The server application:

Why I used thread pool?

If a server wants to work efficiently, the main thread can only receive new connections and for each connection it creates a thread to handle it. Although creating a thread to implement a task is cheaper than a process, but it is still an operation that takes resources and time. In order to solve this problem, the concept of thread pool was introduced. Instead of creating a thread for each task, the main thread creates a constant number of working threads during its initialization, and they live all the time the server is up. When a task is received it is,inserted to a task list and a working thread is waked up to handle it. When no task is available the working threads will do nothing (wait on a condition variable).

The Grade Server

The server application will be called “GradeServer” and receive 1 parameter in the command line:

- GradeServer *port*
  - *port* – server port number

The server is a multi-threaded application in which the main thread listens for new connections and a pool of working threads handles user connections.

When the server starts, the **main thread** will:

- initialize the data structures of the server.
  - The ids and passwords of the TAs are read from a file called assistants.txt which resides in the same directory of the server application. Each line of the file has a single id:password couple (No spaces, id is nine digits).
  - The ids and passwords of the students are read from a file called students.txt which resides in the same directory of the server application. Each line of the file has a single id:password couple (No spaces).
- Created N=5 working threads which will be ready to handle clients.
- Opened a socket with the port specified in the command line.
- The server will run an infinite loop in which it will accept users’ connections. For each connection a new task is added to the task list and a working thread is waked up to handle the task.
- The server ends only when it receives a kill command or Ctrl-C from the command prompt. The signal handler closes the socket and exit.

All **Working threads** runs the same function. The function runs an infinite loop which it takes a task from the task list and handles the user's queries. When the list is empty the thread will wait on a condition variable until new tasks are inserted. All working threads work with the same global tables and should synchronize access to them.