2024 考研 408

计算机学科专业基础综合

复习笔记

wjl

2023 年 12 月 3 日

考查目标

数据结构

1. 掌握数据结构的基本概念、基本原理和基本方法。

2. 掌握数据的逻辑结构、存储结构及基本操作的实现，能够对算法进行基本的时间复杂度与空间复杂度的分析。

3. 能够运用数据结构基本原理和方法进行问题的分析与求解，具备采用 C 或 C++ 语言设计与实现算法的能力。

计算机组成原理

1. 理解单处理器计算机系统中主要部件的工作原理、组成结构以及相互连接方式。

2. 掌握指令集体系结构的基本知识和基本实现方法，对计算机硬件相关问题进行分析，并能够对相关部件进行设计。

3. 理解计算机系统的整机概念，能够综合运用计算机组成的基本原理和基本方法，对高级编程语言 (C 语言) 程序中的相关问题进行分析，具备软硬件协同分析和设计能力。

操作系统

1. 掌握操作系统的基本概念、方法和原理，了解操作系统的结构、功能和服务，理解操作系统所采用的的策略、算法和机制。

2. 能够从计算机系统的角度理解并描述应用程序、操作系统内核和计算机硬件协作完成任务的过程。

3. 能够运用操作系统原理，分析并解决计算机系统中与操作系统相关的问题。

计算机网络

1. 掌握计算机网络的基本概念、基本原理和基本方法。

2. 掌握典型计算机网络的结构、协议、应用以及典型网络设备的工作原理。

3. 能够运用计算机网络的基本概念、基本原理和基本方法进行网络系统的分析、设计和应用。

目录

1 数据结构

1.1 线性表

线性表的基本概念、线性表的实现：1. 顺序存储 2. 链式存储

1.1.1 线性表的应用

1. 单链表原地逆置

```
void reverse(Node *l) {
  Node *q = l->next, *r = NULL; l->next = NULL;
  while (q != NULL) {
    r = q; q = q->next;
    r->next = l->next; l->next = r;
  }
}
```

2. 双指针合并两个有序单链表

```
void merge(Node *a, Node *b) {
  Node *p = a->next, *q = b->next, *r = NULL;
  a->next = NULL;
  while (p != NULL && q != NULL) {
    if (p->data < q->data) {
      if (r == NULL) { a->next = p; } else { r->next = p; }
      r = p; p = p->next;
    } else {
      if (r == NULL) { a->next = q; } else { r->next = q; }
      r = q; q = q->next;
    }
  }
  if (q != NULL) p = q;
  while (p != NULL) {
    if (r == NULL) { a->next = p; } else { r->next = p; }
    r = p; p = p->next;
  }
}
```

1.2 栈、队列和数组

栈和队列的基本概念、栈和队列的顺序存储结构：1. 顺序栈 2. 循环队列、**栈和队列的链式存储结构**

对于 n 个不同元素进栈，出栈序列的个数为 $\frac{1}{n+1}C_{2n}^{n}$.

1.2.1 多维数组的存储

二维数组行下标与列下标的范围分别为 $[0,h_1]$ 与 $[0,h_2]$，每个数组元素所占的存储单元为 L，
行优先：$LOC(a_{i,j})=LOC(a_{0,0})+[i\times(h_2+1)+j]\times L$.
列优先：$LOC(a_{i,j})=LOC(a_{0,0})+[j\times(h_1+1)+i]\times L$.

1.2.2 特殊矩阵的压缩存储

1. 对称矩阵：$k=\begin{cases}\dfrac{i(i-1)}{2}+j-1, & i\geqslant j\ \text{（下三角区和主对角元素）}\\ \dfrac{j(j-1)}{2}+i-1, & i<j\ \text{（上三角区元素）}\end{cases}$

2. 上三角矩阵：$k=\begin{cases}\dfrac{(i-1)(2n-i+2)}{2}+j-i, & i\leqslant j\ \text{(上三角区和主对角元素)}\\ \dfrac{n(n+1)}{2}, & i>j\ \text{(下三角区元素)}\end{cases}$

3. 下三角矩阵：$k=\begin{cases}\dfrac{i(i-1)}{2}+j-1, & i\geqslant j\ \text{(下三角区和主对角元素)}\\ \dfrac{n(n+1)}{2}, & i<j\ \text{(上三角区元素)}\end{cases}$

4. 三对角矩阵：$k=2i+j-3$.

5. 稀疏矩阵的三元组既可以采用数组存储，也可以采用十字链表法存储。

1.2.3 栈、队列和数组的应用

栈在递归中的应用：
通常需要借助栈将递归程序转化为非递归程序，但不是必须的。计算斐波那契数列仅需一个迭代过程。

1.3 树与二叉树

1.3.1 树的基本概念

完全二叉树的最后一个分支结点的序号为 $\lfloor n/2\rfloor$.

1.3.2 二叉树

1. 二叉树的定义及其主要特征 2. 二叉树的顺序存储结构和链式存储结构

顺序存储：下标从 0 开始，当前结点下标为 i，父节点下标 $(i-1)/2$，左孩子 $2*i+1$，右孩子 $2*i+2$.

3. 二叉树的遍历

1）先序遍历（迭代）：

```
void preOrder(BNode *t) {
  int M = 32, top = -1; BNode *stack[M], *p = t;
  while (p || top != -1) {
    if (p) { visit(p); stack[++top] = p; p = p->left; }
    else { p = stack[top--]; p = p->right; }
  }//end while
}
```

2）中序遍历（迭代）:

```
void inOrder(BNode *t) {
  int M = 32, top = -1; BNode *stack[M], *p = t;
  while (p || top != -1) {
    if (p) { stack[++top] = p; p = p->left; }
    else { p = stack[top--]; visit(p); p = p->right; }
  }//end while
}
```

3）后序遍历（迭代）:

```
void postOrder(BNode *t) {
  int M = 32, top = -1; BNode *stack[M], *p = t, *r = NULL;
  while (p || top != -1) {
    if (p) { stack[++top] = p; p = p->left; }
    else {
      p = stack[top];
      if (p->right && p->right != r) p = p->right;
      else { p = stack[top--]; visit(p); r = p; p = NULL; }
    }
  }//end while
}
```

4）层次遍历:

```
void levelOrder(BNode *t) {
  int M = 32, front = 0, rear = 0; BNode *queue[M], *p = t;
  queue[rear] = p; rear = (rear + 1) % M;
  while (front != rear) {
    p = queue[front]; front = (front + 1) % M; visit(p);
    if (p->left != NULL) {
      queue[rear] = p->left; rear = (rear + 1) % M;
    }
    if (p->right != NULL) {
      queue[rear] = p->right; rear = (rear + 1) % M;
    }
  }//end while
}
```

5）二叉树深度:

```
int depth(BNode *t) {
  int M = 32, front = 0, rear = 0, depth = 0;
  BNode *queue[M], *p = t, *last = t, *nlast = NULL;
  queue[rear] = p; rear = (rear + 1) % M;
  while (front != rear) {
    p = queue[front]; front = (front + 1) % M;
    if (p->left != NULL) {
      queue[rear] = p->left; rear = (rear + 1) % M; nlast = p->left; }
    if (p->right != NULL) {
      queue[rear] = p->right; rear = (rear + 1) % M; nlast = p->right; }
    if (p == last) { last = nlast; depth++; }
  }//end while
  return depth;
}
```

4. 线索二叉树的基本概念和构造

1.3.3 树、森林

1. 树的存储结构 2. 森林与二叉树的转换 3. 树和森林的遍历

1）先根遍历：先访问根结点，再依次遍历根结点的每棵子树。与这棵树相应二叉树的先序序列 相同。
2）后根遍历：先依次遍历根结点的每棵子树，再访问根结点。与这棵树相应二叉树的中序序列 相同。

树与二叉树的应用：1. 哈夫曼 (Huffman) 树和哈夫曼编码 2. 并查集及其应用

1.4 图

图的基本概念

1.4.1 图的存储及基本操作

1. 邻接矩阵 2. 邻接表

```
typedef struct ArcNode { int adjvec; struct ArcNode *next; } ArcNode;
typedef struct VNode { int data; ArcNode *firstArc; } VNode;
typedef struct Graph { int n, e; VNode adjlist[M]; } Graph;
```

3. 邻接多重表、十字链表

邻接多重表是无向图 的一种链式存储结构，十字链表是有向图 的一种链式存储结构。

1.4.2 图的遍历

1. 广度优先搜索 2. 深度优先搜索

2020-06 深度优先搜索，将输出顶点语句移到退出循环前，若输出结果包含全部顶点，则为逆拓扑有序序列。

```
void _bfs(Graph *g, int v) {
  visit(v); visited[v] = 1;
  int queue[M], front = 0, rear = 0; queue[rear] = v; rear = (rear + 1) % M;
  while (front != rear) {
    v = queue[front]; front = (front + 1) % M;
    for (int w = firstNbr(g, v); w >= 0; w = nextNbr(g, v, w)) {
      if (!visited[w]) {
        visit(w); visited[w] = 1; queue[rear] = w; rear = (rear + 1) % M;
      }
    }
  }//end while
}
void bfs(Graph *g) {
  for (int i = 0; i < g->n; i++) {
    if (!visited[i]) _bfs(g, i);
  }
}
void _dfs(Graph *g, int v) {
  visit(v); visited[v] = 1;
  for (int w = firstNbr(g, v); w >= 0; w = nextNbr(g, v, w)) {
    if (!visited[w]) _dfs(g, w);
  }
}
void dfs(Graph *g) {
  for (int i = 0; i < g->n; i++) {
    if (!visited[i]) _dfs(g, i);
  }
}
```

1.4.3 图的基本应用

1. 最小（代价）生成树

当带权连通图的任意一个环中所包含的边的权值均不相同 时，其 MST 是唯一的。

2. 最短路径

Dijkstra 算法：**2016-08, 2021-08**
1. 初始化：集合 S 初始为 $\{0\}$，$dist[]$ 的初始值 $dist[i]=arcs[0][i], i=1,2,\ldots,n-1$.
2. 从顶点集合 $V-S$ 中选出 v_j，满足 $dist[j]=Min\{dist[i]|v_i\in V-S\}$，$v_j$ 就是当前求得的一条从 v_0 出发的最短路径的终点，令 $S=S\cup\{j\}$.
3. 修改从 v_0 出发到集合 V-S 上任意一个顶点 v_k 可达的最短路径长度：若 $dist[j]+arcs[j][k]<dist[k]$，则更新 $dist[k]=dist[j]+arcs[j][k]$.
4. 重复 $2\sim3$ 操作共 $n-1$ 次，直到所有的顶点都包含在 S 中。

