Assignment 4 Design Document

Seyedsajad Hosseini
Student No.: 101205877

Pseudo-code

1. User runs pps program
   1.1 prompt the user to enter the name of Pokemon database
   1.2 open Pokemon file
   1.3 set up a TCP server socket and wait for clients.
2. User runs pqc program
   2.1 initialize a mutex
   2.2 set up a TCP client socket and connect to the server.
3. pqc prompts the user to enter one of the options a, b, or c
   3.1. when option a (type search) is chosen
      3.1.1 prompt the user for a search word
      3.1.2 create a thread and, in the thread function, send the search word to the server
      3.1.3 in pps
         3.1.3.1 read the Pokemon file line by line
         3.1.3.2 if "type 1" field of a line matches the search word, add the line to a string of saved Pokemon lines
         3.1.3.3 return the string of Pokemon lines to the client
      3.1.4 in pqc
         3.1.4.1 lock the mutex
         3.1.4.1 save response blocks (based on a buffer size) in a loop
         3.1.4.2 unlock mutex
   3.2 when option b (save results) is chosen
      3.2.1 prompt the user for a file name
         3.2.1.1 if a file cannot be opened using the given name, prompt the user to enter the name of the file again

3.2.2 create a thread

3.2.3 add the server response saved in step 3.1.4.2 to the file

3.2.4 if the file name has not been entered already, add it to an array of saved file names

3.3 when option c (exit the program) is chosen

3.3.1 send a message to the server to close the connection with this client

3.3.2 close the client socket

3.3.3 print the total number of queries completed

3.3.4 print the names of the saved files (from the array in 3.2.4)

3.3.5 free allocated memories

3.3.6 exit the pqc program

NFR1 and Multi-threading

The specifications require responsiveness of the program to the user input while queries are being executed or while data is being written to the disk. To achieve this, `pthread` library is used to create two threads: one thread to communicate with the server, and another one to write saved Pokemon records to the file. To prevent data corruption, a mutex is locked and unlocked in threads to control access to the shared data.

NFR2 and File Structure

In order to improve code reusability and modularity, the code is divided into 2 c files and 1 header file for the client-side program, 2 c files and 1 header file for the server-side program, and a shared h file. Server-side files include:

- pps.c: starts the server, listens and responds to clients through functions defined in serverUtil.c.
- serverUtil.c: contains all functions for server-side functionality.
- serverUtil.h: contains functions prototypes for serverUtil.c.

Client-side files include:

- pqc.c: opens the client socket, sends queries to the server, receives responses, and saves data to file through functions defined in clientUtil.c.
- clientUtil.c: contains all functions for client-side functionality.
- clientUtil.h: contains functions prototypes for serverUtil.c and definitions of data structures used in the client-side.

In addition, there is a shared header filed named constants.h that contains constant values used in both server-side and client-side functions. As most these constants (e.g. `SERVER_PORT` and `SEARCH_WORD_SIZE`) are used in both server and client functions, it was decided to use only one header file for both sides. This reduces code duplication and possibility of error due to change in one header file and not applying the same changes in the other file.

Client-Socket Connection

There were two choices for coding the transport layer between client and server sockets: TCP and UDP. Characteristics such as connection-based communication, reliability of data delivery, and correct order of packets delivered resulted in selecting TCP as the protocol of communication for this program. As a result, these libraries were needed to create sockets and transfer data: `sys/socket`, `netinet/in`, and `arpa/inet`.

In this program, the server sends found Pokemon records to the client in the string format. When query does not result in any Pokemon records, the server sends a `NOT_FOUND` macro to the client in order to communicate its failure in finding requested Pokemon type. As sending a positive number or any word could be interpreted as part of a Pokemon record, `NOT_FOUND` message is set to be “-1”. Therefore, when “-1” is received by the client, it breaks the infinite loop of receiving data and returns to the user.

When the user selects option c to exit the pqc program, it sends a "disconnect" message to pps to disconnect from the server so that other clients can connect to the server. The "disconnect" message is saved in the `DISCONNECT` macro.