Design Document Report

Network Application Development Project

A Simple File Transfer Service

By

Alexandre Vallières	#40157223
Samson Kaller	#40136815

Course

COEN366: Communication Networks and Protocols

Presented to

Professor Chadi Assi

December 12th, 2022

Concordia University

1.0 INTRODUCTION

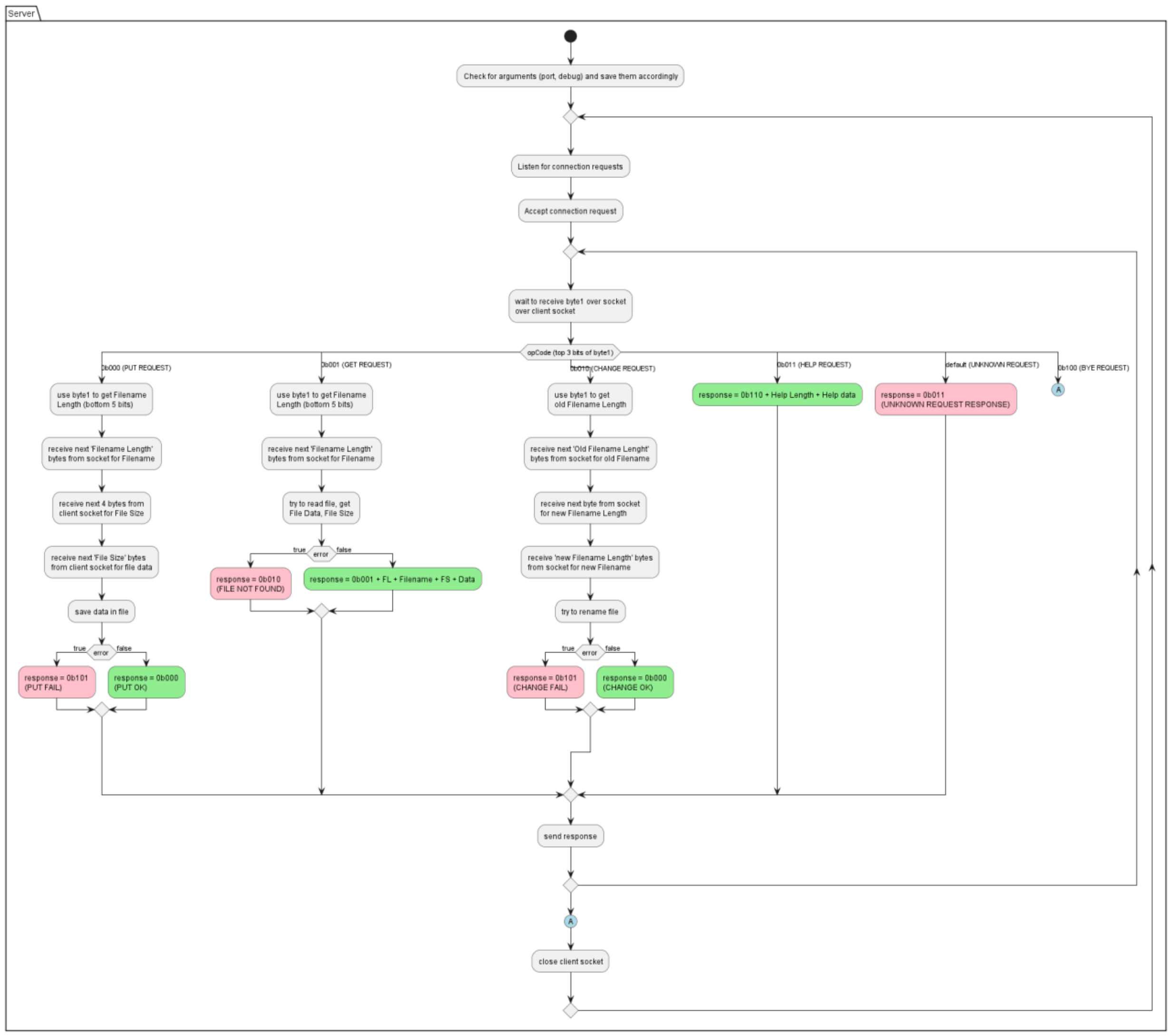
This report contains the design documentation for the server and client python applications developed for the project. These python scripts implement a simple file transfer service according to the protocol specifications in the Project Description. This report will first cover flowcharts that describe the algorithms behind each program, client and server, followed by descriptions of the functions used in each script.

2.0 FLOWCHARTS

This section contains flowcharts describing the algorithms for the Server and Client python scripts. First the Server will be covered followed by the Client. PlantUML was used to model and generate the flowcharts.

