Министерство образования и науки РФ
Новосибирский Государственный Технический Университет
Кафедра ПМт

Лабораторная работа №3

«Создание приложения интерактивной переписки»

по курсу «Сетевые информационные технологии»

Вариант 1

Факультет: ПМИ
Группа: ПММ-81
Студенты: Михайлов А. А.,
Санина А. А.

Преподаватель: Долозов Н. Л.

Новосибирск 2012

1. Цель работы

Изучить основные принципы разработки многопользовательских приложений, построенных на основе технологии клиент-сервер с использованием стека протоколов TCP/IP.

С помощью API-интерфейса реализовать простой chat.

Каждая бригада должна написать chat-сервер и chat-клиента. Сервер должен поддерживать соединение сразу от нескольких клиентов. Обмен между клиентами осуществляется через сервер. При получении сообщения от какого-либо клиента, сервер дублирует его на своем экране и оповещает всех подсоединенных клиентов, отправляя каждому из них данное сообщение. При подсоединении нового клиента к chat-серверу, сервер оповещает каждого клиента о новом пользователе, посылая им его IP-адрес и имя.

2. Задание

В лабораторной работе №2 было реализовано простейшее взаимодействие, в котором участвовали один клиент и один сервер. В данной лабораторной работе предлагается модифицировать программы таким образом, чтобы сервер мог осуществлять взаимодействие с несколькими клиентами сразу.

3. Решение

3.1. Основные принципы

Главным вопросом, который нужно было решить в ходе разработки программного продукта, стал выбор способа организации работы сервера со многими клиентами. В основном, рассматривались два варианта:

- Выделение работы с каждым клиентом в отдельный поток (или даже процесс) и последующая одновременная обработка сообщений от всех клиентов;
- Работа со всеми клиентами в одном потоке.

В первом случае проще реализуется получение сообщения и ответ для единственного клиента, однако реализация сообщения между клиентами затруднена. Кроме того, как показывает опыт, такие решения не отличаются высоким быстродействием (частое переключение контекста между многими потоками — дорогая операция).

Если же мы работаем с клиентами в единственном потоке, мы теряем в производительности на многоядерной платформе (всегда задействуется только одно ядро), однако такое решение будет проще отлаживать. Те немногие серверные решения, которые на настоящий момент способны решить проблему 10K (десяти тысяч одновременных соединений с клиентами) придерживаются именно этого подхода.

Каждое клиентское соединение в случае использования стека протоколов TCP/IP реализуется в виде сокета — специального программного интерфейса, организующего ожидание сообщений и отправку сообщений отдельному клиенту. По умолчанию, сокеты являются *блокирующими*. Это означает, что процесс, который запросил прослушивание сообщений на сокете, приостанавливается до момента фактического получения сообщения, либо до превышения таймаута. При работе с *неблокирующими* сокетами приложение само решает, когда проверить наличие сообщений на сокете.

Событийно-ориентированное программирование (EBP, event-based programming) — это стиль программирования, при котором центральное место занимает цикл обработки *событий*. Событием, например, может быть факт готовности оборудования, срабатывание таймера, получение приложением сетевого пакета, сигнал операционной системы и т.п.

3.2. Реализация

Была разработана библиотека async-server, представляющая собой универсальную платформу для TCP/IP-сервера, способного работать со многими клиентами.

Основной класс, IOLoop реализует цикл генерации и обработки сообщений от клиентов. Сообщениями являются следующие события:

- Подключение клиента;
- Получение сообщения от клиента;
- Передача сообщения;
- Отключение клиента;
- Получение управляющего сообщения.

Перечисленные события отслеживаются системным вызовом epoll — это метод, который возвращает управление программе, если в одном из зарегистрированных для прослушивания файловых дескрипторов (которыми могут быть: файлы, каналы, сокеты, устройства и т.п.) появились свежие данные.

В случае подключения нового клиента, IOLoop делает обращение к объекту класса BaseHandlerFactory и последний создаёт новый объект типа BaseHandler (обработчик).

Таким образом, для каждого подключённого клиента будет создан свой объект-обработчик. Если обнаружена активность на сокете, либо готовы данные для передачи на сокет, вызываются соответствующие методы соответствущего обработчика.

Реализация конкретного серверного приложения сводится к реализации конкретных классов, дочерних от BaseHandlerFactory и BaseHandler.

