Assignment Code

###sender.py

```
from sys import argv
import socket
import select
from common import *

CLOSE_REQUESTED = False

def send(sin, sout, file):
    """
    Send thread. Sends the file contents to a
    receiver via the channel.

    """
    _next = 0
    exit_flag = False
    num_sent_packets = 0

    while not (exit_flag or CLOSE_REQUESTED):
        block = file.read(BLOCK_SIZE)
        if len(block) == 0:
            exit_flag = True
            packet = Packet(bytes(), _next)
        else:
            packet = Packet(block, _next)

        while not CLOSE_REQUESTED:
            try:
                sout.send(packet.to_bytes())
                num_sent_packets += 1
            except:
                print("locked")
```

```
                    break
                readable, _, _ = select.select([sin], [], [], 0.
                if readable:
                    sock = readable[0]
                    packet_bytes, address = sock.recvfrom(PACKET_
                    packet = Packet.from_bytes(packet_bytes)
                    if packet.magicno == 0x497E \
                       and packet.packet_type == Packet.ACK \
                       and packet.data_len == 0 \
                       and packet.seqno == _next:
                        _next = 1 - _next
                        break

    print(num_sent_packets)
    print(_next)
    file.close()
    sin.close()
    sout.close()


def main(filename, ports):
    global CLOSE_REQUESTED
    CLOSE_REQUESTED = False

    file = setup_file(filename)

    sin, sout = setup_sockets(ports[0], ports[1], ports[2])

    send(sin, sout, file)


def setup_sockets(s_in_port, s_out_port, c_s_in_port):
    """
    Create and bind the sender_in and sender_out sockets.
    """
    sin = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sout = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    sin.bind((HOST_IP, s_in_port))
```

```
    sout.bind((HOST_IP, s_out_port))

    sout.connect((HOST_IP, c_s_in_port))

    return sin, sout


def setup_file(filename):
    """
    Opens the input file for read
    """
    return open(filename, 'rb')


if __name__ == '__main__':

    number_of_arguments = len(argv)
    if number_of_arguments != 5: # argv[0] is program name
        abort("Incorrect number of parameters")

    if number_of_arguments != len(set(argv)):
        abort("Port numbers not distinct")

    ports = tuple(int(p) for p in argv[1:4])
    for port in ports:
        if (port < 1024) or (port > 64000):
            abort("Port {} not within valid range \
            1024-64000".format(port))

    filename = argv[4]

    main(filename, ports)
```

###channel.py

```
from sys import argv
import socket
```

```
from select import select
import random
from common import *

CLOSE_REQUESTED = False

def channel(packet_loss_rate, csin, csout, crin, crout):
    global CLOSE_REQUESTED
    while not CLOSE_REQUESTED:
        readable_sockets, _, _ = select([csin, crin], [], []
        for sock in readable_sockets:
            rcvd, address = sock.recvfrom(PACKET_SIZE)
            packet = Packet.from_bytes(rcvd)

            if packet.magicno != 0x497E or \
            random.random() <= packet_loss_rate:
                continue

            try:
                if sock == csin:
                    crout.send(packet.to_bytes())
                elif sock == crin:
                    csout.send(packet.to_bytes())
            except:
                CLOSE_REQUESTED = True
                break

    csin.close()
    csout.close()
    crin.close()
    crout.close()

def main(packet_loss_rate, ports):
    global CLOSE_REQUESTED
    CLOSE_REQUESTED = False

    channel(packet_loss_rate, *setup_sockets(*ports))
```

```
def setup_sockets(c_s_in_port, c_s_out_port,\
     s_in_port, c_r_in_port,\
     c_r_out_port, r_in_port):

    csin = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    csout = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    crin = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    crout = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    csin.bind((HOST_IP, c_s_in_port))
    csout.bind((HOST_IP, c_s_out_port))
    crin.bind((HOST_IP, c_r_in_port))
    crout.bind((HOST_IP, c_r_out_port))

    csout.connect((HOST_IP, s_in_port))
    crout.connect((HOST_IP, r_in_port))

    return csin, csout, crin, crout

if __name__ == "__main__":

    number_of_arguments = len(argv)
    if number_of_arguments != 8: # argv[0] is program name
        abort("Incorrect number of parameters")

    if number_of_arguments != len(set(argv)):
        abort("Port numbers not distinct")

    ports = tuple(int(p) for p in argv[1:7])
    for port in ports:
        if (port < 1024) or (port > 64000):
            abort("Port {} not within valid range \
            1024-64000".format(port))

    packet_loss_rate = float(argv[7])
    if (packet_loss_rate < 0) or (packet_loss_rate >= 1):
        abort("Incorrect packet loss rate, must be \
        between 0 and 1")
```

```
    main(packet_loss_rate, ports)
```

