**SOCKET PROGRAMMING**

Laporan ini disusun untuk memenuhi tugas EL5226 – Jaringan Informasi dan Sosial tentang Socket Programming. Laporan ini beserta source-code dapat juga diunduh di github [https://github.com/soedomoto/ITB2015/tree/EL5226/Assignment%201%20-%20Socket%20Programming](https://github.com/soedomoto/ITB2015/tree/EL5226/Assignment 1 - Socket Programming).

**CLIENT-SERVER CHAT**

Client-Server chat merupakan sebuah program ringan berbasis python yang dapat digunakan untuk mengirim pesan text. Client-server chat mendukung koneksi multi-client. Program server, berisikan kode server socket, yang bertugas mem-broadcast pesan dari satu client ke seluruh client yang lain. Sementara, program client berisi kode yang dapat menampilkan pesan dari server dan dapat menerima user input.

chat\_server.py

#!/usr/bin/python

import socket, threading, sys, select

# Use argument to define port

args = sys.argv

if not len(args) == 2:

print 'Usage : python chat\_server.py <port>'

exit()

else:

# Define addr where socket server is listening

host = '0.0.0.0'

port = int(args[1])

sockets = []

recv\_buffer = 4096

# broadcast chat messages to all connected clients

def broadcast (sock\_server, sock, message):

for socket in sockets:

# send the message only to peer

if socket != sock\_server and socket != sock :

try :

socket.send(message)

except :

# broken socket connection

socket.close()

# broken socket, remove it

if socket in sockets:

sockets.remove(socket)

if \_\_name\_\_ == "\_\_main\_\_":

try:

# Create socket server

sock\_server = socket.socket()

sock\_server.bind((host, port))

sock\_server.listen(5)

print 'Chat server is listening at %s:%s' % (host, port)

sockets.append(sock\_server)

while True:

read\_sockets, write\_sockets, error\_sockets = select.select(sockets, [], [])

for sock in read\_sockets:

# a new connection request recieved

if sock == sock\_server:

new\_sock, addr = sock\_server.accept()

sockets.append(new\_sock)

print "[+] Client %s:%s connected" % addr

broadcast(sock\_server, new\_sock, "[%s:%s]\t: Entered our chatting room\n" % addr)

# a message from a client, not a new connection

else:

cip, cport = sock.getpeername()

# process data recieved from client,

try:

# receiving data from the socket.

data = sock.recv(recv\_buffer)

if data:

# there is something in the socket

broadcast(sock\_server, sock, "[%s:%s]\t: %s" % (cip, cport, data))

else:

# remove the socket that's broken

if sock in sockets:

sockets.remove(sock)

# at this stage, no data means probably the connection has been broken

broadcast(sock\_server, sock, "[%s:%s]\t: Now offline\n" % (cip, cport))

# exception

except:

broadcast(sock\_server, sock, "[%s:%s]\t: Now offline\n" % (cip, cport))

continue

except KeyboardInterrupt:

print "Chat server is stopping..."

sock\_server.close()

print "Chat server is stopped"

chat\_client.py

#!/usr/bin/python

import socket, threading, sys, select

recv\_buffer = 4096

def prompt():

sys.stdout.write('[Me]\t\t\t: ');

sys.stdout.flush()

if \_\_name\_\_ == "\_\_main\_\_":

# Use argument to define port

args = sys.argv

if not len(args) == 3:

print 'Usage : python chat\_client.py <host> <port>'

exit()

else:

# Define addr where socket server is listening

host = args[1]

port = int(args[2])

# Create socket client

sock = socket.socket()

try:

sock.connect((host, port))

except socket.error:

print 'Unable to connect'

exit()

print 'Connected to remote host. You can start sending messages'

prompt()

while 1:

socket\_list = [sys.stdin, sock]

# Get the list sockets which are readable

read\_sockets, write\_sockets, error\_sockets = select.select(socket\_list , [], [])

for s in read\_sockets:

if s == sock:

# incoming message from remote server, s

data = s.recv(recv\_buffer)

if not data :

print '\nDisconnected from chat server'

sys.exit()

else :

#print data

sys.stdout.write('\r' + data)

prompt()

else :

# user entered a message

msg = sys.stdin.readline()

sock.send(msg)

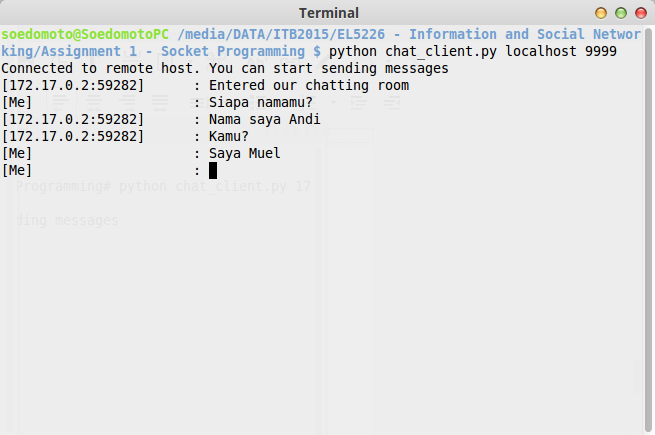
prompt()

