COSC 264

Introduction to Computer Networks and the Internet Assignment

Liz Richardson
Stefan Hall

Due: September 9

```
                                                   """
Sender part of assignment
#--Liz And Stefan--#
To run me type in the command line:
python sender.py portNum PortNum PortNum fileName
"""
#   python3 sender.py 7001 7000 8000 test.txt
import sys
import socket
import os.path #This is being used to check if file exists
import struct
import binascii
import packets
import select
MAX_BYTES = 512
PTYPE_DATA = 0
PTYPE_ACK = 1
MAGICNO = hex(0x497E)
def exit():
    """Exits the program displaying a message"""
    print("----------------------------------------")
    print("------------Program exited--------------")
    print("----------------------------------------")
    sys.exit(0)
def checkPort(num):
    """checks port number to ensure that it is an integer in the range 1024 - 64000
        exits program if it finds and error
        returns portnumber as an integer"""
    try:
        port_num = int(num)
    except ValueError:
        print("Port Number " + str(num) + " was not an integer")
        exit()

    if not(1024 <= port_num and port_num <= 64000):
        print("Port number " + str(num) + "  was not in range 1024 - 64000")
        exit()
    else:
        return port_num

def checkFile(fname):
    """checks for existing file
        exits if file not found"""
    if(os.path.isfile(fname)):
        return fname
    else:
        print("File: " + fname + " does not exist.")
        exit()
def get_params():
    """gets all of the parameters from the command line
        exits if wrong amount of arguments are entered
        checks to ensure input is correct
        returns port numbers and filename"""
    if len(sys.argv) != 5:
        print("\nWrong amount of command line arguments entered")
        exit()
    else:
        sys.argv = sys.argv[1:]
        sin = checkPort(sys.argv[0])
        sout = checkPort(sys.argv[1])
        csin = checkPort(sys.argv[2])
        filename = checkFile(sys.argv[3])
    return  sin, sout, csin, filename
```

```python
def read_file_data(file_obj):
    """reads a max amount of data into a buffer"""
    return file_obj.read(MAX_BYTES)
def  return_resources(socket_list, file_object):
    """closes sockets and file
        giving memory back to system"""
    for sockets in socket_list:
        sockets.close()
    file_object.close()
def raise_socket_error(socket_list, file_object, ERROR_COUNT):
    """if connection to a socket is refused an error is raised
        will wait 10 times and then time out"""
    if ERROR_COUNT >5:
        print()
        print("----------------------------------------")
        print("-----------Connection timeout-----------")
        print("----------Program will now exit---------")
        print("----------------------------------------")
        print()
        return_resources(socket_list, file_object)
        exit()
    else:
        print()
        print("---connection refused, trying again-----")
        print()
    return ERROR_COUNT+1
def main():
    """main function for sender
        sends a file to channel"""
    sin, sout, csin, filename = get_params()
    sockOut = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sockOut.bind(('127.0.0.1', sout))
    sockOut.connect(('127.0.0.1', csin))
    sockIn = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sockIn.bind(('127.0.0.1', sin))
    next = 0
    exitFlag = False
    file_object = open(filename, "rb")
    all_packets_count  = 0
    ack_packet_count = 0
    socket_list = [sockOut, sockIn]
    ERROR_COUNT = 0
    print()
    while not exitFlag:
        current_string = read_file_data(file_object)
        if len(current_string) == 0:
            current_packet = packets.packet(next, len(current_string), current_string)
            exitFlag = True
        elif len(current_string) > 0:
            current_packet = packets.packet(next, len(current_string), current_string)
        encoded_packet = packets.pack_packet(current_packet)
        has_response = False
        while not has_response:
            try:
                sockOut.send(encoded_packet) #sendencoded packet
                all_packets_count += 1
            except:
                ERROR_COUNT = raise_socket_error(socket_list, file_object,
ERROR_COUNT)
            else:
                ERROR_COUNT = 0 #a packet has been successfully sent, reset errorcount
            readable, _, _ = select.select([sockIn], [], [], 1)
            if readable:
                data = sockIn.recv(528)
                unpacked_packet = packets.unpack_packet(data)
                magicno, type, seqno, dataLen, byte_data = unpacked_packet
                if magicno == int(MAGICNO, 0) and type == PTYPE_ACK and dataLen == 0:
                    if seqno == next:
                        if exitFlag:
                            has_response = True
                        next ^= 1
                ack_packet_count +=1
                has_response = True
    print("Total num packets sent: ", all_packets_count)
    return_resources(socket_list, file_object)
    exit()

if __name__ == '__main__':
    main()
```

"""

```python
Channel part of assignment
#--Liz And Stefan--#
