CMPSC 381
Data Communications and Networks
Spring 2016
Bob Roos
http://cs.allegheny.edu/sites/rroos/cs381s2016

Lab 5
25 February 2016
Due via Bitbucket on Thursday, 3 March, 8 a.m.
NOTE THE 8 a.m. DEADLINE!

Summary: Create a file upload protocol with TCP

Details: In class on Wednesday, 24 February, we looked at two programs named `filerecv.py` and `filesend.py`. Program `filerecv.py` listens for connection requests on port 12345; when such a request arrives, `filerecv.py` immediately begins saving the received data to a file until the connection closes. Program `filesend.py` asks the user for the name of a file and then connects to `filerecv.py` and sends the contents of the file.

Think of `filesend` as a primitive backup application that connects to a backup system represented by `filerecv`.

Today you are going to expand this into a file upload protocol that will behave as in Figure 1.

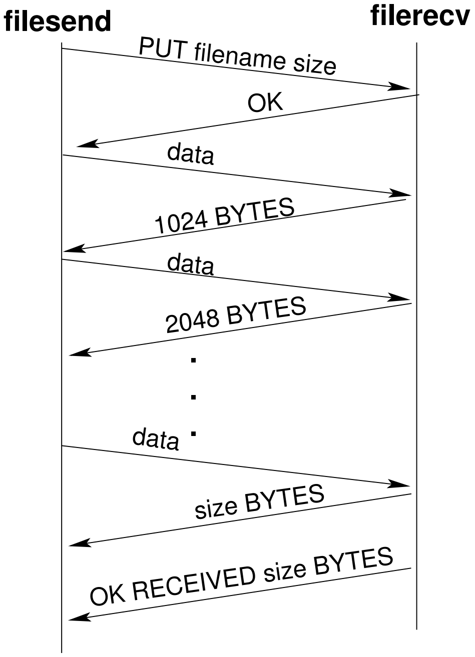
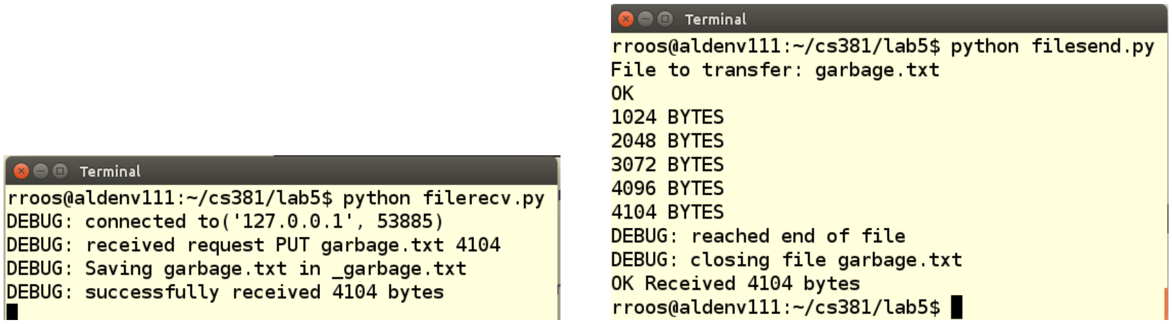


Figure 1: The upload protocol

You will run the `filerecv.py` program first, then use `filesend.py` to upload files. See sample outputs in Figure 2.



```
rroos@aldenv111:~/cs381/lab5$ python filerecv.py
DEBUG: connected to('127.0.0.1', 53885)
DEBUG: received request PUT garbage.txt 4104
DEBUG: Saving garbage.txt in _garbage.txt
DEBUG: successfully received 4104 bytes
```

```
rroos@aldenv111:~/cs381/lab5$ python filesend.py
File to transfer: garbage.txt
OK
1024 BYTES
2048 BYTES
3072 BYTES
4096 BYTES
4104 BYTES
DEBUG: reached end of file
DEBUG: closing file garbage.txt
OK Received 4104 bytes
rroos@aldenv111:~/cs381/lab5$
```

Figure 2: Sample output from `filerecv.py` and `filesend.py`

Here's a quick run-through of the process:

- Program `filerecv.py` opens a socket and listens
- Program `filesend.py` asks the user for the name of a file (example: `garbage.txt`), computes the size of the file (let's say it's 4104 bytes), and sends the message "`PUT garbage.txt 4104`"
- Program `filerecv` extracts the filename and filesize from the message, opens a file named "*_filename*" (e.g., "`_garbage.txt`"), and sends back the message "`OK`". Then it enters a loop to repeatedly read blocks of data
- Inside the loop, `filerecv` receives a block of data, saves it in the file, and sends a message back to the client stating how many bytes have been received so far (this number will grow in steps of 1024 until the very last one)
- Program `filesend.py` will print out all of the server's replies, including the beginning and ending "`OK`" messages and the running totals of the number of bytes uploaded
- The loop in program `filerecv.py` terminates when the number of bytes received equals the size of the file specified in the request (don't simply keep looping until you receive a block of size zero—ask if you don't understand why this won't work here)
- `filerecv.py` sends one more message to the client saying "`OK Received ... bytes`", then closes the connection and listens for another request
- `filesend.py` receives and prints the "`OK`" message, closes the connection, and terminates

To prevent accidental overwriting of files during the debugging process, I added a leading underscore character "`_`" to the file name that was given in the "`PUT`" message (thus, "`PUT garbage.txt ...`" will copy to a file named "`_garbage.txt`")

Finding the Size of a File.

Your `filesend` client must determine the size of the file to be sent—this is done using the "`os.path.getsize(...)`" function:

```
import os.path
       ...
    filename = raw_input('File to transfer: ')
    f = open(filename,'rb')
    filesize = os.path.getsize(filename)
       ...
```

Debugging

I suggest you include lots of print statements to track the progress of your programs—I've showed examples of some of these in the sample output. Note that the only non-debugging output in the sample is when `filesend` echoes the replies from `filerecv`. These are the cumulative bytes received and the two "OK" messages.

[**Submit your work.**] Make sure you have *fully-commented* Python programs named "*yourlastname*-`filerecv.py` and "*yourlastname*-`filesend.py` in your `lab5` folder. Upload this folder to your Bitbucket repository by the lab deadline.

Make sure your name and the honor code pledge appear at the top of both program files.