ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра ВС

Самостоятельная работа

Отказоустойчивая сетевая игра «Виселица»

ВЫПОЛНИЛИ:

студенты гр. ВМ-88

Егоров И.Ю.

Барановский В.А.

ПРОВЕРИЛ:

Майданов Ю.С.

Новосибирск 2011

**Цель работы:**

Разработать сервер-клиент приложение учитывая некоторые аспекты реализации:

1. При отказе клиента сервер должен продолжить функционирование, сообщив второму пользователю об уходе первого.  
2. При отказе сервера клиенты должны это распознать и вывести сообщение, после этого завершив работу.  
3. Во всех процессах передачи данных по сети реализовать проверку протокола и контроль объема переданных данных. При некорректных действиях клиента сервер должен сделать пометку в логе и отключить данного клиента.  
4. Необходимо реализовать альтернативный сервер управления с протоколом синхронизации между основным и альтернативным серверами. При отказе основного сервера клиенты должны переключится на альтернативный сервер сохранив состояние.

**Листинг программы:**

1. При отказе клиента сервер должен продолжить функционирование, сообщив второму пользователю об уходе первого.

**protocol\_server.py:**

try:

text = sock.recv(128)

parse = text.split("@")

except socket.error, detail:

userscount -= 1

**logger.info(name + " has been disconnected!")**

del users[sock]

new = None

**sendmsg(ANSWER\_USERCOUNT + "\_%s@" % (userscount), sock)**

break

if not text:

**logger.info(name + " has been disconnected!") #запись события отключения в лог**

new = None

sleep(1)

userscount -= 1

sock.close()

del users[sock]

**sendmsg(ANSWER\_USERCOUNT + "\_%s@" % (userscount), sock)**

1. При отказе сервера клиенты должны это распознать и вывести сообщение, после этого завершив работу.

**client.py:**

try:

if self.main:

self.sock.connect((CLI\_MAIN\_HOST, CLI\_MAIN\_PORT))

else:

sleep(3)

logger.info("Connecting to alternative server...")

self.sock.connect((CLI\_ALT\_HOST, CLI\_ALT\_PORT))

except socket.error:

if self.main:

try:

sleep(3)

**logger.info("Connecting to alternative server...") #если подключение к основному серверу не удалось**

self.sock.connect((CLI\_ALT\_HOST, CLI\_ALT\_PORT))

except socket.error:

**logger.critical("Connecting to alternative server failed...")**  **#если подключение к альт. серверу не удалось**

**logger.critical("Servers not work!") #вывод ошибки**

sleep(5)

sys.exit(0)

else:

logger.critical("Servers not work!") **#если произошел разрыв соединения с альт. сервером**

try:

self.sock.send(QUERY\_CONN)

self.query\_result = self.sock.recv(4)

if (self.query\_result == CONN\_ALLOW):

logger.info("Connect allowed!")

self.connected = True

else:

logger.error(self.query\_result)

except socket.error, detail:

logger.error(detail)

if self.main:

cli.connect(False)

else:

logger.critical("Alternative server down!") **#если произошел разрыв соединения с альт. сервером**

sys.exit(0)

1. Во всех процессах передачи данных по сети реализовать проверку протокола и контроль объема переданных данных. При некорректных действиях клиента сервер должен сделать пометку в логе и отключить данного клиента.

**protocol\_server.py:**

**if item[0] == "#": первый символ пакета данных — символ «#», иначе — отключение пользователя**

if text[0:4] == QUERY\_CONN:

sock.send(CONN\_ALLOW + "@")

queue\_start.append(sock)

if main\_server:

word = gallows.generate()

usersword = "\*" \* len(word)

s.changed = True

logger.info("\nSecret word generated! [%s]. \nFor users: %s\n" % (word, usersword))

else:

gallows.secret = pinger.secret

gallows.attempts = pinger.attempts

gallows.newuword = pinger.userword

usersword = pinger.userword

gallows.newuword = pinger.userword

gallows.used\_letters = pinger.used\_letters

s.changed = False

sendmsg(PACKET\_USERWORD + "\_%s\_%s@" % (usersword, gallows.attempts), sock)

sendmsg(ANSWER\_USERCOUNT + "\_%s@" % (userscount), sock)

break

lst = item.split("\_")

logger.debug(lst)

if lst[0] == QUERY\_USERCOUNT:

sendmsg(ANSWER\_USERCOUNT + "\_%s@" % (userscount), sock)

logger.info("Userscount is %s" % (userscount))

**if lst[0] == PACKET\_LETTER:**

**if len(lst[1])== 1 and re.match("^[a-z]\*$", lst[1]): #если в пакете PACKET\_LETTER длина данных == 1 и символ входит в множество a—z, то всё в порядке. Иначе — отключение пользователя.**

letter = lst[1]

result = gallows.getletter(usersword, strip(letter))

logger.info("S: %s UW: %s Text: %s Letter: %s Result: %s" % (gallows.secret, usersword, text, letter, result))

usersword = result[1]

s.changed = True

if (gallows.attempts == 0):

sendmsg(WORD\_FAIL + "\_%s\_%s@" % (name, word), sock)

restart = True

else:

if (result[0] != 0):

if (gallows.attempts != 0):

if (result[0] > 0):

sendmsg(LETTER\_WIN + "\_%s\_%s\_%s\_%s@" % (name, letter, usersword, gallows.attempts), sock)

s.changed = True

if (result[0] < 0) or (gallows.attempts == 0):

if (result[0] == -1):

sendmsg(WORD\_WIN + "\_%s\_%s@" % (name, usersword), sock)

s.changed = True

restart = True

if (result[0] == -2):

sendmsg(LETTER\_ALREADY + "\_%s@" % letter, sock)

else:

sendmsg(LETTER\_FAIL + "\_%s\_%s\_%s\_%s@" % (name, letter, usersword, gallows.attempts), sock)

gallows.attempts -= 1

s.changed = True

break

if lst[0] == QUERY\_USERWORD:

sendmsg(PACKET\_USERWORD + "\_%s\_%s@" % (usersword, gallows.attempts), sock)

restart = True

logger.info("Userscount is %s" % (userscount))

**else: #отключение пользователя, если пришел пакет с неверным заголовком**

**kick = True**

**else: #отключение пользователя, если первый символ пакета ≠ «#»**

**kick = True**

if (kick):

logger.info("Client %s kicked!" % name)

sock.send(CONN\_CLOSE\_KICK)

userscount -= 1

del users[sock]

new = None

sendmsg(ANSWER\_USERCOUNT + "\_%s@" % (userscount), sock)

sock.close()

1. Необходимо реализовать альтернативный сервер управления с протоколом синхронизации между основным и альтернативным серверами. При отказе основного сервера клиенты должны переключится на альтернативный сервер сохранив состояние.

