

1、题型及分值

- (1) 名词解释：4 分×5 = 20 分
- (2) 简答：8 分×5 = 40 分
- (3) 设计与计算：40 分（3 题）

其中（3）设计与计算包含：

1、银行家算法（10 分）：见书中例题形式

四、举例说明

假设系统中有五个进程 P0、P1、P2、P3、P4，三种资源 A、B、C，初始状态如下：

进程	Max	Allocation	Need	Available
P0	[7, 5, 3]	[0, 1, 0]	[7, 4, 3]	[3, 3, 2]
P1	[3, 2, 2]	[2, 0, 0]	[1, 2, 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]	

现在，假设进程 P1 发出请求 Request1 = [0, 2, 2]，按照银行家算法步骤：

1. 首先检查 Request1 = [0, 2, 2] 是否小于等于 Need1 = [1, 2, 2]，满足条件，请求合理。
2. 接着检查 Request1 是否小于等于 Available = [3, 3, 2]，满足条件，系统有足够资源。
3. 尝试分配资源：
 - Available = [3, 3, 2] - [0, 2, 2] = [3, 1, 0]
 - Allocation1 = [2, 0, 0] + [0, 2, 2] = [2, 2, 2]
 - Need1 = [1, 2, 2] - [0, 2, 2] = [1, 0, 0]
4. 进行安全性检查：
 - Work = [3, 1, 0]，Finish 向量初始值全为 false。
 - 找到进程 P3，因为 Need3 = [0, 1, 1] <= Work = [3, 1, 0]，满足条件。
 - Work = [3, 1, 0] + [2, 1, 1] = [5, 2, 1]，Finish3 = true。
 - 找到进程 P1，因为 Need1 = [1, 0, 0] <= Work = [5, 2, 1]，满足条件。
 - Work = [5, 2, 1] + [2, 2, 2] = [7, 4, 3]，Finish1 = true。
 - 依次类推，可以发现所有进程的 Finish 向量都能变为 true，系统处于安全状态，所以可以将资源分配给进程 P1。

2、信号量与 PV 操作：（15 分）

① 理发师问题

```
1. semaphore barber_chair = 1;
2. semaphore customer_waiting = 0;
3. semaphore mutex = 1;
4. int waiting = 0;
5. int max_waiting = n;
6.
7. barber() {
8.     while (true) {
9.         P(customer_waiting);
10.        P(mutex);
11.        waiting--;
12.        V(mutex);
13.        理发;
14.        V(barber_chair);
```

```

15.     }
16.}
17.
18.customer() {
19.    P(mutex);
20.    if (waiting < max_waiting) {
21.        waiting++;
22.        V(mutex);
23.        V(customer_waiting);
24.        P(barber_chair);
25.        等待理发;
26.    } else {
27.        V(mutex);
28.        离开;
29.    }
30.}

```

② 读者写者

```

1. semaphore rw_mutex = 1;    // 读写互斥
2. semaphore mutex = 1;      // 更新read_count 的互斥
3. int read_count = 0;
4.
5. reader() {
6.    P(mutex);
7.    read_count++;
8.    if (read_count == 1) P(rw_mutex);
9.    V(mutex);
10.
11.    读操作;
12.
13.    P(mutex);
14.    read_count--;
15.    if (read_count == 0) V(rw_mutex);
16.    V(mutex);
17.}
18.
19.writer() {
20.    P(rw_mutex);
21.    写操作;
22.    V(rw_mutex);
23.}

```

③ 吃水果问题

```

1. semaphore plate = 1;
2. semaphore apple = 0, orange = 0;
3.
4. father() {
5.    while (true) {
6.        P(plate);

```

```

7.      放苹果;
8.      V(apple);
9.  }
10.}
11.
12.mother() {
13.    while (true) {
14.        P(plate);
15.        放橘子;
16.        V(orange);
17.    }
18.}
19.
20.son() {
21.    while (true) {
22.        P(orange);
23.        拿橘子;
24.        V(plate);
25.        吃橘子;
26.    }
27.}
28.
29.daughter() {
30.    while (true) {
31.        P(apple);
32.        拿苹果;
33.        V(plate);
34.        吃苹果;
35.    }
36.}

