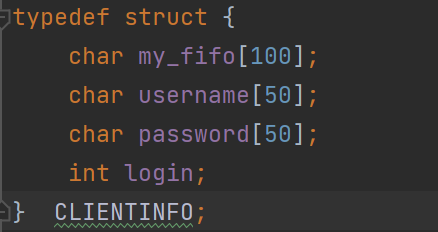
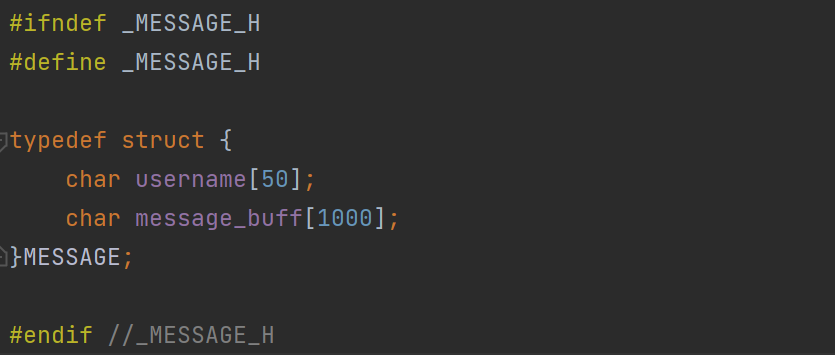
构建一个用于存储客户端信息的结构体，命名为clientinfo，其中my\_fifo用于存储客户私有管道的路径，username和password分别存储客户的用户名和密码。并为每一个客户设置login变量保存其登录状态，在客户成功注册时，将登录状态设置为0，当成功登录后，才将其设置为1。



