Assignment 2

Objective

In this assignment, we will explore the use of locks in OpenMP. We will use what we learned about locking, to implement a thread-safe linked list.

Background

A linked list is data structure that organizes a set of elements, called nodes, in a linear sequence. Linked lists do not keep the sequence of nodes in a contiguous location in physical memory as arrays do. Instead, each node is a separate entity that also contains a pointer to the next node in the list. Each node has two fields: a data field which contains the data, and a link or reference field, which links it to the other elements in the sequence.

The simplest example of a linked list is a singly linked list. Each node in a singly linked list keeps a reference to the next node, as seen in Figure 1. The list shown contains 3 nodes. The first node (data = 12) links to the second (data = 99), the second to the third (data = 37), and the last node (data = 37) links to null.

The power of linked lists lies in the fact that they are allocated dynamically without any limitations at compile time. They grow and shrink in memory footprint as nodes are added and deleted at runtime. It is also easy to add and delete nodes at any position by rearranging the link field among nodes. However, this flexibility comes at the expense of the extra storage needed for this link field.

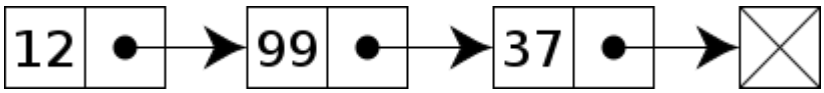

Figure 1: An example of a singly linked list

Assignment

We want to implement a thread-safe singly linked list. A thread-safe linked list means that multiple threads can operate on the linked list in parallel without affecting functionality and correctness of the lists' operations. In the assignment file, we have given you a working implementation of a non-thread-safe singly linked list in C. Your assignment is to augment the implementation, using only

OpenMP locks, to make the implementation thread-safe. In this assignment, you are not allowed to use barriers, critical sections, or atomics.

The linked list operations are:

- `init_list`: initializes a new linked list.
- `append`: inserts a node at the end of the list.
- `add_first`: inserts a node at the beginning of the list.
- `insert`: inserts a node at the index specified by the input.
- `pop`: removes the node at the beginning of the list.
- `remove_by_index`: removes the node at the index specified by the input.
- `remove_by_value`: removes the first node with a value equals to that specified by the input.
- `count`: returns the number of nodes in a list.
- `search_by_value`: returns the index of the first node whose value equals to that specified by the input.
- `print_list`: prints all elements in a list.
- `delete_list`: deletes a list along with its elements.

Development infrastructure

On the course's Moodle page, we have provided the tarball A3.tgz, which contains:

- `assignment3.c` – the file contains:
  - The structure of each linked list node.
  - The implementation of the non-thread-safe linked list functions.
  - `main()` – which provides a sample use of the linked list functions.

  You must augment the linked list functions in this file with OpenMP locks to ensure a thread-safe implementation of the linked list.
- `Makefile` – which compiles the environment using the current gcc version; please refer to Assignment 1 to make sure the `Makefile` is suitable for your environment.

Deliverables

You need to turn in a tarball archive called `a3_<yourSCIPER>.tgz` to the Moodle link specified for Assignment 3, containing:

- An augmented version of `assignment3.c` that is thread-safe with mandatory comments on your augmentation.
- A report, named `a2_<yourSCIPER>.pdf`, that complies to the description that follows.

The tarball should not contain a directory. Create your tarball using:

```
$tar -cvzf a3_<yourSCIPER>.tgz /path/to/a3_<yourSCIPER>.pdf /path/to/assignment3.c
```

Report

The report should be a one-page explanation of the lock primitives that you added to ensure that the linked list is thread-safe. The explanation should first describe the requirements of a thread-safe linked list, and then describe how your additions apply to each operation.

Grading

We will read your submissions (report and code) and grade them based on:

- The correctness, conciseness, and accuracy of the report in describing the requirements.
- The correctness of your code.
- The correctness, conciseness, and accuracy of the comments in your code.