20 | 大名鼎鼎的select：看我如何同时感知多个I/O事件

2019-09-23 盛延敏

网络编程实战 进入课程 >

讲述：冯永吉
时长 10:27　大小 9.59M

你好，我是盛延敏，这里是网络编程实战的第 20 讲，欢迎回来。

这一讲是性能篇的第一讲。在性能篇里，我们将把注意力放到如何设计高并发高性能的网络服务器程序上。我希望通过这一模块的学习，让你能够掌握多路复用、异步 I/O、多线程等知识，从而可以写出支持并发 10K 以上的高性能网络服务器程序。

还等什么呢？让我们开始吧。

什么是 I/O 多路复用

在第 11 讲中，我们设计了这样一个应用程序，该程序从标准输入接收数据输入，然后通过套接字发送出去，同时，该程序也通过套接字接收对方发送的数据流。

我们可以使用 fgets 方法等待标准输入，但是一旦这样做，就没有办法在套接字有数据的时候读出数据；我们也可以使用 read 方法等待套接字有数据返回，但是这样做，也没有办法在标准输入有数据的情况下，读入数据并发送给对方。

I/O 多路复用的设计初衷就是解决这样的场景。我们可以把标准输入、套接字等都看做 I/O 的一路，多路复用的意思，就是在任何一路 I/O 有“事件”发生的情况下，通知应用程序去处理相应的 I/O 事件，这样我们的程序就变成了“多面手”，在同一时刻仿佛可以处理多个 I/O 事件。

像刚才的例子，使用 I/O 复用以后，如果标准输入有数据，立即从标准输入读入数据，通过套接字发送出去；如果套接字有数据可以读，立即可以读出数据。

select 函数就是这样一种常见的 I/O 多路复用技术，我们将在后面继续讲解其他的多路复用技术。使用 select 函数，通知内核挂起进程，当一个或多个 I/O 事件发生后，控制权返还给应用程序，由应用程序进行 I/O 事件的处理。

这些 I/O 事件的类型非常多，比如：

- 标准输入文件描述符准备好可以读。
- 监听套接字准备好，新的连接已经建立成功。
- 已连接套接字准备好可以写。
- 如果一个 I/O 事件等待超过了 10 秒，发生了超时事件。

select 函数的使用方法

select 函数的使用方法有点复杂，我们先看一下它的声明：

复制代码

```
int select(int maxfd, fd_set *readset, fd_set *writeset, fd_set *exceptset, const struct
返回：若有就绪描述符则为其数目，若超时则为 0，若出错则为 -1
```

在这个函数中，n 表示的是待测试的描述符基数，它的值是待测试的最大描述符加 1。比如现在的 select 待测试的描述符集合是{0,1,4}，那么 maxfd 就是 5，为啥是 5，而不是 4

呢? 我会在下面进行解释。

紧接着的是三个描述符集合，分别是读描述符集合 readset、写描述符集合 writeset 和异常描述符集合 exceptset，这三个分别通知内核，在哪些描述符上检测数据可以读，可以写和有异常发生。

那么如何设置这些描述符集合呢？以下的宏可以帮助到我们。

复制代码

```
void FD_ZERO(fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_CLR(int fd, fd_set *fdset);
int  FD_ISSET(int fd, fd_set *fdset);
```

如果你刚刚入门，理解这些宏可能有些困难。没有关系，我们可以这样想象，下面一个向量代表了一个描述符集合，其中，这个向量的每个元素都是二机制数中的 0 或者 1。

复制代码

```
a[maxfd-1], ..., a[1], a[0]
```

我们按照这样的思路来理解这些宏：

FD_ZERO 用来将这个向量的所有元素都设置成 0；

FD_SET 用来把对应套接字 fd 的元素，a[fd] 设置成 1；

FD_CLR 用来把对应套接字 fd 的元素，a[fd] 设置成 0；

FD_ISSET 对这个向量进行检测，判断出对应套接字的元素 a[fd] 是 0 还是 1。

其中 0 代表不需要处理，1 代表需要处理。

怎么样，是不是感觉豁然开朗了？

实际上，很多系统是用一个整型数组来表示一个描述字集合的，一个 32 位的整型数可以表示 32 个描述字，例如第一个整型数表示 0-31 描述字，第二个整型数可以表示 32-63 描述字，以此类推。