Реализовано через отдельный классы:

**protocol\_server.py**

**class Pinger(Thread): #«синхронизирующий», основной сервер**

def \_\_init\_\_(self):

Thread.\_\_init\_\_(self)

self.packets = None

self.lastsync = None

def parsesync(self, packets):

packets = packets[0].split("\_")

if (len(packets) == 5) and (packets[0] == SYNC\_SERVER\_PACKET):

self.secret = packets[1]

self.attempts = int(packets[2])

self.userword = packets[3]

self.used\_letters = list(packets[4])

start()

def run(self):

try:

self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.sock.connect((HOST\_PONG, PORT\_PONG))

except:

logger.critical("Main server don't work! %s %s" % (HOST\_PING, PORT\_PING))

sys.exit(0)

while True:

try:

self.sock.send(CONN\_PING + "@")

data = self.sock.recv(128)

except socket.error, detail:

logging.error(detail)

logger.error("Ping server error! %s" % self.packets)

if self.lastsync:

self.parsesync(self.lastsync)

else:

logger.error("Main server not send sync packet! :(")

sleep(5)

sys.exit(0)

break

if not data: logger.error("Ping server error! %s" % self.sock.fileno())

else:

try:

self.packets = data.strip()

self.packets = self.packets.split("$")

for pack in self.packets:

answers = pack.strip()

answers = pack.split("\_")

for item in answers:

if len(item) > 0:

if item[0] == "#":

code = item[:4]

if code == CONN\_PONG:

logger.info("Ping server success! %s" % self.sock.fileno())

self.sock.send(CONN\_PING + "$")

elif code == SYNC\_SERVER\_PACKET:

print "SYNC: " + str(answers)

self.lastsync = self.packets

self.sock.send(SYNC\_SERVER\_PACKET\_APPLY + "$")

else:

logger.error("Ping server error! CODE: '%s'" % code)

self.parsesync(self.lastsync)

except socket.error:

logger.error("Ping server error! %s" % self.packets)

self.parsesync(self.lastsync)

**class Ponger(Thread): #«синхронизуемый», альтерантивный сервер**

def \_\_init\_\_(self):

Thread.\_\_init\_\_(self)

def run(self):

try:

pong = socket.socket(AF\_INET, SOCK\_STREAM)

pong.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

pong.bind((HOST\_PONG, PORT\_PONG))

logger.debug("Ponger server binded")

pong.listen(5)

logger.debug("Ponger server listen on port " + str(PORT\_PONG))

sleep(15)

except socket.error, detail:

logger.error(detail)

pingsock, addr = pong.accept()

logger.info("Ponger connected %s" % str(addr))

try:

while True:

data = pingsock.recv(32)

ping = data.strip()

ping = ping.split("$")

if not data:

logger.error("Pong server error! %s" % pingsock.fileno())

sleep(5)

pingsock, addr = pong.accept()

pingsock.send(SYNC\_SERVER\_PACKET + "\_%s\_%s\_%s\_%s$" % (gallows.secret, str(gallows.attempts), gallows.newuword, str(gallows.used\_letters)))

else:

for item in ping:

if len(ping) > 0:

if len(item) > 0:

if item[0] == "#":

code = item[:4]

if code == CONN\_PING:

if (hasattr(gallows, 'secret') and not main\_server) or (s.changed and main\_server):

pingsock.send(SYNC\_SERVER\_PACKET + "\_%s\_%s\_%s\_%s$" % (gallows.secret, str(gallows.attempts), gallows.newuword, str(gallows.used\_letters)))

s.changed = False

logger.info("Send: " + SYNC\_SERVER\_PACKET + "\_%s\_%s\_%s\_%s$" % (gallows.secret, str(gallows.attempts), gallows.newuword, str(gallows.used\_letters)))

sleep(2)

else:

pingsock.send(CONN\_PONG + "$")

sleep(2)

break

elif code == SYNC\_SERVER\_PACKET\_APPLY:

logger.info("SYNC OK! %s" % pingsock.fileno())

elif item[0] != "#":

break

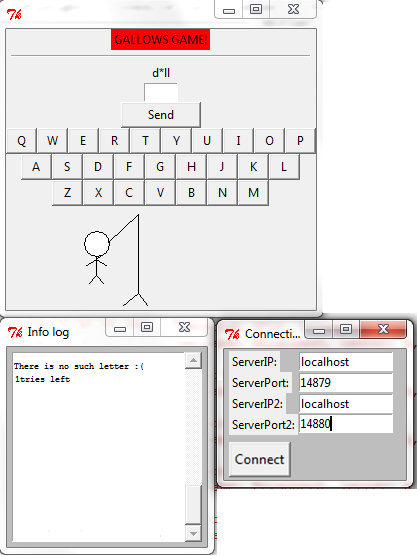
else:

logger.info("Pong server unknown answer code! CODE: %s" % code)

except socket.error, detail:

logger.error("Pong server error! %s" % pingsock.fileno())

1. Так же в программе предусмотрен графический интерфейс для удобства работы:



1. Окно “Connection” позволяет пользователю ввести IP адрес и порт для главного и альтернативного серверов.
2. Окно “Info log” информирует пользователя о количестве оставшихся попыток, наличии в слове введенной буквы и разрывах в соединении.
3. “Tk” – главное окно для игры c рисунком, который отображает количество попыток