3. 拓扑排序 4. 关键路径

关键路径：**2011-41, 2013-09, 2019-05, 2020-08**
1. 事件 v_k 的最早发生时间 $ve(k)$：$ve(\text{源点})=0$，　$ve(k)=\text{Max}\{ve(j)+\text{Weight}(v_j,v_k)\}$
2. 事件 v_k 的最迟发生时间 $vl(k)$：$vl(\text{汇点})=ve(\text{汇点})$，　$vl(k)=\text{Min}\{vl(j)-\text{Weight}(v_k,v_j)\}$
3. 活动 a_i 的最早开始时间 $e(i)$：该活动弧的起点所表示的事件的最早发生时间。
若边 $<v_k,v_j>$ 表示活动 a_i，则有 $e(i)=ve(k)$。
4. 活动 a_i 的最迟开始时间 $l(i)$：该活动弧的终点所表示事件的最迟发生时间与该活动所需时间之差。
若边 $<v_k,v_j>$ 表示活动 a_i，则有 $l(i)=vl(j)-\text{Weight}(v_k,v_j)$。

表 1: 图算法的时间复杂度

图算法	时间复杂度	
	邻接矩阵	邻接表
深度优先搜索	$O(V^2)$	$O(V+E)$
广度优先搜索	$O(V^2)$	$O(V+E)$
拓扑排序	$O(V^2)$	$O(V+E)$
Prim	$O(V^2)$	$O(V^2)$
Dijkstra	$O(V^2)$	$O(V^2)$

1.5 查找

查找的基本概念、顺序查找法、分块查找法、折半查找法

1. 顺序查找的平均查找长度：查找成功：$\frac{n+1}{2}$；查找失败：无序表：$n+1$，有序表：$\frac{n}{2}+\frac{n}{n+1}$.
2. 分块查找：将长度为 n 的查找表均匀分成 b 块，每块有 s 个记录，在等概率情况下，若在块内和索引表中均采用顺序查找，平均查找长度为：$\text{ASL}=L_I+L_S=\frac{s^2+2s+n}{2s}$.
3. 折半查找选取中间结点时，可以向上或向下取整。

1.5.1 树型查找

1. 二叉树搜索树

删除并插入叶结点，得到的二叉树与原来的相同；删除并插入中间结点，得到的二叉树与原来的必不相同。

2. 平衡二叉树：**2009-04, 2010-04, 2012-04, 2013-03, 2015-04, 2019-04**

1. 以 n_h 表示深度为 h 的平衡树中含有的最少结点树，则 $n_0 = 0, n_1 = 1, n_2 = 2, n_h = n_{h-1} + n_{h-2} + 1$.
2. 删除并插入叶结点或中间结点，得到的平衡树与原来的可能相同。

3. 红黑树

红黑树插入：待插入结点 z 着红色。
1.z 的父节点是黑色，无需调整；z 是根结点，着为黑色；
2.z 不是根结点且 z 的父节点是红色的：
1）z 的叔结点是红色的，z 的父节点和叔结点着为黑色，z 的爷结点着为红色并作为新 z 结点重复；
2）z 的叔结点是黑色的，类似平衡树的四种情况，LL、LR、RR、RL。

1.5.2 B 树及其基本操作、B+ 树的基本概念

m 阶 B 树：**2009-08, 2012-09, 2013-10, 2014-09, 2016-10, 2017-09, 2018-08, 2020-10**
1. 每个结点至多 m 棵子树，至多含 $m-1$ 个关键字。若根结点不是叶节点，则至少有 2 棵子树。
2. 除根结点外所有非叶结点至少有 $\lceil m/2 \rceil$ 棵子树，至少含有 $\lceil m/2 \rceil - 1$ 个关键字。
3. 若含 n 个关键字，高度为 h，则 $\log_m(n+1) \leqslant h \leqslant \log_{\lceil m/2 \rceil}((n+1)/2) + 1$.

1.5.3 散列 (Hash 表)

散列表的平均查找长度：**2010-41, 2013-42, 2018-09, 2019-08**
长度为 n 的散列表，散列函数为 $H(key) = key\%m(m < n)$，采用线性探查法解决冲突。
1. 查找成功的平均查找长度：按插入的关键字序列计算即可。
2. 查找失败的平均查找长度：查找失败时的地址可能有 m 个，若填入的关键字序列最高位地址为 $t(t > m-1)$，则地址为 0 的关键字比较次数为 $1 + (t+1)$(线性探查) 次，以此类推。

字符串模式匹配、查找算法的分析及应用

1.6 排序

排序的基本概念、直接插入排序、折半插入排序、起泡排序、简单选择排序、希尔排序、快速排序

```
void insertSort(int *a, int n) {
  int j, tmp;
  for (int i = 1; i < n; i++)
    if (a[i] < a[i - 1]) {
      tmp = a[i];
      for (j = i - 1; a[j] > tmp && j >= 0; j--) { a[j + 1] = a[j]; }
      a[j + 1] = tmp;
    }
}
```

```
int partition(int *a, int lo, int hi) {
  int pivot = a[lo];
  while (lo < hi) {
    while (lo < hi && a[hi] >= pivot) { hi--; } a[lo] = a[hi];
    while (lo < hi && a[lo] <= pivot) { lo++; } a[hi] = a[lo];
  }
  a[lo] = pivot; return lo;
}
void _quickSort(int *a, int lo, int hi) { // [lo, hi]
  if (lo < hi) {
    int pivot = partition(a, lo, hi);
    _quick_sort(a, lo, pivot - 1); _quick_sort(a, pivot + 1, hi);
  }
}
void quickSort(int *a, int n) { _quick_sort(a, 0, n - 1); }
```

堆排序、二路归并排序、基数排序

1.6.1 外部排序

最佳归并树：**2013-04, 2019-11**
初始归并段不足以构成一颗严格 k 叉树时，需添加长度为 0 的“虚段”。设度为 0 的结点 $n_0 = n$ 个，度为 k 的结点有 n_k 个，对严格 k 叉树有 $n_0 = (k-1)n_k + 1$，由此可得 $n_k = (n_0 - 1)/(k-1)$.
1. 若 $(n_0 - 1)\%(k-1) = 0$，则说明这 n_0 个结点正好可以构成 k 叉树，内结点有 n_k 个。
2. 若 $(n_0 - 1)\%(k-1) = u \neq 0$，则说明这 n_0 个结点有 u 个结点多余，应在原有的 n_k 个结点基础上增加一个内结点，即再加上 $k - u - 1$ 个空归并段。

排序算法的分析和应用:

选取排序方法需要考虑的因素：数据规模，元素本身信息量的大小，数据初始状态，算法的稳定性，存储结构

表 2: 各种排序算法的性质

算法种类	时间复杂度			空间复杂度	是否稳定
	最好情况	平均情况	最坏情况		
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	是
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	是
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	否
快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(\log_2 n)$	否
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	否
2 路归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	是
基数排序	$O(d(n+r))$	$O(d(n+r))$	$O(d(n+r))$	$O(r)$	是

1.7 往年代码题

2009-42 查找带头结点的单链表中倒数第 k 个位置的结点。查找成功输出 $data$ 域并返回 1，否则只返回 0。
思路： 双指针，q 比 q 先移动 k，再同时移动，q 到达链表尾时 p 指向倒数第 k 个元素。

```
int findLast(Node *list, int k) {
  Node *p = list->link, *q = list->link; int count = 0;
  while (q != NULL) {
    if (count < k) count++;
    else p = p->link;
    q = q->link;
  }
  if (count < k) return 0;
  printf("%d", p->data);
  return 1;
}
```

2010-42 将长为 n 的数组循环左移 p 个位置。
思路： $ab \to a^{-1}b^{-1} \to ba$

```
void swap(int *a, int *b) { int tmp = *a; *a = *b; *b = tmp; }
void reverse(int *a, int lo, int hi) { // [lo, hi]
  while (lo < hi) swap(&a[lo++], &a[hi--]);
}
void shift(int *a, int n, int p) {
  reverse(a, 0, p - 1);
  reverse(a, p, n - 1);
  reverse(a, 0, n - 1);
}
```

2011-42 求两个等长升序序列 a 和 b 的中位数。
思路： 归并思想，双指针

```
int findMid(int *a, int *b, int n) {
  int i = 0, j = 0, k = 0;
  while (i < n && j < n) {
    k++;
    if (a[i] == b[j]) return a[i];
    if (a[i] < b[j]) { i++; if (k == n) return a[i - 1]; }
    else { j++; if (k == n) return b[j - 1]; }
  }
}
```

2012-42 求两个带有头节点的字符类型的单链表的相同后缀位置。
思路：双指针，根据表长调整初始位置，再同时移动

```
Node *findSuffix(Node *str1, Node *str2) {
  int m = listLen(str1), n = listLen(str2), ahead = abs(m - n);
  Node *p = str1->next, *q = str2->next;
  if (m < n) { for (int i = 0; i < ahead; i++) q = q->next; }
  else { for (int i = 0; i < ahead; i++) p = p->next; }
  while (p != NULL && q != NULL) {
    if (p->data == q->data) return p;
    p = p->next; q = q->next;
  }
  return NULL;
}
```

2013-41 查找数组中的主元素。
思路：空间换时间，存每个元素出现的次数，找最大值，比较

```
int findMain(int *a, int n) {
  int b[n]; for (int i = 0; i < n; i++) b[i] = 0;
  int max = 0;
  for (int i = 0; i < n; i++) {
    b[a[i]]++; if (b[a[i]] > b[max]) max = a[i];
  }
  if (b[max] <= n / 2) max = -1;
  return max;
}
```

2014-41 二叉树的带权路径长度。
思路：先序遍历，递归解决；层次遍历，用两个指针记录最近访问的层最后一个结点求结点深度

```
int _wpl(Node *p, int depth) {
  if (p->left == NULL && p->right == NULL) return (p->weight * depth);
  else return _wpl(p->left, depth + 1) + _wpl(p->right, depth + 1);
}
int wpl(Node *root) {
  return _wpl(root, 0);
}
```

```
int wpl(Node *root) {
  int M = 32, front = 0, rear = 0, wpl = 0, depth = 0;
  Node *queue[M], *p = root, *l = root, *nl = NULL;
  queue[rear] = p; rear = (rear + 1) % M;
  while (front != rear) {
    p = queue[front]; front = (front + 1) % M;
    if (p->left == NULL && p->right == NULL) wpl += depth * p->weight;
    if (p->left != NULL) {
      queue[rear] = p->left; rear = (rear + 1) % M; nl = p->left;
    }
    if (p->right != NULL) {
      queue[rear] = p->right; rear = (rear + 1) % M; nl = p->right;
    }
    if (p == l) { l = nl; depth++; }
  }
  return wpl;
}
```

