

**BCS2213 FORMAL METHODS**

**SEMESTER 1 SESSION 2015/2016**

**GROUP PROJECT**

**TITLE: MODELLING HAZARD POINTERS**

**LECTURER:**

**PROF. DR. VITALIY MEZHUYEV**

**GROUP MEMBER:**

|  |  |  |
| --- | --- | --- |
| **No** | **Names** | **Matric ID** |
| **1.** | **SYED NABIL NAIM BIN SYED MOHD AMUDIN** | **CB14010** |
| **2.** | **AMIRAH BINTI MOHAMAD** | **CB14018** |
| **3.** | **KHAIRUNNISA BINTI CHE HASSAN** | **CB14139** |
| **4.** | **LAKHVINDER KAUR MAHINDER SINGH** | **CB141136** |

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| No | Contents | Page Numbers |
| 1. | Introduction | **3 - 5** |
| 2. | TLA Model | **6 - 9** |
| 3. | UPPAL Model | **10 - 12** |
| 4. | Conclusion | **13** |
| 5. | Reference | **14** |

**INTRODUCTION**

Hazard pointers are one approach to solving the problems posed by [dynamic memory management](https://en.wikipedia.org/wiki/Dynamic_memory_management) of the nodes in a [lock-free](https://en.wikipedia.org/wiki/Non-blocking_algorithm) [data structure](https://en.wikipedia.org/wiki/Data_structure). Arise only in environments that don't have [automatic garbage collection](https://en.wikipedia.org/wiki/Automatic_garbage_collection) any lock-free data structure that uses the [compare-and-swap](https://en.wikipedia.org/wiki/Compare-and-swap) primitive must deal with the [ABA problem](https://en.wikipedia.org/wiki/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. In a hazard-pointer system, each [thread](https://en.wikipedia.org/wiki/Thread_(computer_science)) keeps a [list](https://en.wikipedia.org/wiki/List_(computing)) of hazard pointers indicating which nodes the thread is currently accessing. Nodes on the hazard pointer list must not be modified or deal located 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 deleting hands off it."

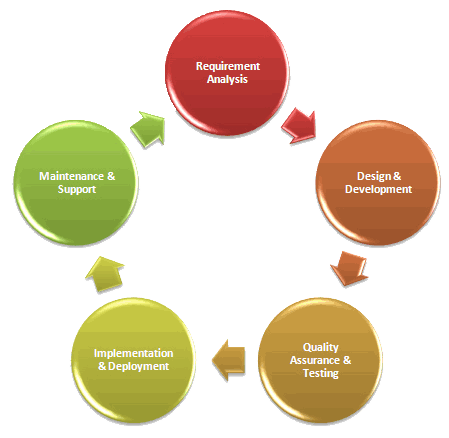
**Problem Statement**

1. The condition of at least one node of the hazard pointer has be continuously holding that reference from the period where the node was definitely safe for the thread.
2. Complier sometimes not clear to determine whether the node is no longer reachable.
3. How to allow the removed nodes to be free up based on the memory reclamation access.

**Benefit of Hazard Pointer**

Lock-free hazard pointer objects offer significant performance and reliability advantages over conventional lock-based objects. However, the lack of an efficient portable lock-free method for the reclamation of the memory occupied by dynamic nodes removed from such objects is a major obstacle to their wide use in practice. Hazard pointers, a memory management methodology that allows memory reclamation for arbitrary reuse.. It is suitable for user-level applications—as well as system programs—without dependence on special kernel or scheduler support. It is wait-free. It requires only single-word reads and writes for memory access in its core operations. It allows reclaimed memory to be returned to the operating system. In addition, it offers a lock-free solution for the ABA problem using only practical single-word instructions. Our experimental results on a multiprocessor system show that the new methodology offers equal and, more often, significantly better performance than other memory management methods, in addition to its qualitative advantages regarding memory reclamation and independence of special hardware support. We also show that lock-free implementations of important object types, using hazard pointers, offer comparable performance to that of efficient lock-based implementations under no contention and no multiprogramming, and outperform them by significant margins under moderate multiprogramming and/or contention, in addition to guaranteeing continuous progress and availability, even in the presence of thread failures and arbitrary delays.

