质朴的原理

TCP 四元组

源 IP	源端口	目标 IP	目标端口
2^32	2^16	1	1

=2^48

= 281万亿

理论上：每个 server 能接收的连接是两百多万亿

临时端口分配策略

蹩脚的特性：奇偶之分

/proc/sys/net/ipv4/ip_local_port_range 文件指定了临时端口号的下界 low 和上界 high，默认情况，low 是偶数，在我的电脑上 low 和 high 的值分别是 32768 和 60999。

- 优先给 bind(0) 分配随机的与 low 奇偶性不同的端口，也就是奇数端口。如果奇数端口号分配完了，才去尝试分配偶数端口
- 优先给 connect 分配与 low 奇偶性相同的临时端口，也就是偶数端口。如果偶数端口号分配完了，才去尝试分配奇数端口

too many open files

linux 下一切皆文件，包括 socket

ulimit -n：单个用户能打开的文件上限

fs.nr_open：单个进程上可打开的文件数

fs.file-max：整个系统上可打开的最大文件数

端口号限制

net.ipv4.ip_local_port_range

$ sysctl -a | grep net.ipv4.ip_local_port_range

net.ipv4.ip_local_port_range = 32768 60999

端口范围：28232

sudo sysctl -w net.ipv4.ip_local_port_range="1025 61999"

黑带程序员

1人正在看

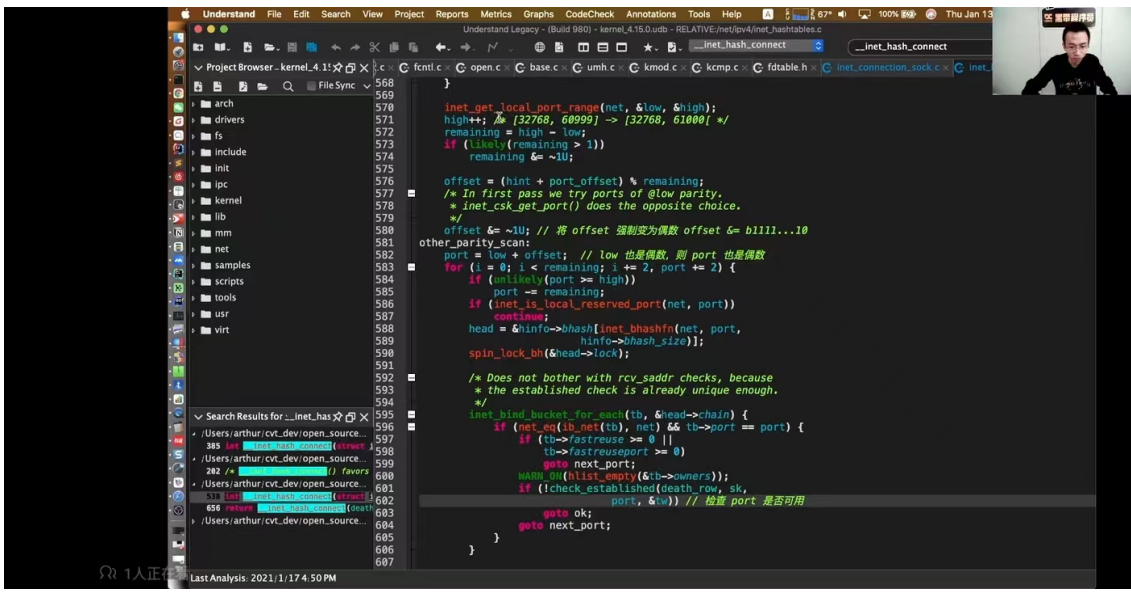
怎么解决

手动 bind 端口！有多少人工，就有多少性能

```c
int bind_local_ip_port(int sockfd, char *ip, int port) {
    struct sockaddr_in cli_addr;
    cli_addr.sin_family = AF_INET;
    cli_addr.sin_addr.s_addr = inet_addr(ip);
    cli_addr.sin_port = htons(port);
    return bind(sockfd, (struct sockaddr *) &cli_addr, sizeof(cli_addr));
}
int main() {

    int port = 1026;
    int fd = socket(AF_INET, SOCK_STREAM, 0);
    int ret = bind_local_ip_port(fd, local_ip, port);

    struct sockaddr_in addr;
    create_inet_addr(&addr, remote_ip, remote_port);

    int rc = connect(fd, (struct sockaddr *) &addr, sizeof(addr));
```

