Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Управление ресурсами в вычислительных системах

Лабораторная работа №8

Факультет: прикладной математики и информатики

Группа: ПМ-63

Студенты: Кожекин М.В.

Утюганов Д.С.

Преподаватели: Стасышин В. М.

Хайленко Е. А.

Новосибирск

2019

1. Условие (Вариант №4)

Организовать взаимодействие типа клиент-сервер. Клиент отсылает строку серверу. Сервер отсылает данную строку на другие сервера, список которых хранится в файле, а там уже осуществляется поиск файлов содержащих данную строку. Результаты поиска отсылаются клиенту.

1. Анализ задачи

**Клиент**

Создаём сокет

Присоединяемся к основному серверу

Отправляем строку на основной сервер

Принимаем результат поиска

Освобождаем сокет

**Основной сервер**

Создаём сокет

Настраиваем адрес и порт

Слушаем

Принимаем соединение клиента

Получаем запрос

Открываем файл результатов поиска

Открываем файл с данными вспомогательных серверов

Считываем адрес и порт

Создаём дочерний поток

Создаём сокет

Присоединяемся к вспомогательному серверу

Отправляем строку на вспомогательный сервер

Принимаем результат поиска и записываем его в файл

Освобождаем сокет

Отправляем файл с результатами клиенту

Освобождаем сокет

**Вспомогательный сервер**

Создаём сокет

Настраиваем адрес и порт

Слушаем

Принимаем соединение клиента

Получаем запрос

Выполняем поиск перенаправив поток в файл

Отправляем файл на основной сервер

Освобождаем сокет

1. Используемые программные средства

**Функции:**

socket() – создание нового сокета

bind(ip, port) – настройка сервер

listen(count) – ожидание соединений

connect((ip, port)) – присоединение к серверу

accept() – приём соединения

send(data) – отправка данных

recv(buf\_size) – приём данных

close() – уничтожение сокета

Thread(function, parameters) – создание дочернего потока

system(command) – выполнение команды в консоли

open(file\_path, mode) – открытие файла

read(buf\_size) – чтение из файла **buf\_size** байт

write(string) – запись в файл

close() – закрытие файла

1. Спецификация

Программа находится в папке “~/lab8/python2”

Чтобы запустить программу, нужно использовать команду “make run”

Если адрес окажется занятым нужно ввести команду “make clean”

В результате работы программы, будет произведён поиск файлов содержащих строку при помощи основного сервера и нескольких ведомых им.

1. Исходный код

**Файл client.py**

|  |
| --- |
| # -\*- coding: utf-8 -\*-  import socket  from sys import argv  # Парсим адрес и порт главного сервера  ipAddress = str(argv[1])  port = int(argv[2])  pattern = "abc"    conn = socket.socket() # создаём сокет  conn.connect((ipAddress, port)) # присоединяемся к основному серверу  conn.send(pattern.encode("utf-8")) # отправляем запрос на основной сервер  # Принимаем ответ от главного сервера  line = conn.recv(1024).decode("utf-8")  while (line):  print(line)  line = conn.recv(1024).decode("utf-8")  conn.close() # освобождаем сокет |

**Файл mainServer.py**

|  |
| --- |
| # -\*- coding: utf-8 -\*-  import socket  from threading import Thread  from sys import argv  # Парсим адрес и порт главного сервера  ipAddress = str(argv[1])  port = int(argv[2])  # Поток присоединённый к одному из серверов  def findData(\_ip, \_port, \_pattern, f):  \_ip = str(\_ip)  conn = socket.socket()  print("Trying connect to ", \_ip, \_port)  conn.connect((\_ip, \_port))  conn.send(pattern)  l = conn.recv(1024)  while (l):  f.write(l)  l = conn.recv(1024)  conn.close()    #-------------------------------------------------------------------------------  #-------------------------------------------------------------------------------  sock = socket.socket() # создаём сокет  sock.bind((ipAddress, port)) # настраиваем адрес  sock.listen(1) # прослушиваем порт  conn\_to\_client, addr = sock.accept() # принимаем соединение  pattern = conn\_to\_client.recv(1024) # принимаем запрос  #-------------------------------------------------------------------------------  data=[]  threads=[]  f = open("answer.txt", "wb")  with open("servers.txt", 'r') as f\_s:  for line in f\_s:  data.append([ str(x) for x in line.split()])  for i in data:  t = Thread(target=findData, args=(str(i[0]), int(i[1]), pattern, f))  t.start()  threads.append(t)  print(i)  # Закрываем дочерние потоки  for i in threads:  i.join()  f.close()  f = open("answer.txt", "rb")  line = f.read(1024)  while(line):  conn\_to\_client.send(line)  line = f.read(1024)  f.close()  sock.close() # освобождаем сокет  print("Основной сервер отправил ответ клиенту") |