这个时候再来理解为什么描述字集合{0,1,4}，对应的 maxfd 是 5，而不是 4，就比较方便了。

因为这个向量对应的是下面这样的：

复制代码

```
a[4],a[3],a[2],a[1],a[0]
```

待测试的描述符个数显然是 5， 而不是 4。

三个描述符集合中的每一个都可以设置成空，这样就表示不需要内核进行相关的检测。

最后一个参数是 timeval 结构体时间：

复制代码

```
struct timeval {
  long    tv_sec; /* seconds */
  long    tv_usec; /* microseconds */
};
```

这个参数设置成不同的值，会有不同的可能：

第一个可能是设置成空 (NULL)，表示如果没有 I/O 事件发生，则 select 一直等待下去。

第二个可能是设置一个非零的值，这个表示等待固定的一段时间后从 select 阻塞调用中返回，这在第 12 讲超时的例子里曾经使用过。

第三个可能是将 tv_sec 和 tv_usec 都设置成 0，表示根本不等待，检测完毕立即返回。这种情况使用得比较少。

程序例子

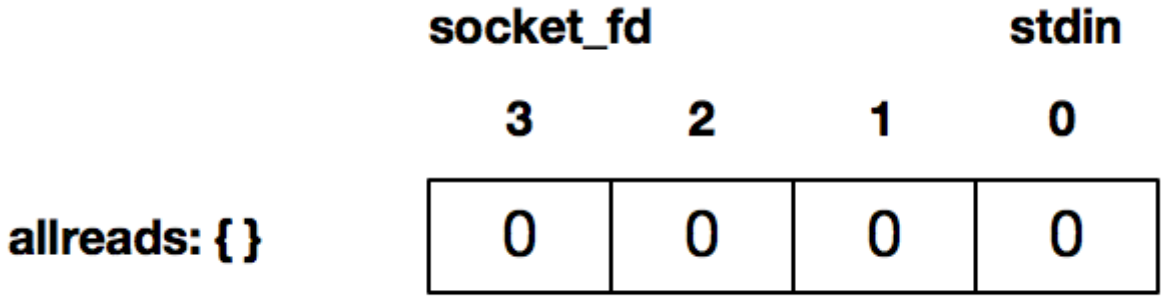
下面是一个具体的程序例子，我们通过这个例子来理解 select 函数。

复制代码

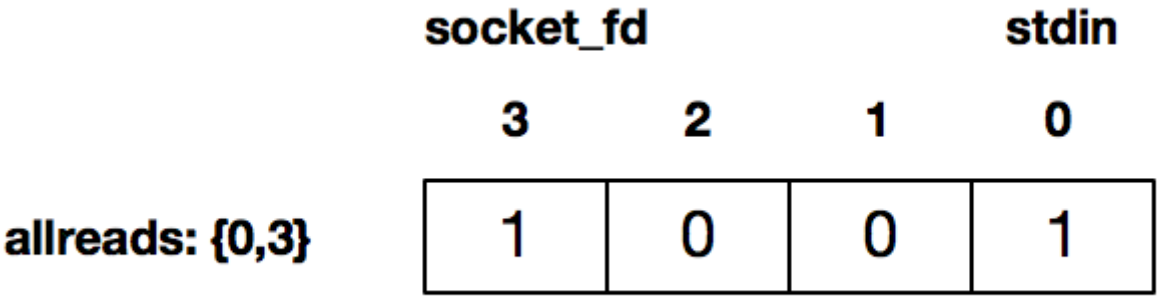
```
int main(int argc, char **argv) {
    if (argc != 2) {
        error(1, 0, "usage: select01 <IPaddress>");
    }
    int socket_fd = tcp_client(argv[1], SERV_PORT);

    char recv_line[MAXLINE], send_line[MAXLINE];
    int n;

    fd_set readmask;
    fd_set allreads;
    FD_ZERO(&allreads);
    FD_SET(0, &allreads);
    FD_SET(socket_fd, &allreads);

    for (;;) {
        readmask = allreads;
        int rc = select(socket_fd + 1, &readmask, NULL, NULL, NULL);

        if (rc <= 0) {
            error(1, errno, "select failed");
        }

        if (FD_ISSET(socket_fd, &readmask)) {
            n = read(socket_fd, recv_line, MAXLINE);
            if (n < 0) {
                error(1, errno, "read error");
            } else if (n == 0) {
                error(1, 0, "server terminated \n");
            }
