University of Puerto Rico Mayaguez Campus

CIIC4030

Device Communications Project

Maria Cordero

Julio Aguilar

December 11, 2019

SukET Device Connector

I. Introduction

SukET is a language to simplify communication with devices by allowing the creation of both client/servers.
It uses two Python libraries to achieve this; socket and SLY.

Socket library

Socket programming library is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

The socket library was used to create the sockets responsible for allowing the communication in the clients and servers created—it establishes the socket in the respective connections and either listens or connects, according to its client or server nature.

SLY library

SLY is a library for writing parsers and compilers. It is loosely based on the traditional compiler construction tools lex and yacc and implements the same LALR(1) parsing algorithm. SLY provides two separate classes Lexer and Parser. The Lexer class is used to break input text into a collection of tokens specified by a collection of regular expression rules. The Parser class is used to recognize language syntax that has been specified in the form of a context free grammar. The two classes are typically used together to make a parser.

II. Execution

The Repo consists of 3 Python files, a test folder from previous practice with sockets and a ReadMe file. To execute this language run the python file called **parser.py**. It will initiate a loop taking input from the console. Typing in “exit” will close the program.

III. Tokens

#Rules for Tokens

The r in front of every variable token is regular expression to allow the parser to know until when to read for the tokens.

CREATE = r'create'	CLIENT = r'client'
SERVER = r'server'	CLOSE = r'close'
CONNECT = r'connect'	BIND = r'bind'
RECEIVE = r'receive'	LISTEN = r'listen'
SEND = r'send'	ACCEPT = r'accept'

IV. **Grammar**

statement : CREATE CLIENT	statement : SEND
statement : CREATE SERVER	statement : RECEIVE
statement : CLOSE CLIENT	statement : BIND
statement : CLOSE SERVER	statement : LISTEN
statement : CONNECT	statement : ACCEPT

Example of grammar functions in code:

```
#Create the Server
@_('CREATE SERVER')
def statement(self,p):
    self.sSocket = Server.create_socket(self)
    self.gCommands[p.SERVER] = self.sSocket
    return self.sSocket

@_('CONNECT')
def statement(self,p):
    Client.connect(self, self.cSocket)
```

V. **Detailed Statements**

- create client/server – creates a client/server space for user using sockets
- bind – associate socket with a specific port, used to bind to the address

- listen – enable a server the acceptation of a connection
- connect – connect with desired device
- client/server close – close the connection with client/server
- accept – to accept a connection with a server
- receive – receive the server message

VI. Example of code to run

1. create client
2. create server
3. bind (Input desired port)
4. listen
5. connect (Input desired IP address and Port, for local host use: 127.0.0.1)
6. accept
7. send (Type any message)
8. client close
9. server close