To run me type in the command line:
python channel.py portNum PortNum PortNum PortNum S(in) R(in) P
#   python3 channel.py 8000 8001 8003 8002 7001 9000 0.1
"""
import sys
import socket
import select
import packets
import random
def exit():
    """Exits the program displaying a message"""
    print("Program will now exit\n")
    sys.exit(0)
def checkPort(num):
    """checks port number to ensure that it is an integer in the range 1024 - 64000
        exits program if it finds and error
        returns portnumber as an integer"""
    try:
        port_num = int(num)
    except ValueError:
        print("Port Number " + str(num) + " was not an integer")
        exit()
    if not(1024 <= port_num and port_num <= 64000):
        print("Port number " + str(num) + "  was not in range 1024 - 64000")
        exit()
    else:
        return port_num
def checkProbability(p):
    """checks to make sure the probability enteed is a float in the range [0,1)
        exits if there is an error
        returns probability P """
    try:
        p = float(p)
    except ValueError:
        print("Probability not an integer")
        exit()
    else:

        if not(0 <= p < 1):
            print("Probability not in range 0-1")
            exit()
        else:
            return p
def get_params():
    """gets all of the parameters from the command line
        exits if wrong amount of arguments are entered
        checks to ensure input is correct
        reutrns port numbers and P"""
    if len(sys.argv) != 8:
        print("\nWrong amount of command line arguments entered")
        exit()
    else:
        sys.argv = sys.argv[1:] #chops off the name of the program
        csin = checkPort(sys.argv[0])
        csout = checkPort(sys.argv[1])
        crin = checkPort(sys.argv[2])
        crout = checkPort(sys.argv[3])
        sin = checkPort(sys.argv[4])
        rin = checkPort(sys.argv[5])
        P = checkProbability(sys.argv[6])

    return  csin, csout, crin, crout, sin, rin, P
```

```python
def can_send(P):
    """generates a uniformly distrubuted number between 0 and 1
        drops packet if u < P
        returns true if packet can be sent"""
    to_send = False
    u = random.uniform(0, 1)
    if u >= P:
        to_send =  True
    return to_send
def  return_resources(socket_list):
    """closes sockets and file
        giving memory back to system"""
    for sockets in socket_list:
        sockets.close()
def raise_socket_error(socket_list, error_count):
    """If connection resets """
    if error_count >5:
        print()
        print("----------------------------------------")
        print("-----------Connection refused-----------")
        print("----------Program will now exit---------")
        print("----------------------------------------")
        print()
        return_resources(socket_list)
        exit()
    else:
        print()
        print("---connection refused, trying again-----")
        print()
    return error_count + 1
def raise_listen_error(socket_list, listening_flag):
    """Raises error if has been listening too long"""
    if listening_flag > 5:
        print()
        print("----------------------------------------")
        print("--------No responses in a while---------")
        print("------ has been listening too long -----")
        print("---------Program will now exit----------")
        print("----------------------------------------")
        print()
        return_resources(socket_list)
        exit()
def main():
    """main fuction for channel
        makes four sockets then listens waiting for a packet from sender or receiver
        randomly decides to drop packet or not
        if packet it isnt dropped it passes it through"""
    csin, csout, crin, crout, sin, rin, P = get_params()
    sockCSOut = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sockCSOut.bind(('127.0.0.1', csout))
    sockCSOut.connect(('127.0.0.1', sin))  # So we don't have to specify where we send
to
    sockCSIn = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sockCSIn.bind(('127.0.0.1', csin))
    sockCROut = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sockCROut.bind(('127.0.0.1', crout))
    sockCROut.connect(('127.0.0.1', rin))  # So we don't have to specify where we send
to
    sockCRIn = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sockCRIn.bind(('127.0.0.1', crin))
    socket_list = [sockCROut, sockCRIn, sockCSIn, sockCSOut]
    error_count = 0
    listening_flag = 0
    while True:
        readable, _, _ = select.select([sockCSIn, sockCRIn], [], [], 1)
        print("\nlistening\n")
        if len(readable) == 0:
             listening_flag += 1
             if listening_flag > 20:
                 raise_listen_error(socket_list, listening_flag)
        else:
             listening_flag = 0 #reset listening time
             for sockets in readable:
                 if sockets is sockCSIn: #if data is rome channel
                     data, addr = sockCSIn.recvfrom(528)
                     unpacked_packet = packets.unpack_packet(data)
                     magicno, type, seqno, dataLen, byte_data = unpacked_packet
                     if packets.magicNoCheck(magicno):
                          print("recieved Datapack...")