2015-41 删除带头节点的单链表中第二次出现的绝对值相同的节点。
思路： 空间换时间，利用数组记录出现次数，第二次出现即删除

```
void duplicate(Node *head, int n) {
  int *count = (int *) malloc(sizeof(int) * (n + 1));
  for (int i = 0; i < n + 1; i++) count[i] = 0;
  Node *p = head->link, *r = head, *q;
  while (p != NULL) {
    if (count[abs(p->data)] == 0) {
      count[abs(p->data)] = 1;
      r = p; p = p->link;
    } else {
      q = p; p = q->link;
      r->link = p;
      free(q);
    }
  }
}
```

2016-43 将长为 n 的数组划分为最小的 $\lfloor n/2 \rfloor$ 个元素和最大的的 $\lceil n/2 \rceil$ 个元素。
思路： 快速排序划分子数组的思想，将数组划分为前 $n/2-1$ 个元素都比后半部分小

```
int partition(int *a, int n) {
  int lo = 0, hi = n - 1, llo = 0, lhi = n - 1, k = n / 2 - 1, pivot;
  while (1) {
    pivot = a[lo];
    while (lo < hi) {
      while (lo < hi && pivot <= a[hi]) hi--; if (lo != hi) a[lo] = a[hi];
      while (lo < hi && pivot >= a[lo]) lo++; if (lo != hi) a[hi] = a[lo];
    }
    a[lo] = pivot;
    if (lo == k) break;
    if (lo < k) { llo = ++lo; hi = lhi; }
    else { lhi = --hi; lo = llo; }
  }
  int s1 = 0, s2 = 0;
  for (int i = 0; i < lo + 1; i++) s1 += a[i];
  for (int i = lo + 1; i < n; i++) s2 += a[i];
  return s2 - s1;
}
```

2017-41 二叉树转中缀表达式，通过括号反映操作数的计算次序。
思路： 叶结点直接输出，中间结点且不为根，利用中序遍历思想，外层加括号

```
void _expr(BTree *root, int depth) {
  if (root == NULL) return;
  if (root->left == NULL && root->right == NULL) {
    printf("%s", root->data); return;
  }
  if (depth > 1) printf("(");
  _expr(root->left, depth + 1);
  printf("%s", root->data);
  _expr(root->right, depth + 1);
  if (depth > 1) printf(")");
}
void expr(BTree *root) { _expr(root, 1); }
```

2018-41 给定一个含 $n(\geqslant 1)$ 个整数的数组，请设计一个在时间上尽可能高效的算法，找出数组中未出现的最小正整数。例如，数组 {-5,3,2,3} 中未出现的最小正整数是 1；数组 {1,2,3} 中未出现的最小正整数是 4。
思路： 默认元素不大于数组元素个数（纯放屁），和主元素一个思路（2013-41）

```
int findMin(int *a, int n) {
  int m = 0, k;
  for (int i = 0; i < n; i++) if (a[i] > m) m = a[i];
  int *b = (int *) malloc(sizeof(int) * (m + 1));
  for (int i = 0; i < m + 1; i++) b[i] = 0;
  for (int i = 0; i < n; i++) if (a[i] > 0) b[a[i] - 1]++;
  for (k = 0; k < m + 1; k++) if (b[k] == 0) break;
  return k + 1;
}
```

2019-41 线性表 $L = (a_1, a_2, a_3, \ldots, a_{n-2}, a_{n-1}, a_n)$ 采用带头结点的单链表保存，请设计一个空间复杂度为 $O(1)$ 且时间上尽可能高效的算法, 重新排列 L 中的各结点, 得到线性表 $L' = (a_1, a_n, a_2, a_{n-1}, a_3, an-2, \ldots)$.
思路： 分成两半，原地逆置后半部分，再和前半部分交叉相连
1. 分成两半：双指针，q 比 p 每次多移动一步，q 到达链表尾时 p 指向链表中间节点
2. 原地逆置：单链表的尾插法
3. 交叉相连：双指针，类似二路归并链表

```
void replace(Node *l) {
  Node *p = l, *q = p, *r;
  while (q->next != NULL) {
    p = p->next; q = q->next;
    if (q->next != NULL) q = q->next;
  }
  q = p->next; p->next = NULL;
  while (q != NULL) {
    r = q; q = q->next;
    r->next = p->next;
    p->next = r;
  }
  q = p->next; p->next = NULL; p = l->next;
  while (q != NULL) {
    r = q; q = q->next;
    r->next = p->next; p->next = r;
    p = r->next;
  }
}
```

2020-41 定义三元组 (a, b, c)(其中 a, b, c 均为正数) 的距离 $D = |a - b| + |b - c| + |c - d|$。给定 3 个非空整数集合 $S1$、$S2$ 和 $S3$，按升序分别存储在 3 个数组中。设计一个尽可能高效的算法，计算并输出所有可能的三元组 $(a, b, c)(a \in S1, b \in S2, c \in S3)$ 中的最小距离。例如 $S1 = \{-1, 0, 9\}, S2 = \{-25, -10, 10, 11\}, S3 = \{2, 9, 17, 30, 41\}$，则最小距离为 2，相应的三元组为 $(9, 10, 9)$。
思路： 三指针，根据三元素大小情况进行移动

```
int trip(int *s1, int n1, int *s2, int n2, int *s3, int n3) {
  int i = 0, j = 0, k = 0, a, b, c, d, min = 0x7fffffff;
  while (i < n1 && j < n2 && k < n3) {
    a = s1[i]; b = s2[j]; c = s3[k];
    d = abs(a-b) + abs(b-c) + abs(c-a); if (d < min) min = d;
    if (a < b) { a < c ? i++ : k++; }
    else { b < c ? j++ : k++; }
  }
  return min;
}
```

2021 计算邻接矩阵表示的无向图的度为奇数的顶点个数，若为不大于 2 的偶数，返回 1，否则返回 0。
思路：遍历邻接矩阵，计算各个顶点的度数，并统计度为奇数的个数，若为 0 或2（**SB 还在这里犯错！！！**），返回 1，否则返回 0。

```
int IsExistEL(MGraph G) {
  int count = 0, d;
  for (int i = 0; i < G.numVertices; i++) {
    d = 0;
    for (int j = 0; j < G.numVertices; j++) d += G.Edge[i][j];
    if (d % 2 == 1) count++;
  }
  if (count == 0 || count == 2) return 1;
  else return 0;
}
```

2022 思路：

2023 思路：

2 计算机组成原理

2.1 计算机系统概述

计算机系统层次结构：1. 计算机系统的基本组成 2. 计算机硬件的基本组成 3. 计算机软件和硬件的关系 4. 计算机系统的工作原理

MAR 位数与地址线位数相同，MDR 位数与数据线位数相同。

“存储程序”工作方式，高级语言程序与机器语言程序之间的转换，程序和指令的执行过程

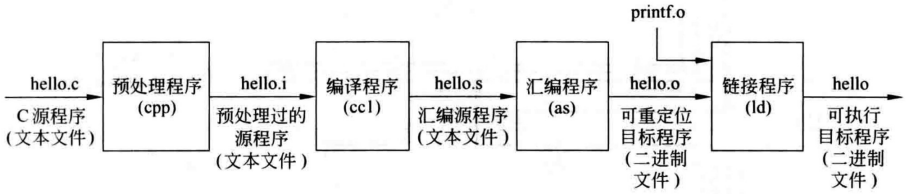


图 1: hello.c 源程序文件到可执行目标文件的转换过程

计算机性能指标：吞吐量、响应时间；CPU 时钟周期、主频、CPI、CPU 执行时间；MIPS、MFLOPS、GFLOPS、TFLOPS、PFLOPS、EFLOPS、ZFLOPS

2.2 数据的表示和运算

数制与编码：1. 进位计数制及其数据之间的相互转换 2. 定点数的编码表示

2.2.1 运算方法和运算电路

1. 基本运算部件：加法器，算术逻辑部件 (ALU) 2. 加减运算：补码加/减运算器，标志位的生成

1. 补码运算：$Sub=1$ 时，减法，$X+\overline{Y}+1=[x]_{补}+[-y]_{补}$，$Sub=0$ 时，加法，$X+Y=[x]_{补}+[y]_{补}$.
2. 标志位的生成：
1）溢出标志 OF：$OF=1$ 表示带符号数运算发生溢出；无符号数运算没有意义. $OF=C_n\oplus C_{n-1}$.
2）符号标志 SF：结果的符号，无符号数没有意义. $SF=F_{n-1}$.
3）零标志 ZF：$ZF=1$ 表示结果为 0，无符号/带符号整数都有意义. $ZF=F=0$.
4）进位/借位标志 CF：加法时，$CF=1$ 表示无符号数加法溢出，减法时，$CF=1$ 表示有借位，不够减；带符号整数没有意义. $CF=Cout\oplus Cin$.