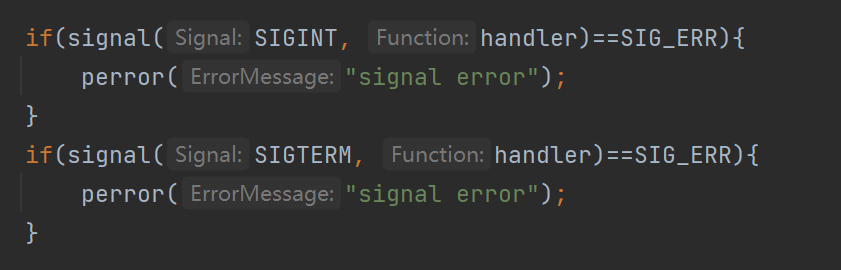
构建一个存储信息的结构体，用来存储目标客户和需要传递的信息内容。



接下来，对于服务器的代码实现进行解析。

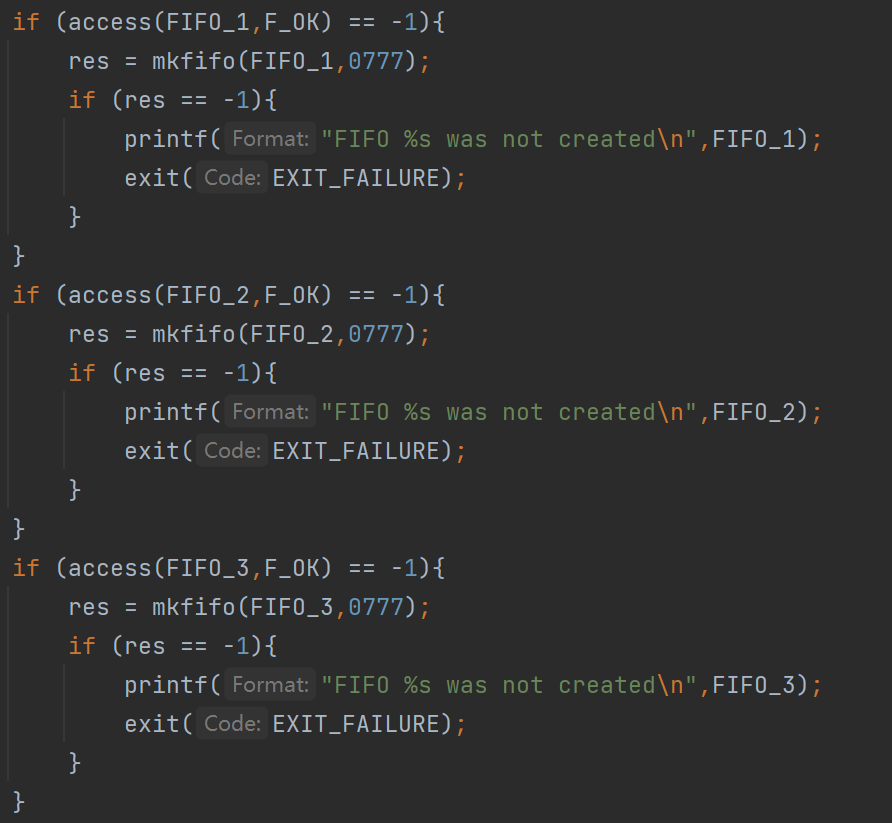
该服务器的实现对于I/O多路复用的实现是基于epoll实现的。

首先，为中断信号注册信号处理函数，用于将管道文件关闭。

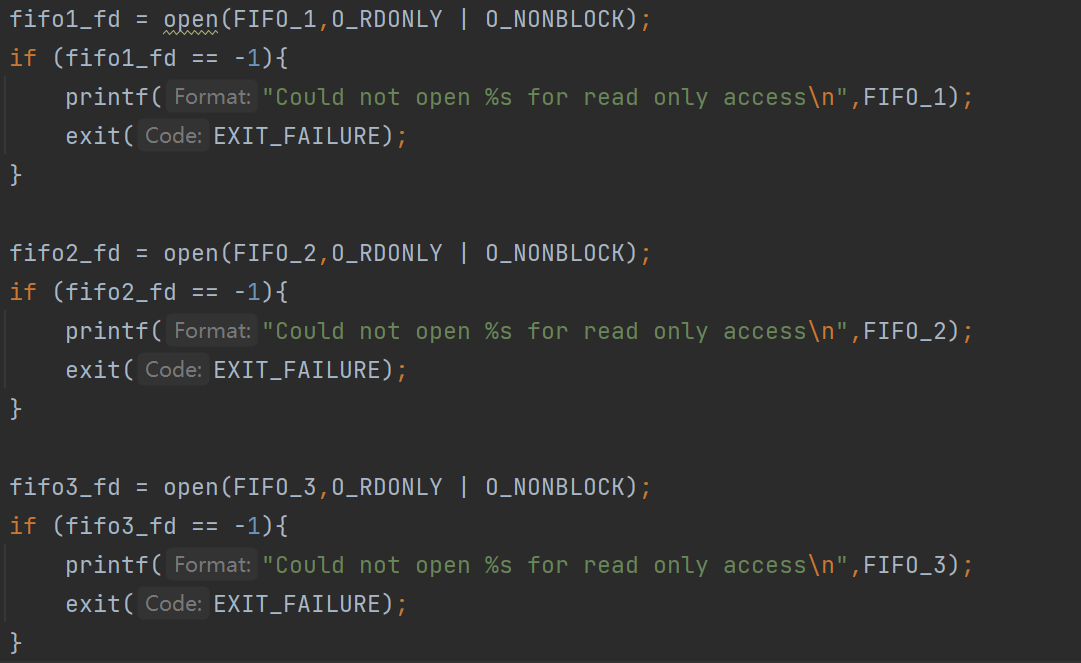


其中，不需要对于SIGKILL信号注册信号处理函数，因为SIGKILL信号无法被捕捉。

然后，判断三个众所周知管道的管道文件是否存在，如果不存在，则创建管道文件，并将权限设置为可读可写可执行。

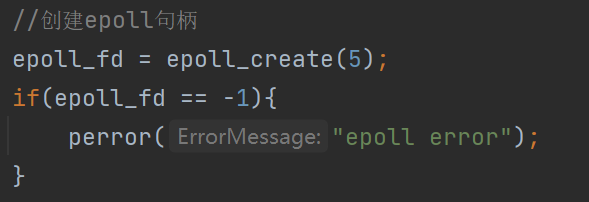


分别以非阻塞读的模式打开各个管道文件，并且将管道文件的文件描述符保存在对应变量中。



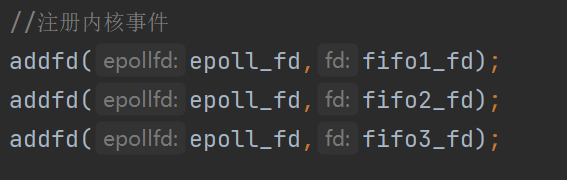
使用非阻塞读的目的是为了提高服务器的可复用性，若不使用非阻塞模式，在管道另一端无进程以写管道的方式打开管道时，服务器进程会被阻塞在开启管道的这段代码中，导致服务器可复用性极差。

创建一个epoll句柄



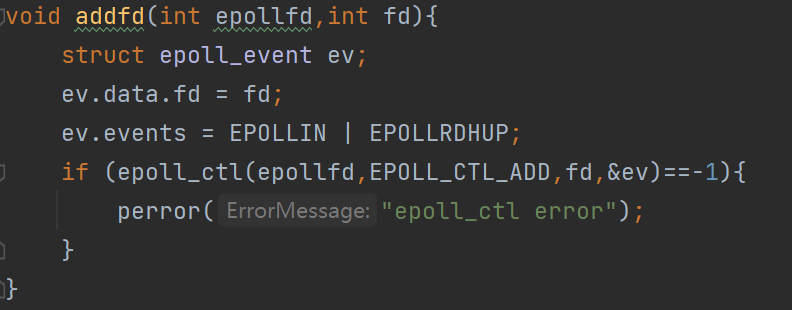
在epoll\_create中，内核会创建一个eventpoll对象，并将该对象的描述符返回。结构体包含三部分，一个就绪链表，一个等待链表和一个用于监听的红黑树。