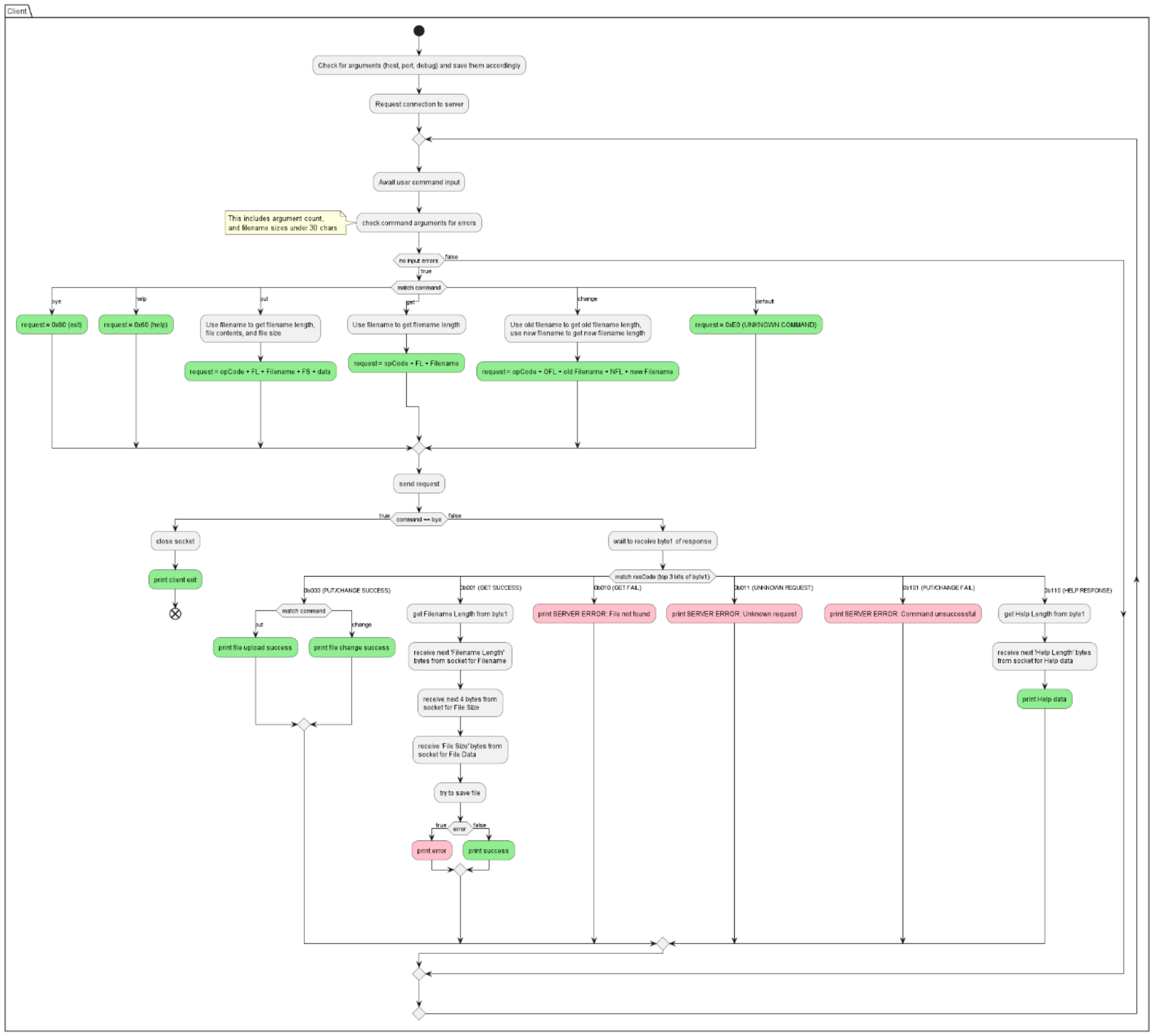
2.1 Server Algorithm

The server algorithm is very straightforward. In short: it listens for client connections, waits to receive a client request, then handles the request and sends a response back to the client. Functions have also been created to handle these responses which can be found further on in section 3.0 FUNCTIONS.


Server
Check for arguments (port, debug) and save them accordingly
Listen for connection requests
Accept connection request
wait to receive byte1 over socket over client socket
opCode (top 3 bits of byte1)
0b000 (PUT REQUEST)
use byte1 to get Filename Length (bottom 5 bits)
receive next 'Filename Length' bytes from socket for Filename
receive next 4 bytes from client socket for File Size
receive next 'File Size' bytes from client socket for file data
save data in file
true
error
false
response = 0b101 (PUT FAIL)
response = 0b000 (PUT OK)
0b001 (GET REQUEST)
use byte1 to get Filename Length (bottom 5 bits)
receive next 'Filename Length' bytes from socket for Filename
try to read file, get File Data, File Size
true
error
false
response = 0b010 (FILE NOT FOUND)
response = 0b001 + FL + Filename + FS + Data
0b010 (CHANGE REQUEST)
use byte1 to get old Filename Length
receive next 'Old Filename Lenght' bytes from socket for old Filename
receive next byte from socket for new Filename Length
receive 'new Filename Length' bytes from socket for new Filename
try to rename file
true
error
false
response = 0b101 (CHANGE FAIL)
response = 0b000 (CHANGE OK)
0b011 (HELP REQUEST)
response = 0b110 + Help Length + Help data
default (UNKNOWN REQUEST)
response = 0b011 (UNKNOWN REQUEST RESPONSE)
0b100 (BYE REQUEST)
A
send response
A
close client socket

2.2 Client Algorithm

The client algorithm is a little more complex. It connects to the server then waits for user to input commands. Then it sends the commands to the server and waits for a response. After receiving the response, it handles it appropriately. Functions have also been created to handle these requests and responses which can be found further on in section 3.0 FUNCTIONS.

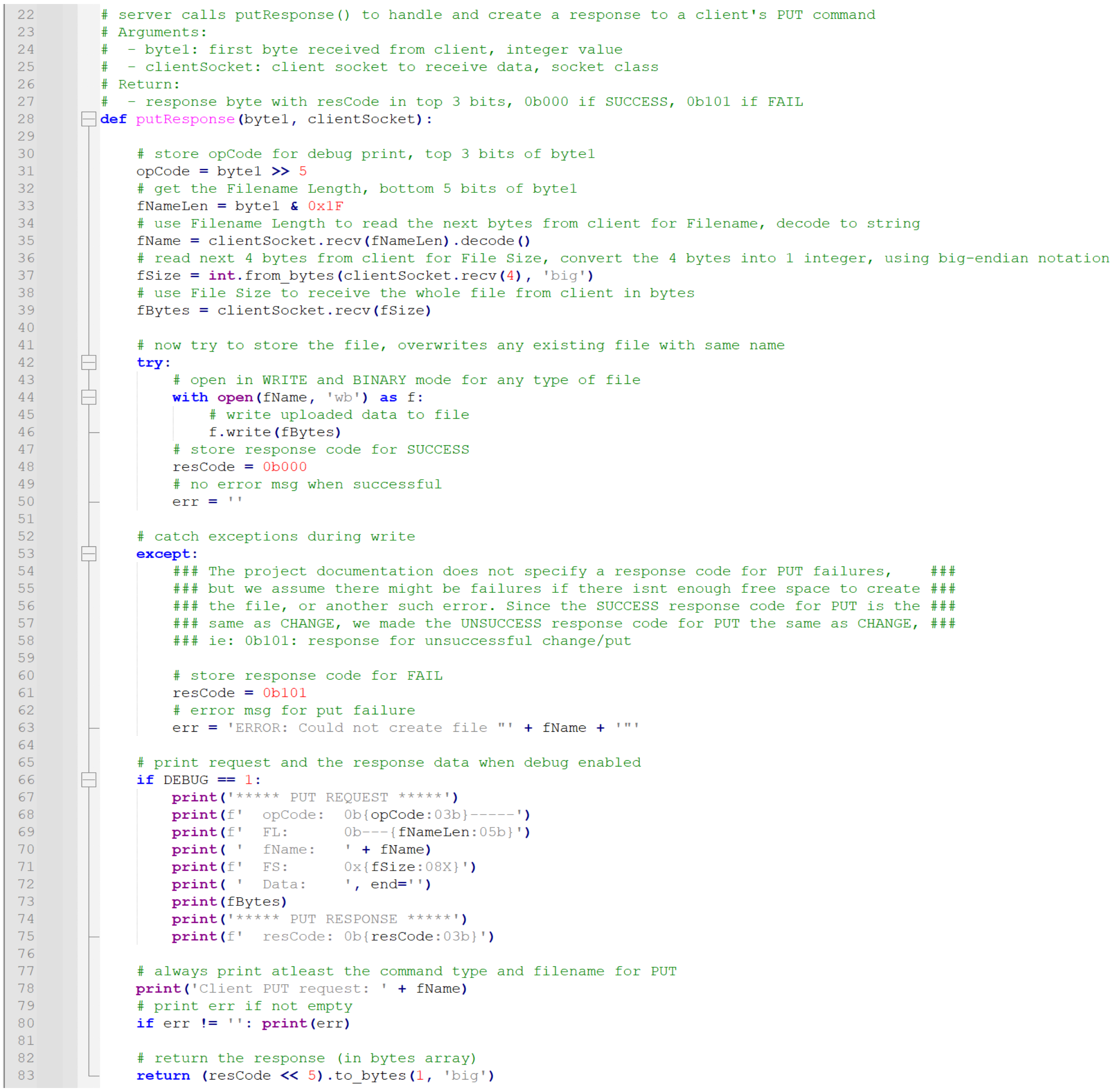


Client
Check for arguments (host, port, debug) and save them accordingly
Request connection to server
Await user command input
This includes argument count, and filename sizes under 30 chars
check command arguments for errors
no input errors
false
true
match command
bye
help
put
get
change
default
request = 0x80 (exit)
request = 0x60 (help)
Use filename to get filename length, file contents, and file size
request = opCode + FL + Filename + FS + data
Use filename to get filename length
request = opCode + FL + Filename
Use old filename to get old filename length, use new filename to get new filename length
request = opCode + OFL + old Filename + NFL + new Filename
request = 0xE0 (UNKNOWN COMMAND)
send request
true
command == bye
false
close socket
print client exit
wait to receive byte1 of response
match resCode (top 3 bits of byte1)
0b000 (PUT/CHANGE SUCCESS)
match command
put
change
print file upload success
print file change success
0b001 (GET SUCCESS)
get Filename Length from byte1
receive next 'Filename Length' bytes from socket for Filename
receive next 4 bytes from socket for File Size
receive 'File Size' bytes from socket for File Data
try to save file
true
error
false
print error
print success
0b010 (GET FAIL)
print SERVER ERROR: File not found
0b011 (UNKNOWN REQUEST)
print SERVER ERROR: Unknown request
0b101 (PUT/CHANGE FAIL)
print SERVER ERROR: Command unsuccessful
0b110 (HELP RESPONSE)
get Help Length from byte1
receive next 'Help Length' bytes from socket for Help data
print Help data