移位运算：带符号负数左移移出位不全为 1 时溢出，正数不全为 0 时溢出。

循环移位：
1）小循环左移：最高位移入进位标志位，同时也移入最低位。
2）小循环右移：最低位移入进位标志位，同时也移入最高位。
3）大循环左移：最高位移入进位标志位，而进位标志位移入最低位。
4）大循环右移：最低位移入进位标志位，而进位标志位移入最高位。

3. 乘除运算：乘/除法运算的基本原理，乘法电路和除法电路的基本结构

表 3: 条件转移指令中标志信息

	条件	标志位
无符号整数	$A > B$	$CF = 0$ AND $ZF = 0$
	$A \geqslant B$	$CF = 0$ OR $ZF = 1$
	$A < B$	$CF = 1$ AND $ZF = 0$
	$A \leqslant B$	$CF = 1$ OR $ZF = 1$
有符号整数	$A > B$	$SF = OF$ AND $ZF = 0$
	$A \geqslant B$	$SF = OF$ OR $ZF = 1$
	$A < B$	$SF \neq OF$ AND $ZF = 0$
	$A \leqslant B$	$SF \neq OF$ OR $ZF = 1$

1. 补码除法：被除数和除数同号相减，异号相加。
2. 乘法运算溢出判断（n 位乘法，$2n$ 位乘积）：
带符号整数乘法：高 33 位非全 0 或非全 1 溢出；无符号整数乘法：高 32 位非全 0 溢出。

整数的表示和运算：1. 无符号整数的表示和运算 2. 带符号整数的表示和运算

2.2.2 浮点数的表示和运算

1. 浮点数的表示：IEEE754 标准**2012-14, 2018-14**

1）全 0 阶码全 0 尾数，$+0/-0$，零的符号取决于数符 S。
2）全 1 阶码全 0 尾数，$+\infty/-\infty$，引入无穷大数是为了计算过程出现异常的情况下程序能继续进行。

表 4: IEEE754 浮点数的范围

格式	最小值	最大值
单精度	$(-1)^s \times 1.0 \times 2^{1-127} = (-1)^s \times 2^{-126}$	$(-1)^s \times 1.11\ldots1 \times 2^{254-127} = (-1)^s \times 2^{127} \times (2 - 2^{-23})$
双精度	$(-1)^s \times 1.0 \times 2^{1-1023} = (-1)^s \times 2^{-1022}$	$(-1)^s \times 1.11\ldots1 \times 2^{2046-1023} = (-1)^s \times 2^{1023} \times (2 - 2^{-52})$

2. 浮点数的加减运算

1. 溢出判断：右规和尾数舍入都有可能引起阶码上溢；左规可能引起阶码下溢；尾数溢出结果不一定溢出。
2. 判断浮点数是否是规格化数：
1）原码编码的尾数：尾数第一位是否为 1。
2）补码编码的尾数：符号位和尾数最高位是否相反。

2.3 存储器层次结构

存储器的分类

1. 随机存取存储器（RAM）：按地址访问，SRAM（Cache），DRAM（主存）。
2. 相联存储器（TLB）：按内容访问。

层次化存储器的基本结构、半导体随机存取存储器：1. SRAM 存储器 2. DRAM 存储器 3. Flash 存储器
主存储器：1.DRAM 芯片和内存条 2. 多模块存储器 3. 主存和 CPU 之间的连接
外部存储器：1. 磁盘存储器 2. 固态硬盘 (SSD)

2.3.1 高速缓冲存储器（Cache）

1.Cache 的基本原理 2.Cache 和主存之间的映射方式 3.Cache 中主存块的替换算法 4.Cache 写策略

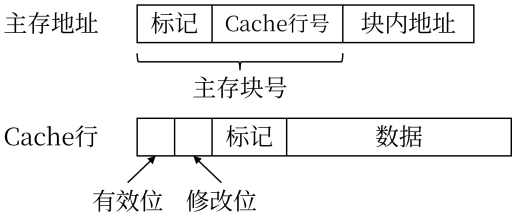


图 2: Cache 直接映射

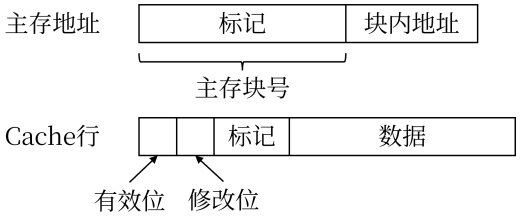


图 3: Cache 全相联映射

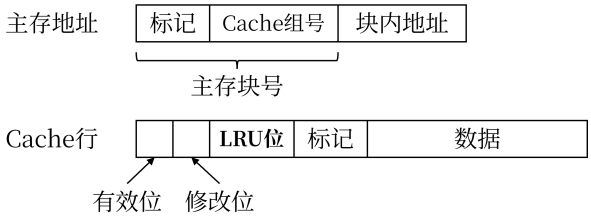


图 4: Cache 组相联映射

2.3.2 虚拟存储器

1. 虚拟存储器的基本概念 2. 页式虚拟存储器：基本原理，页表，地址转换，TLB（快表）3. 段式虚拟存储器 4. 段页式虚拟存储器

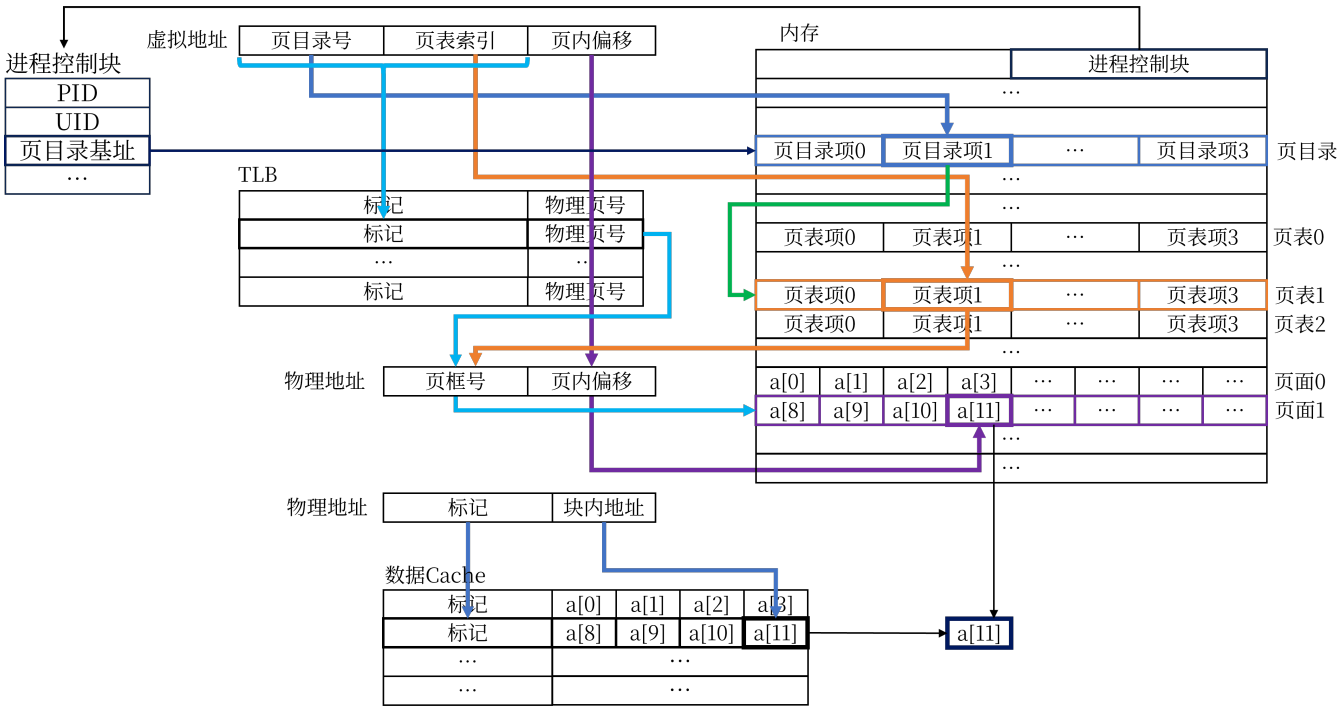


图 5: TLB 和 Cache 的访问过程

2.4 指令系统

指令系统的基本概念、指令格式、寻址方式、数据的对齐和大/小端存放方式、CISC 和 RISC 的基本概念

高级语言程序与机器级代码之间的对应：1. 编译器，汇编器和链路器的基本概念 2. 选择结构语句的机器级表示 3. 循环结构语句的机器级表示 4. 过程（函数）调用对应的机器级表示

2.5 中央处理器 (CPU)

CPU 的功能和基本结构、指令执行过程

2.5.1 数据通路的功能和基本结构

数据通路：指令执行过程中数据所经过的路径，包括路径上的部件称为数据通路。ALU、通用寄存器、状态寄存器、cache、MMU、浮点运算逻辑、异常和中断处理逻辑 等都是指令执行过程中数据流经的部件。

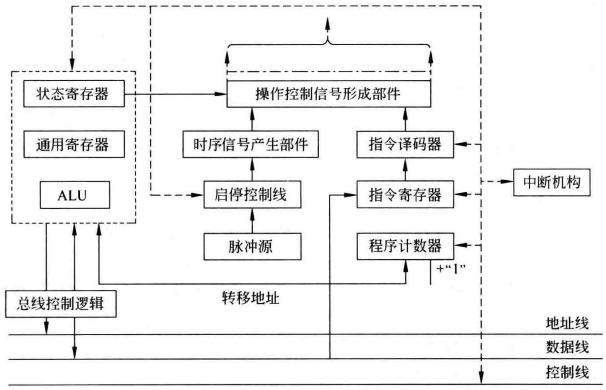


图 6: CPU 基本组成原理图

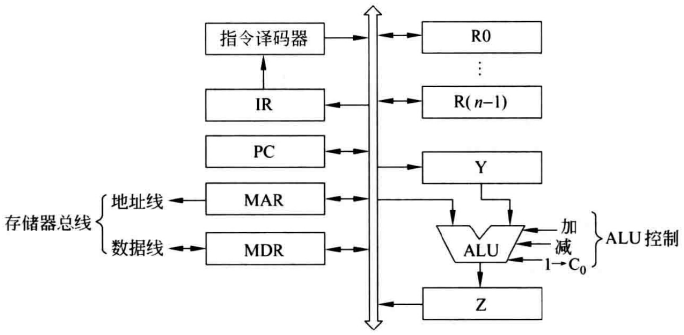


图 7: 单总线数据通路

2.5.2 控制器的功能和工作原理

1. 每条机器指令对应一个微程序，每个微程序包含若干微指令，每条微指令对应一个或几个微操作命令。
2. 微命令是微操作的控制信号，微操作是微命令的执行过程。
3.控制存储器：在 CPU 内部，由 ROM 构成，存放微程序，按地址访问。
4. 微指令的编码方式：字段直接编码：相容性微命令分在不同段，每个字段留出一个状态表示不发出微命令。
5. 微指令的格式：
1）水平型微指令：微程序短，执行速度快；微指令长，编写微程序麻烦。
2）垂直型微指令：微指令短，便于编写；微程序长，执行速度慢，工作效率低。

2.5.3 异常和中断机制

1. 异常和中断的基本概念 2. 异常和中断的分类 3. 异常和中断的检测与响应

1. 中断的基本概念：**2009-22, 2015-22, 2016-22, 2020-18, 2020-21**
1）“缺页”或“溢出”等异常事件是由特定指令在执行过程中产生的；中断相对于指令的执行则是异步的，中断不和任何指令相关联。CPU 只需要在开始一个新指令之前检测是否有外部发来的中断请求。
2）异常的发生和异常事件的类型是由 CPU 自身发现和识别的，不必通过外部的某个信号通知 CPU，而 CPU 必须通过对外部中断请求线进行采样，才能获知哪个设备发生了何种中断。
2. 自陷：除转移指令外，自陷处理完成后返回到陷阱指令的下一条指令执行。