**Протоколы взаимодействия**

**Протокол синхронизации**

В модуле protocol\_server.py реализованы два класса Pinger и Ponger. Ponger – синхронизуемый, альтернативный сервер, а Pinger в свою очередь синхронизирующий, основной. В синхронизации используются несколько заголовков пакетов, такие как

SYNC\_SERVER\_PACKET = "#720" **#заголовок, за которым следует загаданное слово, количество оставшихся попыток, слово которое видно пользователям (с засекреченными буквами и список с уже использованными буквами**

SYNC\_SERVER\_PACKET\_APPLY = "#721" **#ответ успешной синхронизации от синхронизуемого сервера**

CONN\_PING = "#698" **#ping-пакет**

CONN\_PONG = "#699" **#pong-пакет**

**Протокол клиент-серверного взаимодействия**

**Заголовки пакетов:**

В кодах заголовков:

**первая цифра** означает тип пакета (запрос на соединение, пакет с буквой от пользователя; запрос, связанный с загаданным словом; использованными буквами и т.д.)

**вторая цифра –** источник отправкипакета (0 – клиент, 1 – сервер)

**третья цифра –** дополнительная, идентификация пакета

**constants.py**

QUERY\_CONN = "#100" **#запрос подключения от клиента**

CONN\_ALLOW = "#111" **#разрешение от сервера**

PACKET\_LETTER = "#201" **#пакет с буквой от клиента**

PACKET\_ALLOW = "#211" **#пакет принят**

QUERY\_USERWORD = "#203" **#запрос на слово**

PACKET\_USERWORD = "#213" **#ответ, содержащий в себе слово**

QUERY\_USERCOUNT = "#204" **#запрос на количество онлайн пользователей**

ANSWER\_USERCOUNT = "#214" **#ответ с количеством пользователей**

QUERY\_USEDLETTERS = "#205" **#запос на использованные буквы**

ANSWER\_USEDLETTERS = "#215" **#ответ**

LETTER\_FAIL = "#310" **#ответ сервера: буква не угадана**

LETTER\_WIN = "#311" **#ответ сервера: буква угадана**

LETTER\_ALREADY = "#320" **#ответ сервера: буква уже была использована**

WORD\_FAIL = "#410" **#ответ сервера: слово не угадана по истечению всех попыток**

WORD\_WIN = "#411" **#ответ сервера: слово угадано**

CONN\_CLOSE\_CLI = "#500" **#клиент закрывает соединение**

CONN\_CLOSE\_SERV = "#510" **#сервер закрывает соединение**

CONN\_CLOSE\_KICK = "#511" **#сервер принудительно закрывает соединение с клиентом**

**Листинг программы:**

**client.py**

#-\*- coding:utf-8 -\*-

"""

@module: client module for gallows

@license: GNU GPL v2

@author: Egorov Ilya

@version: 0.1

"""

from socket import \*

from sys import exit

from select import select

from time import \*

from threading import \*

from constants import \*

from string import \*

import socket, string, sys, threading, select, time, logging, constants, string, re

global HOST, PORT, LOG, usersword, connected, code, param

connected = [""]

logger = logging.getLogger("client")

logger.setLevel(logging.DEBUG)

logstream = logging.StreamHandler()

logstream.setLevel(logging.DEBUG)

formatter = logging.Formatter("%(asctime)s: %(message)s")

logstream.setFormatter(formatter)

logger.addHandler(logstream)

def answerparse(code="", param=""):

lst = [""]

parse = {}

if param != "":

param = param.strip()

lst = param.split('\_')

if code == PACKET\_USERWORD: #userword

if lst.\_\_len\_\_() == 2:

parse[PACKET\_USERWORD] = lst

logger.debug("Userword: %s, attempts: %s" % (lst[0], lst[1]))

else:

parse[PACKET\_USERWORD] = None

logger.debug("USERWORD Undefined params! [%s] Lst: %s" % (code, lst))

return parse

if code == ANSWER\_USERCOUNT:

if lst.\_\_len\_\_() == 1:

parse[ANSWER\_USERCOUNT] = lst

logger.debug("Userscount: %s" % (lst[0]))

else:

parse[ANSWER\_USERCOUNT] = None

logger.debug("Userscount undefined!")

return parse

elif code == LETTER\_FAIL or code == LETTER\_WIN: #letter fail or win

if lst.\_\_len\_\_() == 4:

logger.debug("Username: %s, letter: %s, word: %s, attempts: %s" % (lst[0], lst[1], lst[2], lst[3]))

if(code == LETTER\_WIN):

parse[LETTER\_WIN] = lst

else:

parse[LETTER\_FAIL] = lst

else:

parse[LETTER\_FAIL] = None

logger.debug("LETTER\_FAIL\_OR\_WIN Undefined params! [%s] Lst: %s" % (code, lst))

return parse

elif code == LETTER\_ALREADY: #letter already exists. TO-DO: Return letter (lst[0])

parse[LETTER\_ALREADY] = lst

logger.debug("Letter already exists!")

return parse

elif code == WORD\_FAIL or code == WORD\_WIN: #word fail

if lst.\_\_len\_\_() == 2:

if code == WORD\_FAIL:

parse[WORD\_FAIL] = lst

logger.debug("Word fail!")

else:

parse[WORD\_WIN] = lst

logger.debug("Word win!")

cli.send("", QUERY\_USERWORD)

logger.debug("Username: %s, Word: %s" % (lst[0], lst[1]))

else:

parse[WORD\_FAIL] = None

parse[WORD\_WIN] = None

logger.debug("WORD\_FAIL\_OR\_WIN Undefined params! [%s] Lst: %s" % (code, lst))

return parse

elif code == CONN\_CLOSE\_CLI: #client close

if lst.\_\_len\_\_() == 1:

parse[CONN\_CLOSE\_CLI] = lst

logger.debug("%s has been disconnected" % lst[0])

else:

parse[CONN\_CLOSE\_CLI] = None

logger.debug("CONN\_CLOSE\_CLI Undefined params! [%s] Lst: %s" % (code, lst))

return parse

elif code == CONN\_CLOSE\_SERV: #server close

cli.connected = False

parse[CONN\_CLOSE\_SERV] = lst

logger.debug("Server close connection") #lst[0] -- isAlternativeServer

return parse

elif code == CONN\_CLOSE\_KICK: #server close

cli.connected = False

logger.warning("You has been kicked!")

