CSCI320: Homework 4

Due: Friday, February 23rd.

General Information

Your solutions to the homework assignment should be submitted electronically on Canvas.

- Please submit a ZIP folder containing your files.
  - Only include the source code.
  - Include a README with your name and student ID on two lines, and additional information on lines below. (Collaboration Information).
- Coding Guidelines:
  - Minimize compiler warnings! Be careful with casts!
  - Remember to free memory appropriately
  - **Make sure your program compiles!**

Collaboration Policy

- I expect you might discuss the homework with other members of the class. This is allowed. However, note the other policies!
- I expect you to write your own code.
- You may collaborate, look over code, hunt for bugs, etc, but *you should be the only one typing your own code.*
- If you assist or collaborate with other class members on coding, please note it in the README file.
- (You will not be penalized for this.)

1 Linked List. (14 pts)

A linked list is a fundamental data structure in computer science. In this problem, you will implement a doubly linked list in C which uses synchronization appropriately in order to be thread-safe.

You should create your own program in *linkedlist.c* which implements the appropriate functions for a linked list.

Note that there are some files provided to you. These test files are provided to give you some basic checks before you turn in your code. They are not very extensive tests (and do not really test for all kinds of race conditions). It is up to you to write your code to avoid bad behaviors.

Nevertheless, at a minimum, the provided test files should not produce any strange results!

- *linkedlist.h* is a header file which defines the function signatures you should use in your *linkedlist.c* file.
- *test_basics.c* contains a short, basic test of the features of your linked list.
- *test_threaded.c* contains a longer, threading based test of inserts/deletes.

Implement your linked list with functions that match the appropriate ones in the header file.

- The header file has an intentionally vague definition of a list node. You should think about what sort of information each node must have. Remember, it should be a doubly linked list.
- At some point, you will need to include locks for the list. Think about how you can have a lock associated with a particular list!
- Your list should have a sentinel *head* node which is before any real item in the list.
- Every list node should be dynamically allocated (use malloc appropriately).
- You must implement all the functions specified by the header file.
  - Each function acts upon a void pointer to the head of the list, along with other appropriate parameters. The void pointer indicates *which* list to change, if there are multiple lists.
  - **Create** should create a new, empty list which can be referenced by the pointer it returns.
  - For the function with the Safe variants (Insert and SafeInsert, Delete and SafeDelete, etc), the Safe variants should be thread-safe if used in conjuction with the other Safe functions.
  - **Display** should display a string that is a good representation of the list. It should also return the length of the list.
  - **Destroy** should destroy every node in the list by freeing the memory they correspond to, allowing the operating system to regain possession of their memory.
  - **Insert** and **SafeInsert** should insert an item at the index passed in to the function. If the location is beyond the end of the list, it should append a new node to the end. It should return 1 when successful.
  - **Delete** and **SafeDelete** should delete an item at the index passed in to the function. For these functions, if the index is beyond the end of the list, it should remove the last element. If it succeeds in deleting something, it should return 1. If there is nothing to remove, it should just return 0.
  - **Find** and **SafeFind** acts on a target value. If the target value is within the list, it returns a pointer to the corresponding node. Otherwise it should return null.
- Remarks
  - Be careful with memory allocation and freeing!
  - Be careful with casting pointers to other types of pointers!
  - Think through your code logically! Races might just happen to not show up in your testing, but if it shows up in mine you will lose points!

2 Hints

- If you don't remember the commands to compile:

  ```
  gcc -Wall -lpthread <program_name.c> -o <output_name>
  ```

- To compile multiple files which must be linked together, include multiple files in the command line:

  ```
  gcc -Wall -lpthread <program_part1.c> <program_part2.c> -o <output_name>
  ```

- ***Data types are important!***