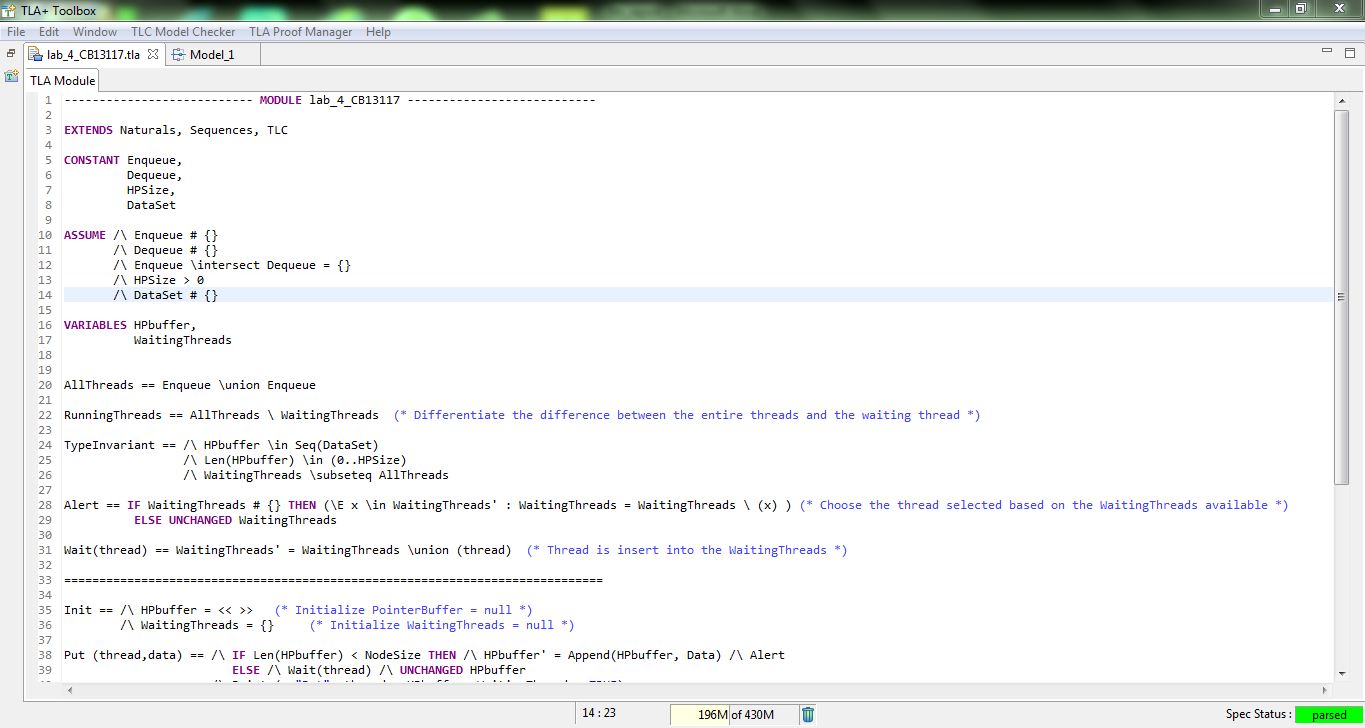
**Methodology**

****

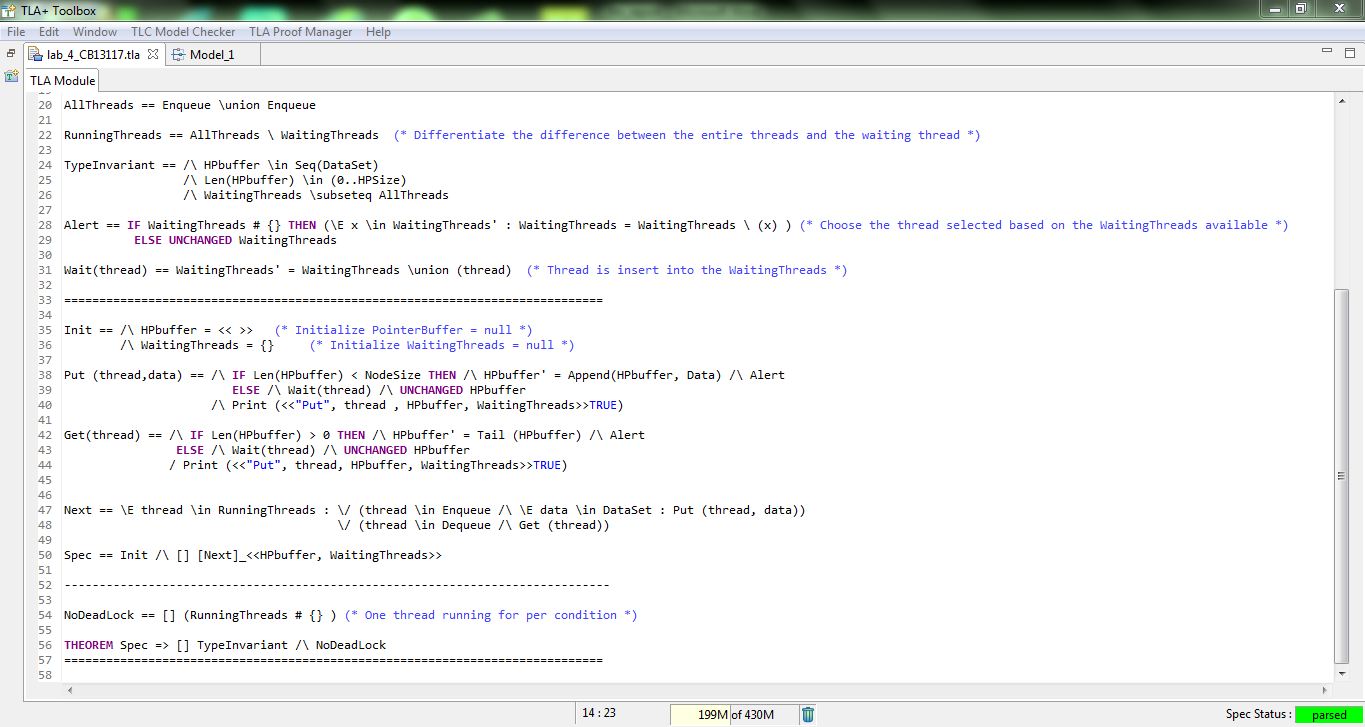
The methodology that we had implement in this project is system development life cycle (SDLC).The methodology is primarily based on the observation that, in the vast majority of algorithms for lock-free dynamic objects, a thread holds only a small number of references that may later be used without further validation for accessing the contents of dynamic nodes, or as targets or expected values of ABA-prone atomic comparison operations. The core idea of the new methodology is associating a number of single-writer multireader shared pointers, called hazard pointers, with each thread that may operate on the associated objects. The number of hazard pointers per thread depends