parse[CONN\_CLOSE\_KICK] = True

else:

logger.debug("Error code: param: %s" % (code, param))

parse[code] = None

return parse

class Client():

parsedanswer = []

def connect(self, main = True):

self.main = main

self.connected = False

self.disconnected = False

self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

try:

if self.main:

self.sock.connect((CLI\_MAIN\_HOST, CLI\_MAIN\_PORT))

else:

sleep(3)

logger.info("Connecting to alternative server...")

self.sock.connect((CLI\_ALT\_HOST, CLI\_ALT\_PORT))

except socket.error:

if self.main:

try:

sleep(3)

logger.info("Connecting to alternative server...")

self.sock.connect((CLI\_ALT\_HOST, CLI\_ALT\_PORT))

except socket.error:

logger.critical("Connecting to alternative server failed...")

logger.critical("Servers don't worked!")

sleep(5)

sys.exit(0)

else:

logger.critical("Servers don't worked!")

try:

self.sock.send(QUERY\_CONN)

self.query\_result = self.sock.recv(4)

if (self.query\_result == CONN\_ALLOW):

logger.info("Connect allowed!")

self.connected = True

else:

logger.error(self.query\_result)

except socket.error, detail:

logger.error(detail)

if self.main:

cli.connect(False)

else:

logger.critical("Alternative server down!")

sys.exit(0)

class Listen(Thread):

def \_\_init\_\_(self):

Thread.\_\_init\_\_(self)

def run(self):

while True:

try:

data = cli.sock.recv(128)

except socket.error, detail:

logger.error(detail)

cli.sock.close()

cli.connected = False

if cli.main:

cli.connect(False)

else:

logger.critical("Alternative server down!")

sys.exit(0)

break

if not data:

if cli.main:

cli.connect(False)

else:

logger.critical("Alternative server down!")

sys.exit(0)

break

else:

answer = data.strip()

answer = answer.split("@")

for item in answer:

if len(item) > 0:

if item[0] == "#":

code = item[:4]

if code != CONN\_ALLOW:

if len(item) > 4:

param = item[5:]

cli.parsedanswer.append(answerparse(code, param))

else:

cli.parsedanswer.append(answerparse(code))

listen = Listen()

listen.start()

def send(self, msg = "", code = PACKET\_LETTER):

if self.connected == False:

logger.error("You are not connected")

else:

try:

if code == PACKET\_LETTER:

msg = msg.lower()

if re.match("^[a-z]\*$", msg):

self.sock.send(code + "\_" + str(msg))

else:

logger.error("Entered char not letter!")

except socket.error, detail:

print detail

try:

if self.main:

try:

sleep(10)

self.sock.close()

logger.info("Connecting to alternative server...")

cli.connect(False)

except:

cli.disconnect()

else:

cli.disconnect()

except NameError, detail:

print detail

def disconnect(self):

global getout

try:

self.sock.send("#500")

except error, detail:

logger.error(detail)

self.sock.close()

sleep(1)

logger.info("Client closed the connection\n")

logger.info("Client closed")

cli.connected = False

cli.disconnected = True

sys.exit(0)

cli = Client()

if \_\_name\_\_ == '\_\_main\_\_':

cli.connect()

**constants.py**

# -\*- coding: utf-8 -\*-

"""

@module: constants for server protocol gallows

@license: GNU GPL v2

@author: Egorov Ilya

@version: 0.7

"""

global QUERY\_CONN, CONN\_ALLOW, CONN\_DENY, \

PACKET\_SIZE, PACKET\_LETTER, PACKET\_ALLOW, PACKET\_USERWORD, \

LETTER\_FAIL, LETTER\_WIN, WORD\_FAIL, WORD\_WIN, \

CONN\_CLOSE\_CLI, CONN\_CLISE\_SERV

CLI\_MAIN\_HOST, CLI\_MAIN\_PORT = "localhost", 14879

CLI\_ALT\_HOST, CLI\_ALT\_PORT = "localhost", 14880

SERV\_MAIN\_HOST, SERV\_MAIN\_PORT = (CLI\_MAIN\_HOST, CLI\_MAIN\_PORT)

SERV\_ALT\_HOST, SERV\_ALT\_PORT = (CLI\_ALT\_HOST, CLI\_ALT\_PORT)

HOST\_PONG, PORT\_PONG = (SERV\_MAIN\_HOST, 14881)

HOST\_PING, PORT\_PING = (SERV\_ALT\_HOST, 14881)

QUERY\_CONN = "#100"

CONN\_ALLOW = "#111"

CONN\_DENY = "#110"

PACKET\_SIZE = "#200"

PACKET\_LETTER = "#201"

PACKET\_ALLOW = "#211"

QUERY\_USERWORD = "#203"

PACKET\_USERWORD = "#213"

QUERY\_USERCOUNT = "#204"

ANSWER\_USERCOUNT = "#214"

QUERY\_USEDLETTERS = "#205"

ANSWER\_USEDLETTERS = "#215"

LETTER\_FAIL = "#310"

LETTER\_WIN = "#311"

LETTER\_ALREADY = "#320"

WORD\_FAIL = "#410"

WORD\_WIN = "#411"

CONN\_CLOSE\_CLI = "#500"

CONN\_CLOSE\_SERV = "#510"

CONN\_CLOSE\_KICK = "#511"

CONN\_PING = "#698"

CONN\_PONG = "#699"

SYNC\_SERVER\_PACKET = "#720"

SYNC\_SERVER\_PACKET\_APPLY = "#721"

**gallows\_logic.py**

# -\*- coding: utf-8 -\*-

"""

