Advance Operating System Lab (CSD-416)
National Institute of Technology, Hamirpur

ADVANCE OPERATING SYSTEM LAB (CSD-416) ASSIGNMENT 4

Causal ordering of messages using BSS protocol

AKSHAT RAJ VANSH (185520)

December 11, 2021

Computer Science Department
National Institute of Technology, Hamirpur

Contents

1 Causal ordering of messages using BSS protocol

1.1 Code - Server

```
import socket as socket
import _thread
import threading


class Server:
    def __init__(self, port, admin, host="",):
        self.admin = admin
        self.host = host
        self.port = port
        self.connection = []
        self.members = []
        self.flags = []
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def configure(self):
        try:
            self.server.bind((self.host, self.port))
            print("Server binded to port", self.port)
            self.server.listen(5)
            print("Server is listening")
        except Exception as e:
            print(e)

    def decode(self, value):
        return value.decode('ascii')

    def encode(self, value):
        return value.encode('ascii')

    def broadcast(self, client, message):
        for x in self.connection:
            if x != client:
                x.send(self.encode(message))

    def threaded(self, client, client_addr, client_name):
        intial_message = "{} joined the chatroom".format(client_name)
        self.flags.append(0)
        self.broadcast(client, intial_message)
        while True:
            try:
                data = client.recv(1024)
                if not data or str(self.decode(data)) == "./leave":
                    self.broadcast(
                        client, "{} left the chatroom".format(client_name))
                    self.members.remove(client_name)
                    self.connection.remove(client)
                    break
                data = str(self.decode(data))
                if data == "./members":
                    client.send(self.encode(str(self.members)))
                    continue
                message = "{} : {}".format(client_name, data)
                self.flags[self.members.index(client_name)] += 1
```

```
55                  self.broadcast(client, message)
56                  print(self.flags)
57              except Exception as e:
58                  print(e)
59                  break
60          client.close()
61
62      def start(self):
63          while True:
64              client, client_addr = self.server.accept()
65              client_name = client.recv(1024)
66              self.connection.append(client)
67              self.members.append(self.decode(client_name))
68              print('Connected to :', client_addr[0], ':', client_addr[1])
69
70              _thread.start_new_thread(
71                  self.threaded, (client, client_addr, client_name.decode('ascii')))
```

1.2 Code - Controller

```python
import socket
from server import Server
import _thread
import threading
import json


class Controller:
    def __init__(self, port, host=''):
        self.port = port
        self.host = host
        self.controller = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        self.configure()

    def configure(self):
        self.controller.bind((self.host, self.port))
        print("Server binded to port", self.port)
        self.controller.listen(5)
        print("Server is listening")

    def decode(self, value):
        return value.decode('ascii')

    def encode(self, value):
        return value.encode('ascii')

    def threaded(self, admin, port):
        try:
            server = Server(port=port, admin=admin)
            server.configure()
            server.start()
        except Exception as e:
            print("Thread exception", e)
            self.port += 1
            self.threaded(admin, self.port)

    def listen(self):
        self.count = self.port + 1
        while True:
            client, client_addr = self.controller.accept()
            command = client.recv(1024)
            if self.decode(command) == "./createRoom":
                _thread.start_new_thread(
                    self.threaded, (client_addr, self.count))
                result = {
                    "status": 200,
                    "port": self.count,
                    "type": "creation",
                    "message": "CHATROOM CREATED"}
                client.send(self.encode(json.dumps(obj=result)))
                self.count += 1


if __name__ == '__main__':
    controller = Controller(12343)
```

```
57      controller.listen()
```

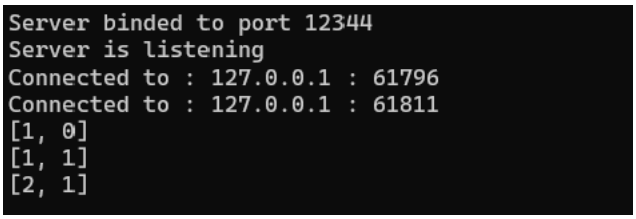
1.3 Code - Client

```
1   import socket
2   import json
3   import _thread
4   import threading
5
6
7   class Error:
8       commandInputError = Exception("Please enter correct command")
9       portInputError = Exception("Please enter correct port number")
10      controllerError = Exception("Controller Error. Try After Sometime")
11      createRoomError = Exception("Error in creating the room")
12
13
14  class Client:
15      def __init__(self, host, port):
16          self.host = host
17          self.port = port
18          self.flag = 0
19
20      def createSocket(self, port):
21          client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
22          client.connect((self.host, port))
23          return client
24
25      def decode(self, value):
26          return value.decode('ascii')
27
28      def encode(self, value):
29          return value.encode('ascii')
30
31      def listen(self, client):
32          while True:
33              data = client.recv(1024)
34              if not data:
35                  break
36              print(self.decode(data))
37          client.close()
38          exit(0)
39
40      def send(self, client):
41          while True:
42              message = input("")
43              self.flag += 1
44              if(message == "./leave"):
45                  break
46              client.send(self.encode(message))
47
48          client.send(self.encode(message))
49          self.flag += 1
50          client.close()
51          exit(0)
52
53      def joinChatRoom(self, port):
```

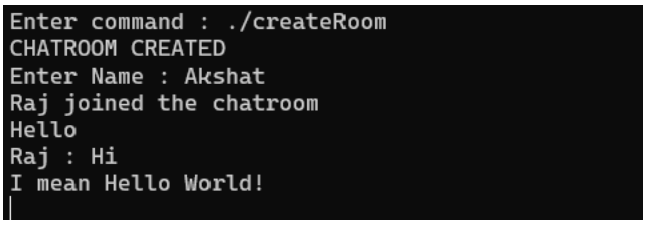
```
54          try:
55              client = self.createSocket(port)
56              name = input("Enter Name : ")
57              client.send(name.encode('ascii'))
58              _thread.start_new_thread(self.listen, (client,))
59              _thread.start_new_thread(self.send, (client,))
60              while True:
61                  continue
62          except Exception as e:
63              print(e)
64              self.joinChatRoom(port+1)
65
66      def start(self):
67          client = self.createSocket(self.port)
68          while True:
69              try:
70                  command = input("Enter command : ")
71                  if(command == "./join"):
72                      port = input("Enter port of 5 digits: ")
73                      assert(len(port) == 5)
74                      client.close()
75                      self.joinChatRoom(int(port))
76                      break
77                  elif(command == "./createRoom"):
78                      client.send(self.encode(command))
79                      reply = client.recv(1024)
80                      if not reply:
81                          raise Error.controllerError
82                          continue
83                      result = json.loads(str(self.decode(reply)))
84                      if(result["status"] == 200):
85                          if(result["type"] == "creation"):
86                              client.close()
87                              print(result["message"])
88                              self.joinChatRoom(result["port"])
89                              break
90                      else:
91                          raise Error.createRoomError
92                  else:
93                      raise Error.commandInputError
94              except Exception as e:
95                  print(e)
96                  continue
97
98
99  if __name__ == '__main__':
100     client = Client('127.0.0.1', 12343)
101     client.start()
```

1.4 Output

Causal ordering of messages using BSS protocol Output: Controller



Causal ordering of messages using BSS protocol Output: Client 1



Causal ordering of messages using BSS protocol Output: Client 2