```

④ 和尚打水

```

1. // 信号量定义
2. semaphore mutex_well = 1;    // 水井互斥
3. semaphore mutex_vat = 1;    // 水缸互斥
4. semaphore empty = 10;    // 水缸空位
5. semaphore full = 0;    // 水缸有水
6. semaphore bucket = 3;    // 可用水桶
7.
8. // 小和尚进程 (打水入缸)
9. young_monk() {
10.    while (true) {
11.        P(bucket);    // 申请水桶
12.        P(mutex_well);    // 申请水井
13.        从井中打水;
14.        V(mutex_well);    // 释放水井
15.
16.        P(empty);    // 申请水缸空位
17.        P(mutex_vat);    // 申请水缸
18.        将水倒入水缸;

```

```

19.      V(mutex_vat);          // 释放水缸
20.
21.      V(full);                // 增加水量计数
22.      V(bucket);              // 归还水桶
23.  }
24.}
25.
26.// 老和尚进程（从缸取水）
27.old_monk() {
28.    while (true) {
29.        P(full);                // 等待水缸有水
30.        P(bucket);              // 申请水桶
31.
32.        P(mutex_vat);           // 申请水缸
33.        从水缸中取水;
34.        V(mutex_vat);           // 释放水缸
35.
36.        V(empty);               // 增加空位计数
37.        V(bucket);              // 归还水桶
38.
39.        喝水;
40.    }
41.}

```

⑤ 生产者消费者问题

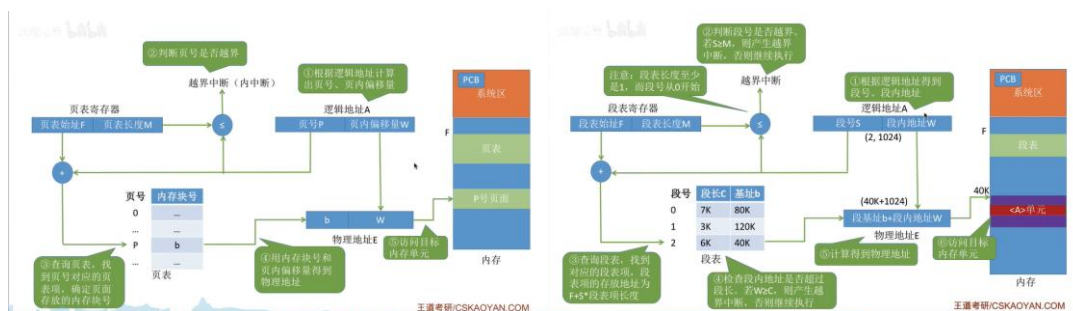
```

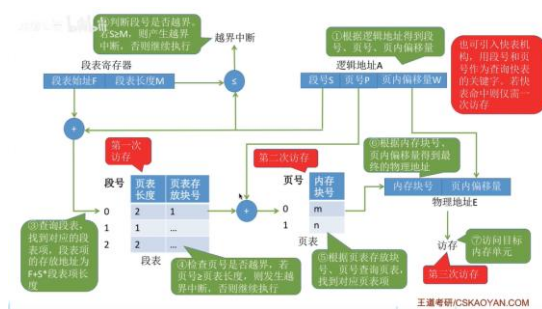
1. semaphore mutex = 1;        // 互斥访问缓冲区
2. semaphore empty = n;         // 空缓冲区数量
3. semaphore full = 0;          // 满缓冲区数量
4.
5. producer() {
6.     while (true) {
7.         生产一个产品;
8.         P(empty);
9.         P(mutex);
10.        将产品放入缓冲区;
11.        V(mutex);
12.        V(full);
13.    }
14.}
15.
16.consumer() {
17.    while (true) {
18.        P(full);
19.        P(mutex);
20.        从缓冲区取出产品;
21.        V(mutex);
22.        V(empty);
23.        消费产品;
24.    }

```

3、存储管理：（15 分）

- ① 页式、段式、段页式：逻辑地址转换为物理地址的过程描述（画图）
(求余，要去处用简单的方法就行，是 16 进制的)
- ② 给定逻辑地址，能够转化为物理地址
- ③ 分页式存储管理中，逻辑地址为什么可以直接划分为页号和页内偏移？
- ④ 分段式存储管理中，逻辑地址为什么可以直接划分为短号和段内偏移？