@module: game-logic module for gallows

@license: GNU GPL v2

@author: Egorov Ilya

@version: 0.7

"""

from string import \*

from threading import \*

from random import randrange

import string, threading, time, logging

global HOST, PORT, LOG, usersword, ATTEMPT\_MAX, USERNAME, ALT\_SERVER

ALT\_SERVER = False

HOST, PORT = "localhost", 14880

ATTEMPT\_MAX = 10

USERNAME = "Prisoner"

logger = logging.getLogger("gallows")

logger.setLevel(logging.DEBUG)

logstream = logging.StreamHandler()

logstream.setLevel(logging.DEBUG)

formatter = logging.Formatter("%(asctime)s: %(message)s")

logstream.setFormatter(formatter)

logger.addHandler(logstream)

class Gallows:

def \_\_init\_\_(self):

self.attempts = ATTEMPT\_MAX

self.generated = False

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

@return: слово, сгенерированное из словаря

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

def generate(self):

wordsfile = open('words.txt', 'r')

words = wordsfile.readlines()

length = len(words)

wordindex = randrange(0, length)

wordsfile.close()

self.secret = strip(words[wordindex])

self.used\_letters = []

self.generated = True

logger.debug("Generated TRUE")

self.newuword = "\*" \* len(self.secret)

return self.secret

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

@param:

uword: слово, видимое игрокам

letter: буква, предложенная пользователем

@return:

guessed: количество верно угаданных букв (если -1, то угадано слово)

newuword: слово, видимое игрокам, с учетом параметра letter

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

def getletter(self, uword, letter):

if (uword.count(letter) > 0) or (self.used\_letters.count(letter) > 0):

logger.info("Letter already opened")

guessed = -2

self.newuword = uword

else:

self.used\_letters.append(letter)

iter1 = True

guessed = self.secret.count(letter)

tmp2 = uword

for x in range(len(self.secret)):

if (self.secret.find(letter) != -1):

pos = self.secret.find(letter)

tmp = self.secret

logger.debug("Position:" + str(pos))

self.secret = tmp[:pos] + "@" + tmp[pos + 1:]

if (iter1):

tmp2 = uword[:pos] + letter + uword[pos + 1:]

else:

tmp2 = tmp2[:pos] + letter + tmp2[pos + 1:]

iter1 = False

logger.info("Letter %s; Attempts: %d" % (letter, self. attempts))

if tmp2.count("\*") == 0:

guessed = -1

self.newuword = tmp2

logger.info("New user word: %s" % self.newuword)

return [guessed, self.newuword]

**GUI.py**

from client import \*

import time

import threading

import constants

from Tkinter import \*

#actions&log

class thread1(threading.Thread):

def run(self):

while 1:

while len(cli.parsedanswer):

pop=cli.parsedanswer.pop()

#if CONN\_ALLOW:

# text1.insert(END,"Connection allowed"+'\n')

#if CONN\_CLOSE\_KICK:

#text1.insert(END,"You were kicked!"+'\n')

for x in pop.keys():

if CONN\_ALLOW:

text1.insert(END,"Connection allowed"+'\n')

if x==CONN\_DENY:

text1.insert(END,"Connection deny"+'\n')

if x==PACKET\_USERWORD:

for items in pop[x]:

v.set(pop[PACKET\_USERWORD][0])

text1.insert(END,pop[PACKET\_USERWORD][1]+'tries left'+'\n')

if pop[PACKET\_USERWORD][1]=='9':

circle = canvas.create\_oval(25, 25, 50, 50, fill="white")

if pop[PACKET\_USERWORD][1]=='8':

canvas.create\_line(37, 50, 37, 69)

if pop[PACKET\_USERWORD][1]=='7':

canvas.create\_line(37, 69, 29, 78)

if pop[PACKET\_USERWORD][1]=='6':

canvas.create\_line(37, 69, 45, 78)

if pop[PACKET\_USERWORD][1]=='5':

canvas.create\_line(37, 53, 26, 60)

if pop[PACKET\_USERWORD][1]=='4':

canvas.create\_line(37, 53, 48, 60)

if pop[PACKET\_USERWORD][1]=='3':

canvas.create\_line(37, 53, 42, 49) #rope

canvas.create\_line(49, 36, 79, 8) #rope

if pop[PACKET\_USERWORD][1]=='2':

canvas.create\_line(79, 8, 79, 87)

if pop[PACKET\_USERWORD][1]=='1':

canvas.create\_line(79, 87, 63, 100)

canvas.create\_line(79, 87, 89, 100)

if x==LETTER\_FAIL:

text1.insert(END,"There is no such letter :("+'\n')

if x==LETTER\_WIN:

text1.insert(END,"BINGO!"+'\n')

if x==LETTER\_ALREADY:

text1.insert(END,"This letter already showed!"+'\n')

if x==WORD\_FAIL:

text1.insert(END,"Game over, try another time!"+'\n')

if x==WORD\_WIN:

text1.insert(END,"Congratulations, you win!"+'\n')

if x==CONN\_CLOSE\_SERV:

text1.insert(END,"Server has been closed :("+'\n')

thread1().start()

import Tkinter as tk

root = Tk()

# Connection options

MAIN\_HOST=StringVar()

MAIN\_PORT=StringVar()

SECOND\_HOST=StringVar()

SECOND\_PORT=StringVar()

#window scale

opt= Toplevel(root,bd=5,bg="grey")

opt.title("Connection")

opt.geometry("250x200+200+300")

#main server

IP\_lab=Label(opt,text="ServerIP:", bg="red")

IP\_ent=Entry(opt,textvariable=MAIN\_HOST,width="15")

PORT\_lab=Label(opt,text="ServerPort:",bg="green")

PORT\_ent=Entry(opt,textvariable=MAIN\_PORT,width="15")

#alt server

IP\_lab1=Label(opt,text="ServerIP2:", bg="red")

IP\_ent1=Entry(opt,textvariable=SECOND\_HOST,width="15")

PORT\_lab1=Label(opt,text="ServerPort2:",bg="green")

PORT\_ent1=Entry(opt,textvariable=SECOND\_PORT,width="15")

