Personal Project - Tasksify

Adam Wu

Personal Project - Summer 2023

1 Purpose

The aim of this project is to create a usable task manager that works on a local machine that is efficient [1]. Additionally, I am trying to utilize the skills I have learned from my previous class, cse13s, and implementing my own projects. This will be a challenge project for me! For those who are reading this report, if you play around with my program and want to help or give suggestions, feel free to let me know! I will be happy for those who want to collaborate with me to improve this project!

2 How to Use the Program

The first step will be to get all the files from my Github repository in order to use the program. To retrieve all the files from my Github repository, you can type *git clone https://github.com/ATOMiNATiON/tasksify.git* into your terminal. By cloning my repository, you will have all the files necessary to use my program. The next thing you will need to understand is compiling my program, running my program, and cleaning up the files generated during compilation. There will be 3 subsections below about how to compile the program, run the program, and removing the files during the compilation of the program.

2.1 Compiling the program

Once you have cloned the repository, you can now compile the program to create the executable file. To compile, just type *make* in your terminal (make sure you are in the correct directory when you type *make*). After typing *make*, you should see a new file in the directory called *task*. If you have that, then we can move on to the next step.

2.2 Running the program

To use this program, type *./task -[OPTIONS]*, the OPTIONS for this programs are [”a”, ”l”, ”r”, ”h”]. I will make a list of what each option does below:

- -a taskname.......This option allows you to ADD a specified task to your todo list
- -l ....This option allows you to LIST out the task(s) on your todo list
- -r tasknumber.....This option allows you to REMOVE a specified task on your todo list
- -h ....This option will present you the help menu, showing you how to use the program

These are currently the basic options that I have implemented, I will demonstrate below on how you would add a task below:

[1] This project was inspired by ”taskwarrior”

Usage example

```
$ ./task -a "Math homework"
Adding a task...
Math homework
$
$ ./task -l
Listing out all tasks...
#1: Math homework
$
```

2.3 Removing files

To remove the files that you do not wish to see after the *make* of this file, you may type: *make clean*, to clean up your directory for what it looked like before. But if you want to run this program again, you **will** need to *make* it again.

3 Program Design

I am currently planning to have 3 files that will work alongside each other to make this program work. The first file is the main file called *task.c*, the second file will be called *sll.c* which is the singly linked list file holding the abstract data types and its functions, and the third file will be called *sll.h* which is the linker file for the singly linked list to be linked to the main file.

Once the program is designed in a more usable way, I will update this design document. This will be my journey in creating my own program that I have learned from my cse13s class (from UC Santa Cruz).

3.1 Data Structures

For this program, I used singly linked list abstract data type to hold the tasks in order. I chose to use this because it would be easy to implement as well as being efficient comparing it to using a stack. There was a challenge when doing going through each task to find the correct task number throughout the list. Currently thinking about it, if I were to have a big list of task and needing to find a certain task, it may take a long time and inefficient. Therefore, in the next update, I might have to change the data structure to using a stack perhaps.

For the singly linked list data type, it contained three important information. The first is the task id for whichever task is being assigned, the second is the task name, and the third is where the next item is being pointed to. This data structure was created in the sll.c file, and the sll.h file will allow the main file to connect with the sll.c file. I should also mention that there is a *.taskrc* file that I created, this file is meant to store the tasks that users add and delete. I purposely made it *.taskrc* instead of something like *task.txt* so that normal users would not be able to see this file, unless they show hidden files as well. Since the *.taskrc* file contains the tasks, one should push the *.taskrc* file being empty.

3.2 Algorithms

The only algorithm used was traversing through the linked list to find the correct task to remove, which is something very basic. Though in the future, I could perhaps implement something like a priority queue for the tasks. If that were to happen, then some type of algorithm could be used, such as heap sort.

3.3 Function Descriptions

For this project, the functions are mainly built in the sll.c file. I will describe to you what each function does, so that future me or future programmers who would like to improve this project may do so.

void free_list(List **list);

This function takes in a List structure as a double pointer so that I can free the list items that I used for allocating memory and so that I can free the object itself after.

void list_tasks(List *list);

This function takes in a List structure and displays the list of tasks by printing it to the terminal. It gathers the task by reaching into the ".taskrc" file (NO ONE SHOULD TOUCH THIS FILE AS IT STORES THE TASKS).

void add_task(const char *taskname, List **list, const char *filename);

This function takes in a taskname, List object, and a filename. It's purpose is to add a task to the current list. It does so by adding it to the linked list, but also updates the ".taskrc" file.

void remove_task(int task_id, List **list);

This function takes in a task id and a list object. The purpose of this function is to remove a task from the current list from the linked list but also updates the ".taskrc" file.

4 Results

Currently, the results seems fine. I have not thought about any edge cases yet, but for now as I have implemented the three basic functions of a to-do list and works well for me. I will show some images of how it is working so far.

4.1 Error Handling

It currently handles errors like users inputting invalid command line options (CLI) to the terminal. It also handles errors such as not giving any CLI. For these errors that happen, I have made the program to display the help menu if any of these cases happen. As a programmer, if anyone else finds an edge case where there is an error not being handled properly, please let me know. I will fix it as soon as possible.