GRASS design

Matteo Rizzo, Lucio Romerio, Romain Mendez

This document describes the design of our GRASS implementation.
We build our client and server as 32-bit executables. We used C++ to implement our project. The project has no dependencies other than a C++ compiler that supports C++17 and the C++ standard library. The project uses some Linux-specific functions and as such only works on Linux.

Server

Most of the functionality of the server is implemented in `grass::ServerManager` (`src/server_manager.cpp`) and `grass::Session` (`src/session.cpp`).

The server has a listening socket, which it uses to wait for new connections from clients, and a data socket for each client, which it uses to communicate with the clients. The server is built around an event loop which uses the `select()` system call to block execution until there is an event on any of the sockets. This design allows the server to not busy-wait for new events and at the same time does not require a multi-threaded server, simplifying the design. The only places where the server uses multiple threads are the implementations of `put` and `get`, where the server spawns a worker thread to transfer the file.

`grass::ServerManager` contains the event loop and manages a `Session` object for each connected client. Each instance of `grass::Session` is responsible for handling commands coming from its client. We implemented most of the commands ourselves but resorted to spawning an external process for some. We used POSIX regular expressions, which are implemented in libc, to implement grep.

Client

Similarly to the server, the client is also structured around an event loop. The difference in this case that the client only has to check 2 file descriptors for input (`stdin` and the socket connected to the server). Just like the server, the client only spawns threads to transfer files and it is otherwise single threaded. Because the server does not send the name of the file that is being transferred when it responds to a `put` or `get` request, our client maintains two queues of file names. When the user requests a file transfer, the name of the file is added to the appropriate queue, and it is removed when the server initiates the transfer.