注册内核事件



将三个众所周知管道注册为内核的监听事件。

addfd函数的具体实现如下：

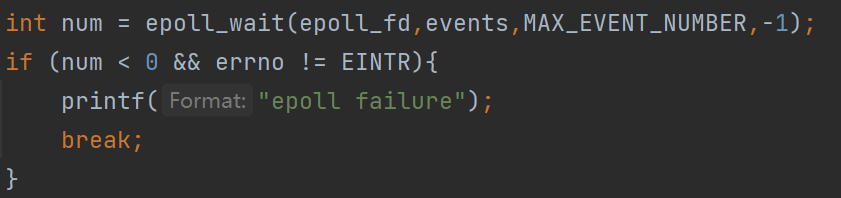


创建一个epoll\_event结构体，并在众所周知管道的描述符存入结构体中。最后通过epoll\_ctl将该结构体挂到eventpoll对象的红黑树中。

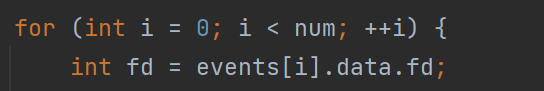
至此，内核开始了对于三个众所周知管道的监听。

接下来，就是多路复用的实现部分。

首先调用epoll\_wait函数查看就绪链表中是否有事件就绪，若无，则将当前进程阻塞，并将进程的描述符注册到等待队列中。若有时间就绪，返回事件数，并将事件从内核态拷贝到用户态，存储在events数组中。



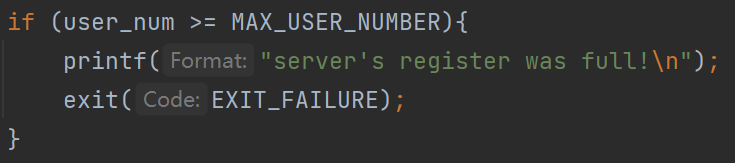
通过遍历events数组，获得管道的文件描述符。



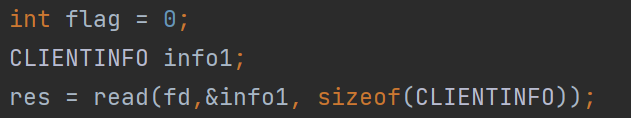
接下来就是对于三个管道如何完成任务的实现。

对于管道1，我们需要实现客户注册功能。

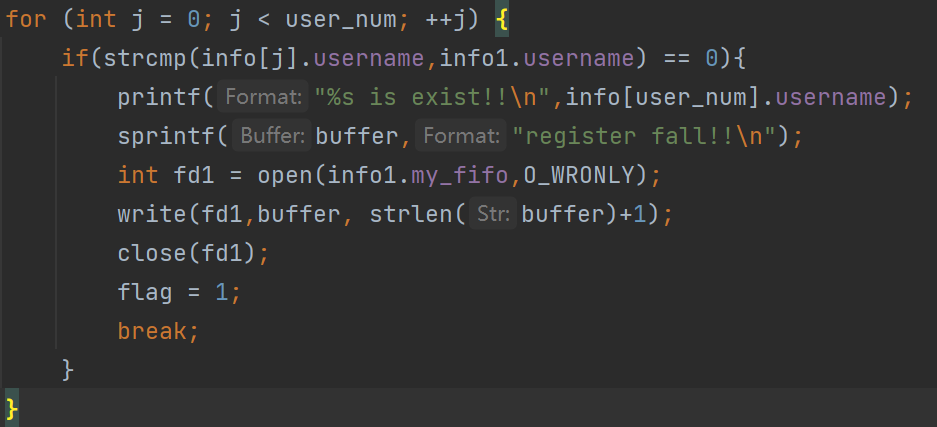
首先，我们判断当前服务器挂载的用户数量是否超过了服务器的最大承担数量，在这里对于最大承担数量的定义由clinetinfo结构体数组的大小决定。