2

3

在分页系统中，逻辑地址能直接划分为页号和页内偏移量，根本原因在于页的大小是固定且由系统硬件预先统一规定的（例如 **4KB**）。这使得整个地址空间成为一个规整的一维线性空间。

4

在分段系统中，逻辑地址的划分源于程序编译或链接时的逻辑结构。编译器会将程序中的不同逻辑单元（如主函数、子函数、全局变量区、栈等）划分为不同的段，并为每个段分配一个段号。段内偏移量则代表了该数据或指令在自身所在段内的位置。

假设在一个分页系统中，页面大小为 **1KB (0x400)**，某进程的页表部分内容如下：页号 **0** 对应物理块号 **3**，页号 **1** 对应物理块号 **7**，页号 **2** 对应物理块号 **11**。

现有逻辑地址 **0x0A5C**，求其物理地址。

计算页号与页内偏移（利用十六进制计算更简便）：

页面大小 **0x400**，页内偏移占低 **10** 位（因为 $2^{10}=1024=0x400$ ）。

页号 $P = 0x0A5C / 0x400 = 0x0A5C \gg 10 = 2$ 。

页内偏移 $D = 0x0A5C \% 0x400 = 0x0A5C \& 0x3FF = 0x25C$ 。

查询页表：页号 **2** 对应的物理块号为 **11**（十进制），即十六进制 **0xB**。

合成物理地址：

物理地址 = 物理块号 $\times$ 页面大小 + 页内偏移 = $0xB * 0x400 + 0x25C = 0x2C00 + 0x25C = 0x2E5C$ 。

第一章 绪论

1、名词：

(1) 操作系统

操作系统是管理计算机硬件与软件资源的系统软件，为用户和应用程序提供接口，负责进程管理、内存管理、文件系统管理、设备管理等核心功能。

(2) 多道程序设计系统

多道程序设计系统是指系统中允许多个程序同时进入内存并交替执行，通过 **CPU 调度** 和资源分配提高系统资源利用率。

2、简答：

(1) 多道程序设计的资源利用率分析

单道程序中，处理机与 **I/O** 设备串行工作，处理机存在大量空闲；多道程序中，一个程序 **I/O** 时，另一个程序占用处理机，资源重叠使用，处理机利用率显著提升。

(2) 操作系统的基本功能

① 资源分配与竞争；

- ② 进程调度;
- ③ 内存地址冲突;
- ④ 进程同步互斥;
- ⑤ 死锁预防与解决。

(3) OS 管理资源 (处理机, 内存, 设备, 文件)

- ① 处理机: 调度、控制进程, 提高利用率
- ② 内存: 分配回收、地址转换、保护与扩充
- ③ 设备: 分配、I/O 控制、缓冲管理
- ④ 文件: 创建、读写、存储、保护文件

第二章 处理器管理

1、名词:

(1) 进程/线程/进程控制块

进程: 程序的一次执行过程, 是资源分配的基本单位。

线程: 进程中的一个执行单元, 是 CPU 调度的基本单位。

进程控制块 (PCB): 记录进程状态、程序计数器、寄存器、内存分配等信息的系统数据结构。

(2) 原语

原语是指由若干条指令组成、执行过程中不可中断的操作序列, 常用于操作系统内核中实现进程同步、通信等功能。

(3) 作业周转时间/带权作业周转时间/平均作业周转时间/平均带权作业周转时间

周转时间 = 完成时间 - 到达时间

带权周转时间 = 周转时间 / 运行时间

平均周转时间 = 所有作业周转时间之和 / 作业数量

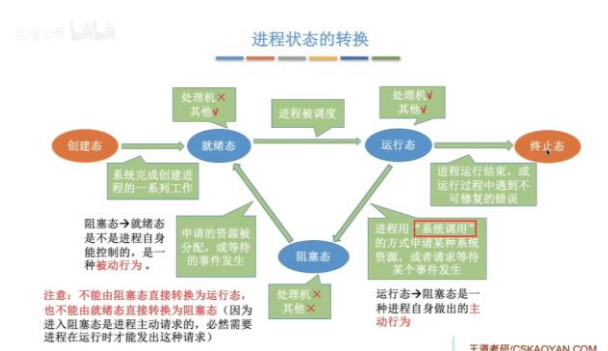
平均带权周转时间 = 所有作业带权周转时间之和 / 作业数量

(4) 响应比

$1 + \text{等待时间} / \text{要求服务时间}$

2、简答:

(1) 进程三态模型、五态模型图



(2) 先来先服务算法/最短优先算法/最短剩余时间优先算法/响应比最高者优先算法:
给定作业序列, 能够写出作业调度顺序, 并计算“作业周转时间/带权作业周转时间/平均作业周转时间/平均带权作业周转时间”

周转时间 = 完成时间 - 到达时间

带权周转时间 = 周转时间 / 运行时间

平均周转时间 = 所有作业周转时间之和 / 作业数量

平均带权周转时间 = 所有作业带权周转时间之和 / 作业数量

进程	到达时间	运行时间
P1	0	7
P2	2	4
P3	4	1
P4	5	4

先来先服务

进程	到达时间	运行时间
P1	0	7
P2	2	4
P3	4	1
P4	5	4

先来先服务调度算法：按照到达的先后顺序调度，事实上就是等待时间越久的越优先得到服务。
因此，调度顺序为：P1 → P2 → P3 → P4



周转时间 = 完成时间 - 到达时间

P1=7-0=7; P2=11-2=9; P3=12-4=8; P4=16-5=11

带权周转时间 = 周转时间/运行时间

P1=7/7=1; P2=9/4=2.25; P3=8/1=8; P4=11/4=2.75

等待时间 = 周转时间 - 运行时间

P1=7-7=0; P2=9-4=5; P3=8-1=7; P4=11-4=7

平均周转时间 = (7+9+8+11)/4 = 8.75

平均带权周转时间 = (1+2.25+8+2.75)/4 = 3.5

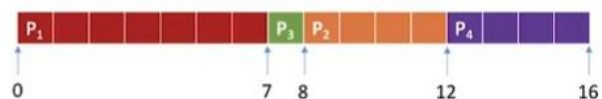
平均等待时间 = (0+5+7+7)/4 = 4.75

注意：本例中的进程都是纯计算型的进程，一个进程到达后要么在等待，要么在运行。如果是又有计算、又有I/O操作的进程，其等待时间就是周转时间 - 运行时间 - I/O操作的时间

最短作业优先

进程	到达时间	运行时间
P1	0	7
P2	2	4
P3	4	1
P4	5	4

短作业/进程优先调度算法：每次调度时选择当前已到达且运行时间最短的作业/进程。
因此，调度顺序为：P1 → P3 → P2 → P4



周转时间 = 完成时间 - 到达时间

P1=7-0=7; P3=8-4=4; P2=12-2=10; P4=16-5=11

带权周转时间 = 周转时间/运行时间

P1=7/7=1; P3=4/1=4; P2=10/4=2.5; P4=11/4=2.75

等待时间 = 周转时间 - 运行时间

P1=7-7=0; P3=4-1=3; P2=10-4=6; P4=11-4=7

平均周转时间 = (7+4+10+11)/4 = 8

平均带权周转时间 = (1+4+2.5+2.75)/4 = 2.56

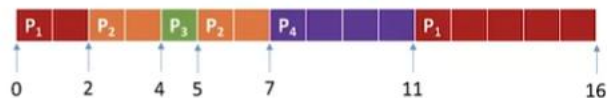
平均等待时间 = (0+3+6+7)/4 = 4

对比FCFS算法的结果，显然SJF算法的平均等待/周转/带权周转时间都要更低

最短剩余时间

进程	到达时间	运行时间
P1	0	7
P2	2	4
P3	4	1
P4	5	4

最短剩余时间优先算法：每当有进程加入就绪队列改变时就需要调度，如果新到达的进程剩余时间比当前运行的进程剩余时间更短，则由新进程抢占处理机，当前运行进程重新回到就绪队列。另外，当一个进程完成时也需要调度