            recv_line[n] = 0;
            fputs(recv_line, stdout);
            fputs("\n", stdout);
        }

        if (FD_ISSET(STDIN_FILENO, &readmask)) {
            if (fgets(send_line, MAXLINE, stdin) != NULL) {
                int i = strlen(send_line);
                if (send_line[i - 1] == '\n') {
                    send_line[i - 1] = 0;
                }

                printf("now sending %s\n", send_line);
                size_t rt = write(socket_fd, send_line, strlen(send_line));
                if (rt < 0) {
```

```
46                      error(1, errno, "write failed ");
47                  }
48                  printf("send bytes: %zu \n", rt);
49              }
50          }
51      }
52
53  }
```

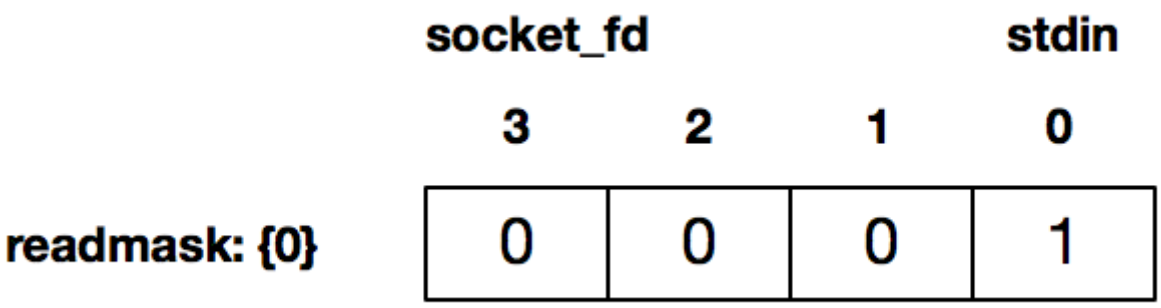
程序的 12 行通过 FD_ZERO 初始化了一个描述符集合，这个描述符读集合是空的：



接下来程序的第 13 和 14 行，分别使用 FD_SET 将描述符 0，即标准输入，以及连接套接字描述符 3 设置为待检测：



接下来的 16-51 行是循环检测，这里我们没有阻塞在 fgets 或 read 调用，而是通过 select 来检测套接字描述字有数据可读，或者标准输入有数据可读。比如，当用户通过标准输入使得标准输入描述符可读时，返回的 readmask 的值为：



这个时候 select 调用返回，可以使用 FD_ISSET 来判断哪个描述符准备好可读了。如上图所示，这个时候是标准输入可读，37-51 行程序读入后发送给对端。

如果是连接描述字准备好可读了，第 24 行判断为真，使用 read 将套接字数据读出。

我们需要注意的是，这个程序的 17-18 行非常重要，初学者很容易在这里掉坑里去。

第 17 行是每次测试完之后，重新设置待测试的描述符集合。你可以看到上面的例子，在 select 测试之前的数据是{0,3}，select 测试之后就变成了{0}。

这是因为 select 调用每次完成测试之后，内核都会修改描述符集合，通过修改完的描述符集合来和应用程序交互，应用程序使用 FD_ISSET 来对每个描述符进行判断，从而知道什么样的事件发生。

第 18 行则是使用 socket_fd+1 来表示待测试的描述符基数。切记需要 +1。

套接字描述符就绪条件

当我们说 select 测试返回，某个套接字准备好可读，表示什么样的事件发生呢?

