balloonwj.github.io

# 主线程与工作线程的分工 · 高性能服务器开发 技术专栏

*easy_coder*

9-11 minutes

---



## 主线程与工作线程的分工

服务器端为了能流畅处理多个客户端链接，一般在某个线程A里面accept新的客户端连接并生成新连接的socket fd，然后将这些新连接的socketfd给另外开的数个工作线程B1、B2、B3、B4，这些工作线程处理这些新连接上的网络IO事件（即收发数据），同时，还处理系统中的另外一些事务。这里我们将线程A称为主线程，B1、B2、B3、B4等称为工作线程。工作线程的代码框架一般如下：

```
while (!m_bQuit)
{
    epoll_or_select_func();

    handle_io_events();

    handle_other_things();
}
```

在epoll_or_select_func()中通过select()或者poll/epoll()去检测

socket fd上的io事件，若存在这些事件则下一步handle_io_events()来处理这些事件（收发数据），做完之后可能还要做一些系统其他的任务，即调用handle_other_things()。

这样做有三个好处：

1. 线程A只需要处理新连接的到来即可，不用处理网络IO事件。由于网络IO事件处理一般相对比较慢，如果在线程A里面既处理新连接又处理网络IO，则可能由于线程忙于处理IO事件，而无法及时处理客户端的新连接，这是很不好的。
2. 线程A接收的新连接，可以根据一定的负载均衡原则将新的socket fd分配给工作线程。常用的算法，比如round robin，即轮询机制，即，假设不考虑中途有连接断开的情况，一个新连接来了分配给B1，又来一个分配给B2，再来一个分配给B3，再来一个分配给B4。如此反复，也就是说线程A记录了各个工作线程上的socket fd数量，这样可以最大化地来平衡资源，避免一些工作线程“忙死”，另外一些工作线程“闲死”的现象。
3. 即使工作线程不满载的情况下，也可以让工作线程做其他的事情。比如现在有四个工作线程，但只有三个连接。那么线程B4就可以在handle_other_thing()做一些其他事情。

下面讨论一个很重要的效率问题：

在上述while循环里面，epoll_or_selec_func()中的epoll_wait/poll/select等函数一般设置了一个超时时间。如果设置超时时间为0，那么在没有任何网络IO时间和其他任务处理的情况下，这些工作线程实际上会空转，白白地浪费cpu时间片。如果设置的超时时间大于0，在没有网络IO时间的情况，epoll_wait/poll/select仍然要挂起指定时间才能返回，导致handle_other_thing()不能及时执行，影响其他任务不能及时处理，也就是说其他任务一旦产生，其处理起来具有一定的延时性。这样也不好。那如何解决该问题呢？

其实我们想达到的效果是，如果没有网络IO时间和其他任务要处理，那么这些工作线程最好直接挂起而不是空转；如果有其他任务要处理，这些工作线程要立刻能处理这些任务而不是在epoll_wait/poll/selec挂起指定时间后才开始处理这些任务。

我们采取如下方法来解决该问题，以linux为例，不管epoll_fd上有没有文件描述符fd，我们都给它绑定一个默认的fd，这个fd被称为唤醒fd。当我们需要处理其他任务的时候，向这个唤醒fd上随便写入1个字节的，这样这个fd立即就变成可读的了，epoll_wait()/poll()/select()函数立即被唤醒，并返回，接下来马上就能执行handle_other_thing()，其他任务得到处理。反之，没有其他任务也没有网络IO事件时，epoll_or_select_func()就挂在那里什么也不做。

这个唤醒fd，在linux平台上可以通过以下几种方法实现：

1. 管道pipe，创建一个管道，将管道绑定到epoll_fd上。需要时，向管道一端写入一个字节，工作线程立即被唤醒。
2. linux 2.6新增的eventfd：

```
int eventfd(unsigned int initval, int flags);
```

步骤也是一样，将生成的eventfd绑定到epoll_fd上。需要时，向这个eventfd上写入一个字节，工作线程立即被唤醒。

1. 第三种方法最方便。即linux特有的socketpair，socketpair是一对相互连接的socket，相当于服务器端和客户端的两个端点，每一端都可以读写数据。

```
int socketpair(int domain, int type, int protocol,
int sv[2]);
```

调用这个函数返回的两个socket句柄就是sv[0]，和sv[1]，在一个其中任何一个写入字节，在另外一个收取字节。

将收取的字节的socket绑定到epoll_fd上。需要时，向另外一个写入的socket上写入一个字节，工作线程立即被唤醒。如果是使用socketpair，那么domain参数一定要设置成AFX_UNIX。

由于在windows，select函数只支持检测socket这一种fd，所以Windows上一般只能用方法3的原理。而且需要手动创建两个socket，然后一个连接另外一个，将读取的那一段绑定到select的fd上去。这在写跨两个平台代码时，需要注意的地方。

## results matching ""

## No results matching ""