若当前服务器负载未超过最大负载，则读入管道信息，存储在info1变量中。

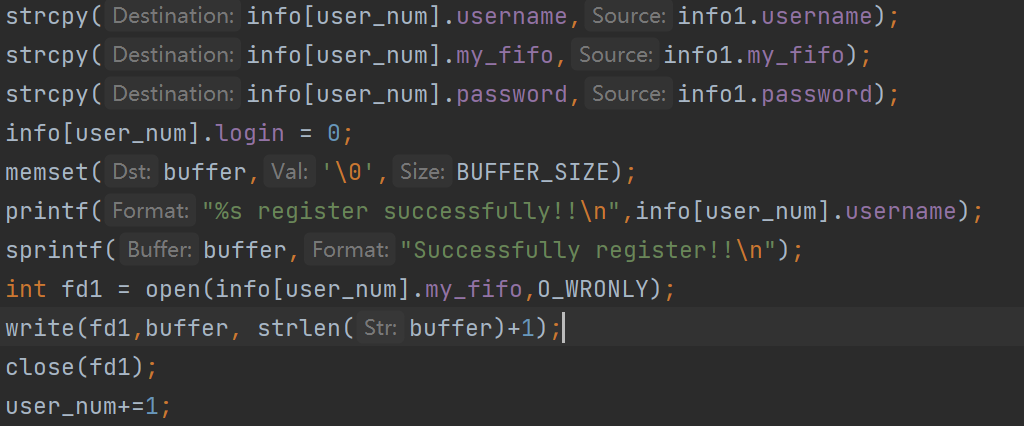


若成功读取到管道内容，判断是否注册了已经存在的用户。



若注册的用户名已存在，那么在服务器打印出相应信息，并将注册失败的信息通过私有管道返回给客户端。

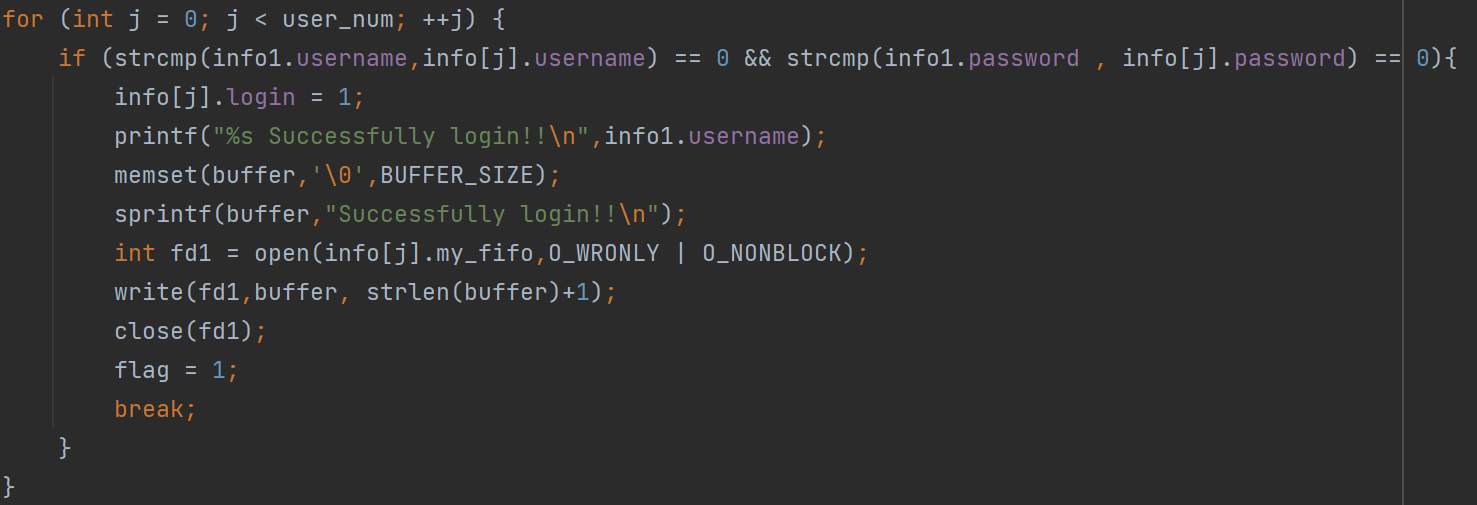
若不存在该用户名，则将信息记录在服务器中。



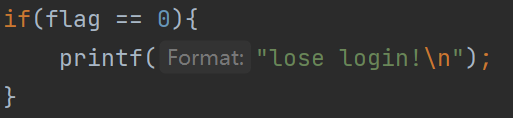
成功记录信息后，在服务器打印成功注册的信息，并且在将成功注册信息通过私有管道返回给客户机。最后将记录客户数量的变量加1。