周转时间 = 完成时间 - 到达时间

P1=16-0=16; P2=7-2=5; P3=5-4=1; P4=11-5=6

带权周转时间 = 周转时间/运行时间

P1=16/7=2.28; P2=5/4=1.25; P3=1/1=1; P4=6/4=1.5

等待时间 = 周转时间 - 运行时间

P1=16-7=9; P2=5-4=1; P3=1-1=0; P4=6-4=2

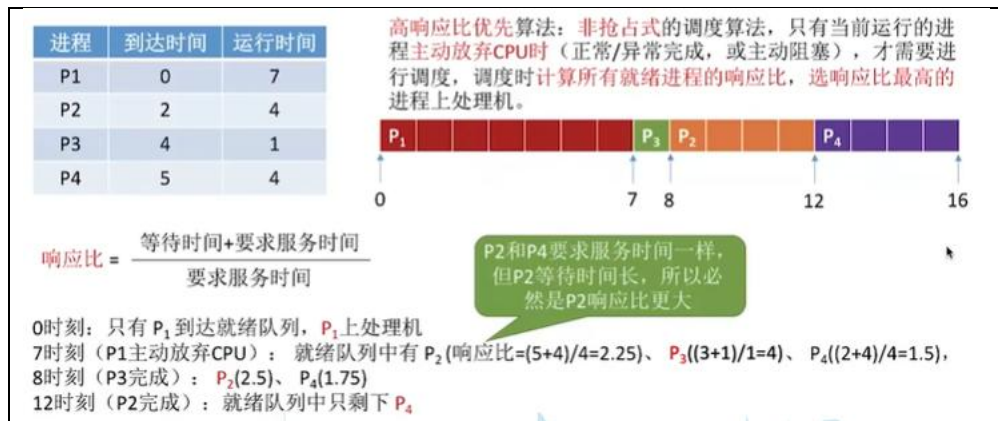
平均周转时间 = (16+5+1+6)/4 = 7

平均带权周转时间 = (2.28+1.25+1+1.5)/4 = 1.50

平均等待时间 = (9+1+0+2)/4 = 3

对比非抢占式的短作业优先算法，显然抢占式的这几个指标又要更低

相应比最高



(3) 证明最短（剩余）时间优先算法具有最小的平均等待时间

最短作业优先及其抢占式版本最短剩余时间优先算法在特定条件下可以实现最小的平均等待时间。

最优性条件：当所有进程同时可运行时（即所有进程几乎同时到达系统），SJF 调度算法的平均等待时间最少。对于进程不同时到达的情况，最短剩余时间优先算法能获得更少的平均等待时间。

证明思路（交换论证法）：

假设存在一个最优调度序列 S，不是按最短作业优先顺序排列。

必然存在相邻的两个作业 A 和 B，其中 A 先执行但运行时间比 B 长。

如果交换 A 和 B 的执行顺序，可以计算其他作业的等待时间不变，但 A 和 B 的总等待时间会减少。

因此，任何非 SJF 顺序的调度都可以通过交换相邻逆序对来减少总等待时间。

所以 SJF 顺序能实现最小的平均等待时间。

需要强调的是，这一结论的前提是所有进程同时可用。在实际系统中，由于进程到达时间不同，最短剩余时间优先算法更能有效减少平均等待时间。

(4) 进程与线程的关系与区别

线程是“轻量级进程”，是进程中的一个执行实体。

一个进程可以包含多个线程，这些线程共享进程的地址空间和资源。

进程是资源分配的基本单位，而线程是 CPU 调度的基本单位。

第三章 同步、通信与死锁

1、名词：

(1) Bernstein 条件

读写集合无交集

(2) 并发/并行

并发：指在一个时间段内，有多个程序都处于开始运行到运行结束之间。在单个处理器上，这些程序通过时间片轮转等调度技术交替执行，宏观上看似同时，微观上并非同一时刻执行。它是一种逻辑上的“同时”。