2.5.4 指令流水线

1. 指令流水线的基本概念

理想情况下，每个时钟周期都有一条指令进入流水线，每个时钟周期都有一条指令完成，CPI=1。

2. 指令流水线的基本实现 3. 结构冒险、数据冒险和控制冒险的处理

1. 结构冒险：采用数据 cache 和代码 cache 分离的方式。
2. 数据冒险：
1）插入空操作指令：在软件上采取措施，使相关指令延迟执行；
2）插入气泡：在硬件上采取措施，使相关指令延迟执行；
3）采用转发技术（数据旁路）：将数据通路中生成的中间数据直接转发到 ALU 的输入端。
3. 控制冒险：
1）对转移指令进行分支预测，尽早生成转移目标地址；
2）预取转移成功和不成功两个控制流方向上的目标指令。

4. 超标量和动态流水线的基本概念

1. 超标量流水线技术：每个时钟周期内可并发多条独立指令，需配置多个功能部件。多数超标量 CPU 都结合动态流水线调度技术，通过动态分支预测等手段提高指令并行性。
2. 超流水线技术：通过提高流水线主频的方式提升流水线性能。

2.5.5 多处理器基本概念

1.SISD、SIMD、MIMD、向量处理器的基本概念 2. 硬件多线程的基本概念

表 5: 硬件多线程的基本概念

	细粒度多线程	粗粒度多线程	同时多线程（SMT）
指令发射	轮流发射各线程的指令	连续几个时钟周期，都发射同一线程的指令序列，流水线阻塞时切换另一线程	一个时钟周期内同时发射多个线程的指令
线程切换频率	每个时钟周期切换一次线程	只有流水线阻塞时才切换一次线程	-
线程切换代价	低	高，需要重载流水线	-
并行性	指令级并行，线程间不并行	指令级并行，线程间不并行	指令级并行，线程级并行

3. 多核处理器 (multi-core) 的基本概念 4. 共享内存多处理器 (SMP) 的基本概念

2.6 总线和输入/输出系统

2.6.1 总线

1. 总线的基本概念 2. 总线的组成及性能指标 3. 总线事务和定时

1. 数据线：传输数据、命令或地址（数据线和地址线复用）。
2. 地址线：主存单元或 I/O 端口的地址，地址线是单向的。
3. 控制线：控制对数据线和地址线的访问和使用，传输定时信号和命令信息。
4. 同步总线采用公共的时钟信号进行定时，适合于存取时间相差不大的多个功能部件之间的通信。
5. 更多总线采用异步串行方式进行传输。串行总线每次在一根信号线上传送数据位，传输速率可以比并行总线高得多。可以实现比传统并行总线高得多的数据传输带宽。
6.总线带宽（总线的最大数据传输率）= 总线宽度（总线上同时能够传送的数据位数）× 总线频率。

2.6.2 I/O 接口 I/O 控制器

1.I/O 接口的功能和基本结构

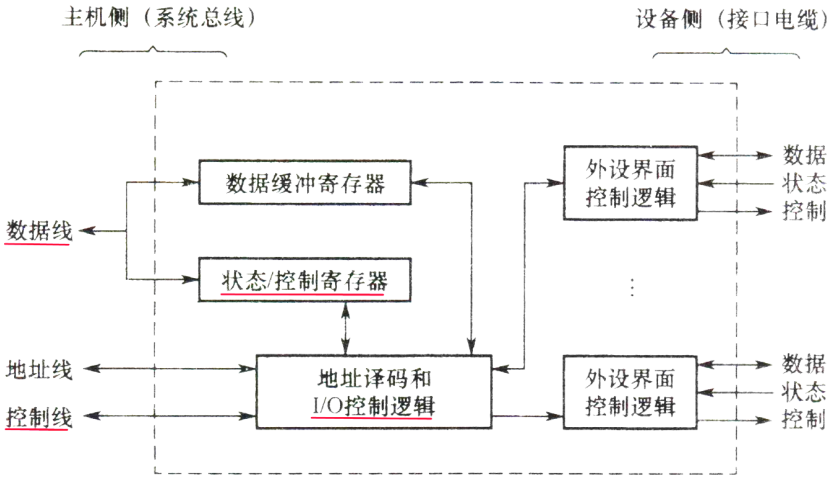


图 8: I/O 接口的基本结构

I/O 总线的控制线只是对端口进行读写控制，命令字通过数据线传输。

2.I/O 端口及其编址

1.I/O 端口：可被 CPU 直接访问的寄存器，CPU 对数据端口可读可写，状态端口只读，控制端口只写。
2. 编址方式：
1）统一编址：I/O 端口当作存储器的单元进行地址分配，使用统一的访存指令访问 I/O 端口。
2）独立编址：设置专门的I/O 指令 来访问 I/O 端口。

2.6.3 I/O 方式

1. 程序查询方式
2. 程序中断方式：中断的基本概念；中断响应过程；中断处理过程；多重中断和中断屏蔽的概念

1. 中断响应过程（硬件实现）：**2010-21, 2011-21, 2012-22, 2013-22, 2017-22, 2018-22, 2019-22**
1）关中断：屏蔽掉所有可屏蔽中断请求。
2）保护断点：将 PC 和 PSW 送入栈或特殊寄存器。
3）识别中断源并转中断服务程序：取优先级最高中断源的中断服务程序首址和初始 PSW，并送 PC 和 PSWR。
2. 中断优先级：
1）中断响应优先级：CPU 响应中断请求的先后顺序。通常是通过硬件排队器实现的。
2）中断处理优先级：多重中断实际 优先级处理次序，中断屏蔽技术动态调整，灵活调整中断服务程序优先级。

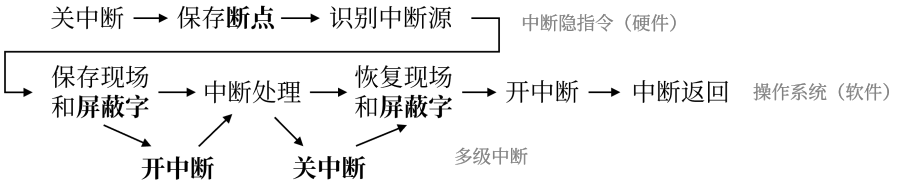


图 9: 中断过程

3.DMA 方式：DMA 控制器的组成，DMA 传送过程

1.DMA 传送方式：
1）停止 CPU 访存：传送一块数据过程中 CPU 不可访问主存。
2）周期挪用：DMA 控制器传送完一个数据立即释放总线。
3）交替访存：分时控制总线。
2. 中断方式不适合快速设备，DMA 也可用于低速设备。

图 10: DMA 请求

3 操作系统

3.1 操作系统概述

操作系统的基本概念、操作系统的发展历程

多道程序系统：资源利用率高，吐量大，CPU 和其他资源保持“忙碌”状态；用户响应时间长，无交互能力。

程序运行环境：1.CPU 运行模式：内核模式、用户模式 2. 中断和异常的处理 3. 系统调用 4. 程序的链接与装入 5. 程序运行时内存映像与地址空间

特权指令：有关对 I/O 设备操作的指令；有关访问程序状态的指令；存取特殊寄存器指令等。

操作系统结构：分层，模块化，宏内核，微内核，外核、**操作系统引导、虚拟机**

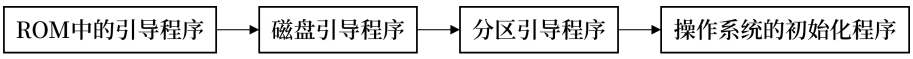


图 11: 操作系统启动

3.2 进程管理

进程与线程：1. 进程与线程的基本概念 2. 进程/线程的状态与转换 3. 线程的实现：内核支持的线程，线程库支持的线程 4. 进程与线程的组织与控制 5. 进程间通信：共享内存，消息传递，管道

3.2.1 CPU 调度与上下文切换

1. 调度的基本概念 2. 调度的目标

1. 周转时间：指从作业提交到作业完成所经历的时间：周转时间 = 作业完成时间-作业提交时间。
2. 响应时间：指从用户提交请求到系统首次产生响应所用的时间。

3. 调度的实现：调度器/调度程序 (scheduler)，调度的时机与调度方式（抢占式/非抢占式），闲逛进程，内核级线程与用户级线程调度

4. 典型调度算法：先来先服务调度算法；短作业（短进程、短线程）优先调度算法；时间片轮转调度算法；优先级调度算法；高响应比优先调度算法；多级队列调度算法，多级反馈队列调度算法

响应比 $=$ (等待时间 $+$ 执行时间)/执行时间，高响应比优先综合考虑了进程等待时间和执行时间

5. 上下文及其切换机制

3.2.2 同步与互斥

1. 同步与互斥的基本概念 2. 基本的实现方法：软件方法；硬件方法 3. 锁 4. 信号量 5. 条件变量

1. 同步信号量初值为 0，互斥信号量初值为可用资源数 n.
2. 条件变量作用类似于信号量，都用于实现进程同步。
3. 管程：在同一时刻，管程中只能有一个进程 在执行，wait() 操作会使当前进程阻塞。

6. 经典同步问题：生产者-消费者问题；读者-写者问题；哲学家进餐问题

1）生产者-消费者问题

```
semaphore mutex = 1; /* 互斥访问缓冲区 */
semaphore empty = n; /* 缓冲区空的个数 */
semaphore full  = 0; /* 缓冲区满的个数 */
```

```
void Tproduce() /* 生产者进程 */
  while (1) {
    P(&empty);
    P(&mutex);
    /* 放入缓冲区 */
    V(&mutex);
    V(&full);
  }
```

```
void Tconsume() /* 消费者进程 */
  while (1) {
    P(&full);
    P(&mutex);
    /* 从缓冲区拿出 */
    V(&mutex);
    V(&empty);
  }
```

2）读者-写者问题（“写优先”）

```
semaphore wlock    = 1; /* 写者锁 */
semaphore rlock    = 1; /* 读者锁 */
semaphore rwlock   = 1; /* 读写锁 */
semaphore mutex_rc = 1; /* 互斥访问rcount */
int       rcount   = 0; /* 读者数量计数 */
```

```
void Tread()
  while (1) {
    P(&rwlock);==================\
    P(&mutex_rc);--------------\ |
    rcount++;                  | |
    /* 第一个读者准备读 */     | |
    if (rcount == 1) P(&wlock);| |
    V(&mutex_rc);--------------/ |
    V(&rwlock);==================/
    /* 读文件 */
    P(&mutex_rc);--------------\
    rcount--;                  |
    /* 最后一个读者读完 */     |
    if (rcount == 0) V(&wlock);|
    V(&mutex_rc);--------------/
  }
```

```
void Twrite()
  while (1) {
    P(&rwlock);=======\
    P(&wlock);------\ |
    /* 写文件 */    | |
    V(&wlock);------/ |
    V(&rwlock);=======/
  }
```