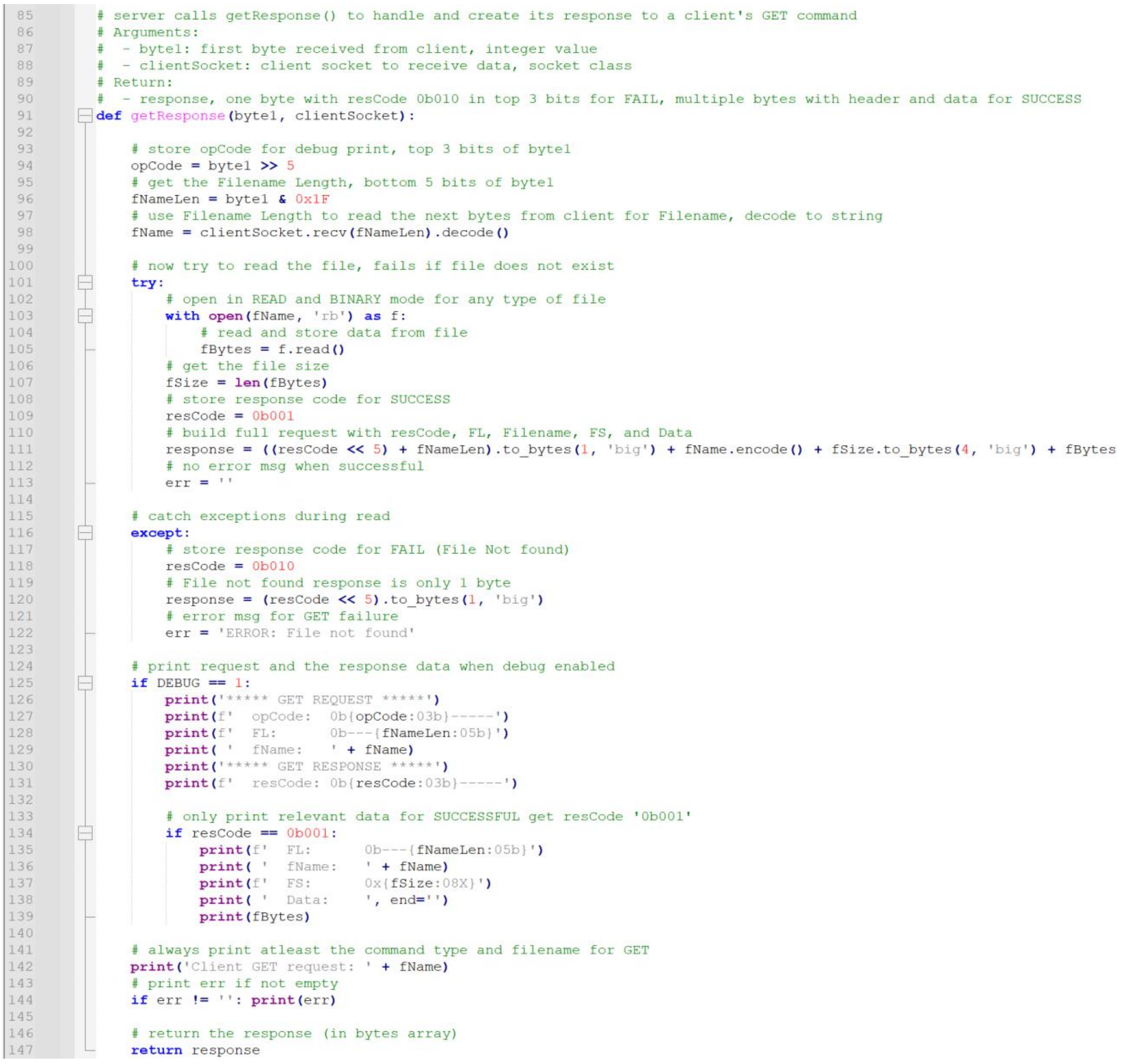
3.0 FUNCTIONS

This section contains descriptions of the functions developed for the server and client applications. First the server functions will be covered followed by the client functions. These descriptions are pretty much the function definitions from the application python scripts, but they have been included here for documentation purposes.