并行：指在同一时刻，有多个指令在多个处理器核心上真正同时执行。它要求有多个计算单元，是物理上的同时。并行是并发的一种特例，旨在实现物理执行效率的线性提升。

(3) 互斥/同步

互斥：指保证对临界资源的独占式访问。某一共享资源在同一时刻最多只允许一个进程（或线程）使用，以解决资源竞争问题。互斥是同步的一种特例，其核心

是“谁可以进来”的问题。

同步：指在互斥的基础上，协调多个进程（或线程）的执行顺序，使它们按照预定的先后次序合作完成任务。它解决的是线程间的协作问题，核心是“什么时候进来”的问题。例如，一个进程需要等待另一个进程提供数据后才能继续执行。

(4) 临界区/临界资源

临界资源：指一次仅允许一个进程访问的共享资源。既包括打印机、磁带机等物理硬件，也包括变量、缓冲区、队列等软件资源。若多个进程同时访问，可能导致数据不一致等与时间相关的错误。

临界区：指每个进程中访问临界资源的那段代码。进程在进入临界区前需执行“进入区”代码申请资源，退出时执行“退出区”代码释放资源。保证诸进程互斥地进入各自的临界区，即可实现安全共享临界资源。

(5) 信号量

信号量是一种用于进程/线程同步与互斥的机制，由一个整数值和相关的等待队列组成。其值表示可用资源的数量。对信号量的操作主要是 P（wait）和 V（signal）

原语操作。P 操作申请一个单位资源，信号量值减 1，若值小于 0 则进程阻塞等待；V 操作释放一个单位资源，信号量值加 1，若值不大于 0 则唤醒一个等待进程。

(6) 死锁

死锁是指两个或多个进程因竞争资源而造成的一种互相等待、无法继续执行的状态

(7) 死锁定理

死锁定理是用于判断系统是否已发生死锁的准则。其核心是通过简化系统资源分配图来检测死锁。如果资源分配图是不可完全简化的（即无法通过一系列操作消除图中所有边），则系统处于死锁状态。该定理为死锁检测算法提供了理论基础。

