WIKIPEDIA

Hazard pointer

In a multithreaded computing environment, **hazard pointers** are one approach to solving the problems posed by dynamic memory management of the nodes in a lock-free data structure. These problems generally arise only in environments that don't have automatic garbage collection.[1]

Any lock-free data structure that uses the compare-and-swap primitive must deal with the ABA problem. For example, in a lock-free stack represented as an intrusively linked list, one thread may be attempting to pop an item from the front of the stack (A → B → C). It remembers the second-from-top value "B", and then performs `compare_and_swap(target=&head, newvalue=B, expected=A)`. Unfortunately, in the middle of this operation, another thread may have done two pops and then pushed A back on top, resulting in the stack (A → C). The compare-and-swap succeeds in swapping `head` with `B`, and the result is that the stack now contains garbage (a pointer to the freed element "B").

Furthermore, any lock-free algorithm containing code of the form

```
Node* currentNode = this->head;  // assume the load from "this->head" is atomic
Node* nextNode = currentNode->next;  // assume this load is also atomic
```

suffers from another major problem, in the absence of automatic garbage collection. In between those two lines, it is possible that another thread may pop the node pointed to by `this->head` and deallocate it, meaning that the memory access through `currentNode` on the second line reads deallocated memory (which may in fact already be in use by some other thread for a completely different purpose).

Hazard pointers can be used to address both of these problems. In a hazard-pointer system, each thread keeps a list of hazard pointers indicating which nodes the thread is currently accessing. (In many systems this "list" may be probably limited to only one[1][2] or two elements.) Nodes on the hazard pointer list must not be modified or deallocated by any other thread.

> Each reader thread owns a single-writer/multi-reader shared pointer called "hazard pointer." When a reader thread assigns the address of a map to its hazard pointer, it is basically announcing to other threads (writers), "I am reading this map. You can replace it if you want, but don't change its contents and certainly keep your `delete`ing hands off it."
>
> — Andrei Alexandrescu and Maged Michael, Lock-Free Data Structures with Hazard Pointers[2]

When a thread wishes to remove a node, it places it on a list of nodes "to be freed later", but does not actually deallocate the node's memory until no other thread's hazard list contains the pointer. This manual garbage collection can be done by a dedicated garbage-collection thread (if the list "to be freed later" is shared by all the threads); alternatively, cleaning up the "to be freed" list can be done by each worker thread as part of an operation such as "pop" (in which case each worker thread can be responsible for its own "to be freed" list).

In 2002, Maged Michael of IBM filed an application for a U.S. patent on the hazard pointer technique,[3] but the application was abandoned in 2010.

Alternatives to hazard pointers include reference counting.[1]

See also

- Concurrent data structure
- Hazard (computer architecture)
- Finalizer

References

1. Anthony Williams. *C++ Concurrency in Action: Practical Multithreading.* Manning:Shelter Island, 2012. See particularly Chapter 7.2, "Examples of lock-free data structures".
2. Andrei Alexandrescu and Maged Michael (2004). "Lock-Free Data Structures with Hazard Pointers" (http://www.drdobbs.com/lock-free-data-structures-with-hazard-po/184401890). *Dr Dobbs*. (C++ oriented article)
3. US application 20040107227 (https://worldwide.espacenet.com/textdoc?DB=EPODOC&IDX=US20040107227) Maged M. Michael, "Method for efficient implementation of dynamic lock-free data structures with safe memory reclamation." Filed on 3 December 2002.

- Maged Michael (2004). "Hazard Pointers: Safe Memory Reclamation for Lock-Free Objects" (http://www.research.ibm.com/people/m/michael/ieeetpds-2004.pdf) (PDF). *IEEE Transactions on Parallel and Distributed Systems*. **15** (8): 491–504. doi:10.1109/TPDS.2004.8 (https://doi.org/10.1109%2FTPDS.2004.8).

External links

- Concurrent Building Blocks (http://amino-cbbs.sourceforge.net/) - C++ implementation of Hazard Pointer (called "SMR") and other lock-free data structures. Also has Java interfaces.
- Concurrency Kit (http://concurrencykit.org/) - C implementation of Hazard Pointer and lock-free data structures
- Atomic Ptr Plus (http://atomic-ptr-plus.sourceforge.net/) - C/C++ library that has a Hazard Pointer implementation
- The parallelism shift and C++'s memory model (http://www.johantorp.com/) - Contains C++ implementation for Windows in appendices
- libcds (http://libcds.sourceforge.net/) - C++ library of lock-free containers and Hazard Pointer implementation