**Screenshot**

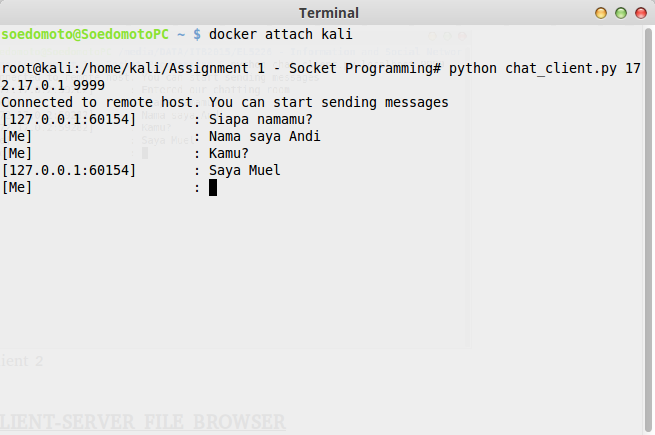
Server



Client 1



Client 2



**CLIENT-SERVER FILE BROWSER**

Client-Server file browser merupakan sebuah program ringan berbasis python yang dapat digunakan untuk mem-browser file/folder yang ada di server melalui client. Client-server file browser mendukung koneksi multi-client. Program server, berisikan kode server socket, yang memberi response kepada client dalam 2 rule : 1) Jika yang direquest oleh client adalah sebuah folder, maka server akan merensponse dengan mengirimkan list content dari folder tersebut, 2) Jika yang direquest oleh client adalah sebuah file, maka server akan meresponse dengan mengirimkan isi dari file tersebut. Sementara, program client berisi kode yang dapat menampilkan pesan dari server dan dapat menerima user input (dapat diketik).

file\_server.py

#!/usr/bin/python

import socket, threading, sys, select, os

# Use argument to define port

args = sys.argv

if not len(args) == 2:

print 'Usage : python file\_server.py <port>'

exit()

else:

# Define addr where socket server is listening

host = '0.0.0.0'

port = int(args[1])

sockets = []

last\_data = {}

recv\_buffer = 4096

def column(matrix, i):

return [row[i] for row in matrix]

def list\_file(sock, inpdir):

cip, cport = sock.getpeername()

data = []

data.append(['dir ', os.pardir])

for path, subdirs, files in os.walk(inpdir, topdown=True):

for name in subdirs:

data.append(['dir ', name])

for name in files:

data.append(['file', name])

break

last\_data[cip + ':' + str(cport)] = [inpdir, column(data, 1)]

counter = 1

str\_list = 'Please select folder/file below :\n'

for f in data:

str\_list = str\_list + "[{}] {}".format(counter, '{} {}\n'.format(f[0], f[1]))

counter = counter + 1

sock.send(str\_list)

if \_\_name\_\_ == "\_\_main\_\_":

try:

# Create socket server

sock\_server = socket.socket()

sock\_server.bind((host, port))

sock\_server.listen(5)

print 'File server is listening at %s:%s' % (host, port)

sockets.append(sock\_server)

while True:

read\_sockets, write\_sockets, error\_sockets = select.select(sockets, [], [])

for sock in read\_sockets:

# a new connection request recieved

if sock == sock\_server:

new\_sock, addr = sock\_server.accept()

sockets.append(new\_sock)

print "[+] Client %s:%s connected" % addr

list\_file(new\_sock, os.getcwd())

# a message from a client, not a new connection

else:

cip, cport = sock.getpeername()

# process data recieved from client,

try:

# receiving data from the socket.

data = sock.recv(recv\_buffer)

if data:

# convert to integer

try:

data = int(data)

except:

sock.send('Your selection must be an integer\n')

continue

# there is something in the socket

if cip + ':' + str(cport) in last\_data:

l\_data = last\_data[cip + ':' + str(cport)]

try:

f = os.path.join(l\_data[0], l\_data[1][data-1])

if os.path.isdir(f):

f = os.path.abspath(f)

list\_file(sock, f)

elif os.path.isfile(f):

with open(f, 'r') as handler:

content = handler.read()

content = '===============================================================================\n' + \

'Content of file : ' + l\_data[1][data-1] + '\n' + \

'-------------------------------------------------------------------------------\n' + \

content + '\n' + \

'-------------------------------------------------------------------------------\n'

sock.send(content)

list\_file(sock, l\_data[0])

except:

list\_file(sock, l\_data[0])

else:

# remove the socket that's broken

if sock in sockets:

sockets.remove(sock)

# exception

except:

continue

except KeyboardInterrupt:

print "File server is stopping..."

sock\_server.close()

print "File server is stopped"

file\_client.py

#!/usr/bin/python

import socket, threading, sys, select

recv\_buffer = 4096

def prompt():

sys.stdout.write('[Selection]\t: ');

sys.stdout.flush()

if \_\_name\_\_ == "\_\_main\_\_":

# Use argument to define port

args = sys.argv

if not len(args) == 3:

print 'Usage : python file\_client.py <host> <port>'

exit()

else:

# Define addr where socket server is listening

host = args[1]

port = int(args[2])

# Create socket client

sock = socket.socket()

try:

sock.connect((host, port))

except socket.error:

print 'Unable to connect'

exit()

print 'Connected to remote host. You can start sending messages'

prompt()

while 1:

socket\_list = [sys.stdin, sock]

# Get the list sockets which are readable

read\_sockets, write\_sockets, error\_sockets = select.select(socket\_list , [], [])

for s in read\_sockets:

if s == sock:

# incoming message from remote server, s

data = s.recv(recv\_buffer)

if not data :

print '\nDisconnected from chat server'

sys.exit()

else :

#print data

sys.stdout.write('\r' + data)

prompt()

else :

# user entered a message

msg = sys.stdin.readline()

sock.send(msg)

sys.stdout.write('\n')

prompt()

**Screenshot**

Server



Client