2、简答：

(1) 并发进程中可能出现的与时间相关的错误：（会举例并分析）

① 结果不唯一（售票问题例子）

② 永远等待（内存申请例子）

1. 两进程同时读共享变量，指令交错导致计数错误

2. 两进程各持部分资源，相互等待释放，陷入死锁

(2) 临界区调度原则

互斥性、空闲让进、有限等待、让权等待。

(3) 一般信号量及对应的 PV 操作/二元信号量及对应的 PV 操作：伪代码形式写出其定义

略

(4) 产生死锁的条件

互斥条件、请求与保持条件、不剥夺条件、循环等待条件。

(5) 证明：层次化、按序分配策略可以预防死锁

资源按层次编号，进程需按递增顺序申请资源，破坏循环等待条件，故无死锁。

第四章 存储管理

1、名词：

(1) 静态/动态地址重定位

静态：程序装入时一次性完成地址转换

动态：程序执行时通过硬件(如重定位寄存器)完成地址转换

(2) 页/页框

页：进程逻辑地址空间的固定大小划分

页框（页帧）：物理内存中与页大小相同的存储块

(3) 段

按程序逻辑关系划分的连续地址空间单位（如代码段、数据段），长度可变

(4) 逻辑地址空间/物理地址空间

逻辑地址空间：程序生成的地址集合，从 0 开始编址

物理地址空间：内存中实际存储单元的地址集合

(5) 页表/段表

页表：记录进程页号到物理块号的映射

段表：记录段号到内存起始地址和段长的映射。

(6) 虚拟内存

将硬盘空间作为后备存储，使程序无需全部装入内存即可运行，提供比实际内存更大的地址空间

(7) 程序局部性原理

程序执行时表现出访问的局部性，包括时间局部性（最近访问的项可能再次被访问）和空间局部性（访问项附近的项也可能被访问）。

(8) 外页表

当页表非常大时，可将其部分页表项存放在外存中

(9) 缺页中断率

衡量缺页中断发生频率的指标，计算公式为：缺页中断率 = 缺页次数 / 总的页面访问次数。

2、简答：

(1) 为什么提出分页/分段/段页式存储管理？

分页：解决连续分配产生的外部碎片问题，提高内存利用率。将进程和内存划分为固定大小的页/页框，实现离散分配。

分段：满足程序逻辑结构需求，便于实现共享和保护。按逻辑模块（如代码段、数据段）划分地址空间。

段页式：结合分段和分页的优点。先分段以满足逻辑需求和共享保护，段内再分页以解决外部碎片问题。

(2) 页的大小为什么设计为 2 的整数次幂？

简化地址转换的硬件实现。逻辑地址到页号和页内偏移的划分可以通过简单的位操作完成。

(3) 为什么说分段式存储管理中程序的逻辑地址是二维地址？

因为在分段管理中，程序员或编译器在标识一个地址时，需要显式或隐式地指定段号（或段名）和段内偏移地址两个部分。这与分页管理中由系统自动生成的一维线性地址不同，分段地址反映了程序的逻辑结构，因而是二维的。

(4) 各种页面替换策略：给定页面访问序列，求缺页中断率（见书上的例子）

最佳置算法：淘汰未来最长时间不被访问的页面（理论最优，但难以实现）。

先进先出算法：淘汰最先进入内存的页面（可能产生 Belady 异常）。

最近最久未使用算法：淘汰最近最久未被访问的页面（性能接近 OPT）。

缺页中断率计算：缺页中断率 = 缺页次数 / 总页面访问次数。

最佳置算法

最佳置换算法 (OPT)

最佳置换算法 (OPT, Optimal)：每次选择淘汰的页面将是以后永不使用，或者在最长时间内不再被访问的页面，这样可以保证最低的缺页率。

例：假设系统为某进程分配了三个内存块，并考虑到有一下页面号引用串（会依次访问这些页面）：
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
内存块1	7	7	7	2		2		2		2			2					7		
内存块2		0	0	0		0		4		0			0				0			
内存块3			1	1		3		3		3			1				1			
是否缺页	√	√	√	√		√		√		√		√		√			√			

选择从 0, 1, 7 中淘汰一页。按最佳置换的规则，往后寻找，最后一个出现的页号就是要淘汰的页面

整个过程缺页中断发生了9次，页面置换发生了6次。
注意：缺页时未必发生页面置换。若还有可用的空闲内存块，就不用进行页面置换。

缺页率 = $9/20 = 45\%$

先进先出算法

先进先出置换算法 (FIFO)

先进先出置换算法 (FIFO)：每次选择淘汰的页面是最早进入内存的页面
实现方法：把调入内存的页面根据调入的先后顺序排成一个队列，需要换出页面时选择队头页面即可。
队列的最大长度取决于系统为进程分配了多少个内存块。

例：假设系统为某进程分配了三个内存块，并考虑到有以下页面号引用串：

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

访问页面	3	2	1	0	3	2	4	3	2	1	0	4
内存块1	3	3	3	0	0	0	4			4	4	
内存块2		2	2	2	3	3	3			1	1	
内存块3			1	1	1	2	2			2	0	
是否缺页	√	√	√	√	√	√	√			√	√	



分配三个内存块时，缺页次数：9次

最近最久未使用算法

最近最久未使用置换算法 (LRU)

最近最久未使用置换算法 (LRU, least recently used)：每次淘汰的页面是最近最久未使用的页面
实现方法：赋予每个页面对应的页表项中，用访问字段记录该页面自上次被访问以来所经历的时间t。当需要淘汰一个页面时，选择现有页面中t值最大的，即最近最久未使用的页面。

页号	内存块号	状态位	访问字段	修改位	外存地址
----	------	-----	------	-----	------

例：假设系统为某进程分配了四个内存块，并考虑到有以下页面号引用串
1, 8, 1, 7, 8, 2, 7, 2, 1, 8, 3, 8, 2, 1, 3, 1, 7, 1, 3, 7

该算法的实现需要专门的硬件支持，虽然算法性能好，但是实现困难，开销大

访问页面	1	8	1	7	8	2	7	2	1	8	3	8	2	1	3	1	7	1	3	7
内存块1	1	1	1	1							1					1				
内存块2		8		8	8					8						7				
内存块3			7	7						3						3				
内存块4					2					2						2				
缺页否	√	√		√	√					√					√					

在手动做题时，若需要淘汰页面，可以逆向检查此时在内存中的几个页面号。在逆向扫描过程中最后一个出现的页号就是要淘汰的页面。