对于管道2，实现用户登录功能

服务器需要对于客户端传递的用户名和密码查询客户信息中是否存在完全匹配的记录，若存在，返回成功登录的提示，并且将用户的登录状态置1。反之，返回登陆失败的提示。

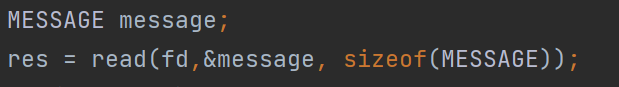


登录失败的提示：

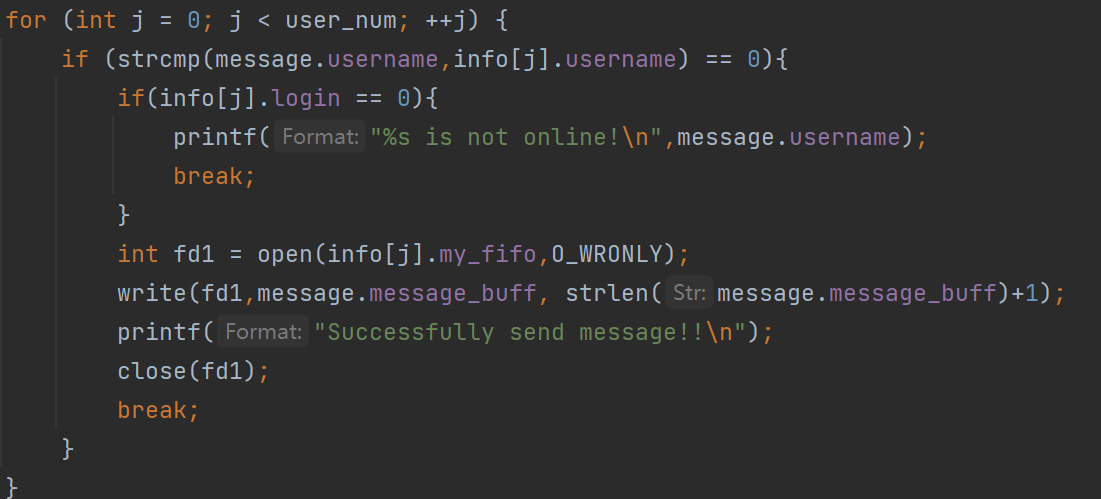


对于管道3，实现用户之间传递消息的功能

在这部分代码的实现，我们使用到了开头定义的message结构体，用于存储消息的目标用户。



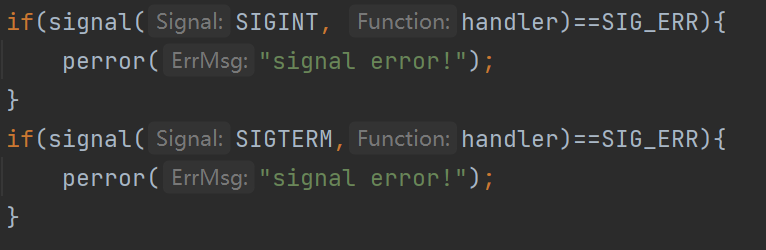
同理，在服务器的客户信息中查找是否存在用户名匹配的记录，若存在，则查看该用户的登录状态，若为0，则返回用户不在线信息，反之，查看目标用户私有管道，将消息通过私用管道传输。



其中，对于写消息的方式使用阻塞写的方式和非阻塞写的方式都可，若以阻塞写的方式，那么在客户机未准备好时，服务器会被阻塞在此，而以非阻塞写的方式，那么在客户机为准备好时，就得保证消息内容不能过大，在管道读端未读走消息时，消息会临时存放在管道中，若消息过大，管道被写满，进程依然会被阻塞。

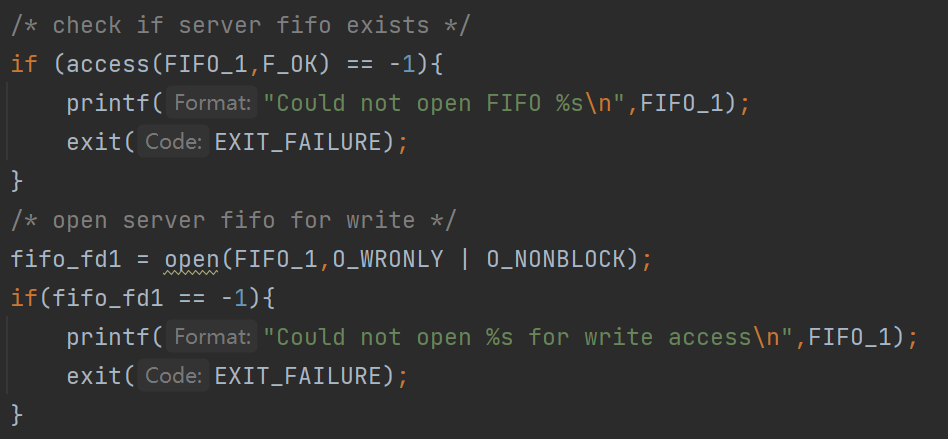
至此，服务器端的代码工作完成。接下来是客户端的代码实现。

注册信号处理函数，过程同服务端实现。



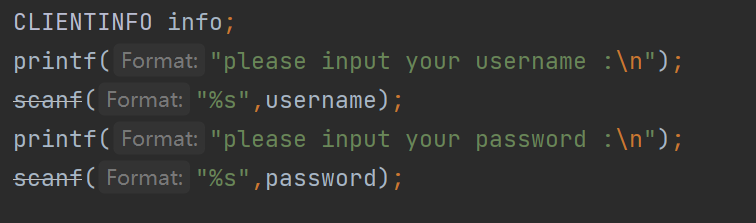
在客户端获取终端输入，有五个可选操作。

操作1，发起注册请求。



首先判断众所周知管道1是否打开。

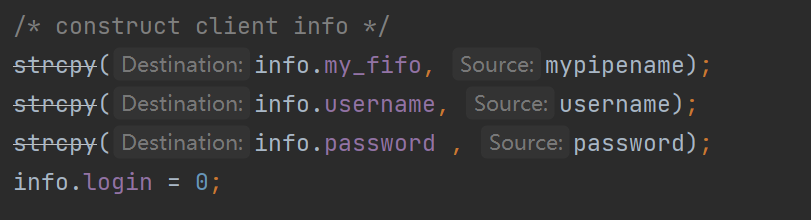
然后获取用户将要注册的用户名和密码。



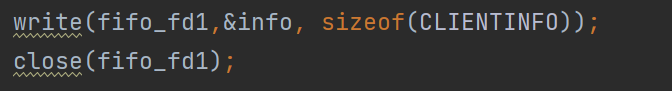
由于客户端的使用同一个代码文件，所以对于私有管道的命名，通过加入进程号来区分私有管道。创建私有管道，并且设置错误捕捉，方便在程序出现bug时查找bug来源。