3.3. Исходные тексты программ

3.3.1. IOLoop.py

```
1  #!/usr/bin/env python
2
3  import socket
4  import select
5  if not "EPOLLRDHUP" in dir(select):
6      select.EPOLLRDHUP = 0x2000
7
8  import os
9  from pty import STDIN_FILENO
10
11 def createServerSocket(host, port):
12     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13     s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
14     s.bind((host, port))
15     s.setblocking(0)
16     return s
17
18
19 class IOLoop(object):
20     def __init__(self):
21         self._host            = None
22         self._port            = None
23         self._serverSock      = None
24         self._handlerFactory  = None
25         self._epoll           = select.epoll()
26 #         self._manager         = os.open('chatserver.pipe', os.O_RDWR)
27         self._manager         = STDIN_FILENO
28         self._epoll.register(self._manager, select.EPOLLIN)
29         self._connections     = {}
30         self._handlers        = {}
31         self._responses       = {}
32         self._addresses       = {}
33
34     # IOLoop is the singleton
35     def __new__(cls):
36         if not hasattr(cls, 'instance'):
37             cls.instance = super(IOLoop, cls).__new__(cls)
38         return cls.instance
39
40     def listenTCP(self, handlerFactory, host = 'localhost', port = 20001):
41         self._handlerFactory = handlerFactory
42         self._host = host
43         self._port = port
44         if self._serverSock:
45             self._serverSock.close()
46             self._serverSock = None
47         self._serverSock = createServerSocket(self._host, self._port)
48
49     def send(self, fileno, data):
50         self._responses[fileno] = data
51         self._epoll.modify(fileno, select.EPOLLOUT)
52
53     def breakConnection(self, fileno):
54         self._epoll.modify(fileno, select.EPOLLRDHUP)
55
56     def _onClientJoin(self):
57         connection, address = self._serverSock.accept()
58         connection.setblocking(0)
59         self._epoll.register(connection.fileno(), select.EPOLLIN | select.EPOLLOUT | select.EPOLLRDHUP)
60         self._connections[connection.fileno()] = connection
61         self._handlers[connection.fileno()] = self._handlerFactory.buildHandler(address)
62         self._handlers[connection.fileno()].setSender(lambda data: self.send(connection.fileno(), data))
63         self._handlers[connection.fileno()].setBreaker(lambda: self.breakConnection(connection.fileno()))
64         self._handlers[connection.fileno()].connectionMade()
65         self._responses[connection.fileno()] = b''
66         self._addresses[connection.fileno()] = '[%s:%d]' % address
67
68     def _onDataReceived(self, fileno, event):
69         data = self._connections[fileno].recv(0x400)
70         if len(data):
71             print '%s ==> %dB received' % (self._addresses[fileno], len(data))
72             self._handlers[fileno].dataReceived(data)
73
```

```
74      def _onDataSent(self, fileno, event):
75          byteswritten = self._connections[fileno].send(self._responses[fileno])
76          self._responses[fileno] = self._responses[fileno][byteswritten:]
77          bytesleft = len(self._responses[fileno])
78          if bytesleft:
79              self._epoll.modify(fileno, select.EPOLLOUT)
80          else:
81              self._epoll.modify(fileno, select.EPOLLIN)
82          print '%s <== %dB written, %dB left' % (self._addresses[fileno], byteswritten, bytesleft)
83
84      def _onClientDisconnect(self, fileno, event):
85          self._epoll.unregister(fileno)
86          self._handlers[fileno].connectionLost()
87          del self._handlers[fileno]
88          self._connections[fileno].close()
89          del self._connections[fileno]
90
91      def run(self):
92          if not self._serverSock:
93              raise Exception, 'ioloop.listenTCP() must be called before ioloop.run()'
94          self._serverSock.listen(1)
95          self._epoll.register(self._serverSock.fileno())
96          try:
97              print 'Starting main loop on [%s:%d]...' % ( self._host, self._port )
98              while True:
99                  events = self._epoll.poll(1)
100                 for fileno, event in events:
101                     if fileno == self._serverSock.fileno():
102                         self._onClientJoin()
103                     elif fileno == self._manager:
104                         data = os.read( self._manager, 100)
105                         if 'exit' in data or 'quit' in data or 'kill' in data:
106                             return
107                     elif event & select.EPOLLIN:
108                         self._onDataReceived(fileno, event)
109                     elif event & select.EPOLLOUT:
110                         self._onDataSent(fileno, event)
111                     elif event & select.EPOLLRDHUP:
112                         self._onClientDisconnect(fileno, event)
113         except KeyboardInterrupt:
114             print 'Interrupted'
115         #except Exception, e:
116         #    print e.__class__, e
117         finally:
118             self.finish()
119
120     def finish(self):
121         print 'Finishing main loop on [%s:%d]...' % ( self._host, self._port ),
122         self._epoll.unregister(self._serverSock.fileno())
123         self._epoll.close()
124         self._serverSock.close()
125         print 'ok'
126
127
128
129 ioloop = IOLoop()
```

3.3.2. BaseHandlerFactory.py

```
1  #!/usr/bin/env python
2
3  class BaseHandlerFactory(object):
4      def buildHandler(self, addr):
5          raise NotImplementedError, 'Cannot create an object of abstract class: %s' % self.__class__.
               __name__
```