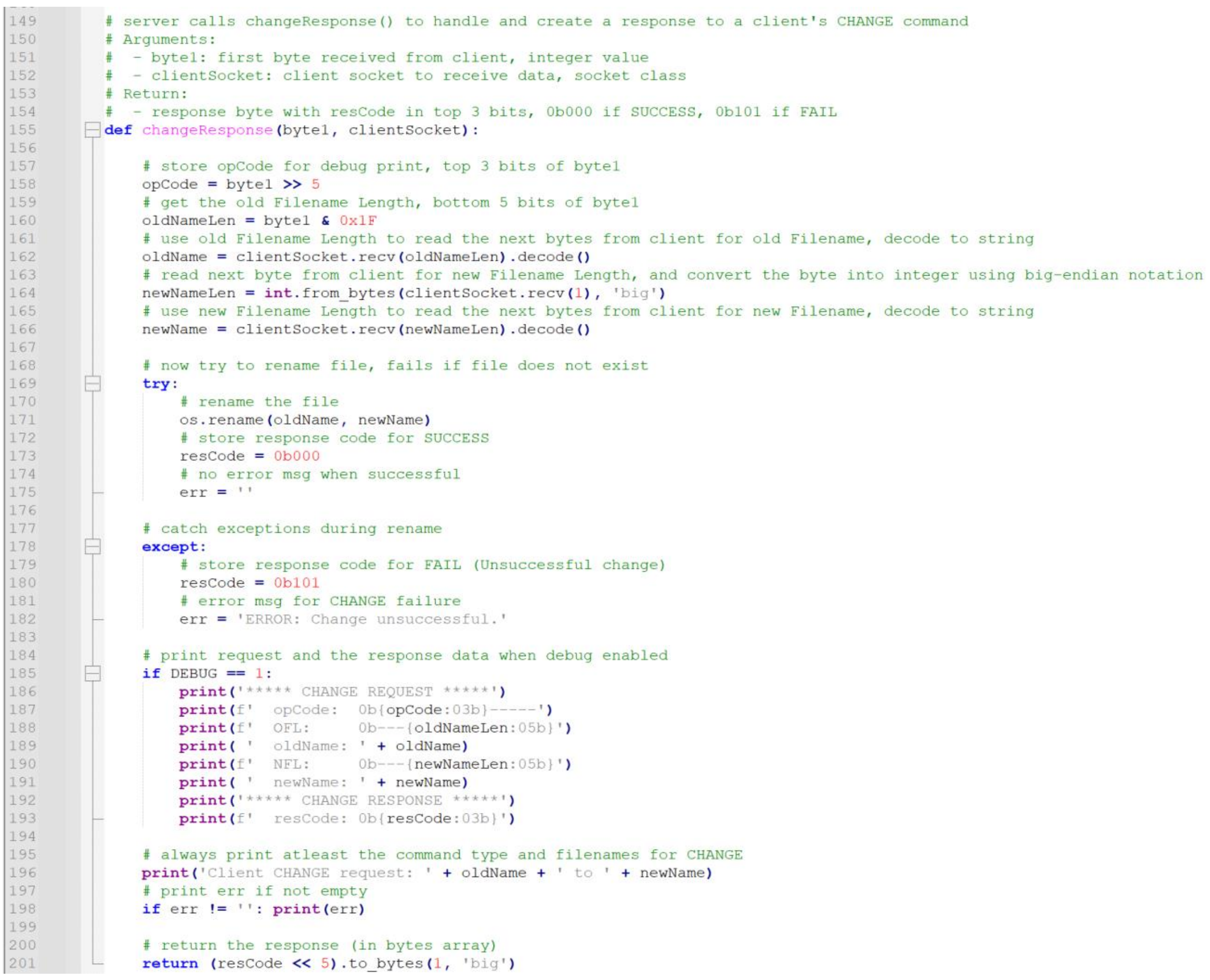
3.1 Server Functions



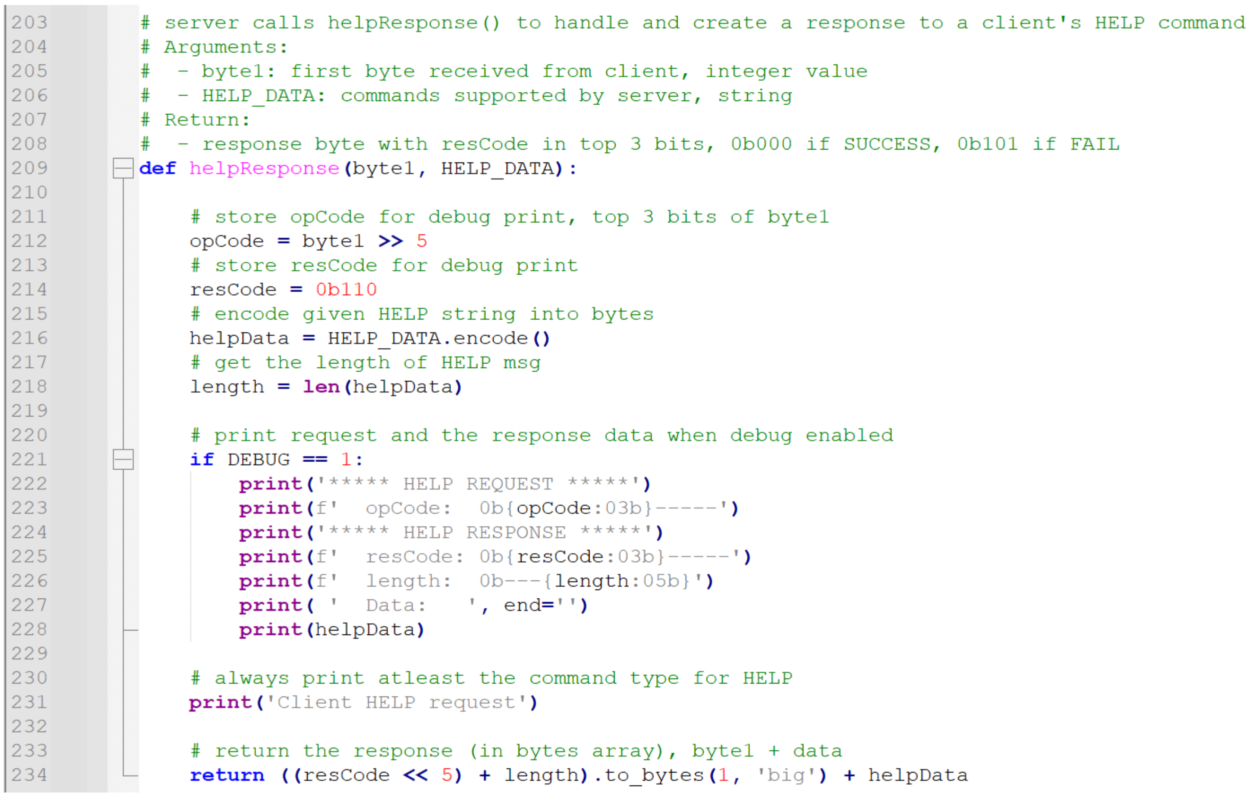
```python
# server calls putResponse() to handle and create a response to a client's PUT command
# Arguments:
#   - byte1: first byte received from client, integer value
#   - clientSocket: client socket to receive data, socket class
# Return:
#   - response byte with resCode in top 3 bits, 0b000 if SUCCESS, 0b101 if FAIL
def putResponse(byte1, clientSocket):

    # store opCode for debug print, top 3 bits of byte1
    opCode = byte1 >> 5
    # get the Filename Length, bottom 5 bits of byte1
    fNameLen = byte1 & 0x1F
    # use Filename Length to read the next bytes from client for Filename, decode to string
    fName = clientSocket.recv(fNameLen).decode()
    # read next 4 bytes from client for File Size, convert the 4 bytes into 1 integer, using big-endian notation
    fSize = int.from_bytes(clientSocket.recv(4), 'big')
    # use File Size to receive the whole file from client in bytes
    fBytes = clientSocket.recv(fSize)

    # now try to store the file, overwrites any existing file with same name
    try:
        # open in WRITE and BINARY mode for any type of file
        with open(fName, 'wb') as f:
            # write uploaded data to file
            f.write(fBytes)
        # store response code for SUCCESS
        resCode = 0b000
        # no error msg when successful
        err = ''

    # catch exceptions during write
    except:
        ### The project documentation does not specify a response code for PUT failures,    ###
        ### but we assume there might be failures if there isnt enough free space to create ###
        ### the file, or another such error. Since the SUCCESS response code for PUT is the ###
        ### same as CHANGE, we made the UNSUCCESS response code for PUT the same as CHANGE, ###
        ### ie: 0b101: response for unsuccessful change/put

        # store response code for FAIL
        resCode = 0b101
        # error msg for put failure
        err = 'ERROR: Could not create file "' + fName + '"'

    # print request and the response data when debug enabled
    if DEBUG == 1:
        print('***** PUT REQUEST *****')
        print(f'  opCode:  0b{opCode:03b}-----')
        print(f'  FL:      0b---{fNameLen:05b}')
        print( '  fName:   ' + fName)
        print(f'  FS:      0x{fSize:08X}')
        print( '  Data:    ', end='')
        print(fBytes)
        print('***** PUT RESPONSE *****')
        print(f'  resCode: 0b{resCode:03b}')

    # always print atleast the command type and filename for PUT
    print('Client PUT request: ' + fName)
    # print err if not empty
    if err != '': print(err)

    # return the response (in bytes array)
    return (resCode << 5).to_bytes(1, 'big')
```



```python
# server calls getResponse() to handle and create its response to a client's GET command
# Arguments:
#  - byte1: first byte received from client, integer value
#  - clientSocket: client socket to receive data, socket class
# Return:
#  - response, one byte with resCode 0b010 in top 3 bits for FAIL, multiple bytes with header and data for SUCCESS
def getResponse(byte1, clientSocket):

    # store opCode for debug print, top 3 bits of byte1
    opCode = byte1 >> 5
    # get the Filename Length, bottom 5 bits of byte1
    fNameLen = byte1 & 0x1F
    # use Filename Length to read the next bytes from client for Filename, decode to string
    fName = clientSocket.recv(fNameLen).decode()

    # now try to read the file, fails if file does not exist
    try:
        # open in READ and BINARY mode for any type of file
        with open(fName, 'rb') as f:
            # read and store data from file
            fBytes = f.read()
        # get the file size
        fSize = len(fBytes)
        # store response code for SUCCESS
        resCode = 0b001
        # build full request with resCode, FL, Filename, FS, and Data
        response = ((resCode << 5) + fNameLen).to_bytes(1, 'big') + fName.encode() + fSize.to_bytes(4, 'big') + fBytes
        # no error msg when successful
        err = ''

    # catch exceptions during read
    except:
        # store response code for FAIL (File Not found)
        resCode = 0b010
        # File not found response is only 1 byte
        response = (resCode << 5).to_bytes(1, 'big')
        # error msg for GET failure
        err = 'ERROR: File not found'

    # print request and the response data when debug enabled
    if DEBUG == 1:
        print('***** GET REQUEST *****')
        print(f'  opCode:  0b{opCode:03b}-----')
        print(f'  FL:      0b---{fNameLen:05b}')
        print( '  fName:   ' + fName)
        print('***** GET RESPONSE *****')
        print(f'  resCode: 0b{resCode:03b}-----')

        # only print relevant data for SUCCESSFUL get resCode '0b001'
        if resCode == 0b001:
            print(f'  FL:      0b---{fNameLen:05b}')
            print( '  fName:   ' + fName)
            print(f'  FS:      0x{fSize:08X}')
            print( '  Data:    ', end='')
            print(fBytes)

    # always print atleast the command type and filename for GET
    print('Client GET request: ' + fName)
    # print err if not empty
    if err != '': print(err)

    # return the response (in bytes array)
    return response
```