第一种情况是套接字接收缓冲区有数据可以读，如果我们使用 read 函数去执行读操作，肯定不会被阻塞，而是会直接读到这部分数据。

第二种情况是对方发送了 FIN，使用 read 函数执行读操作，不会被阻塞，直接返回 0。

第三种情况是针对一个监听套接字而言的，有已经完成的连接建立，此时使用 accept 函数去执行不会阻塞，直接返回已经完成的连接。

第四种情况是套接字有错误待处理，使用 read 函数去执行读操作，不阻塞，且返回 -1。

总结成一句话就是，内核通知我们套接字有数据可以读了，使用 read 函数不会阻塞。

不知道你是不是和我一样，刚开始理解某个套接字可写的时候，会有一个错觉，总是从应用程序角度出发去理解套接字可写，我开始是这样想的，当应用程序完成相应的计算，有数据准备发送给对端了，可以往套接字写，对应的就是套接字可写。

其实这个理解是非常不正确的，select 检测套接字可写，**完全是基于套接字本身的特性来说**的，具体来说有以下几种情况。

第一种是套接字发送缓冲区足够大，如果我们使用非阻塞套接字进行 write 操作，将不会被阻塞，直接返回。

第二种是连接的写半边已经关闭，如果继续进行写操作将会产生 SIGPIPE 信号。

第三种是套接字上有错误待处理，使用 write 函数去执行读操作，不阻塞，且返回 -1。

总结成一句话就是，内核通知我们套接字可以往里写了，使用 write 函数就不会阻塞。

总结

今天我讲了 select 函数的使用。select 函数提供了最基本的 I/O 多路复用方法，在使用 select 时，我们需要建立两个重要的认识：

描述符基数是当前最大描述符 +1；

每次 select 调用完成之后，记得要重置待测试集合。

思考题

和往常一样，给大家布置两道思考题：

第一道， select 可以对诸如 UNIX 管道 (pipe) 这样的描述字进行检测么？如果可以，检测的就绪条件是什么呢？

第二道，根据我们前面的描述，一个描述符集合哪些描述符被设置为 1，需要进行检测是完全可以知道的，你认为 select 函数里一定需要传入描述字基数这个值么？请你分析一下这样设计的目的又是什么呢？

欢迎你在评论区写下你的思考，也欢迎把这篇文章分享给你的朋友或者同事，一起交流一下。

上一篇　期中大作业丨题目以及解答剖析

精选留言 (4)

写留言

安排
2019-09-23

第一道：可以，就绪条件是有数据可读(检测可读事件)。是否可以监测可写事件不太清楚，没有实验过。

第二道：不一定需要传入，那样的话内核中for循环需要遍历整个集合，效率低。传入基数可以减小遍历范围，提高效率。...

展开

2

Keep-Moving
2019-09-23

allreads = {0, 3};
老师，这一步是怎么实现的？没看出来

莫珣
2019-09-23

我有些疑问，select的FD数组大小默认是1024，但是Linux的文件描述符大小一定不是1024，假设现在使用ulimit将一个进程可以打开的文件数设置成了65535，那么大于1024的文件描述符怎么加到FD数组中去呢，如果按照文本里说的，文件描述符代表数组下标的话不就加不进去了？

...

展开

Linuxer
2019-09-23

第二题是为了减少检测的范围吧

展开