对于将要传递的clientinfo进行初始化。

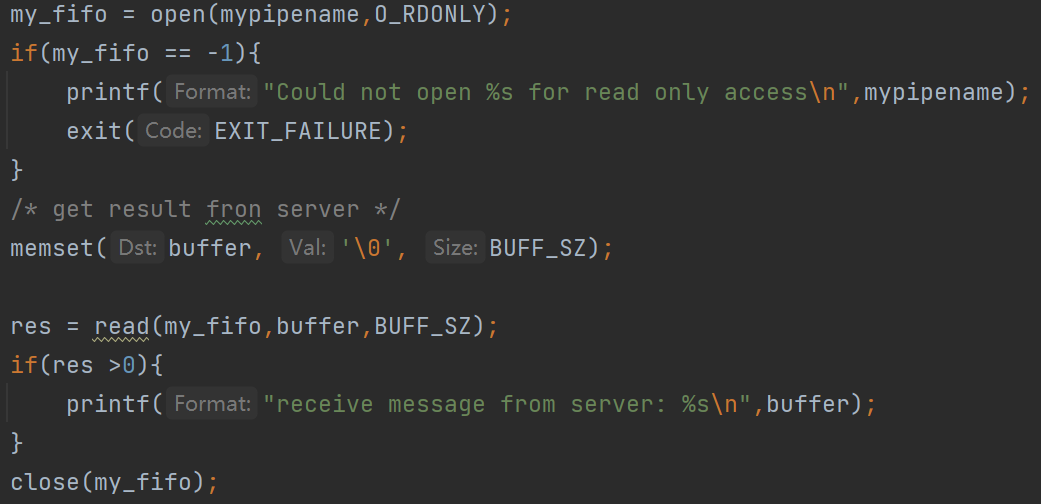


然后将clinetinfo结构体通过众所周知管道写到服务器。



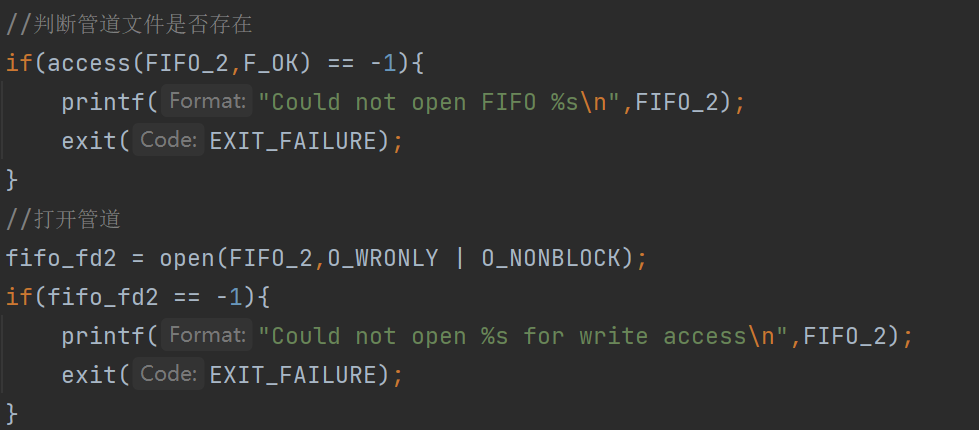
并且在完成写任务后立刻关闭写端，防止错误操作导致写入多余信息。

最后通过获取私有管道内容并打印的方法，将来自服务器端的反馈信息打印给用户查看。

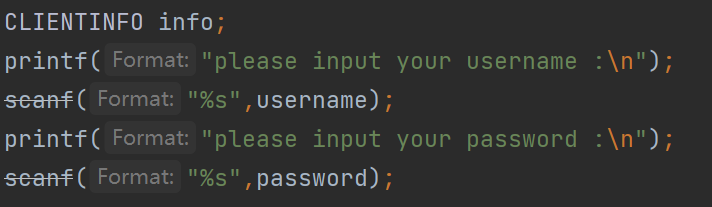


接下来是对于操作2的实现

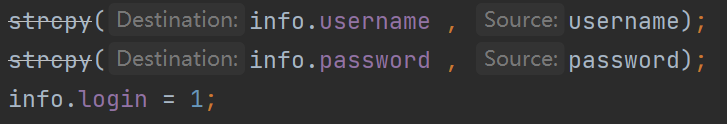
同上，首先判断管道文件是否存在，若存在则打开管道。



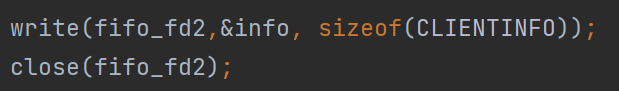
获取客户端的登录信息



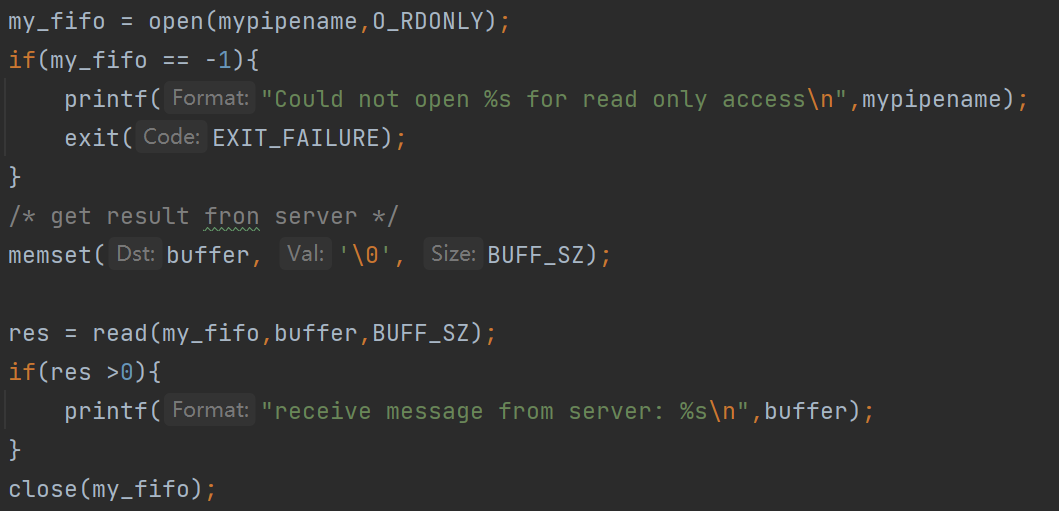
通过获取的信息初始化结构体



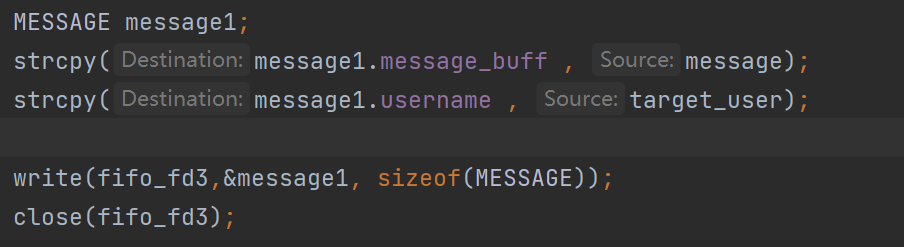
将结构体通过众所周知管道写给服务端



最后将来自服务器的反馈消息打印出来。