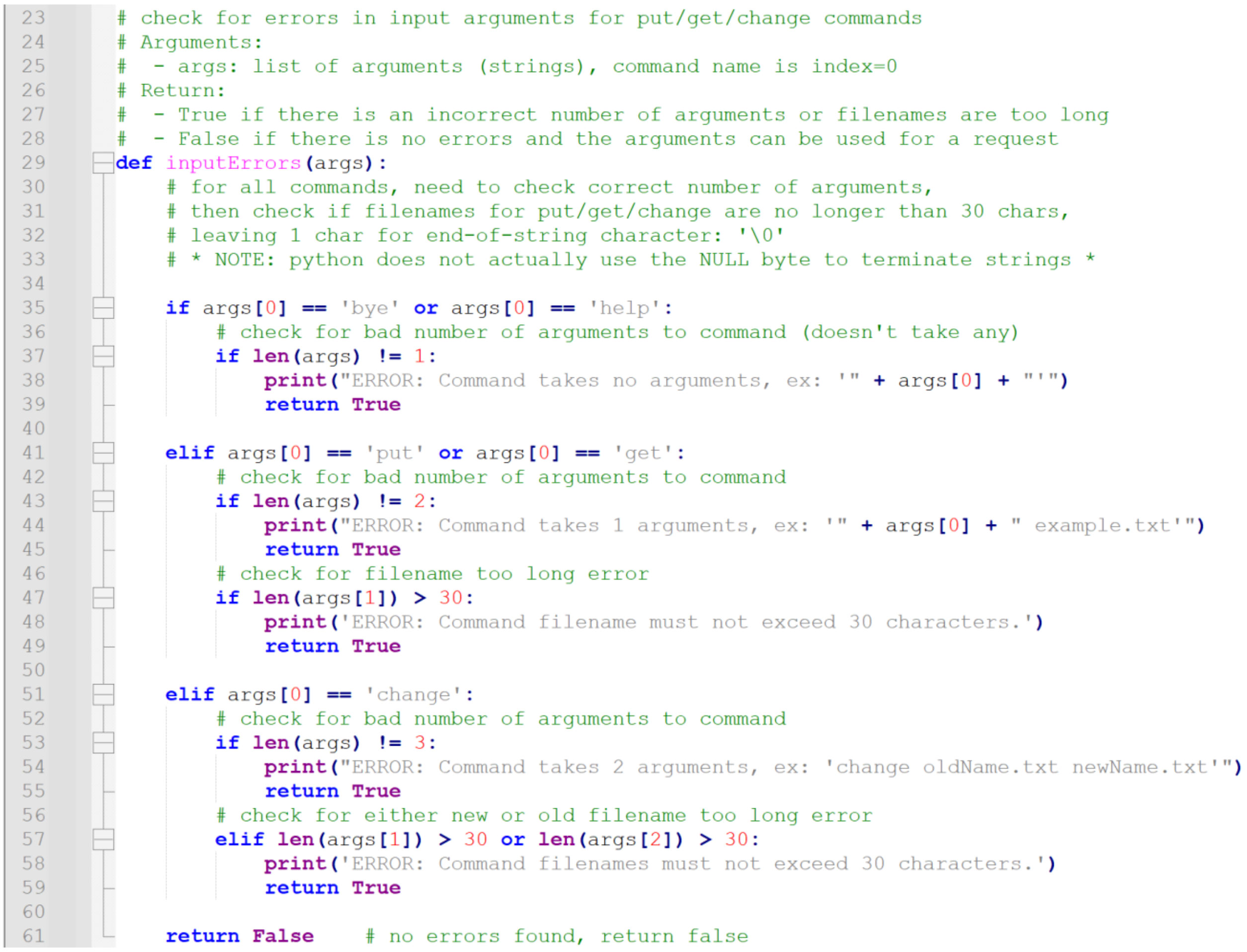


```python
# server calls changeResponse() to handle and create a response to a client's CHANGE command
# Arguments:
#  - byte1: first byte received from client, integer value
#  - clientSocket: client socket to receive data, socket class
# Return:
#  - response byte with resCode in top 3 bits, 0b000 if SUCCESS, 0b101 if FAIL
def changeResponse(byte1, clientSocket):

    # store opCode for debug print, top 3 bits of byte1
    opCode = byte1 >> 5
    # get the old Filename Length, bottom 5 bits of byte1
    oldNameLen = byte1 & 0x1F
    # use old Filename Length to read the next bytes from client for old Filename, decode to string
    oldName = clientSocket.recv(oldNameLen).decode()
    # read next byte from client for new Filename Length, and convert the byte into integer using big-endian notation
    newNameLen = int.from_bytes(clientSocket.recv(1), 'big')
    # use new Filename Length to read the next bytes from client for new Filename, decode to string
    newName = clientSocket.recv(newNameLen).decode()

    # now try to rename file, fails if file does not exist
    try:
        # rename the file
        os.rename(oldName, newName)
        # store response code for SUCCESS
        resCode = 0b000
        # no error msg when successful
        err = ''

    # catch exceptions during rename
    except:
        # store response code for FAIL (Unsuccessful change)
        resCode = 0b101
        # error msg for CHANGE failure
        err = 'ERROR: Change unsuccessful.'

    # print request and the response data when debug enabled
    if DEBUG == 1:
        print('***** CHANGE REQUEST *****')
        print(f'  opCode:   0b{opCode:03b}-----')
        print(f'  OFL:      0b---{oldNameLen:05b}')
        print( '  oldName: ' + oldName)
        print(f'  NFL:      0b---{newNameLen:05b}')
        print( '  newName: ' + newName)
        print('***** CHANGE RESPONSE *****')
        print(f'  resCode: 0b{resCode:03b}')

    # always print atleast the command type and filenames for CHANGE
    print('Client CHANGE request: ' + oldName + ' to ' + newName)
    # print err if not empty
    if err != '': print(err)

    # return the response (in bytes array)
    return (resCode << 5).to_bytes(1, 'big')
```

