网络程序设计
实 验 报 告

聊天程序软件

班 级 131114

学号 13111387

姓名 胡俊峰

实验目的：

1. 使用 socket 进行网络编程，熟悉网络编程的基本知识，掌握在课本上学到的理论知识

2. 编写聊天室程序，熟悉网络编程的一般套路

实验原理：

1. 服务端使用单进程，采用 epoll 来支持并发．

2. 使用 postgres 存储用户帐号密码

3. 使用 SQLAlchemy 定义表结构

4. 使用 psycopg2 连接 postgres 数据库服务器

5. 使用 Python 语言编写

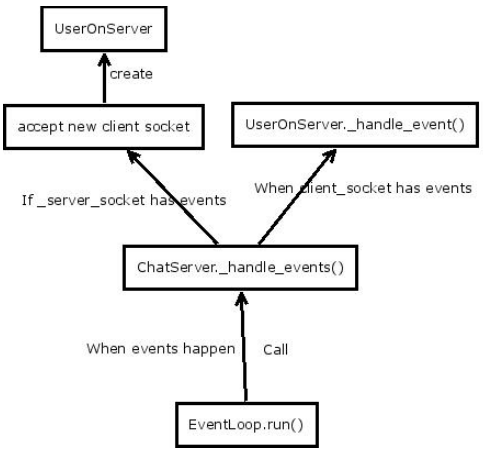
6. 非阻塞式 socket

7. server 端初始加入 listen socke 到 epoll 的模拟下，监听 server socket 的读状态，将 accept 来的客户 socket 再加入到 epoll 的模拟之下，同时以 client socket 初始话一个 UserOnServer 实例

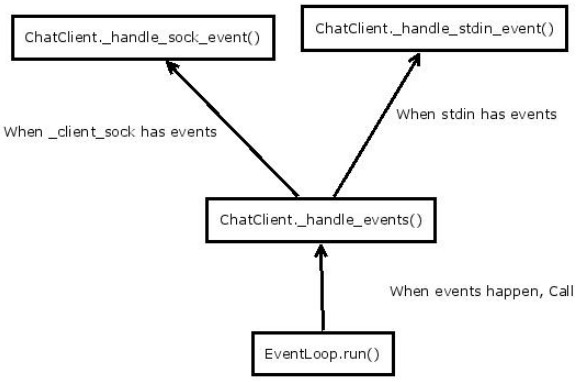
8. client 端同样采用 noblocking connect socket, 使用 epoll 监控 connected socket 和标准输入，接收标准输入的数据，发送到服务端，接收服务端的数据显示在标准输出上．

9. 使用&字符作为消息之间的分隔符，用户消息采用 json 传输，同时使用 urllib.quote()对用户数据进行转义

架构图：



关键代码：



ChatServer:

```
    def _handle_events(self, events):
        for sock, fd, event in events:
            if sock:
                logging.info('fd %d %s', fd,
                             eventloop.EVENT_NAMES.get(event, event))
            if sock == self._server_socket:
```

```
            if event & eventloop.EPOLL_ERR:
                # TODO
                raise Exception('server_socket error')
            try:
                client_sock, address = self._server_socket.accept()
                logging.info('accept client from %s', address)

                UserOnServer(self, self._fd_to_users, self._userid_login_status,
                        self._userid_to_fd, self._eventloop, client_sock)
            except (OSError, IOError) as e:
                error_no = eventloop.errno_from_exception(e)
                if error_no in (errno.EAGAIN, errno.EINPROGRESS):
                    continue
                else:
                    logging.error(e)
                    traceback.print_exc()
        else:
            if sock:
                user = self._fd_to_users.get(fd, None)
                if user:
                    user.handle_event(sock, event)
            else:
                logging.warn('poll removed fd')
```

UserOnServer:

```
    def handle_event(self, sock, event):
        if self.closed():
            logging.error("Got events for closed sock: %d", self._user_sock.fileno())
            return
        if event & eventloop.EPOLL_IN:
            self._on_read()
        if self.closed():
            return
        if event & eventloop.EPOLL_OUT:
            self._on_write()
        if self.closed():
            return
```

```
        if event & (eventloop.EPOLL_ERR | eventloop.EPOLL_HUP):
            self.close()
        state = self._STATE
        if len(self._writebuf) > 0:
            state |= eventloop.EPOLL_OUT
        self._loop.modify(self._user_sock, state)
```

ChatClient:

```
    def _handle_events(self, events):
        for f, fd, event in events:
            #if f:
            #   logging.info('f:%s fd %d %s', str(f), fd,
            #            eventloop.EVENT_NAMES.get(event, event))
            if f == self._chat_socket:
                self._handle_sock_event(event)
            elif f == self._stdin:
                self._handle_stdin_event(event)
            else:
                logging.error("don't know type f: %s", str(f))

        state = self._STATE
        if len(self._writebuf) > 0:
            state |= eventloop.EPOLL_OUT
        self._loop.modify(self._chat_socket, state)
```

EventLoop:

```
    def run(self):
        logging.info("starting eventloop...")
        while not self.stopping:
            try:
                events = self.poll(self.TIMEOUT)
            except (OSError, IOError) as e:
                if errno_from_exception(e) == errno.EPIPE:
                    # Happens when the client closes the connection
                    logging.error('poll:%s', e)
                    continue
                else:
```

```
                logging.error('poll:%s', e)
                traceback.print_exc()
                continue
        for handler in self._handlers:
            # TODO when there are a lot of handlers
            try:
                handler(events)
            except (OSError, IOError) as e:
                logging.error(e)
                traceback.print_exc()
```

总结：

通过本次 chat server, client 的编写，我了解到了自己的不足．对于 TCP socket 编程还很生疏．一直是在应用层上编程，到了 TCP socket 层，没有 HTTP, 不用应用框架开发，就有的心有余而力不足了．以后应该加强底层方面的修炼，特别是网络编程方面．本次实验，我也学到了不少．了解了 epoll 编程的基本框架，监控 server socket 的读，如果有新连接过来，server socket accept 就能立即成功．
把新的连接加入到 epoll 下，然后当发生 events 时，区别对待 server socket 和 client socket.同时因为 socket 读写都是 noblocking, 所以应该加入读写缓冲区．同时了解到新建立的 client socket 在 client 和 server 都是可写的．初始状态不能监控 client sock 的写状态，只有当有数据要写时，再动态添加 client socket 监听写事件

备注：

代码托管地址：https://github.com/junfeng-hu/sockchat.git