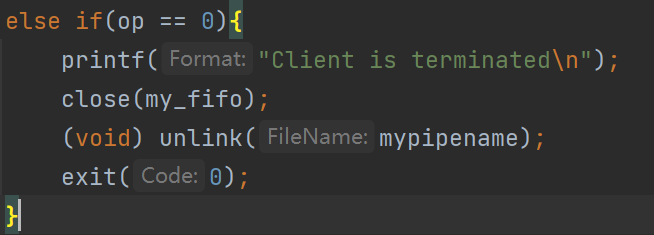


对于操作3的实现，较为简单，直接将结构体message写入服务器即可

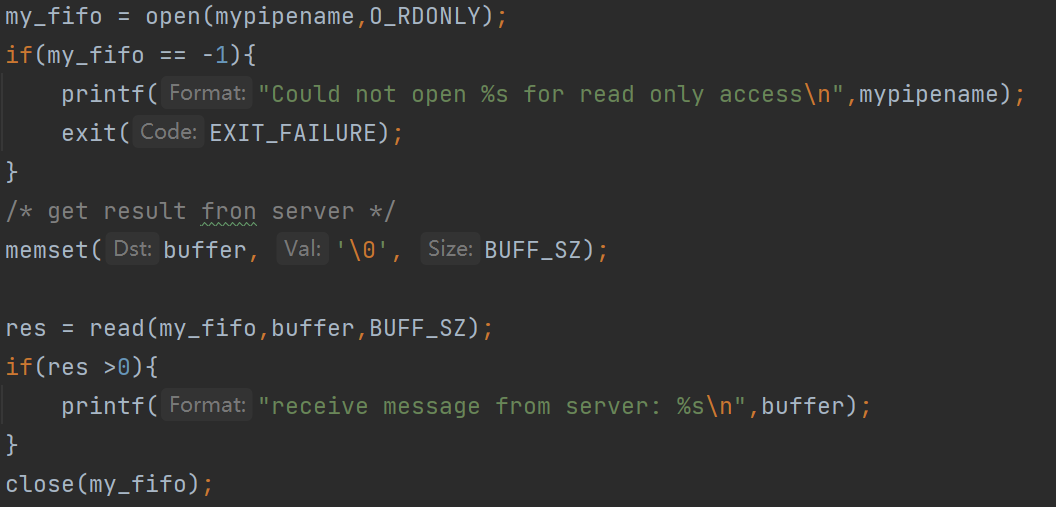


除了实现要求的三个操作外，我创建了两个新的操作用于辅助上述三个操作。

首先是关闭客户端的操作，我们除了在终端键入中断信号以外，也可以输入0，告知客户端主动终止进程，并且释放资源。

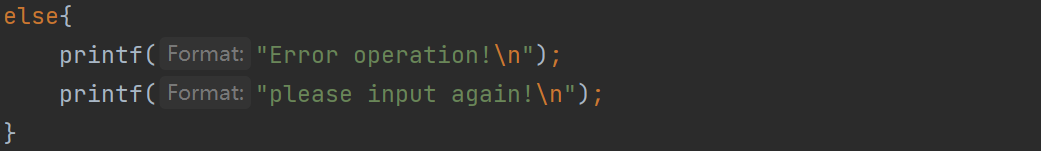


然后是对于收取消息的操作实现，当处于登录状态时，客户端键入4，表示处于等待状态，等待来自别的用户的信息，这样就避免了服务器获取到信息后无法通知给目标用户。



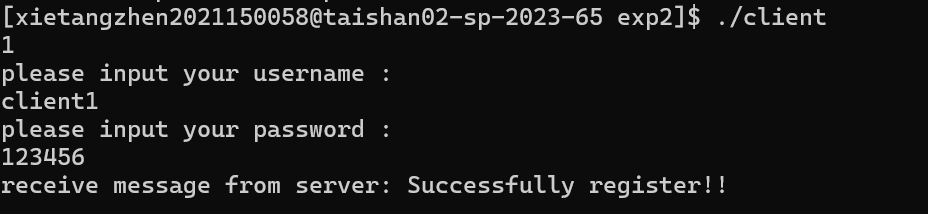
具体实现和读取来自服务器端的反馈信息一样。

最后，若键入了其他操作符号，则返回错误信息，并且要求重新键入。

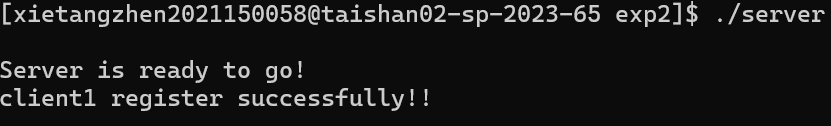


结果展示：

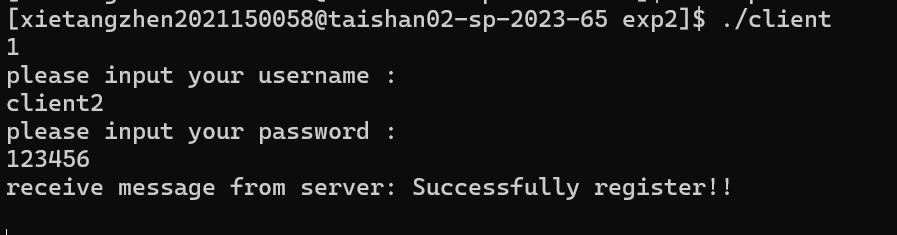