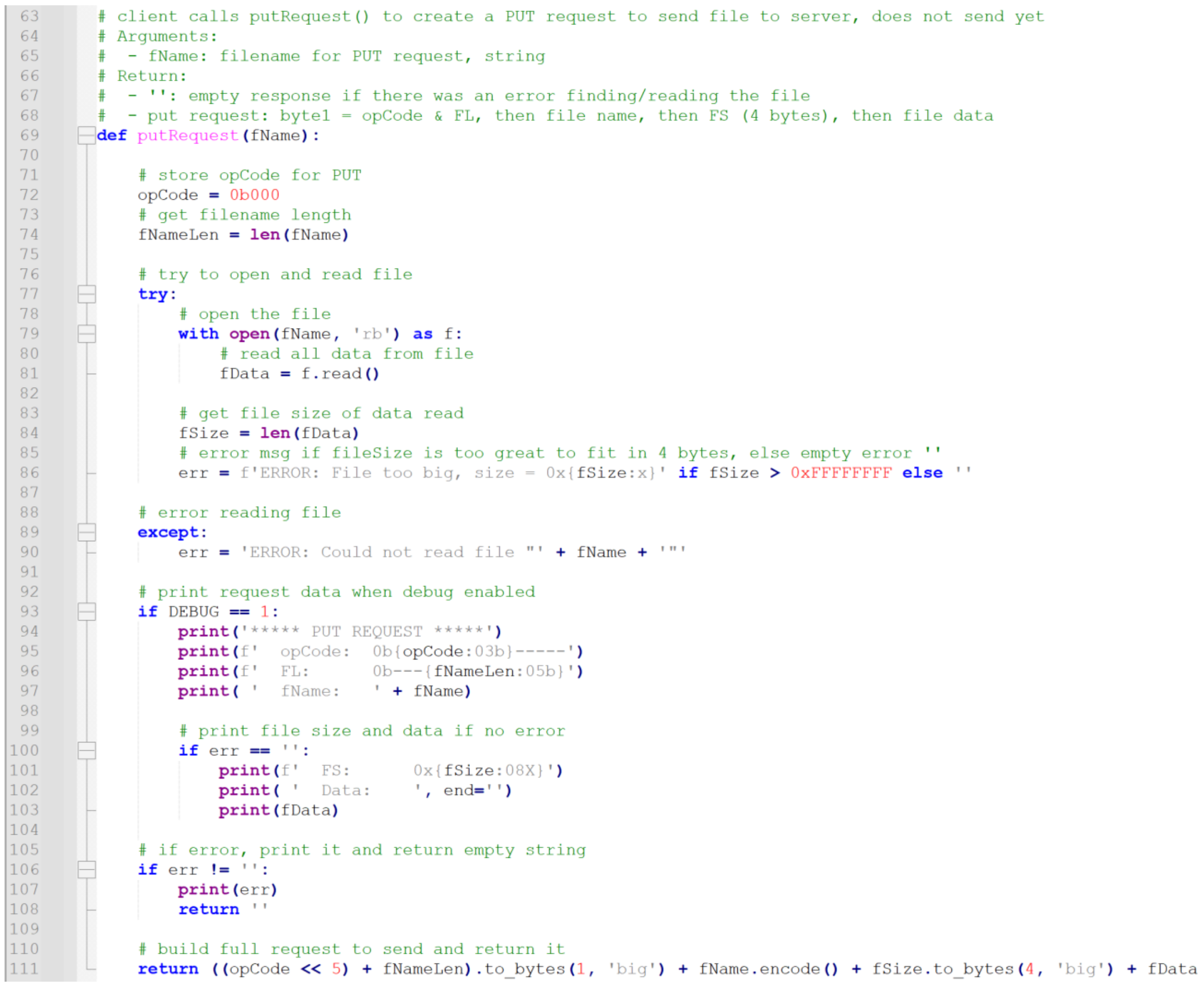


```python
# server calls helpResponse() to handle and create a response to a client's HELP command
# Arguments:
#   - byte1: first byte received from client, integer value
#   - HELP_DATA: commands supported by server, string
# Return:
#   - response byte with resCode in top 3 bits, 0b000 if SUCCESS, 0b101 if FAIL
def helpResponse(byte1, HELP_DATA):

    # store opCode for debug print, top 3 bits of byte1
    opCode = byte1 >> 5
    # store resCode for debug print
    resCode = 0b110
    # encode given HELP string into bytes
    helpData = HELP_DATA.encode()
    # get the length of HELP msg
    length = len(helpData)

    # print request and the response data when debug enabled
    if DEBUG == 1:
        print('***** HELP REQUEST *****')
        print(f'  opCode:  0b{opCode:03b}-----')
        print('***** HELP RESPONSE *****')
        print(f'  resCode: 0b{resCode:03b}-----')
        print(f'  length:  0b---{length:05b}')
        print( '  Data:   ', end='')
        print(helpData)

    # always print atleast the command type for HELP
    print('Client HELP request')

    # return the response (in bytes array), byte1 + data
    return ((resCode << 5) + length).to_bytes(1, 'big') + helpData
```

