Extra Credit Programming Assignment #3

This assignment gives you some practice with threads and synchronization (locks). The goal is to create a program that will implement a single linked list. The difference this time, is that the code should be thread-safe, meaning, it should function properly when multiple threads are inserting and removing nodes from the list. You should use the first assignment, threadlist, as a starting point. (Actually, any single linked list code that **YOU** wrote can be used as a starting point.) The list is really only implementing to functions: *insert* and *remove*. The insert function keeps the list sorted by placing the item in the correct location in the list. That's why it's *insert* and not *push_front* or *push_back*. To keep things simple, the data in the list are just integers.

The goal, as before, is not to gain any great insight into linked-lists, but to learn the process of coding multi-threaded linked list. And to keep the complexity to a minimum, that's why the lists will be singly-linked. This is the definition of the node that will be used by the single-linked list known as a *threadlist*:

```
#define LABEL_SIZE 16
typedef struct tl_node
{
  char label[LABEL_SIZE];
  int value;
  struct tl_node* next;
}tl_node;
```

Each node (*tl_node*) in a list contains a text label, an integer, and a pointer to another `tl_node`. This node structure is intentionally kept simple so you can focus on the list aspect of this assignment. This is the interface:

```
void tl_insert(int value, const char *label);
void tl_insert_values(int *values, int count, const char *label);
void tl_remove(int value);
void tl_remove_values(int *values, int count);
void tl_resetlist(int *values, int count);
void tl_destroy(void);
tl_node * const *tl_getlist(void);
void tl_uselocks(bool enable);
void tl_dump(const tl_node *list);
```

In the previous assignment, all of the functions took a list (a pointer to the first node) as their first parameter. This time, all of the functions work on the same list. This list is defined in threadlist.c, which is solely responsible for it. If every thread had its own list, this would be a trivial assignment. We need them to share a "global" list. This is actually a static variable in threadlist.c called gTList defined at the top of the file. The functions in **bold** require protection (locks):

Function	Description
`tl_insert`	Insert a new node into the proper position based on its value.
`tl_insert_values`	Given an array of values, insert them into the list. This function just calls *tl_insert* multiple times.
`tl_remove`	Remove the item from the list that matches the value given. It will only remove the first occurrence of the value. Be sure to free the memory that was allocated for the node. If the list does not contain the value, nothing is removed.
`tl_remove_values`	Given an array of values, remove them from the list. This function just calls *tl_remove* multiple times.
`tl_resetlist`	Sets the "private" list to NULL. This is not thread-safe. It should only be called from the main thread in the driver.
`tl_getlist`	Retrieves a pointer to the head of the list (a pointer to a pointer). This is not thread-safe. It should only be called from the main thread in the driver.
`tl_uselocks`	Enables/disables use of locking. This is not thread-safe. It should only be called from one thread (the main thread in the driver) before any other threads are created.
`tl_destroy`	Walk the list and free all of the nodes. This is not thread-safe. It should only be called from one thread (the main thread in the driver) after all threads have finished.
`tl_dump`	Print out the list. **This is implemented for you. Do not change it.**

The actual linked list part of this assignment is trivial. I don't expect any third-semester Digipen student to struggle with any of this. The only real effort is in implementing a *correct* and *relatively efficient* locking strategy. As was stated in the first assignment (cachelist), that assignment was designed so that when the memory manager assignment (basically, a single linked list assignment) came along, you already had a working single linked list program to build from. That's happening again with this threadlist assignment.

More on back →

Other criteria (mostly from the as-usual department)

1. When you create helper functions, they must be tagged as **`static`.** In C++, you would put them either in the private section of a class, or in an unnamed namespace in the implementation file. Neither of those exist in C, so you must use the **`static`** keyword.
2. Be sure to free all of the memory that you allocate. This should be easy for you to verify because you are required to use *valgrind* to look for memory leaks. I will post a command line that you should use.
3. Be sure to check the return values of all of your library calls (e.g. malloc). Failure to do so will result in a lower score, regardless of whether or not the call was successful. Remember, this is a third-semester course and failing to check return values will not be tolerated at all. YOU HAVE BEEN WARNED. Hint: You should only have one place in your code where you call malloc.
4. Any function that "touches" the list needs synchronization. These are *insert* and *remove*.
5. Do not lock code that doesn't need to be locked (you will lose points if you do). For example, local variables and function parameters don't have to be protected. Only when your code is "touching" shared data (e.g. the list itself). You don' t have to have to protect any of the C library functions (e.g. strncpy, malloc, printf, etc.) as these are already using locks. Don't double lock!!
6. Do not call the synchronization primitives (e.g. *pthread_mutex_lock*) directly in your code. Put them in wrapper functions as demonstrated in the notes on Synchronization posted on the website. Then call these functions from the places in your code that you need to protect.

Deliverables
You must submit the implementation file (`threadlist.c`), the `typescript` file, and the generated `refman.pdf` file. These files must be zipped up and submitted to the appropriate submission area. Notice that you are not submitting the header file, so don't make any changes to that file.

Files	Description
`threadlist.c`	The implementation file. All implementation for the functions goes here. You must document the file (file header comments) and all functions (function header comments) using the appropriate Doxygen tags.
`refman.pdf`	This is the PDF documentation that was generated by *Doxygen* and *pdflatex*.
`typescript`	This is the session that was captured with the `script` command.

Usual stuff
Your code must compile (using the compilers specified) to receive credit. The code must be formatted as per the documentation on the website.

Make sure your name and other info is on all documents.