Globus Backend Coding Challenge

CONFIDENTIALITY NOTICE

This challenge and your answers are confidential to globus.org. Do not share this challenge or your answers with recruiters, coworkers, friends, etc.

Overview

This coding challenge may take 2-3 hours for a senior programmer with recent experience in the subject. It is approximately 200 lines of code. You may only use modules in the standard library.

Deliverables are: client.py, server.py, and answers to the "Questions" section at the bottom.

client.py must run in under 10 minutes. server.py can run on the same machine, using loopback / localhost TCP. A reference implementation runs in under 2 minutes on a 2019 laptop.

Server

The server.py process sends simulated directory listings over a TCP socket. Only one concurrent TCP connection is necessary (the server must not fork, use threads, or use select() to multiplex connections). When the client disconnects, the server must close the socket and wait for another connection.

The simulated root directory ("/") contains 100 subdirectories, which will be referred to as "level 1 subdirectories". Each "level 1 subdirectory" contains 100 subdirectories, aka "level 2". Each "level 2 subdirectory" contains 100 subdirectories, aka "level 3". Each "level 3 subdirectory" contains 3 files.

Thus, there are 1 million (100 * 100 * 100) "level 3 subdirectories", and 3 million total files. The

files and directories are named as follows:

```
/dir_00/dir_00/dir_00/file_00
```

```
/dir_00/dir_00/dir_00/file_01
/dir_00/dir_00/dir_00/file_02
 ...
/dir_99/dir_99/dir_99/file_00
/dir_99/dir_99/dir_99/file_01
/dir_99/dir_99/dir_99/file_02
```

Client

The client.py process connects to the server and scans the entire directory structure, starting at "/". It prints out the total number of files found. The client MUST use pipelining; up to 20 DIRLIST commands must be in the pipeline to the server when possible, to avoid network or CPU latency.

Note that the client can not assume any prior knowledge of the simulated directory structure; it has to start with "/" and take action based on the server responses.

Protocol

The protocol is line based, and each line is ended by a "\r\n" sequence (for simplicity the protocol assumes that sequence will not exist in a path name).

The DIRLIST command lists a single directory, which may be the root ("/") or one of the subdirectories, e.g. "/dir_99/". The directory must start and end with a '/' character. Paths are case sensitive. The syntax is "DIRLIST" <space> <directory>.

A command line of "DIRLIST /" or "DIRLIST /dir_00/dir_23/" will return:

```
BEGIN
  dir_00/
  dir_01/
   ...
  dir_99/
END
```

A command line of "DIRLIST /dir_00/dir_00/dir_00/" will return:

```
BEGIN
```

```
  file_00
  file_01
  file_02
END
```

A successful response is a "BEGIN" line, 0 or more path lines that each start with a space, and a final "END" line. Subdirectories in a response path have a trailing "/" (that is the only place a "/" is allowed).

On error, the server can simply return a line that starts with "ERROR" and close the socket (even if more commands are in the pipeline). e.g.:

```
ERROR Directory must start with a '/' character
```

Command pipelining works like HTTP/1.1: multiple client commands and/or server responses may be pending on the socket at any time.

Questions

If you completed the code challenge, congratulations! Please also answer the following:

1. How long did client.py run, and what was the peak memory usage?
2. Why should the DIRLIST success and/or error response be in JSON format?
3. Why should the DIRLIST success and/or error response **not** be in JSON format?
4. What are reason(s) why DFS (depth first search) would be a bad choice for scanning a directory tree over the network?