ping traceroute

ping

```
C:\Users\Administrator>ping www.baidu.com

正在 Ping www.a.shifen.com [180.101.49.11] 具有 32 字节的数据:
来自 180.101.49.11 的回复: 字节=32 时间=8ms TTL=54
来自 180.101.49.11 的回复: 字节=32 时间=8ms TTL=54
来自 180.101.49.11 的回复: 字节=32 时间=8ms TTL=54
来自 180.101.49.11 的回复: 字节=32 时间=8ms TTL=54

180.101.49.11 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 8ms, 最长 = 8ms, 平均 = 8ms
```

1. ping程序发送一个 类型8编码0（回显请求）的ICMP报文 到达指定主机
2. 目的主机发回一个 类型0编码0（回显回答）的ICMP报文 进行响应

ping是应用层直接使用网络层ICMP的一个例子，它没有通过运输层的TCP或UDP

traceroute

```
C:\Users\Administrator>tracert www.baidu.com

通过最多 30 个跃点跟踪
到 www.a.shifen.com [180.101.49.11] 的路由:

  1    <1 毫秒    <1 毫秒    <1 毫秒 DESKTOP-DKI9NJF [192.168.1.1]
  2     2 ms     2 ms     1 ms  100.99.0.1
  3     8 ms     6 ms     6 ms  58.220.165.113
  4     5 ms     *        5 ms  58.220.106.209
  5     *        *        8 ms  58.213.95.110
  6     *        *        *     请求超时。
  7     7 ms     7 ms     7 ms  58.213.96.110
  8     *        9 ms     8 ms  10.166.50.0 [10.166.50.0]
  9     *        *        *     请求超时。
 10     9 ms     7 ms     8 ms  10.165.3.21 [10.165.3.21]
 11     9 ms     8 ms     8 ms  180.101.49.11

跟踪完成。
```

1. 源主机向目的主机发送一连串的IP数据报，这些数据报封装了 具有不可达端口号的UDP报文段
2. 第一个数据报的TTL为1，第二个的TTL为2，第三个的TTL为3，以此类推...

   【注意每一行都有三个时间，因为对于每个TTL值，源主机都会发送三次同样的IP数据报】

   当第n个数据报到达第n台路由器时，第n台路由器观察到这个数据报的TTL正好过期，路由器丢弃该报文并发送一个 类型11编码0（TTL过期）的ICMP告警报文 给源主机，该告警报文包含了路由器的名字与它的IP地址

3. 随着发送数据报的TTL不断增大，这些数据报之一将最终沿着路径到达目的主机。由于数据报中的UDP报文段端口非法，因此目的主机会向源发送一个`端口不可达的ICMP报文`，当源主机收到该报文后便停止发送探测分组

union

数据对齐：任何K字节的对象的地址必须是K的倍数

- 联合提供了一种方式，能够规避C语言的类型系统，允许以多种类型来引用一个对象
- 一个联合的总的大小等于它最大字段的大小
- 记得联合中的所有字段相对联合首地址的偏移量都是0就行

> show_bytes函数用来打印底层的字节表示

```c
typedef unsigned char *byte_pointer;

void show_bytes(byte_pointer start, int len) {
    for (int i = 0; i < len; i++) {
        printf("%.2x ", start[i]);
    }
    printf("\n");
}

union data {
    int n;
    char ch;
    short m;
    float f;
};

int main() {
    union data a;
    printf("%d, %d\n", sizeof(a), sizeof(union data));
    a.f = 0x7FE26584;
    show_bytes((byte_pointer) &a,  len: 4);
    show_bytes((byte_pointer) &a.n,  len: 4);
    show_bytes((byte_pointer) &a.ch,  len: 1);
    show_bytes((byte_pointer) &a.m,  len: 2);
    show_bytes((byte_pointer) &a.f,  len: 4);
    printf("a's address: %p\n", &a);
    printf("a.n's address: %p\n", &a.n);
    printf("a.ch's address: %p\n", &a.ch);
    printf("a.m's address: %p\n", &a.m);
    printf("a.f's address: %p\n", &a.f);
    return 0;
}
```

learn_cpp

```
C:\Users\Administrator\CLionProjects\learn_cpp\cmake-build-debug\learn_cpp.exe
4, 4
cb c4 ff 4e
cb c4 ff 4e
cb
cb c4
cb c4 ff 4e
a's address: 0061FF1C
a.n's address: 0061FF1C
a.ch's address: 0061FF1C
a.m's address: 0061FF1C
a.f's address: 0061FF1C

Process finished with exit code 0
```

指令流水线

适合流水线的指令集特征【参考袁春风-计算机系统基础 241-244页】

- 指令长度应尽量一致，这样有利于简化取指令和指令译码操作
- 指令格式尽量规整，尽量保证源操作数寄存器的位置相同，有利于在指令未译码时就可以读取寄存器操作数

  e.g：MIPS指令格式中源操作数寄存器rs和rt的位置总是固定在指令的第25~21位和第20~16位
- 采用load/store型指令风格
- 数据和指令在存储器中要“对齐”存放，有利于减少访存次数

流水线冒险【参考袁春风-计算机系统基础 246-247页】

- 结构冒险（硬件资源冲突）
  - 由硬件资源竞争造成，例如两条指令访问同一个存储器而产生访存冲突
  - 解决方案：设置多个独立部件避免资源冲突
- 数据冒险
  - 产生原因：后面指令用到前面指令的运算结果时，前面指令的结果还没产生
  - 解决方案：①在相关指令间插入空操作；②数据转发旁路
- 控制冒险
  - 产生原因：各种转移类指令（条件/无条件转移、调用/返回指令）、中断/异常
  - 解决方案：①插入空操作；②采用分支预测技术

高级流水线【参考袁春风-计算机系统基础 248-249页】

- 超流水线
  - 通过 `增加流水线级数` 使得更多指令在流水线中重叠执行
  - `CPI仍为1`
- 多发射流水线
  - 超长指令字（静态多发射）
    - 由 `编译器静态` 推测来辅助完成“指令打包”和“冒险处理”
    - 指令打包的结果可看成将同时发射的多条指令合并到一个长指令中
  - 超标量（动态多发射）
    - 由 `处理器硬件动态` 进行流水线调度完成“指令打包”和“冒险处理”，每个周期由处理器决定是发射一条或多条指令
    - 乱序执行：流水线阻塞时动态的在后面找一些没有依赖关系的指令提前执行