###receiver.py

```
"""
Receiver Program.

"""
from sys import argv
import socket
import select
from common import *

CLOSE_REQUESTED = False

def receive(rin, rout, file):

    expected = 0

    while not CLOSE_REQUESTED:
        readable, _, _ = select.select([rin], [], [], 0.0005
        if readable:
            sock = readable[0]
            message, address = rin.recvfrom(PACKET_SIZE)
            rcvd_pack = Packet.from_bytes(message)
            if rcvd_pack.magicno == 0x497E \
               and rcvd_pack.packet_type == Packet.DATA:
                ack_pack = Packet(bytes(), rcvd_pack.seqno,\
                 0x497E, Packet.ACK)
                rout.send(ack_pack.to_bytes())

                if rcvd_pack.seqno == expected:
                    expected = 1 - expected
                    if rcvd_pack.data_len > 0:
                        file.write(rcvd_pack.data)
                    else:
```

```
                break


    print(expected)
    file.close()
    rin.close()
    rout.close()


def main(filename, ports):
    global CLOSE_REQUESTED
    CLOSE_REQUESTED = False

    file = setup_file(filename)

    rin, rout = setup_sockets(ports[0], ports[1], ports[2])

    receive(rin, rout, file)


def setup_sockets(r_in_port, r_out_port, c_r_in_port):
    """
    Create and bind the reciever_in and receiver_out sockets
    """
    rin = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    rout = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    rin.bind((HOST_IP, r_in_port))
    rout.bind((HOST_IP, r_out_port))

    rout.connect((HOST_IP, c_r_in_port))

    return rin, rout



def setup_file(filename):
    """
    Opens the output file for write
```

```
    """
    return open(filename, 'wb')



if __name__ == '__main__':

    number_of_arguments = len(argv)
    if number_of_arguments != 5: # argv[0] is program name
        abort("Incorrect number of parameters")

    if number_of_arguments != len(set(argv)):
        abort("Port numbers not distinct")

    ports = tuple(int(p) for p in argv[1:4])
    for port in ports:
        if (port < 1024) or (port > 64000):
            abort("Port {} not within valid range \
            1024-64000".format(port))

    filename = argv[4]

    main(filename, ports)
```

###test.py

```
from common import *
import sender
import channel
import receiver
import threading
import time
from hashlib import md5
from sys import argv
from sys import stdout

def start_channel():
```

```
    channel.main(0.3, (2000, 2001, 2002, 2003, 2004, 2005))

def start_receiver():
    receiver.main("testfile.out", (2005, 2007, 2003))

def start_sender():
    sender.main("testfile.in", (2002, 2006, 2000))

def main(num_tests=1):
    for _ in range(num_tests):

        chan_thread = threading.Thread(target=start_channel)
        recv_thread = threading.Thread(target=start_receiver
        send_thread = threading.Thread(target=start_sender)

        chan_thread.start()
        time.sleep(1)
        recv_thread.start()
        time.sleep(1)
        send_thread.start()

        try:
            send_thread.join()
            recv_thread.join()
        except:
            break
        finally:
            receiver.CLOSE_REQUESTED = True
            sender.CLOSE_REQUESTED = True
            channel.CLOSE_REQUESTED = True

            send_thread.join()
            recv_thread.join()
            chan_thread.join()
            in_sum = md5(
                open("testfile.in", "rb").read())
                .digest()
            out_sum = md5(
                open("testfile.out", "rb").read())
```

```
                .digest()

            stdout.flush()

if __name__ == "__main__":

    if len(argv) > 1:
        num_tests = int(argv[1])
        main(num_tests)
    else:
        main()
```