黑带程序员

1人正在看




```c
    }

    inet_get_local_port_range(net, &low, &high);
    high++; /* [32768, 60999] -> [32768, 61000[ */
    remaining = high - low;
    if (likely(remaining > 1))
        remaining &= ~1U;

    offset = (hint + port_offset) % remaining;
    /* In first pass we try ports of @low parity.
     * inet_csk_get_port() does the opposite choice.
     */
    offset &= ~1U; // 将 offset 强制变为偶数 offset &= b1111...10
other_parity_scan:
    port = low + offset;  // low 也是偶数, 则 port 也是偶数
    for (i = 0; i < remaining; i += 2, port += 2) {
        if (unlikely(port >= high))
            port -= remaining;
        if (inet_is_local_reserved_port(net, port))
            continue;
        head = &hinfo->bhash[inet_bhashfn(net, port,
                                          hinfo->bhash_size)];
        spin_lock_bh(&head->lock);

        /* Does not bother with rcv_saddr checks, because
         * the established check is already unique enough.
         */
        inet_bind_bucket_for_each(tb, &head->chain) {
            if (net_eq(ib_net(tb), net) && tb->port == port) {
                if (tb->fastreuse >= 0 ||
                    tb->fastreuseport >= 0)
                    goto next_port;
                WARN_ON(hlist_empty(&tb->owners));
                if (!check_established(death_row, sk,
                                       port, &tw)) // 检查 port 是否可用
                    goto ok;
                goto next_port;
            }
        }
```

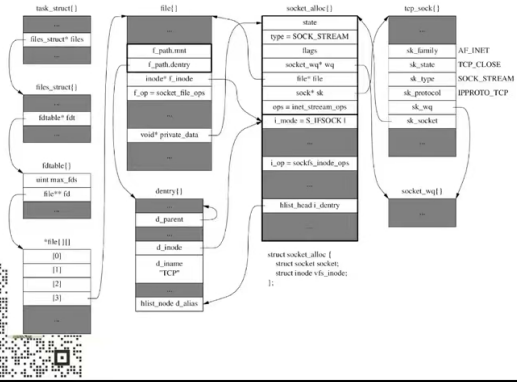
1人正在看

```
time cost[50000]: time: 517
in the end 50000
time cost[50000]: time: 511
in the end 50000

^C
--- 192.168.31.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4050ms
rtt min/avg/max/mdev = 0.440/0.586/1.165/0.289 ms
user-Z270-HD3 :: ~ » ss -nta | grep :8080
user-Z270-HD3 :: ~ » ./c1m_server

  http://192.168.31.197:8080
Hit CTRL-C to stop the server
^Chttp-server stopped.
care001 :: ~ » ss -nat | grep :8080 | wc -l
539967
care001 :: ~ » ss -nat | grep :8080 | wc -l
1000000
care001 :: ~ »

user-Z270-HD3 :: ~ » ss -nat | grep :8080 | wc -l
990151
user-Z270-HD3 :: ~ » ss -nat | grep :8080 | wc -l
1000001
user-Z270-HD3 :: ~ » ss -nat | grep :8080 | wc -l
1000001
user-Z270-HD3 :: ~ » ss -nat | grep :8080 | grep ESTAB |
```

1人正在看

Linux中每个TCP连接最少占用多少内存?

分哪几部分



cat /proc/slabinfo

- struct file，对应每个打开的文件
- struct dentry，文件所在的目录
- struct socket_alloc，包含 struct socket 和 struct inode 两个成员
- struct socket_wq，用于 wait queue