3.3.3. BaseHandler.py

```
1  #!/usr/bin/env python
2
3  class BaseHandler(object):
4      def __init__(self):
5          self._sender  = None
6          self._breaker = None
7
8      def connectionMade(self):
9          pass
10
11     def connectionLost(self):
12         pass
13
14     def dataReceived(self, data):
15         pass
16
17     def sendData(self, data):
18         if not self._sender:
19             raise Exception, 'handler.setSender() must be called before handler.sendData()'
20         self._sender(data)
21
22     def setSender(self, sender):
23         self._sender = sender
24
25     def breakConnection(self):
```

```
26          if not self._sender:
27              raise Exception, 'handler.setBreaker() must be called before handler.breakConnection()'
28          self._breaker()
29
30      def setBreaker(self, breaker):
31          self._breaker = breaker
```

3.3.4. chatserver.py

```
1   #!/usr/bin/env python
2
3   from IOLoop import ioloop
4   from BaseHandlerFactory import BaseHandlerFactory
5   from BaseHandler import BaseHandler
6
7
8
9   class ChatHandler(BaseHandler):
10      def __init__(self, factory, addr):
11          BaseHandler.__init__(self)
12          self._factory = factory
13          self._addr    = addr
14
15      def broadcast(self, msg):
16          print msg
17          for c in self._factory.clients:
18              if c <> self:
19                  c.sendData(msg)
20
21      def connectionMade(self):
22          self.broadcast('Connection established with <%s:%d>' % self._addr)
23          self._factory.clients.add(self)
24
25      def connectionLost(self):
26          self.broadcast('Disconnected: <%s:%d>' % self._addr)
27          self._factory.clients.remove(self)
28
29      def dataReceived(self, data):
30          self.broadcast('[%s:%d] : ' % self._addr + data)
31          if 'exit' in data:
32              self.breakConnection()
33
34
35  class ChatFactory(BaseHandlerFactory):
36      def __init__(self):
37          self.clients = set()
38
39      def buildHandler(self, addr):
40          return ChatHandler(self, addr)
41
42
43
44  if __name__ == '__main__':
45      ioloop.listenTCP(ChatFactory(), '', 2012)
46      ioloop.run()
```

3.3.5. chatclient.py

```
1   #!/usr/bin/env python
2
3   import socket
4   from pty import STDIN_FILENO
5   import select
6   import os
7
8   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9   epoll = select.epoll()
10  try:
11      s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
12
13      s.connect(('localhost', 2012))
14
15      epoll.register(STDIN_FILENO, select.POLLIN)
16      epoll.register(s.fileno(), select.POLLIN | select.POLLERR)
17
18      while True:
19          events = epoll.poll(1)
20          for fileno, event in events:
21              if fileno == STDIN_FILENO:
22                  data = os.read(STDIN_FILENO, 0x500)
23                  s.send(data.rstrip())
24                  if 'exit' in data or 'quit' in data or 'kill' in data:
25                      exit()
26              else:
27                  data = s.recv(0x500)
28                  if not data:
29                      exit()
30                  print data
31
32  finally:
33      epoll.close()
34      s.close()
```

3.4. Пример запуска

3.4.1. Сервер

```
[pmm8101@students async-server]$ python2.7 chatserver.py
Starting main loop on [:2012]...
Connection established with <127.0.0.1:38503>
Connection established with <127.0.0.1:38524>
Connection established with <127.0.0.1:38526>
[127.0.0.1:38524] : Hi!
[127.0.0.1:38503] : Hello!
[127.0.0.1:38526] : Hi there!
[127.0.0.1:38524] : let's go skiing!
[127.0.0.1:38503] : Oh! It's the very wonderful idea!!!!!!!!!
[127.0.0.1:38526] : Oh! No!! I'm ill :(
[127.0.0.1:38524] : Quickly get well, buddy!
[127.0.0.1:38503] : Quickly get well, buddy!
[127.0.0.1:38524] : Good bye!!
[127.0.0.1:38503] : Nighty-night, 38526 :)
[127.0.0.1:38526] : 10x gyus, bye all
[127.0.0.1:38503] : quit
Disconnected: <127.0.0.1:38503>
[127.0.0.1:38524] : exit
Disconnected: <127.0.0.1:38524>
[127.0.0.1:38526] : exit
Disconnected: <127.0.0.1:38526>
kill
Finishing main loop on [:2012]... ok
```

3.4.2. Клиент

```
[pmm8101@students async-server]$ python2.7 chatclient.py
Connection established with <127.0.0.1:38526>
Hi!
[127.0.0.1:38503] : Hello!
[127.0.0.1:38526] : Hi there!
let's go skiing!
[127.0.0.1:38503] : Oh! It's the very wonderful idea!!!!!!!!!
[127.0.0.1:38526] : Oh! No!! I'm ill :(
Quickly get well, buddy!
[127.0.0.1:38503] : Quickly get well, buddy!
Good bye!!
[127.0.0.1:38503] : Nighty-night, 38526 :)
[127.0.0.1:38526] : 10x gyus, bye all
exit
```

4. Ответы на контрольные вопросы к лабораторной работе

Все контрольные вопросы проработаны, затруднений не вызвали.