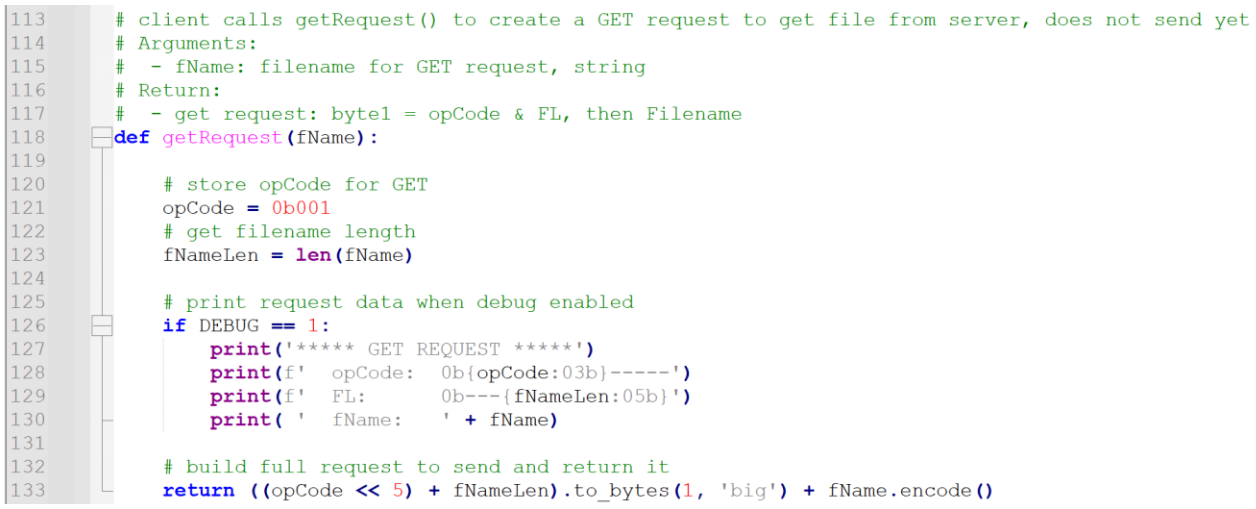
3.2 Client Functions



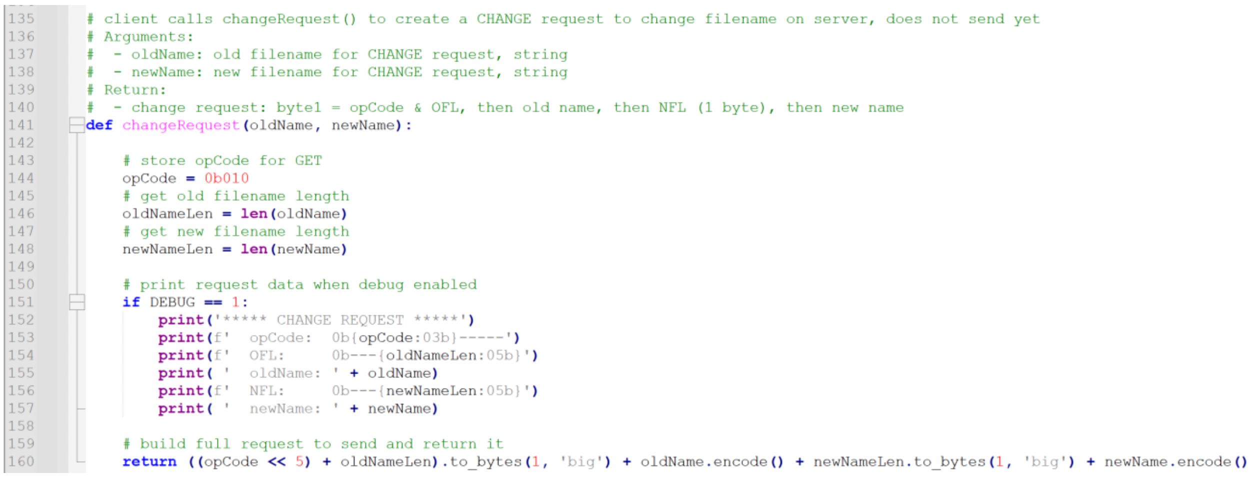
```python
# check for errors in input arguments for put/get/change commands
# Arguments:
#   - args: list of arguments (strings), command name is index=0
# Return:
#   - True if there is an incorrect number of arguments or filenames are too long
#   - False if there is no errors and the arguments can be used for a request
def inputErrors(args):
    # for all commands, need to check correct number of arguments,
    # then check if filenames for put/get/change are no longer than 30 chars,
    # leaving 1 char for end-of-string character: '\0'
    # * NOTE: python does not actually use the NULL byte to terminate strings *

    if args[0] == 'bye' or args[0] == 'help':
        # check for bad number of arguments to command (doesn't take any)
        if len(args) != 1:
            print("ERROR: Command takes no arguments, ex: '" + args[0] + "'")
            return True

    elif args[0] == 'put' or args[0] == 'get':
        # check for bad number of arguments to command
        if len(args) != 2:
            print("ERROR: Command takes 1 arguments, ex: '" + args[0] + " example.txt'")
            return True
        # check for filename too long error
        if len(args[1]) > 30:
            print('ERROR: Command filename must not exceed 30 characters.')
            return True

    elif args[0] == 'change':
        # check for bad number of arguments to command
        if len(args) != 3:
            print("ERROR: Command takes 2 arguments, ex: 'change oldName.txt newName.txt'")
            return True
        # check for either new or old filename too long error
        elif len(args[1]) > 30 or len(args[2]) > 30:
            print('ERROR: Command filenames must not exceed 30 characters.')
            return True

    return False    # no errors found, return false
```



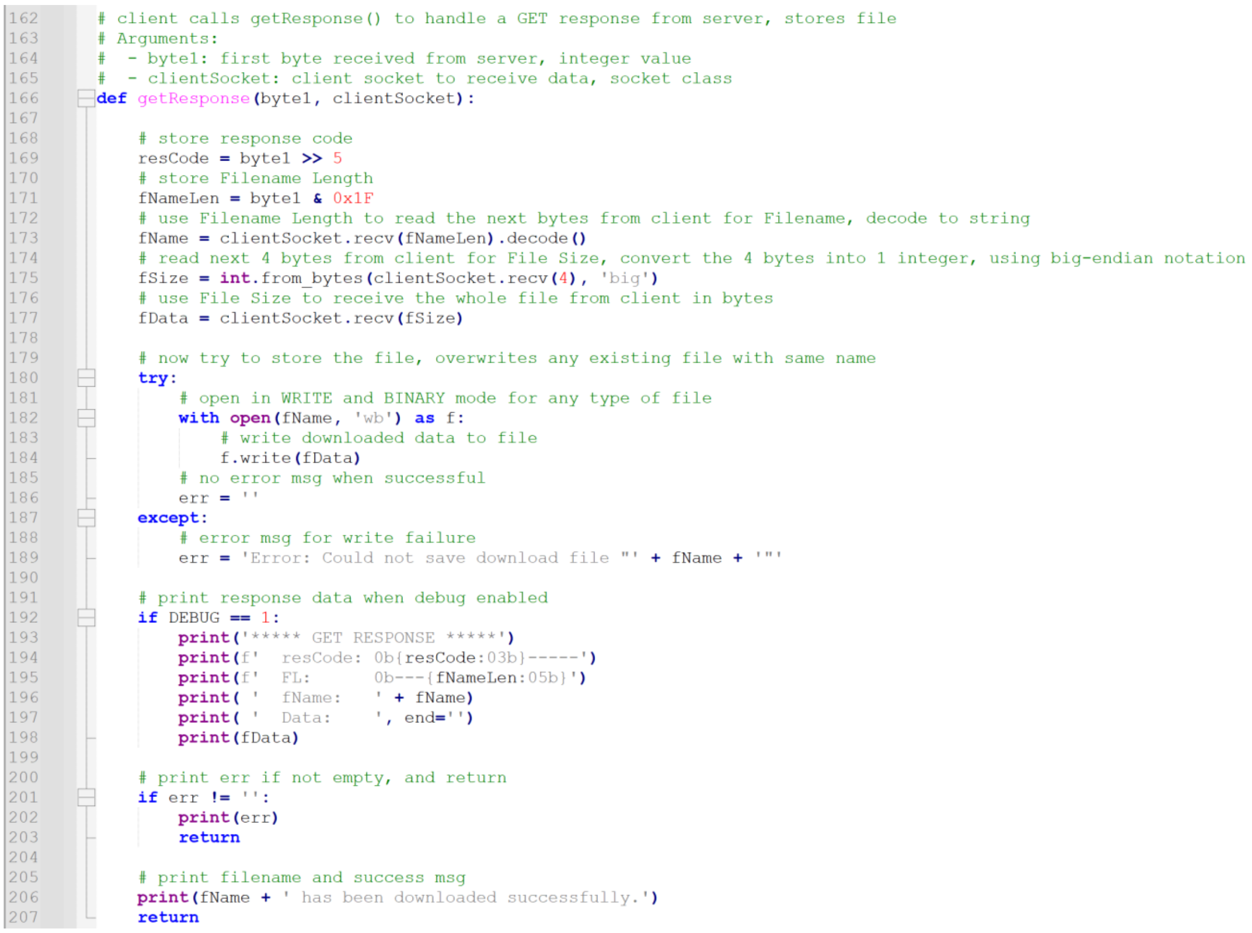
```python
# client calls putRequest() to create a PUT request to send file to server, does not send yet
# Arguments:
#   - fName: filename for PUT request, string
# Return:
#   - '': empty response if there was an error finding/reading the file
#   - put request: byte1 = opCode & FL, then file name, then FS (4 bytes), then file data
def putRequest(fName):

    # store opCode for PUT
    opCode = 0b000
    # get filename length
    fNameLen = len(fName)

    # try to open and read file
    try:
        # open the file
        with open(fName, 'rb') as f:
            # read all data from file
            fData = f.read()

        # get file size of data read
        fSize = len(fData)
        # error msg if fileSize is too great to fit in 4 bytes, else empty error ''
        err = f'ERROR: File too big, size = 0x{fSize:x}' if fSize > 0xFFFFFFFF else ''

    # error reading file
    except:
        err = 'ERROR: Could not read file "' + fName + '"'

    # print request data when debug enabled
    if DEBUG == 1:
        print('***** PUT REQUEST *****')
        print(f'  opCode:   0b{opCode:03b}-----')
        print(f'  FL:       0b---{fNameLen:05b}')
        print( '  fName:    ' + fName)

        # print file size and data if no error
        if err == '':
            print(f'  FS:       0x{fSize:08X}')
            print( '  Data:     ', end='')
            print(fData)

    # if error, print it and return empty string
    if err != '':
        print(err)
        return ''

    # build full request to send and return it
    return ((opCode << 5) + fNameLen).to_bytes(1, 'big') + fName.encode() + fSize.to_bytes(4, 'big') + fData
```