3）哲学家进餐问题

```
semaphore chops[n] = {1,1,...,1};
semaphore mutex    = 1; /* 每一时刻只能一位哲学家拿筷子 */
void Tphilosopher(int i)
  while (1) {
    P(&mutex); /* 同时拿左边和右边的筷子 */
    P(&chops[i]);
    P(&chops[(i + 1) % n]);
    V(&mutex);
    /* 进餐 */
    V(&chops[i]);
    V(&chops[(i + 1) % n]);
  }
```

4）在读者写者问题的基础上增加如下条件：若写者写完的内容还没有被任何一个读者读取，则新的写进程不能进行写操作，直到有至少一个读进程进行了读操作。

```
semaphore wlock    = 1; /* 写者锁 */
semaphore rwlock   = 1; /* 读写锁 */
semaphore wwlock   = 1; /* 写者写完至少读一次 */
semaphore mutex_rc = 1; /* 互斥访问 rcount */
semaphore mutex_rt = 1; /* 互斥访问 rtime */
int       rcount   = 0; /* 读者数量 */
int       rtime    = 0; /* 文件读的次数 */
```

```
void Twrite()
  while (1) {
    P(&wwlock);
    P(&rwlock);------------------\
    P(&wlock);==================\ |
    /* 写文件 */                | |
    P(&mutex_rt);------------\ | |
    /* 写者写完，读次数置0 */| | |
    rtime = 0;               | | |
    V(&mutex_rt);------------/ | |
    V(&wlock);==================/ |
    V(&rwlock);------------------/
  }
```

```
void Tread()
  while (1) {
    P(&rwlock);
    P(&mutex_rc);----------------\
    rcount++;                    |
    if (rcount == 1) P(&wlock);  |
    V(&mutex_rc);----------------/
    V(&rwlock);
    /* 读文件 */
    P(&mutex_rt);================\
    rtime++;                     |
    /* 写者写完第一次读 */       |
    if (rtime == 1) V(&wwlock);  |
    V(&mutex_rt);================/
    P(&mutex_rc);----------------\
    rcount--;                    |
    if (rcount == 0) V(&wlock);  |
    V(&mutex_rc);----------------/
  }
```

3.2.3 死锁

1. 死锁的基本概念 2. 死锁预防 3. 死锁避免 4. 死锁检测和解除 **2013-32, 2015-26, 2019-30**

表 6: 死锁处理策略比较的总结

	资源分配策略	各种可能模式	主要优点	主要缺点
死锁预防	保守，宁可资源闲置	一次请求所有资源，资源剥夺，资源按序分配	适用于突发式处理的进程，不必进行剥夺	效率低，进程初始化时间长，剥夺次数过多，不便灵活申请新资源
死锁避免	在运行时判断是否可能死锁	寻找可能的安全允许顺序	不必进行剥夺	必须知道将来的资源需求，进程不能被长时间阻塞
死锁检测	宽松，只要允许就分配资源	定期检查死锁是否已经发生	不延长进程初始化时间，允许对死锁进行现场处理	通过剥夺解除死锁，造成损失

1. 银行家算法作为一种死锁避免算法，不能判断系统是否处于死锁状态。
2. 死锁检测：通过简化资源分配图可以检测系统是否处于死锁状态。

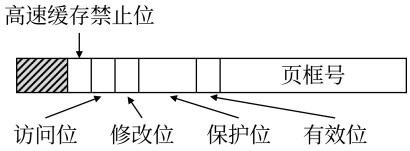
3.3 内存管理

内存管理基础：1. 内存管理的基本概念：逻辑地址空间与物理地址空间，地址变换，内存共享，内存保护，内存分配与回收 2. 连续分配管理方式 3. 页式管理 4. 段式管理 5. 段页式管理

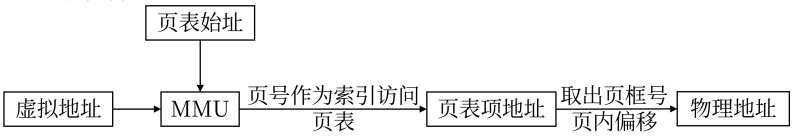
3.3.1 虚拟内存管理

1. 虚拟内存基本概念 2. 请求页式管理 3. 页框分配 4. 页置换算法 5. 内存映射文件 (Memory-Mapped Files) 6. 虚拟存储器性能的影响因素及改进方式

1. 页表项结构：

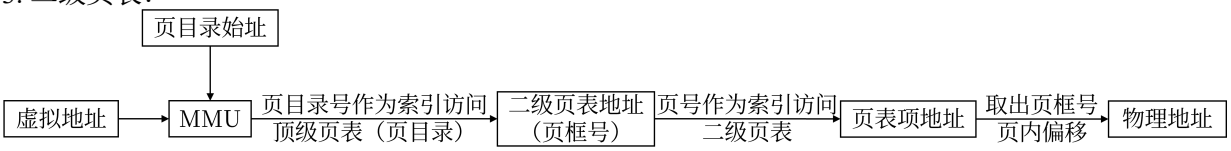


2. 一级页表：



页表项物理地址 = 页表物理始址 + 页号 × 页表项长度.

3. 二级页表：



页目录项物理地址 = 页目录物理始址 + 页目录号 × 页目录项长度.

页表项物理地址 = 二级页表始址（页框号） × 页表长度 + 页号 × 页表项长度.

4. 缺页中断处理：在含有 TLB 的页表系统中，若 TLB 未命中，页表有效位为 0，产生缺页中断，系统将页调入内存，并修改页表和 TLB，程序再读 TLB 得到页框号。

3.4 文件管理

3.4.1 文件

1. 文件的基本概念 2. 文件元数据和索引节点 (inode)

1. 进程控制块（PCB）常驻内存，在进程创建和终止过程中存在。
2. 文件控制块（FCB）、索引结点存放在磁盘中，保存文件元数据，文件存在即存在。
3. 文件分配表（FAT）系统启动时读入内存，保存文件的磁盘块链接关系和空闲的磁盘块。

3. 文件的操作：建立，删除，打开，关闭，读，写

```
int open(const char *pathname, int flags, mode_t mode);
ssize_t read(int fd, void buf[.count], size_t count);
ssize_t write(int fd, const void buf[.count], size_t count);
```

打开文件或 I/O 设备使用文件名或逻辑设备名，读写文件使用文件描述符。

4. 文件的保护 5. 文件的逻辑结构 6. 文件的物理结构

1. 连续分配：文件目录项包含起始地址 和文件长度。支持顺序访问 和直接访问。只适用于长度固定的文件。
2. 链接分配：
1）隐式链接：文件目录项含有文件第一块和最后一块的指针，除最后一个磁盘块外，每个盘块都含有一个指向文件下一个盘块的指针。只适合顺序访问。
2）显示链接：FAT（文件分配表）：磁盘启动时读入内存，一个磁盘一张表，每个表项存放盘块号的下一盘块号的指针。支持直接访问，但需要占用较大的内存空间。FAT 不仅记录了文件各块之间的先后链接关系，还标记了空闲的磁盘块，操作系统可以通过 FAT 对文件存储空间进行管理。
3. 索引分配：一个文件一个索引块，第 i 个条目指向文件的第 i 块。支持直接访问。

3.4.2 目录

1. 目录的基本概念 2. 树形目录 3. 目录的操作 4. 硬链接和软链接

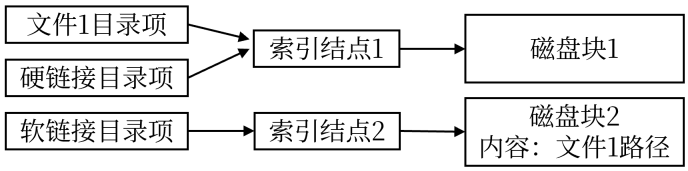


图 12: 文件软硬链接的区别

软链接不改变文件的引用计数，硬链接会改变文件的引用计数，当文件的引用的计数为 0 时才能删除文件。

3.4.3 文件系统

1. 文件系统的全局结构 (layout)：文件系统在外存中的结构，文件系统在内存中的结构
2. 外存空闲空间管理办法 3. 虚拟文件系统 4. 文件系统挂载 (mounting)

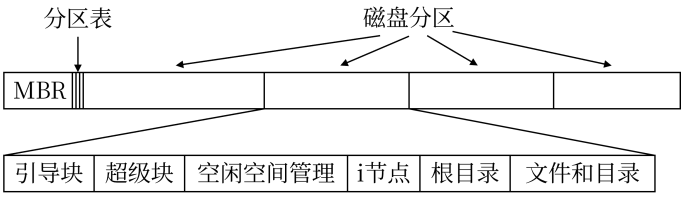


图 13: 文件系统布局

3.5 输入输出 (I/O) 管理

3.5.1 I/O 管理基础

1. 设备：设备的基本概念，设备的分类，I/O 接口，I/O 端口
2.I/O 控制方式：轮询方式，中断方式，DMA 方式
3.I/O 软件层次结构：中断处理程序，驱动程序，设备独立软件，用户层 I/O 软件

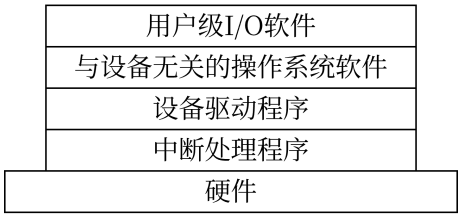


图 14: I/O 软件层次结构

4. 输入/输出应用程序接口：字符设备接口，块设备接口，网络设备接口，阻塞/非阻塞 I/O

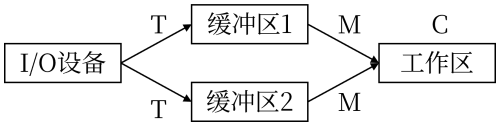
3.5.2 设备独立软件

1. 缓冲区管理

1. 单缓冲:

设初始状态为工作区满，缓冲区空，单缓冲区处理每块数据用时 $\max(C, T) + M$.

2. 双缓冲:



设初始状态为工作区空，缓冲区 1 空，缓冲区 2 满，双缓冲区处理每块数据用时 $\max(C + M, T)$.

