Semester Project

CS 344

Spring 2019

Alex Hall, James Trafny, Matthieu Privat

Contents:

1. Lessons Learned
2. clientWithMenu.c (Dr. Bethelmy, Alex Hall, James Trafny, Matthieu Privat)
3. DieWithError.c (Dr. Bethelmy)
4. linkedList.c (James Trafny)
5. transmssion.c (Dr. Bethelmy, Alex Hall, James Trafny, Matthieu Privat)
6. createProject.c (Alex Hall)
7. editProject.c (Alex Hall)
8. HandleTCPClientWithMenu.c (Alex Hall, James Trafny, Matthieu Privat)
9. serverWithMenu.c (Alex Hall, James Trafny, Matthieu Privat)
10. linkedList.h (Alex Hall, James Trafny, Matthieu Privat)
11. menu.h (Alex Hall, James Trafny, Matthieu Privat)
12. projectStructure.h (Alex Hall, James Trafny, Matthieu Privat)

Lessons Learned

Over the course of this project, a few things stood out as lessons to all of the group members. The first and foremost of these being the importance of modular code. When the code is modular, it helps with the workload since certain functions can simply be called from before. It also helps with group work as multiple members of the group could work on different methods of the same file without stepping too much on each other's toes. Writing the code in modular blocks of functions also helped a lot with debugging, and it was much quicker to isolate the problems encountered. We encountered issues early on because we were not designing with modularity in mind. Towards the end of the project we got better at this, but some functions will stand out for being similar to other functions with the exception of the datatypes (see linkedList.c).

The group has also learned lessons on integration and working together on GitHub in general. For instance, we underestimated just how much time would be needed to integrate our individual work. When each of us set out to do our individual work, we had begun to rewrite code that others depended on; for example: when James was working on his code for file I/O he had to change the project structure to include a pointer that wasn’t there before, this effected the way that Alex and Matt’s code worked and caused conflicts after each merge. This took us a little longer to realize than it should have, and we only came together at the very end.

Time management was a big problem for us early on in this project. Each member of the group was involved with multiple projects and our class and work schedules did not mesh well together. The only time we found to work together was on Saturdays, but this caused a lot of time in between meetings where work was either being delayed until others finished their parts, or work was being done that caused problems after the merges. As discussed earlier, better design could have helped with the modularity, but it is our belief that poor time management on our part played a large role in us not hitting every requirement.

We also learned that we should only be changing on thing at a time, and start small, then extend. While we were developing, often times we found ourselves going too long between testing and had to rework a lot of code at a time, or had trouble finding problems in what we just wrote because we went to far without tests. We should have been refactoring the code each time before we added the next feature to prevent “spaghetti code” from getting unrulily. We also learned that everything takes longer than you think. We routinely ran into unexpected bugs and errors that required huge amounts of time to fix before we could move on; we should have accounted for this from the beginning.

Overall, we are dissatisfied with the final product that we are delivering; however, we all agree that we learned a lot more about how to work together as a team, and how hard managing large systems can be. If we had more time (or if we had started earlier more earnestly,) we feel that we could eventually meet the requirements laid out for this project; we feel that we are submitting more of a “walking skeleton” or proof-of-concept as opposed to a finished product.

clientWithMenu.c

This file prints a menu to the client regarding options they may choose for their project handling.

This can be either creating a project, saving a project, or anything else of the sort.

It contains the functions:

**main():** This function creates our TCP connection with the server and calls the talkToServer() method.

**talkToServer():** This function allows the user to choose an option from the

displayMenuAndSendSelection() method.

**displayMenuAndSendSelection():** A method that displays the menu of options for the user when called.

**sendProjectInformation():** A brief method that sends the information on a project to the client for them to view.

**sendProjectID():** A method that prompts the client to enter an ID number for the current project.

**sendProjectDescription():** A method that sends a description of the project.

**sendProjectCreationDate():** A method that prints the date that the project was created.

**sendProjectDueDate():** A method that prints the date that the project is due.

**sendProjectMemberNum():** A method that send the number of members working on a given project.

DieWithError.c

This file terminates the program when called with an error message.

It contains the function:

**DieWithError():** A method that will exit the program and print an error message.

linkedList.c

Functions supporting LINKED\_LIST structures. Includes functions for adding and removing nodes at specific locations as well and displaying or printing contents to and from a file.

It contains the functions:

**append():** Takes in a linked list and the data that the new node will point to and appends it to the front of the list.

**prepend():** Takes in a linked list and the data that the new node will point to and prepends it to the rear of the list.

**insert():** Takes in a linked list and data to be inserted at the specified position.

**removeFromFront():** Takes in a linked list and removes the front node.

**removeFromBack():** Takes in a linked list and removes the rear node.

**removeAtPosition():** Takes in a linked list and removes the node at the specified position.

**printProject():** Takes in a pointer to a project structure to print.

**sendToFile():** Takes in a pointer to a list of projects and prints it out to the location of the file pointer.

**loadFromFile():** Creates a new project\_list from the contents of a file.

transmission.c

This file runs the transmission between the server and client, and keeps track of the bytes sent between the two.

It contains the functions:

**get():** This method receives bytes from the sender and tallies the total amount of bytes received. It also will terminate if there is an error by calling DieWithError().

**put():** A method that terminates the program if the received buffer is not equal to the size of the message.

createProject.c

This file is used by the server to create and return a new project (PROJECT\_STRUCT) to be added to project\_list for use in the server.

It contains the functions:

**createProject():** Takes a socket and returns a new project to the caller.

**askForProjectID():** Takes a socket, data pointer, and size of that pointer and asks to user for the ID of the project number.

**askForProjectDescription():** Takes a socket, data pointer, and size of that pointer and asks to user for a description of the project.

**askForProjectDateCreated():** Takes a socket, data pointer, and size of that pointer and asks to user for the creation date of the project.

**askForProjectDateDue():** Takes a socket, data pointer, and size of that pointer and asks to user for the due date of the project.

**askForMemberNum():** Takes a socket, data pointer, and size of that pointer and asks to user for the number of members of the project.

editProject.c

Server side for editing a project. Requires the client socket and also a LINKED\_LIST of project structures to index into.

It contains the functions:

**editProject():** Takes in a user socket and position within the list of projects to which project will be edited.

**sendProjectDataAndWaitForResponse():** Sends editing options to client.

HandleTCPClientWithMenu.c

This file is called after a connection has been made to the client. The program flow here goes: send menu, wait for response, trigger fn based on respone, send new menu, repeat.

It contains the functions:

**HandleTCPClient():** Takes in a user socket and sends it a menu, after sending the menu it waits for a response and triggers a function based on the response.

**SendMenuAndWaitForResponse():** Takes in a user socket and sends it a menu.

serverWithMenu.c

This file is the main entry point for the server. The server sets itself up for TCP connection, then waits for a connection. Once a connection is recieved, the server spawns a child process to handle the client communications so the server can await another request to handle.

It contains the functions:

**main():** see file description.

linkedList.h

Header file for linked lists. Contains structures and prototypes needed for DLL manipulations.

This contains no functions.

menu.h

Header file menu structures.

This contains no functions.

projectStructure.h

Header file project structures.

This contains no functions.