#scales

IP\_lab.grid(row=0,sticky=W)

IP\_ent.grid(row=0,column=1,padx=1)

PORT\_lab.grid(row=1,sticky=W)

PORT\_ent.grid(row=1,column=1,padx=1)

IP\_lab1.grid(columnspan=1,rowspan=2,sticky=W)

IP\_ent1.grid(column=1,row=3,padx=1,sticky=W)

PORT\_lab1.grid(rowspan=2,sticky=W)

PORT\_ent1.grid(row=4,column=1,padx=1,sticky=W)

#-----------------------

def OptGet():

CLI\_MAIN\_HOST=IP\_ent.get().strip()

CLI\_MAIN\_PORT=PORT\_ent.get().strip()

CLI\_ALT\_HOST=IP\_ent1.get().strip()

CLI\_ALT\_PORT=PORT\_ent1.get().strip()

text1.insert(END,"Changes applyed"+'\n')

but1=Button(opt,text="SET",command=OptGet)

but1.grid(row=0, column=2, columnspan=2, rowspan=2,

sticky=W+E+N+S, padx=5, pady=5)

Label(text="GALLOWS GAME!", bg="red").pack()

separator = Frame(height=2, bd=1, relief=SUNKEN)

separator.pack(fill=X, padx=5, pady=5)

#main game window

v=StringVar()

lab=Label(root,textvariable=v,wraplength="0", anchor=CENTER, justify=CENTER).pack()

text = Text()

text1=Text()

e = Entry(root,justify=CENTER,width=5)

e.pack()

e.focus\_set()

def callback():

cli.send(e.get().strip())

a=e.get()

e.delete(0, END)

b = Button(root, text="Send", width=10, command=callback).pack()

frame1 = Frame(root,width=500, height=50, bg="yellow")

Q=Button(frame1, text="Q",width=3,command=lambda: e.insert(0,"q")).pack(side=LEFT)

W=Button(frame1, text="W",width=3,command=lambda: e.insert(0,"w")).pack(side=LEFT)

E=Button(frame1, text="E",width=3,command=lambda: e.insert(0,"e")).pack(side=LEFT)

R=Button(frame1, text="R",width=3,command=lambda: e.insert(0,"r")).pack(side=LEFT)

T=Button(frame1, text="T",width=3,command=lambda: e.insert(0,"t")).pack(side=LEFT)

Y=Button(frame1, text="Y",width=3,command=lambda: e.insert(0,"y")).pack(side=LEFT)

U=Button(frame1, text="U",width=3,command=lambda: e.insert(0,"u")).pack(side=LEFT)

I=Button(frame1, text="I",width=3,command=lambda: e.insert(0,"i")).pack(side=LEFT)

O=Button(frame1, text="O",width=3,command=lambda: e.insert(0,"o")).pack(side=LEFT)

P=Button(frame1, text="P",width=3,command=lambda: e.insert(0,"p")).pack(side=LEFT)

frame1.pack()

frame2 = Frame(root,width=500, height=20, bg="green", colormap="new")

A=Button(frame2, text="A",width=3,command=lambda: e.insert(0,"a")).pack(side=LEFT)

S=Button(frame2, text="S",width=3,command=lambda: e.insert(0,"s")).pack(side=LEFT)

D=Button(frame2, text="D",width=3,command=lambda: e.insert(0,"d")).pack(side=LEFT)

F=Button(frame2, text="F",width=3,command=lambda: e.insert(0,"f")).pack(side=LEFT)

G=Button(frame2, text="G",width=3,command=lambda: e.insert(0,"g")).pack(side=LEFT)

H=Button(frame2, text="H",width=3,command=lambda: e.insert(0,"h")).pack(side=LEFT)

J=Button(frame2, text="J",width=3,command=lambda: e.insert(0,"j")).pack(side=LEFT)

K=Button(frame2, text="K",width=3,command=lambda: e.insert(0,"k")).pack(side=LEFT)

L=Button(frame2, text="L",width=3,command=lambda: e.insert(0,"l")).pack(side=LEFT)

frame2.pack()

frame3 = Frame(root,width=500, height=20, bg="white", colormap="new")

Z=Button(frame3, text="Z",width=3,command=lambda: e.insert(0,"z")).pack(side=LEFT)

X=Button(frame3, text="X",width=3,command=lambda: e.insert(0,"x")).pack(side=LEFT)

C=Button(frame3, text="C",width=3,command=lambda: e.insert(0,"c")).pack(side=LEFT)

V=Button(frame3, text="V",width=3,command=lambda: e.insert(0,"v")).pack(side=LEFT)

B=Button(frame3, text="B",width=3,command=lambda: e.insert(0,"b")).pack(side=LEFT)

N=Button(frame3, text="N",width=3,command=lambda: e.insert(0,"n")).pack(side=LEFT)

M=Button(frame3, text="M",width=3,command=lambda: e.insert(0,"m")).pack(side=LEFT)

e.delete(0,END)

frame3.pack()

#------picture-----

canvas = Canvas(root, width=200, height=100)

"""circle = canvas.create\_oval(25, 25, 50, 50, fill="white")

canvas.create\_line(37, 50, 37, 69)

canvas.create\_line(37, 69, 29, 78)

canvas.create\_line(37, 69, 45, 78)

canvas.create\_line(37, 53, 26, 60)

canvas.create\_line(37, 53, 48, 60)

canvas.create\_line(37, 53, 42, 49) #rope

canvas.create\_line(49, 36, 79, 8) #rope

canvas.create\_line(79, 8, 79, 87)

canvas.create\_line(79, 87, 63, 100)

canvas.create\_line(79, 87, 89, 100)"""

canvas.pack()

#-----log-----

log= Toplevel(root,bd=5,bg="grey")

log.title("Info log")

log.geometry("200x200+450+300")

scrollbar = tk.Scrollbar(log)

logging=StringVar()

text1 = tk.Text(log, yscrollcommand=scrollbar.set)

text1.insert(END, "-------LOG-------"+'\n')

scrollbar.config(command=text1.yview)

scrollbar.pack(side='right', fill='y')

text1.pack(side='left', expand=0, fill='both')

root.mainloop()