2. 设备分配与回收 3. 假脱机技术 (SPOOLing) 4. 设备驱动程序接口

3.5.3 外存管理

1. 磁盘：磁盘结构，格式化，分区，磁盘调度方法

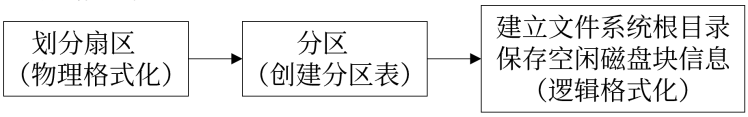
1. 磁盘结构：

柱面号 = ⌊簇号/每个柱面的簇数⌋

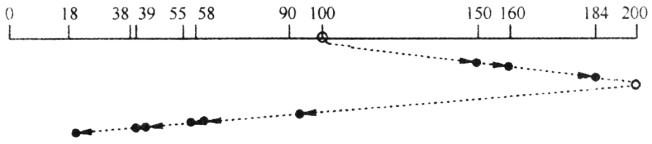
磁头号 = ⌊(簇号%每个柱面的簇数)/每个磁道的簇数⌋

扇区号 = 扇区地址%每个磁道的扇区数

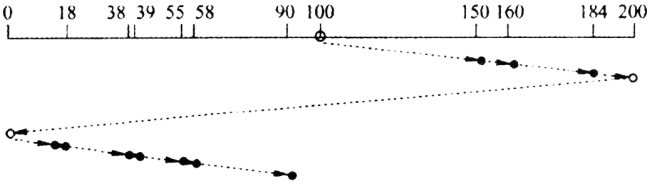
2. 磁盘格式化：



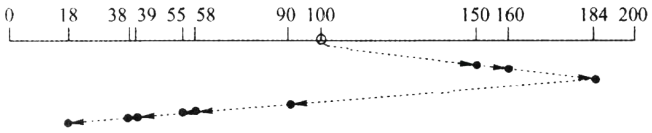
磁盘调度方法：（**注意区分**）



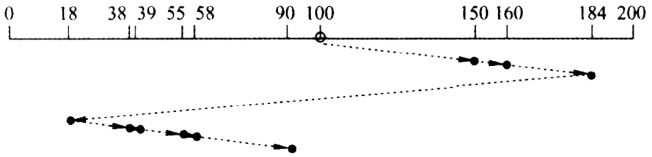
(1) SCAN



(2) C-SCAN



(3) LOOK



(4) C-LOOK

2. 固态硬盘：读写性能特性，磨损均衡

4 计算机网络

4.1 计算机网络概述

计算机网络基本概念：1. 计算机网络的定义、组成与功能 2. 计算机网络的分类 3. 计算机网络主要性能指标

4.1.1 计算机网络体系结构

1. 计算机网络分层结构 2. 计算机网络协议、接口、服务等概念

1. 服务访问点（SAP）：数据链路层是 MAC 地址，网络层是 IP 地址，传输层是端口。
2. 冲突域（第一层概念）：集线器、中继器等所连接的结点都属于同一个冲突域，不能划分冲突域。第二层（网桥、交换机）和第三层（路由器）设备可以划分冲突域。
3. 广播域（第二层概念）：第一层（集线器）和第二层（交换机）设备所连接的结点都属于同一个广播域，路由器作为第三层设备可以划分广播域。

3.ISO/OSI 参考模型和 TCP/IP 模型

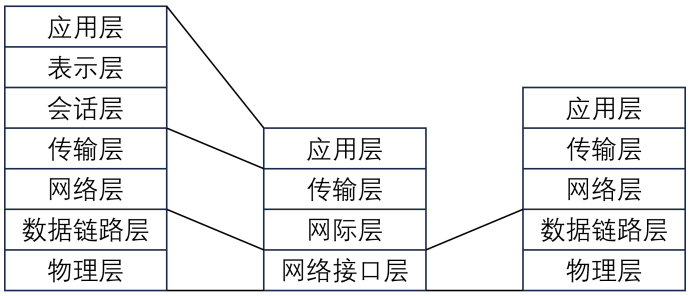


图 15: OSI、TCP/IP 和 5 层模型对比

4.2 物理层

通信基础：1. 信道、信号、带宽、码元、波特、速率、信源与信宿等基本概念 2. 奈奎斯特定理与香农定理 3. 编码与调制 4. 电路交换、报文交换与分组交换 5. 数据报与虚电路

1. 曼彻斯特编码：每个时钟周期中间的跳变表示 0 或 1。
2. 差分曼彻斯特编码：每个时钟周期的起始处，跳变为 0，不跳变为 1。

1. 奈奎斯特定理：理想低通信道下的极限数据传输速率 $= 2W \log_2 V$.
2. 香农定理：信道的极限数据传输速率 $= W \log_2(1 + S/N)$.
3. 虚电路的路由选择体现在连接建立阶段，提供了可靠的通信功能，能保证每个分组正确且有序到达。

传输介质：1. 双绞线、同轴电缆、光纤与无线传输介质 2. 物理层接口的特性

表 7: 以太网的传输介质

参数	**10Base2**	**10Base5**	**10BaseT**	**10BaseFL**	**100BaseT**
传输媒体	同轴电缆（细缆）	同轴电缆（粗缆）	非屏蔽双绞线	光纤对	非屏蔽双绞线
编码	曼彻斯特编码	曼彻斯特编码	曼彻斯特编码	曼彻斯特编码	曼彻斯特编码
拓扑结构	总线形	总线形	星形	点对点	星形

物理层设备：1. 中继器 2. 集线器

4.3 数据链路层

数据链路层的功能、组帧、差错控制：1. 检错编码 2. 纠错编码

4.3.1 流量控制与可靠传输机制

1. 流量控制、可靠传输与滑动窗口机制 2. 停止-等待协议 3. 后退 N 帧协议 (GBN) 4. 选择重传协议 (SR)

滑动窗口协议：**2009-35, 2011-35, 2012-36, 2014-36, 2015-35, 2018-36, 2019-35, 2020-36**
设发送窗口大小 W_T，接收窗口大小 W_R，数据帧发送时间为 t_f，确认帧发送时间为 t_p，单程传播时延为 τ，帧序号 n 比特编号，即 $W_T + W_R \leqslant 2^n$，最大信道利用率取最短帧长。
发送周期：发送方从发送第一个数据帧开始到接收到第一个确认帧为止的时间，即 $T = t_f + t_p + 2\tau$.
N 为一个发送周期 T 内可发送的最大帧数和发送窗口的最小值。
1. 停止-等待协议：一次只允许发一帧，$W_T = W_R = 1$，最大信道利用率 t_f/T.
2. 后退 N 帧协议（GBN）：累积确认，$1 < W_T \leqslant 2^n - 1, W_R = 1$，最大信道利用率 Nt_f/T.
3. 选择重传协议（SR）：只重传出现差错或超时的帧，$1 < W_T = W_R \leqslant 2^{n-1}$，最大信道利用率 Nt_f/T.
4. 对于 GBN：ACK_n 表示该数据帧和之前的数据帧都正确收到；SR：ACK_n 只表示该数据帧正确收到。

4.3.2 介质访问控制

1. 信道划分：频分多路复用、时分多路复用、波分多路复用、码分多路复用的概念和基本原理
2. 随即访问：ALOHA 协议；CSMA 协议；CSMA/CD 协议；CSMA/CA 协议

表 8: 三种不同类型的 CSMA 协议比较

信道状态	1-坚持	非坚持	p-坚持
空闲	立即发送数据	立即发送数据	以概率 p 发送数据， 以概率 1-p 推迟到下一个时隙
忙	继续坚持监听	放弃监听，等待一个随机的时间后监听	持续监听，直至信道空闲

1.最小帧长 = 总线传播时延 × 数据传输速率 × 2.
2.CSMA/CD：工作在总线形或半双工网络，发送前先监听，边发送边监听，一旦出现碰撞马上停止发送。
3.CSMA/CA：发送数据时先广播告知其他结点，让其他结点在某段时间内不要发送数据，以免产生碰撞。使用链路层确认/重传方案，即站点每发送完一帧，要在收到对方的确认帧后才能继续发送下一帧。

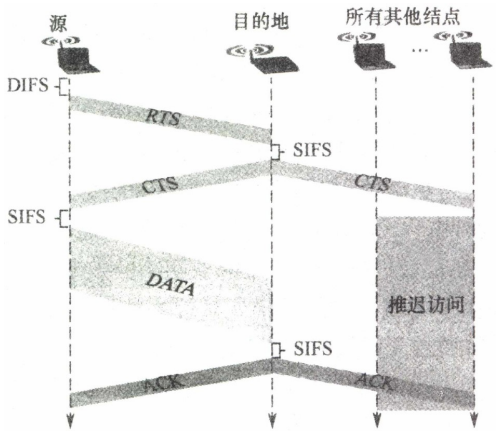


图 16: CSMA/CA 使用 RTS 和 CTS 帧的碰撞避免

3. 轮询访问：令牌传递协议

4.3.3 局域网

1. 局域网的基本概念与体系结构 2. 以太网与 IEEE802.3

1. 前导码：18B，数据：46～1500B，CSMA/CD 算法限制，以太网帧最小长度 64B，数据较少时必须填充。
2. 填充：0～46B，当帧长太短时填充帧，使之达到 64B 的最小长度。

3.IEEE802.11 无线局域网

表 9: 802.11 帧的地址字段最常用的两种情况

去往 AP	来自 AP	地址 1（接收地址）	地址 2（发送地址）	地址 3	地址 4
0	1	目的地址	AP 地址	源地址	–
1	0	AP 地址	源地址	目的地址	–

4. VLAN 基本概念与基本原理

6	6	4	2	46-1500	4
目的地址	源地址	VLAN标签	类型	数据	FCS

图 17: 插入 VLAN 标签的 802.1Q 帧

802.3ac 标准支持 VLAN 的以太网帧格式扩展。首部增加了 4 字节，以太网的最大帧长变为 1522 字节。

4.3.4 广域网

1. 广域网的基本概念 2.PPP 协议

1.PPP 协议是点对点协议，只支持全双工链路，无须采用 CSMA/CD 协议。
2.PPP 提供差错检测但不提供纠错功能，是不可靠传输协议。

数据链路层设备：以太网交换机及其工作原理

直通式交换机: 只检查帧的目的地址 (6B)，帧在接收后几乎能马上被传出去。

4.4 网络层

网络层的功能：1. 异构网络互联 2. 路由与转发 3.SDN 基本概念 4. 拥塞控制
路由算法：1. 静态路由与动态路由 2. 距离-向量路由算法 3. 链路状态路由算法 4. 层次路由

