"Accio" File using TCP (Server)

Accio Server

The Accio server is an extension of the simplified server that:

- processes multiple simultaneous connections in parallel
- saves the received data into the specified folder. Note that filenames should strictly follow what defined in the spec: `1.file`, `2.file`, etc. DO NOT invent any other mechanism that is not explicitly defined in the spec or you will fail the tests.

Revisions

Not yet

Server Application Specification

The server application MUST be implemented in `server.py` Python file, accepting two command-line arguments:

```
$ python3 server.py <PORT> <FILE-DIR>
```

- `<PORT>`: port number on which server will listen on connections. The server must accept connections coming from any interface.
- `<FILE-DIR>`: directory name where to save the received files.

For example, the command below should start the server listening on port `5000` and saving received files in the directory `/save`.

```
$ python3 server.py 5000 /save
```

DO NOT open files in "text" mode. All the code you write should directly work with buffer and buffer strings like `b"foobar-I-am-a-buffer-string"`. Nowhere in your program you should use `.decode('utf-8')` or `.encode('utf-8')`. If you do, you probably not going to pass many of the tests

Requirements:

- The server must open a listening socket on the specified in the command line port number on all interfaces. In order to do that, you should hard-code `0.0.0.0` as a host/IP in `socket.bind` method.
- The server should gracefully process an incorrect port number and exit with a non-zero error code. In addition to exit, the server must print out on standard error (using `sys.stderr.write()`) an error message that starts with `ERROR:` string.
- The server should exit with code zero when receiving `SIGQUIT`, `SIGTERM`, `SIGINT` signal
- The server should be able to accept and process multiple connections (at least 10) from clients at the same time
    - To test, you can open multiple `telnet` sessions to your server. All of them should show `accio` command.
- The server must count all established connections (1 for the first connect, 2 for the second, etc.). The received file over the connection must be saved to `<FILE-DIR>/<CONNECTION-ID>.file` file (e.g., `/save/1.file`, `/save/2.file`, etc.). If the client does not send any data during gracefully terminated TCP connection, the server should create an empty file with the name that corresponds to the connection number.

  NOTE DO NOT invent anything else with the file name. The server should name files as `1.file`, `2.file`, etc. in the specified folder REGARDLESS how they were named originally. Your client MUST NOT transmit any additional information, besides what is defined in the specification. The size of the file is equal to all of the received data of the connection!
- The server must assume an error if no data received from the client for over `10 seconds`. It should abort the connection and write a single `ERROR` string (without end-of-line/carret-return symbol) into the corresponding file. Note that any partial input must be discarded.
- The server should be able to accept large enough files (`100 MiB` or more)

Hints

The suggested way to support simultaneous connections is to introduce threading. Overall, the structure should be following:

- Main thread (main program)
    - processes command-line params
    - creates socket, listen, and has a while loop to accept connections
    - accept connection
        - increase global connection counter

        - create a new thread and pass the accepted connection socket and the value of the global connection counter
    - return to waiting for a new connection
- “Child” thread
    - do all the work processing a single connection