首先在一个终端运行client程序注册用户



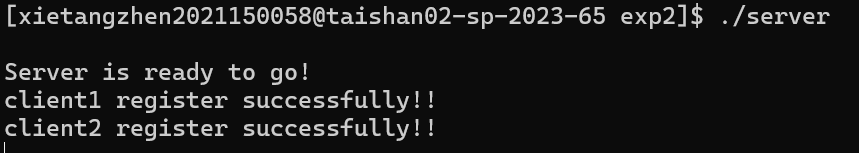
服务器打印出注册成功内容



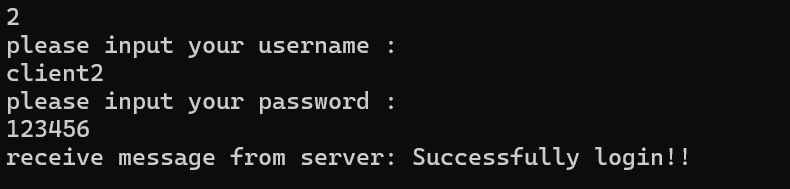
然后在另一个终端运行client程序，并且注册第二个用户



可以看到，这时候服务器也打印出了对应信息

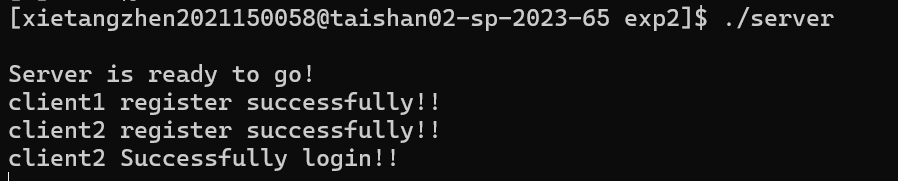


然后实现登录功能，client2用户发起登录请求



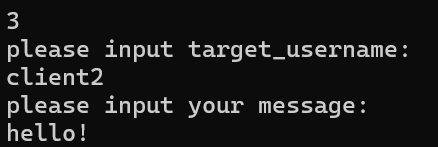
收到来自服务器的成功登录的信息。

接下来我们看看服务器端的信息。

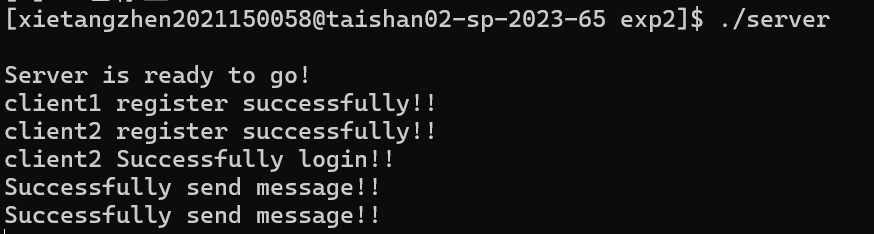


服务器端将成功登陆的信息打印到终端。

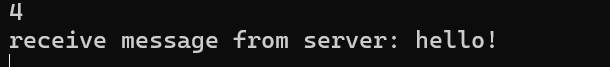
接下来，client1执行操作3，向client2发送信息。



服务器显示成功发送信息的提示



在client2中打印出来自client1发送的消息。



至此一个简单的聊天服务器功能实现。