4.4.1 IPv4

1.IPv4 分组

1. 首部20B。
2. 标志：最低位 MF=1 表示“还有分片”。MF=0 表示最后一个分片。中间一位 DF=0 时才允许分片。
3. 片偏移：某片在原分组中的相对位置，以 8 个字节为偏移单位，每个分片的长度一定是 8B 的整数倍。
4. 协议：分组的数据部分应上交给哪个协议进行处理，6 表示 TCP，17 表示 UDP。

2.IPv4 地址与 NAT 3. 子网划分、路由聚集、子网掩码与 CIDR

图 18: IP 数据报格式

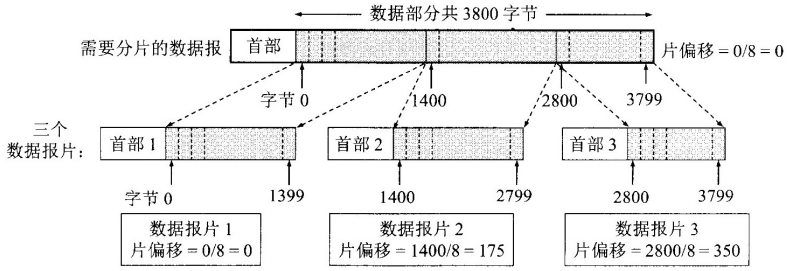


图 19: IP 分片举例

1. 普通路由器仅工作在网络层。NAT 路由器转发数据报时需要查看和转换传输层的端口号。
2. 网桥在转发帧时不改变帧的源地址。

4.ARP 协议、DHCP 协议与 ICMP 协议

1.ARP：
1）请求分组（广播发送）：目的地址为 FF-FF-FF-FF-FF-FF，局域网内所有主机都能收到。
2）响应分组（单播发送）：目的地址为请求主机 MAC 地址。
2.DHCP：应用层协议，基于UDP，通过广播方式进行交互。
3.ICMP：网络层协议，ICMP 报文作为IP 层数据报的数据。

IPv6：1.IPv6 的主要特点 2.IPv6 地址

4.4.2 路由协议

1. 自治系统 2. 域内路由与域间路由 3.RIP 路由协议 4.OSPF 路由协议 5.BGP 路由协议

1.RIP：应用层协议，基于UDP。仅和相邻路由器交换信息，坏消息传得慢。选择的路径不一定是时间最短，但一定是路由器最少的路径。
2.OSPF：网络层协议，直接使用IP 数据报传送。向本自治系统中所有路由器发送信息，每台路由器都能建立全网的拓扑结构图。
3.BGP：应用层协议，基于TCP。只能寻求一条能够到达目的网络比较好的路由，并非寻找一条最佳路由。

IP 组播：1. 组播的概念 2.IP 组播地址

组播：仅 UDP，主机组播时只发送一份数据，只有数据在传送路径出现分岔时才将分组复制后继续转发。对组播数据报不产生 ICMP 差错报文。

移动 IP：1. 移动 IP 的概念 2. 移动 IP 通信过程、**网络层设备**：1. 路由器的组成和功能 2. 路由表与分组转发

1. 同一个网络中传递数据无须路由器的参与，直接交付无须通过路由器。跨网络通信必须通过路由器转发。
2. 分组的实际转发是靠查找转发表，而不是直接查找路由表。

4.5 传输层

传输层提供的服务：1. 传输层的功能 2. 传输层寻址与端口 3. 无连接服务与面向连接服务
UDP 协议：1.UDP 数据报 2.UDP 校验

首部8B（源端口，目的端口，长度，校验和均 2B）。UDP 基于目的端口 分用。

4.5.1 TCP 协议

1.TCP 段

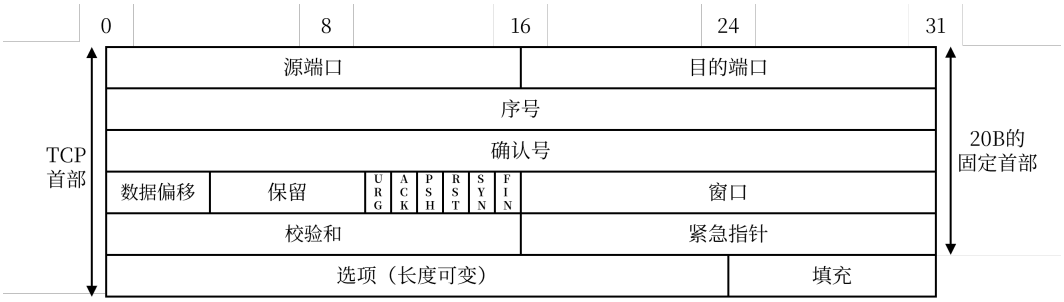


图 20: TCP 报文段

1. 首部20B。
1. 序号：本报文段所发送的第一个数据字节的序号。
2. 确认号：期望收到对方下一个报文段的第一个数据字节的序号。默认使用累积确认。

2.TCP 连接管理

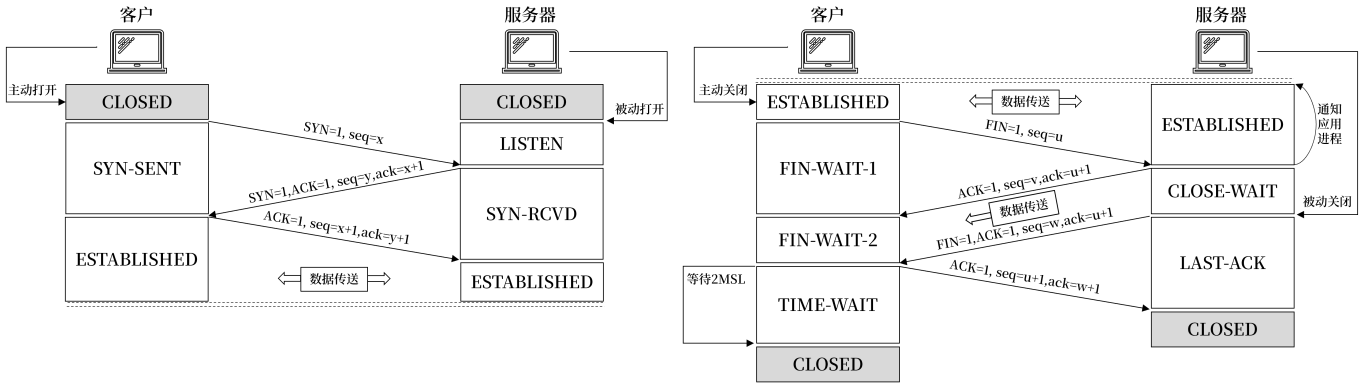


图 21: TCP 连接管理

1.发送方从第三个报文段开始发送数据。
2.SYN 报文段和 FIN 报文段都不能携带数据，但要消耗掉一个序号。
3.A 在 TIME-WAIT 状态必须等待 2MSL：保证 A 发送的最后一个 ACK 报文段能够到达 B。防止已失效的连接请求报文段出现在本连接中。

3.TCP 可靠传输 4.TCP 流量控制 5.TCP 拥塞控制

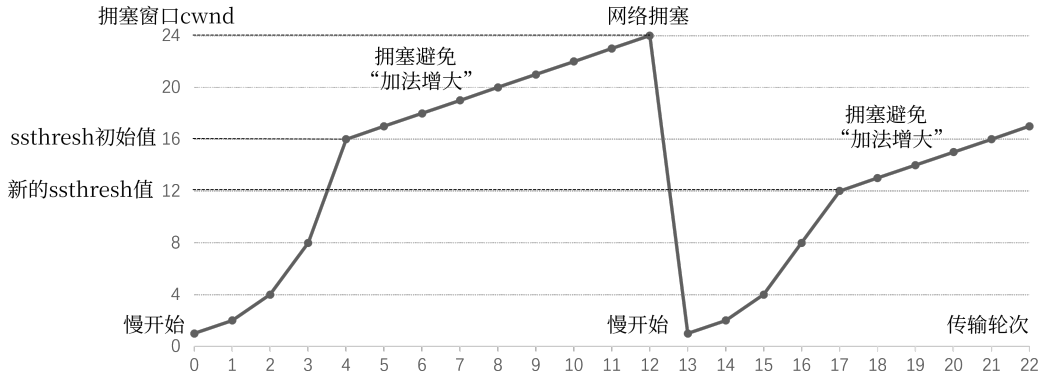


图 22: 慢开始和拥塞避免算法的过程

4.6 应用层

网络应用模型：1. 客户/服务器 (C/S) 模型 2. 对等 (P2P) 模型

4.6.1 DNS 系统

1. 层次域名空间 2. 域名服务器 3. 域名解析过程

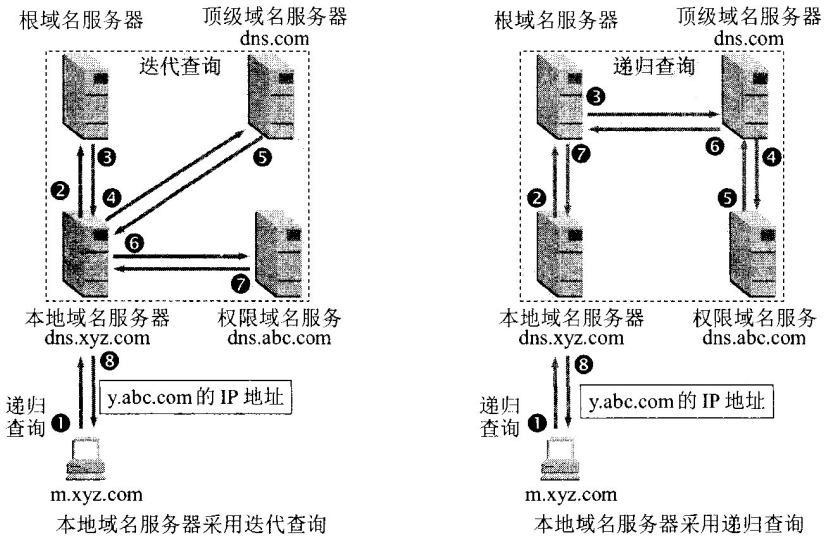


图 23: 两种 DNS 解析方式

FTP：1.FTP 协议的工作原理 2. 控制连接与数据连接

电子邮件：1. 电子邮件系统的组成结构 2. 电子邮件格式与 MIME 3.SMTP 协议与 POP3 协议

图 24: 电子邮件

4.6.2 WWW

1.WWW 的概念与组成结构 2.HTTP 协议

HTTP/1.1 支持持久连接。非流水线方式：客户在收到前一个响应后才能发出下一个请求。