**protocol\_server.py**

# -\*- coding: utf-8 -\*-

"""

@module: server protocol module for gallows

@license: GNU GPL v2

@author: Egorov Ilya

@version: 0.7

"""

from socket import \*

from threading import \*

from select import select

from constants import \*

from gallows\_logic import \*

from time import sleep

from optparse import OptionParser

from sys import exit

import socket, threading, select, logging, gallows\_logic, re, sys

global HOST, PORT, LOG, usersword, ATTEMPT\_MAX, USERNAME, main\_server, pong

ATTEMPT\_MAX = 10

USERNAME = "Prisoner"

logger = logging.getLogger("server\_protocol")

logger.setLevel(logging.DEBUG)

logstream = logging.StreamHandler()

logstream.setLevel(logging.DEBUG)

formatter = logging.Formatter("%(asctime)s: %(message)s")

logstream.setFormatter(formatter)

logger.addHandler(logstream)

def sendmsg(msg, adr):

for sock in users.keys():

sock.send(msg)

def clean():

userscount = 0

users = {}

sockets = [""]

gallows.attempts = ATTEMPT\_MAX

class Pinger(Thread):

def \_\_init\_\_(self):

Thread.\_\_init\_\_(self)

self.packets = None

self.lastsync = None

def parsesync(self, packets):

packets = packets[0].split("\_")

if (len(packets) == 5) and (packets[0] == SYNC\_SERVER\_PACKET):

self.secret = packets[1]

self.attempts = int(packets[2])

self.userword = packets[3]

self.used\_letters = list(packets[4])

start()

def run(self):

try:

self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.sock.connect((HOST\_PONG, PORT\_PONG))

except:

logger.critical("Main server don't work! %s %s" % (HOST\_PING, PORT\_PING))

sys.exit(0)

while True:

try:

self.sock.send(CONN\_PING + "@")

data = self.sock.recv(128)

except socket.error, detail:

logging.error(detail)

logger.error("Ping server error! %s" % self.packets)

if self.lastsync:

self.parsesync(self.lastsync)

else:

logger.error("Main server not send sync packet! :(")

sleep(5)

sys.exit(0)

break

if not data: logger.error("Ping server error! %s" % self.sock.fileno())

else:

try:

self.packets = data.strip()

self.packets = self.packets.split("$")

for pack in self.packets:

answers = pack.strip()

answers = pack.split("\_")

for item in answers:

if len(item) > 0:

if item[0] == "#":

code = item[:4]

if code == CONN\_PONG:

logger.info("Ping server success! %s" % self.sock.fileno())

self.sock.send(CONN\_PING + "$")

elif code == SYNC\_SERVER\_PACKET:

print "SYNC: " + str(answers)

self.lastsync = self.packets

self.sock.send(SYNC\_SERVER\_PACKET\_APPLY + "$")

else:

logger.error("Ping server error! CODE: '%s'" % code)

self.parsesync(self.lastsync)

except socket.error:

logger.error("Ping server error! %s" % self.packets)

self.parsesync(self.lastsync)

class Ponger(Thread):

def \_\_init\_\_(self):

Thread.\_\_init\_\_(self)

def run(self):

try:

pong = socket.socket(AF\_INET, SOCK\_STREAM)

pong.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

pong.bind((HOST\_PONG, PORT\_PONG))

logger.debug("Ponger server binded")

pong.listen(5)

logger.debug("Ponger server listen on port " + str(PORT\_PONG))

sleep(15)

except socket.error, detail:

logger.error(detail)

pingsock, addr = pong.accept()

logger.info("Ponger connected %s" % str(addr))

try:

while True:

data = pingsock.recv(32)

ping = data.strip()

ping = ping.split("$")

if not data:

logger.error("Pong server error! %s" % pingsock.fileno())

sleep(5)

pingsock, addr = pong.accept()

pingsock.send(SYNC\_SERVER\_PACKET + "\_%s\_%s\_%s\_%s$" % (gallows.secret, str(gallows.attempts), gallows.newuword, str(gallows.used\_letters)))

else:

for item in ping:

if len(ping) > 0:

if len(item) > 0:

if item[0] == "#":

code = item[:4]

if code == CONN\_PING:

if (hasattr(gallows, 'secret') and not main\_server) or (s.changed and main\_server):

pingsock.send(SYNC\_SERVER\_PACKET + "\_%s\_%s\_%s\_%s$" % (gallows.secret, str(gallows.attempts), gallows.newuword, str(gallows.used\_letters)))

s.changed = False

logger.info("Send: " + SYNC\_SERVER\_PACKET + "\_%s\_%s\_%s\_%s$" % (gallows.secret, str(gallows.attempts), gallows.newuword, str(gallows.used\_letters)))

sleep(2)

else:

pingsock.send(CONN\_PONG + "$")

sleep(2)

break

elif code == SYNC\_SERVER\_PACKET\_APPLY:

logger.info("SYNC OK! %s" % pingsock.fileno())

elif item[0] != "#":

break

else:

logger.info("Pong server unknown answer code! CODE: %s" % code)

except socket.error, detail:

logger.error("Pong server error! %s" % pingsock.fileno())

class Server:

# listen for connections

def \_\_init\_\_(self):

self.changed = False

class Listen\_for\_connections(Thread):

def \_\_init\_\_(self):

Thread.\_\_init\_\_(self)

def run(self):

global users, server, rl, sockets, userscount, gallows, queue\_start, usersword, letter

gallows = Gallows()

queue\_start = []

try:

server = socket.socket(AF\_INET, SOCK\_STREAM)

server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

server.bind((HOST, PORT))

logger.debug("Server binded")

server.listen(1)

logger.debug("Server listen")

except socket.error, detail:

logger.error(detail)

