A Concurrent Stack

Task

Implement the abstract data type Stack such that multiple threads can perform the operations push and pop concurrently.

Lock the Whole Stack

Using a semaphore or a monitor.

```
Stack : monitor
begin
  ...
  procedure push(number : int)
  begin
    ...
  end
  procedure pop(result number : int)
  begin
    ...
  end
  ...
end
```

Locks: Number and Granularity

Reducing the number and length of sequentially executed code sections is crucial to performance. In the context of locking, this means

- reducing the number of locks acquired, and
- reducing lock granularity, a measure of the number of instructions executed while holding a lock.

Lock the First Node

Implement the stack as a linked list and only lock the first node of the list.

Memory Contention

This solution suffers from memory contention: an overhead in traffic in the underlying hardware as a result of multiple threads concurrently attempting to access the same locations in memory. If the lock protecting the node is implemented in a single memory location, as many simple locks are, then in order to acquire the lock, a thread must repeatedly attempt to modify that location.

Blocking

In any solution that uses locks, if a thread that holds a lock is delayed, then all other threads attempting to get the lock are also delayed. Therefore, this (and the previous) solution is called blocking.

Do Not Lock

Instead of locks, use synchronization instructions, such as compare-and-swap (CAS) and load-linked/store-conditional (LL/SC). All modern processors provide such instructions.

Compare-And-Swap (CAS)

The operation CAS(variable, expected, new) atomically

- loads a value of variable,
- compares that value to expected,
- assigns new to variable if the comparison succeeds, and
- returns the old value of variable.

Distributed Computing

The graduate course CSE 6117 entitled Distributed Computing studies non-blocking algorithms and their properties in detail.