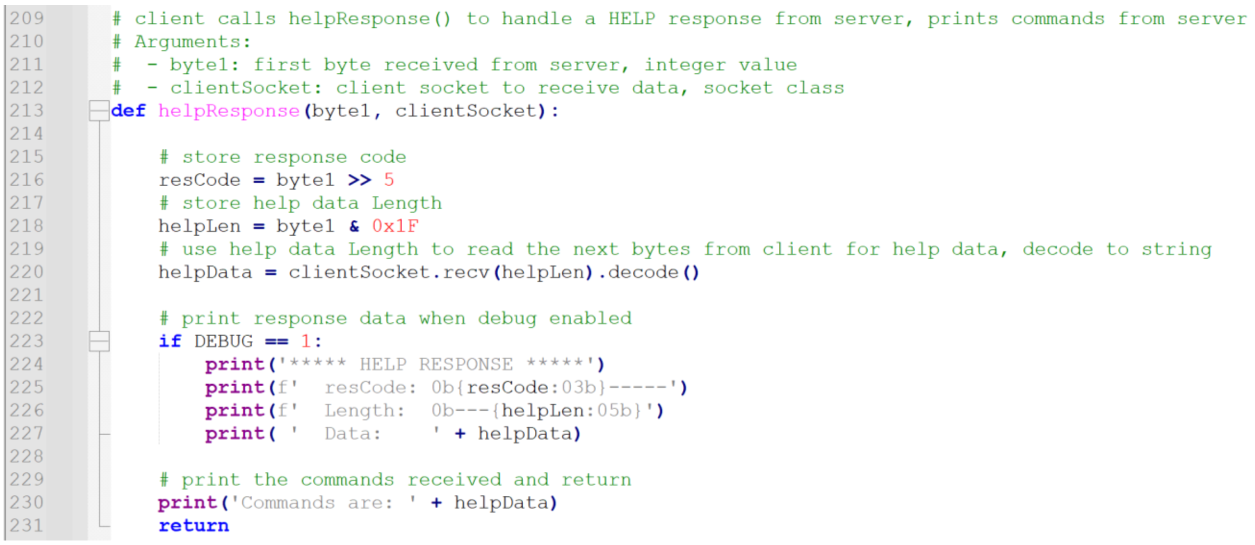
```python
# client calls getRequest() to create a GET request to get file from server, does not send yet
# Arguments:
#   - fName: filename for GET request, string
# Return:
#   - get request: byte1 = opCode & FL, then Filename
def getRequest(fName):

    # store opCode for GET
    opCode = 0b001
    # get filename length
    fNameLen = len(fName)

    # print request data when debug enabled
    if DEBUG == 1:
        print('***** GET REQUEST *****')
        print(f'  opCode:   0b{opCode:03b}-----')
        print(f'  FL:       0b---{fNameLen:05b}')
        print( '  fName:    ' + fName)

    # build full request to send and return it
    return ((opCode << 5) + fNameLen).to_bytes(1, 'big') + fName.encode()
```



```python
# client calls changeRequest() to create a CHANGE request to change filename on server, does not send yet
# Arguments:
#  - oldName: old filename for CHANGE request, string
#  - newName: new filename for CHANGE request, string
# Return:
#  - change request: byte1 = opCode & OFL, then old name, then NFL (1 byte), then new name
def changeRequest(oldName, newName):

    # store opCode for GET
    opCode = 0b010
    # get old filename length
    oldNameLen = len(oldName)
    # get new filename length
    newNameLen = len(newName)

    # print request data when debug enabled
    if DEBUG == 1:
        print('***** CHANGE REQUEST *****')
        print(f'  opCode:   0b{opCode:03b}-----')
        print(f'  OFL:      0b---{oldNameLen:05b}')
        print( '  oldName:  ' + oldName)
        print(f'  NFL:      0b---{newNameLen:05b}')
        print( '  newName:  ' + newName)

    # build full request to send and return it
    return ((opCode << 5) + oldNameLen).to_bytes(1, 'big') + oldName.encode() + newNameLen.to_bytes(1, 'big') + newName.encode()
```



```python
# client calls getResponse() to handle a GET response from server, stores file
# Arguments:
#  - byte1: first byte received from server, integer value
#  - clientSocket: client socket to receive data, socket class
def getResponse(byte1, clientSocket):

    # store response code
    resCode = byte1 >> 5
    # store Filename Length
    fNameLen = byte1 & 0x1F
    # use Filename Length to read the next bytes from client for Filename, decode to string
    fName = clientSocket.recv(fNameLen).decode()
    # read next 4 bytes from client for File Size, convert the 4 bytes into 1 integer, using big-endian notation
    fSize = int.from_bytes(clientSocket.recv(4), 'big')
    # use File Size to receive the whole file from client in bytes
    fData = clientSocket.recv(fSize)

    # now try to store the file, overwrites any existing file with same name
    try:
        # open in WRITE and BINARY mode for any type of file
        with open(fName, 'wb') as f:
            # write downloaded data to file
            f.write(fData)
        # no error msg when successful
        err = ''
    except:
        # error msg for write failure
        err = 'Error: Could not save download file "' + fName + '"'

    # print response data when debug enabled
    if DEBUG == 1:
        print('***** GET RESPONSE *****')
        print(f'  resCode: 0b{resCode:03b}-----')
        print(f'  FL:      0b---{fNameLen:05b}')
        print( '  fName:    ' + fName)
        print( '  Data:     ', end='')
        print(fData)

    # print err if not empty, and return
    if err != '':
        print(err)
        return

    # print filename and success msg
    print(fName + ' has been downloaded successfully.')
    return
```



```python
# client calls helpResponse() to handle a HELP response from server, prints commands from server
# Arguments:
#   - byte1: first byte received from server, integer value
#   - clientSocket: client socket to receive data, socket class
def helpResponse(byte1, clientSocket):

    # store response code
    resCode = byte1 >> 5
    # store help data Length
    helpLen = byte1 & 0x1F
    # use help data Length to read the next bytes from client for help data, decode to string
    helpData = clientSocket.recv(helpLen).decode()

    # print response data when debug enabled
    if DEBUG == 1:
        print('***** HELP RESPONSE *****')
        print(f'  resCode: 0b{resCode:03b}-----')
        print(f'  Length:  0b---{helpLen:05b}')
        print( '  Data:    ' + helpData)

    # print the commands received and return
    print('Commands are: ' + helpData)
    return
```