将处理机状态从用户态转为系统态。
- 由硬件和内核程序进行系统调用的一般性处理，包括保护进程的 CPU 环境和传递用户定义的参数。
- 分析系统调用的类型，转入相应的系统调用处理程序。
- 系统调用处理程序执行完后，恢复 CPU 现场，返回被中断进程，继续执行。