USERNAME = "Prisoner"

userscount = 0

users = {}

sockets = [""]

restart = False

result = []

guessed = 0

fail = {}

kick = False

while True:

if sockets[-1] == "CLOSED":

break

sockets = [server.fileno()]

for sock in users.keys():

sockets.append(sock.fileno())

try:

rl, wl, el = select.select(sockets, [], [], 3)

except select.error, detail:

logger.error(detail)

new = None

break

for n in rl:

if n == server.fileno():

sock, addr = server.accept()

logger.info("New user #" + str(n))

users[sock] = USERNAME + str(userscount)

userscount += 1

new = sock

else:

for sock in users.keys():

if sock.fileno() == n: break

name = users[sock]

try:

text = sock.recv(128)

parse = text.split("@")

except socket.error, detail:

userscount -= 1

logger.info(name + " has been disconnected!")

del users[sock]

new = None

sendmsg(ANSWER\_USERCOUNT + "\_%s@" % (userscount), sock)

break

if not text:

logger.info(name + " has been disconnected!")

new = None

sleep(1)

userscount -= 1

sock.close()

del users[sock]

sendmsg(ANSWER\_USERCOUNT + "\_%s@" % (userscount), sock)

else:

try:

for item in parse:

if item[0] == "#":

if text[0:4] == QUERY\_CONN:

sock.send(CONN\_ALLOW + "@")

queue\_start.append(sock)

if main\_server:

word = gallows.generate()

usersword = "\*" \* len(word)

s.changed = True

logger.info("\nSecret word generated! [%s]. \nFor users: %s\n" % (word, usersword))

else:

gallows.secret = pinger.secret

gallows.attempts = pinger.attempts

gallows.newuword = pinger.userword

usersword = pinger.userword

gallows.newuword = pinger.userword

gallows.used\_letters = pinger.used\_letters

s.changed = False

sendmsg(PACKET\_USERWORD + "\_%s\_%s@" % (usersword, gallows.attempts), sock)

sendmsg(ANSWER\_USERCOUNT + "\_%s@" % (userscount), sock)

break

lst = item.split("\_")

logger.debug(lst)

if lst[0] == QUERY\_USERCOUNT:

sendmsg(ANSWER\_USERCOUNT + "\_%s@" % (userscount), sock)

logger.info("Userscount is %s" % (userscount))

if lst[0] == PACKET\_LETTER:

if len(lst[1])== 1 and re.match("^[a-z]\*$", lst[1]):

letter = lst[1]

result = gallows.getletter(usersword, strip(letter))

logger.info("S: %s UW: %s Text: %s Letter: %s Result: %s" % (gallows.secret, usersword, text, letter, result))

usersword = result[1]

s.changed = True

if (gallows.attempts == 0):

sendmsg(WORD\_FAIL + "\_%s\_%s@" % (name, word), sock)

restart = True

else:

if (result[0] != 0):

if (gallows.attempts != 0):

if (result[0] > 0):

sendmsg(LETTER\_WIN + "\_%s\_%s\_%s\_%s@" % (name, letter, usersword, gallows.attempts), sock)

s.changed = True

if (result[0] < 0) or (gallows.attempts == 0):

if (result[0] == -1):

sendmsg(WORD\_WIN + "\_%s\_%s@" % (name, usersword), sock)

s.changed = True

restart = True

if (result[0] == -2):

sendmsg(LETTER\_ALREADY + "\_%s@" % letter, sock)

else:

sendmsg(LETTER\_FAIL + "\_%s\_%s\_%s\_%s@" % (name, letter, usersword, gallows.attempts), sock)

gallows.attempts -= 1

s.changed = True

for sock in queue\_start:

#sendmsg(PACKET\_USERWORD + "\_%s\_%s@" % (usersword, gallows.attempts), sock)

pass

break

if lst[0] == QUERY\_USERWORD:

sendmsg(PACKET\_USERWORD + "\_%s\_%s@" % (usersword, gallows.attempts), sock)

restart = True

logger.info("Userscount is %s" % (userscount))

if lst[0] == CONN\_CLOSE\_CLI:

sendmsg("Good bye!", sock)

logger.info(name + " has been disconnected!")

userscount -= 1

sock.close()

new = None

del users[sock]

sendmsg("%s has been disconnected!" % (name), sock)

else:

kick = True

else:

kick = True

except socket.error, detail:

logger.error(detail)

new = None

break

logger.debug(str(name) + ": " + text)

if (kick):

logger.info("Client %s kicked!" % name)

sock.send(CONN\_CLOSE\_KICK)

userscount -= 1

del users[sock]

new = None

sendmsg(ANSWER\_USERCOUNT + "\_%s@" % (userscount), sock)

sock.close()

if (new == sock):

sock.send(CONN\_ALLOW)

if (restart):

clean()

word = gallows.generate()

usersword = "\*" \* len(word)

logger.info("\nSecret word generated! [%s]. \nFor users: %s\n" % (word, usersword))

sendmsg(PACKET\_USERWORD + "\_%s\_%s@" % (usersword, gallows.attempts), sock)

s.changed = True

restart = False

lc = Listen\_for\_connections()

def disconnect(self):

global getout

for sock in users.keys():

try:

sock.send("#510")

except error, detail:

logger.error(detail)

sock.close()

del users[sock]

sleep(1)

logger.info("Server closed the connection\n")

server.close()

logger.info("Server closed")

sockets.append("CLOSED")

parser = OptionParser()

parser.add\_option("-t", "--type", dest="type", help="--- SERVER TYPE --- ""Main server: -t m""Alternative server: -t a")

(options, args) = parser.parse\_args()

s = Server()

def start():

s.lc.start()

print "Started..."

if (options.type == "a"):

HOST, PORT = SERV\_ALT\_HOST, SERV\_ALT\_PORT

pinger = Pinger()

pinger.start()

main\_server = False

elif (options.type == "m"):

HOST, PORT = SERV\_MAIN\_HOST, SERV\_MAIN\_PORT

ponger = Ponger()

ponger.start()

main\_server = True

start()

else: sys.exit(0)

**words.txt**

cow

transport

fragile

definition

destination

critic

humanity

porcelain

comunity

doll

jeans

astronomy

monkey

christianity

damnation

biopsy

invasion

trinity

time

chicken