                          if can_send(P):
                              print("forwarding Datapack")
                              try:
                                   sockCROut.send(data)
                              except:
                                   error_count = raise_socket_error(socket_list,
error_count)
                          else:
                              print("PACKET FAILED TO SEND")
                 if sockets is sockCRIn: #if data is from reciever
                     data, addr = sockCRIn.recvfrom(528)
                     unpacked_packet = packets.unpack_packet(data)
                     magicno, type, seqno, dataLen, byte_data = unpacked_packet
                     if packets.magicNoCheck(magicno):
                          print("recieved ackPacket")
                          if can_send(P):
                              # now we can send the packet
                              print("Forwarding AckPack")
                              try:
                                   sockCSOut.send(data)
                                   print("Ackpack sent")
                              except:
                                   error_count = raise_socket_error(socket_list
,error_count)
                          else:
                              print("ACKPACKET FAILED TO SEND")
if __name__ == '__main__':
    #makes it run automatically which is neat
    main()
```

```
                                                                """
Receiver part of assignment
#--Liz And Stefan--#
To run me type in the command line:
python3 receiver.py portNum PortNum PortNum fileName
# python3 receiver.py 9000 9001 8003 test.txt
"""
import sys
import socket
import os.path
import packets
import select
MAX_BYTES = 512
MAGICNO = hex(0x497E)
PTYPE_DATA = 0
PTYPE_ACK = 1
def exit():
    """Exits the program displaying a message"""
    print("----------------------------------------")
    print("------------Program exited--------------")
    print("----------------------------------------")
    sys.exit(0)
def checkPort(num):
    """checks port number to ensure that it is an integer in the range 1024 - 64000
        exits program if it finds and error
        returns portnumber as an integer"""
    try:
        port_num = int(num)
    except ValueError:
        print("Port Number " + str(num) + " was not an integer")
        exit()
    if not(1024 <= port_num and port_num <= 64000):
        print("Port number " + str(num) + "  was not in range 1024 - 64000")
        exit()

    else:
        return port_num

def checkFile(fname):
    """checks for existing file
        exits if file not found"""
    if(os.path.isfile(fname)):
        print("File: " + fname + " already exists.")
        exit()
    else:
        return fname

def get_params():
    """gets all of the parameters from the command line
        exits if wrong amount of arguments are entered
        checks to ensure input is correct
        returns port numbers and filename"""
    if len(sys.argv) != 5:
        print("\nWrong amount of command line arguments entered")
        exit()
    else:
        sys.argv = sys.argv[1:]
        rin = checkPort(sys.argv[0])
        rout = checkPort(sys.argv[1])
        crin = checkPort(sys.argv[2])
        filename = checkFile(sys.argv[3])
    return rin, rout, crin, filename
def  return_resources(socket_list):
    """closes sockets and file
        giving memory back to system"""
    for sockets in socket_list:
        sockets.close()
def raise_socket_error(socket_list, ERRORCOUNT=0):
    """if connection to a socket is refused an error is raised
        will wait 10 times and then time out"""
    if ERRORCOUNT>10:
        print()
        print("----------------------------------------")
        print("-----------Connection refused-----------")
        print("----------Program will now exit---------")
        print("----------------------------------------")
        print()
        return_resources(socket_list)
        exit()
    else:
        print()
        print("---connection refused, trying again-----")
        print()
    return ERRORCOUNT+1
def main():
    """Main function for reciever
     recieves data and turns into file
     sends ack packets"""
    rin, rout, CRIN, FILENAME = get_params()
    sockOut = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sockOut.bind(('127.0.0.1', rout))
    sockOut.connect(('127.0.0.1', CRIN))
    sockIn = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sockIn.bind(('127.0.0.1', rin))
    expected = 0
    recieving_packets = True
    socket_list = [sockOut, sockIn]
    error_count = 0
    file_object = open(FILENAME, "wb+")
    while recieving_packets:
        readable, _, _ = select.select([sockIn], [], [], 1)
        if readable:
            print("recieved Pack")
            data, sender = sockIn.recvfrom(528)
            unpacked_packet = packets.unpack_packet(data)
            magicno, type, seqno, dataLen, data = unpacked_packet
            if magicno == int(MAGICNO, 0) and type == PTYPE_DATA and seqno == 
expected:
                ack_pack = packets.packet(seqno)
                ack_pack.set_ack()
                packed_packet = packets.pack_packet(ack_pack)
                expected ^= 1   #Switches between 0 and 1 with XOR Using bitwise 
operator
                if dataLen > 0: #data in packet
                    file_object.write(data)
                    try:
                        sockOut.send(packed_packet)
                    except:
                        error_count = raise_socket_error(socket_list, error_count)
                    else:
                        error_count = 0 #a packet has been successfully sent so we 
can reset errorcount
                else:   #datalen == 0
                    ack_pack = packets.packet(seqno)
```

```
                        ack_pack.set_ack()
                        packed_packet = packets.pack_packet(ack_pack)
                        try:
                            sockOut.send(packed_packet)
                        except:
                            error_count = raise_socket_error(socket_list, error_count)
                        else:
                            error_count = 0  # a packet has been successfully sent so we
can reset errorcount
                        recieving_packets = False
              elif(magicno == int(MAGICNO, 0) and type == PTYPE_DATA and seqno !=
expected):
                    ack_pack = packets.packet(seqno)
                    ack_pack.set_ack()
                    packed_packet = packets.pack_packet(ack_pack)
                    try:
                        sockOut.send(packed_packet)
                    except:
                        error_count = raise_socket_error(socket_list, error_count)
                    else:
                        error_count = 0  # a packet has been successfully sent so we can
reset errorcount
    file_object.close() #returns resources
    print("RECIEVED ALL DATA")
    exit()
if __name__ == '__main__':
    #makes it run automatically which is neat
    main()
```