Bibliography

Question

How do you cite a journal article?

Bibliography

Question

How do you cite a journal article?

- The names of the authors (for example, all separated by commas apart from the last two which are separated by “and”).
- The title of the article.
- *The title of the journal*, volume(number): pages, month, year.

Bibliography

Question

How do you cite a journal article?

- The names of the authors (for example, all separated by commas apart from the last two which are separated by "and").
- The title of the article.
- *The title of the journal*, volume(number): pages, month, year.

Example

Carla Schlatter Ellis. Concurrent Search and Insertion in AVL Trees. *IEEE Transactions on Computers*, 29(9):811–817, September 1980.

Question

How do you record a journal article in BiBTeX?

```
@article{Ellis80,
  author  = "Carla Schlatter Ellis",
  title   = "Concurrent Search and Insertion in
             {AVL} Trees",
  journal = "IEEE Transactions on Computers",
  volume  = "29",
  number  = "9",
  pages   = "811--817",
  month   = sep,
  year    = "1980"}
```

Bibliography

Question

How do you cite a paper in a conference proceedings?

Question

How do you cite a paper in a conference proceedings?

- The names of the authors (for example, all separated by commas apart from the last two which are separated by "and").
- The title of the paper.
- In, the names of the editors of the proceedings (for example, all separated by commas apart from the last two which are separated by "and"), *the title of the proceedings*, volume of *title of the series*, pages, location where the conference was held, month, year.
- Publisher.

Bibliography

Example

Nathan G. Bronson, Jared Casper, Hassan Chafi, and Kunle Olukotun. A practical concurrent binary search tree. In R. Govindarajan, David A. Padua, and Mary W. Hall, editors, *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 257–268, Bangalore, India, January 2010. ACM.

Question

How do you record a paper in a conference proceedings in BiBTeX?

```
@inproceedings{BronsonCasperChafiOlukotun10,
  author    = "Nathan G. Bronson and Jared Casper
               and Hassan Chafi and Kunle Olukotun"
  title     = "A practical concurrent binary search
  booktitle = "Proceedings of the 15th ACM SIGPLAN
               on Principles and Practice of Parall
  year      = "2010",
  editor    = "R. Govindarajan and David A. Padua a
  pages     = "257--268",
  address   = "Bangalore, India",
  month     = jan,
  publisher = "ACM"}
```

Concurrent Object Oriented Languages

Non-blocking synchronization

wiki.eecs.yorku.ca/course/6490A

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.

Lock the Whole Stack

Using a semaphore.

```
semaphore mutex = 1
node top = null

push(e):
P(mutex)
new = node with element e
new.next = top
top = new
V(mutex)
```

Lock the Whole Stack

Using a semaphore.

Lock the Whole Stack

Using a semaphore.

```
pop:
P(mutex)
if (top == null)
    V(mutex)
    return EMPTY
else
    temp = top
    top = top.next
    V(mutex)
    return element of temp
```

Lock the Whole Stack

Using a monitor.

Lock the Whole Stack

Using a monitor.

```
Stack : monitor
begin
  node top

  procedure push(number : int)
  begin
    new = node with element number
    new.next = top
    top = new
  end
```

Lock the Whole Stack

Using a monitor.

Lock the Whole Stack

Using a monitor.

```
procedure pop(result number : int)
begin
  if (top == null)
    number = EMPTY
  else
    number = element of top
    top = top.next
end

top = null
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

Only lock the first node of the list.

Lock the First Node

Only lock the first node of the list.

```
node top = dummy

push(e):
new = node with element e
lock(top)
new.next = top
lock(new)
temp = top
top = new
unlock(temp)
```

Lock the First Node

Only lock the first node of the list.

Lock the First Node

Only lock the first node of the list.

```
pop():
lock(top)
if (top == dummy)
   unlock(top)
   return EMPTY
else
   number = element of top
   temp = top
   top = top.next
   unlock(temp)
end
```

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

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

ABA Problem

The ABA problem occurs during synchronization, when a location is read twice, has the same value for both reads, and “value is the same” is used to indicate “nothing has changed”. However, another thread can execute between the two reads and change the value, do other work, then change the value back, thus fooling the first thread into thinking “nothing has changed” even though the second thread did work that violates that assumption.

source: wikipedia

ABA Solution

A general solution to the ABA problem is to use a double-length CAS (e.g. on a 32 bit system, a 64 bit CAS). The second half is used to hold a counter. The compare part of the operation compares the previously read value of the variable and the counter, to the current value and counter. If they match, the swap occurs - the new value is written - but the new value has an incremented counter. This means that if ABA has occurred, although the value of the variable will be the same, the counter is exceedingly unlikely to be the same (for a 32 bit value, a multiple of 2^{32} operations would have had to occurred, causing the counter to wrap and at that moment, the value of the variable would have to also by chance be the same).

```
push(e):
new = node with element e;
do
  temp = top;
  new.next = temp;
while (CAS(top, temp, new) != temp);

pop():
do
  temp = top;
  if (temp is undefined)
    return EMPTY
while (CAS(top, temp, temp.next) != temp);
return element of temp;
```