黑带程序员

1人正在看

旁敲侧击

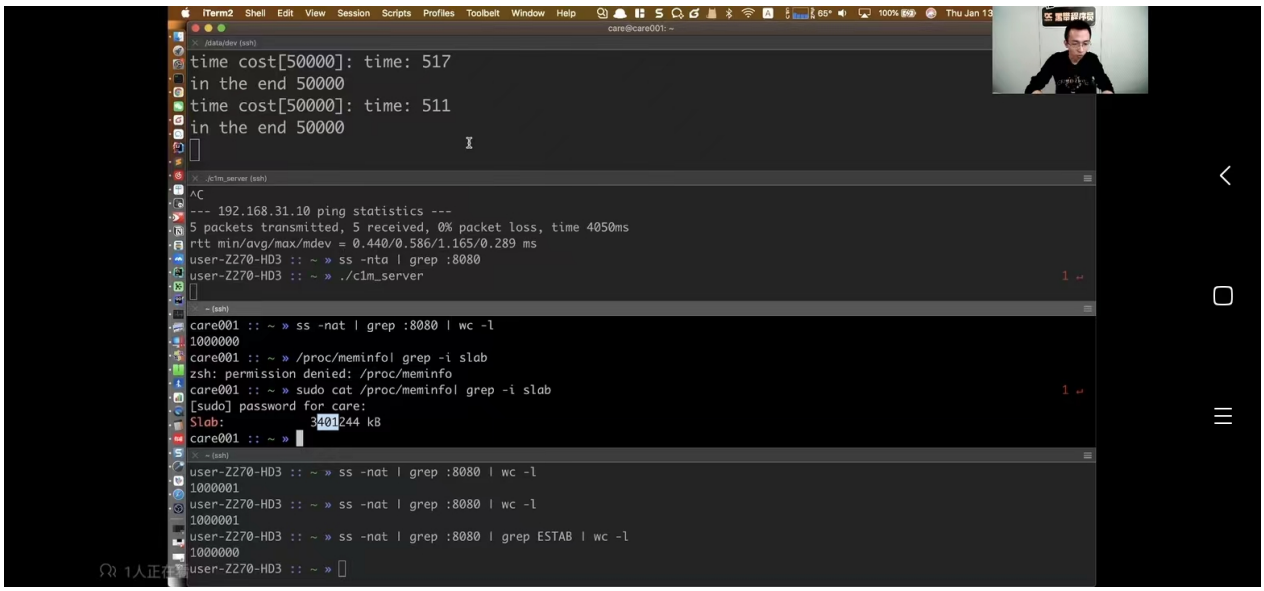
slabtop

OBJS	ACTIVE	USE	OBJ SIZE	SLABS	OBJ/SLAB	CACHE SIZE	NAME
1186962	1180694	0%	0.19K	56522	21	226088K	dentry
1078016	1076305	0%	0.06K	16844	64	67376K	kmalloc-64
1047424	1047424	100%	0.03K	8183	128	32732K	kmalloc-32
1025792	1025602	0%	0.25K	32056	32	256448K	filp
1003421	1003400	0%	0.69K	43627	23	698032K	sock_inode_cache
1002830	1002830	100%	0.02K	5899	170	23596K	lsm_file_cache
1000128	1000128	100%	2.00K	62508	16	2000256K	TCP
291603	291603	100%	0.10K	7477	39	29908K	buffer_head

2.00k+0.02k+0.69k+0.25k+0.03k + 0.06k +0.19k=3.24k

黑带程序员

1人正在看


time cost[50000]: time: 517
in the end 50000
time cost[50000]: time: 511
in the end 50000
^C
--- 192.168.31.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4050ms
rtt min/avg/max/mdev = 0.440/0.586/1.165/0.289 ms
user-Z270-HD3 :: ~ » ss -nta | grep :8080
user-Z270-HD3 :: ~ » ./c1m_server
care001 :: ~ » ss -nat | grep :8080 | wc -l
1000000
care001 :: ~ » /proc/meminfo| grep -i slab
zsh: permission denied: /proc/meminfo
care001 :: ~ » sudo cat /proc/meminfo| grep -i slab
[sudo] password for care:
Slab: 3401244 kB
care001 :: ~ »
user-Z270-HD3 :: ~ » ss -nat | grep :8080 | wc -l
1000001
user-Z270-HD3 :: ~ » ss -nat | grep :8080 | wc -l
1000001
user-Z270-HD3 :: ~ » ss -nat | grep :8080 | grep ESTAB | wc -l
1000000
user-Z270-HD3 :: ~ »


Linux 中每个 TCP 连接最少占用多少内存?
初略计算
(3605592KB-355880 kB)/1000/1000 = 3.249712KB = 3.25KB
黑带程序员

虚拟 IP

ifconfig enp0s31f6:0 192.168.31.10 netmask 255.255.224.0 up

enp0s31f6:0 表示这个 VIP 绑定的目标网卡

192.168.31.10：这个虚拟 IP 地址

up：表示启用这个 VIP

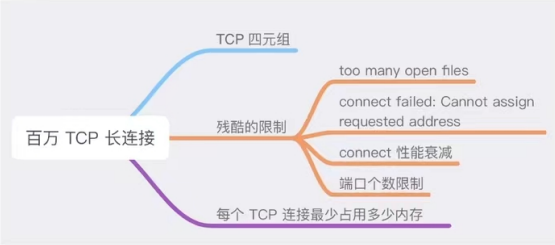
```
192.168.31.10
192.168.31.11
192.168.31.12
192.168.31.13
192.168.31.14
192.168.31.15
192.168.31.16
192.168.31.17
192.168.31.18
192.168.31.19
192.168.31.20
192.168.31.21
192.168.31.22
192.168.31.23
192.168.31.24
192.168.31.25
192.168.31.26
192.168.31.27
192.168.31.28
192.168.31.29
```

```
#!/bin/bash

NETMASK="255.255.224.0"

IFS=$'\n' IPS=($(cat ip.txt))

for i in "${!IPS[@]}"; do
        echo ifconfig enp0s31f6:$i ${IPS[$i]} netmask $NETMASK up
        ifconfig enp0s31f6:$i ${IPS[$i]} netmask $NETMASK up
done
```

黑带程序员

1人正在看

小结



黑带程序员

1人正在看