**Файл server.py**

|  |
| --- |
| # -\*- coding: utf-8 -\*-  import socket  from os import system  from sys import argv  # Парсим адрес, порт, номер и директорию поиска главного сервера  ipAddress = str(argv[1])  port = int(argv[2])  serverNumber = int(argv[3])  folderToSearch = str(argv[4])  print "Сервер ", str(serverNumber), " запущен IP: ", ipAddress + " port: ", (port)  #-------------------------------------------------------------------------------  #-------------------------------------------------------------------------------  sock = socket.socket() # создаём сокет  sock.bind((ipAddress, port)) # настраиваем адрес  sock.listen(1) # прослушиваем порт  conn, addr = sock.accept() # принимаем соединение  pattern = conn.recv(1024).decode("utf-8") # принимаем запрос  bashCommand = "grep -rnw \'" + folderToSearch + "\' -e \'" + pattern + "\' > tmpfile.txt"  os.system(bashCommand) # выполняем команду поиска  #-------------------------------------------------------------------------------  f = open("tmpfile.txt", "rb")  line = f.read(1024)  while(line):  conn.send(line)  line = f.read(1024)  sock.close() # освобождаем сокет  print "Сервер ", str(serverNumber), " отправил ответ на основной сервер" |

**Содержимое файла Makefile**

|  |
| --- |
| # Makefile for lab #8  run:  python server.py 127.0.0.1 8080 1 folder &  sleep 1s  python server.py 127.0.0.2 8080 2 ../../lab5 &  sleep 1s  python server.py 127.0.0.3 8080 3 ~/test &  sleep 1s  python mainServer.py 127.0.0.255 8080 &  sleep 1s  python client.py 127.0.0.255 8080  clean:  lsof - ti : 8080 | xargs --no - run - if - empty kill - 9 |

1. Результат работы программы

Программа работает должным образом

|  |
| --- |
| -bash-4.2$ make run  python server.py 127.0.0.1 8080 1 folder &  sleep 1s  Сервер 1 запущен IP: 127.0.0.1 port: 8080  python server.py 127.0.0.2 8080 2 ../../lab5 &  sleep 1s  Сервер 2 запущен IP: 127.0.0.2 port: 8080  python server.py 127.0.0.3 8080 3 ~/test &  sleep 1s  Сервер 3 запущен IP: 127.0.0.3 port: 8080  python mainServer.py 127.0.0.255 8080 &  sleep 1s  python client.py 127.0.0.255 8080  ('Trying connect to ', '127.0.0.1', 8080)  ['127.0.0.1', '8080']  ('Trying connect to ', '127.0.0.2', 8080)  ['127.0.0.2', '8080']  ('Trying connect to ', '127.0.0.3', 8080)  ['127.0.0.3', '8080']  Сервер 1 отправил ответ на основной сервер  Сервер 2 отправил ответ на основной сервер  Сервер 3 отправил ответ на основной сервер  Основной сервер отправил ответ клиенту  folder/file2:4:abc  folder/file1:3:abc  folder/file1:4:abc  /home/NSTU/brigades/pmi-b6301/test/abcFile.txt:1:abc  /home/NSTU/brigades/pmi-b6301/test/abcFile.txt:1:abc |
| -bash-4.2$ grep -rnw ~/test -e 'abc'  /home/NSTU/brigades/pmi-b6301/test/abcFile.txt:1:abc  -bash-4.2$ grep -rnw ../../lab5 -e 'abc'  -bash-4.2$ grep -rnw folder -e 'abc'  folder/file2:4:abc  folder/file1:3:abc  folder/file1:4:abc |