on the algorithms for associated objects and may vary among threads depending on the types of objects they intend to access. Typically, this number is one or two. For simplicity of presentation, we assume that each thread has the same number K of hazard pointers. The methodology communicates with the associated algorithms only through hazard pointers and a procedure Retire Node that is called by threads to pass the addresses of retired nodes. The methodology consists of two main parts: the algorithm for processing retired nodes, and the condition that lock-free algorithms must satisfy in order to guarantee the safety of memory reclamation and ABA prevention.

**TLA MODEL**

**TLA CODING 1**

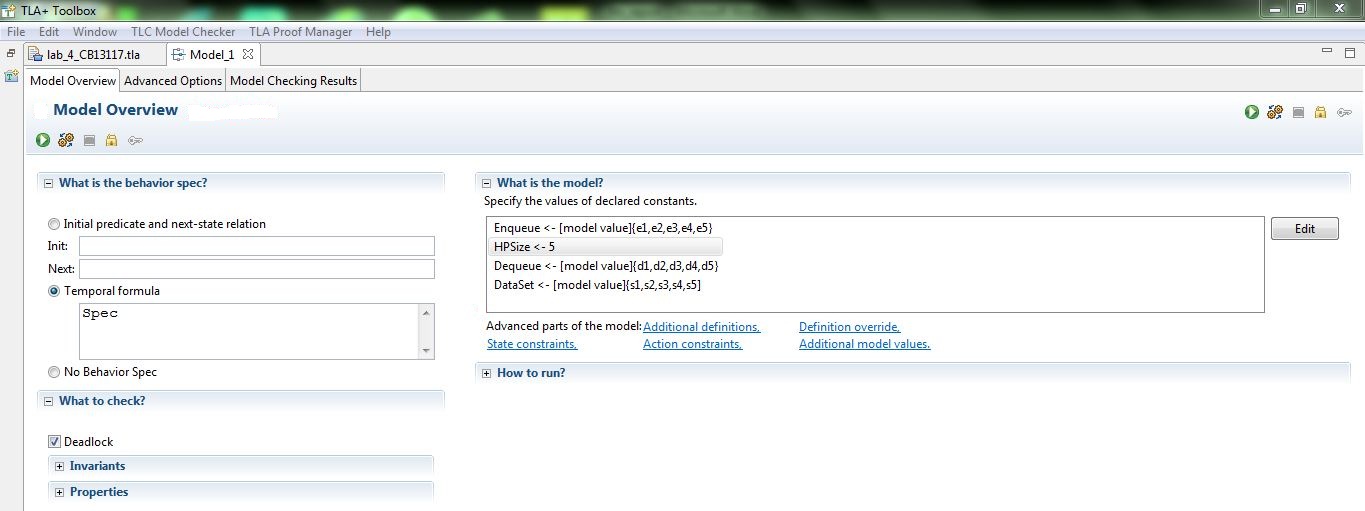
****

**TLA CODING 2 (CONTINUED)**

****

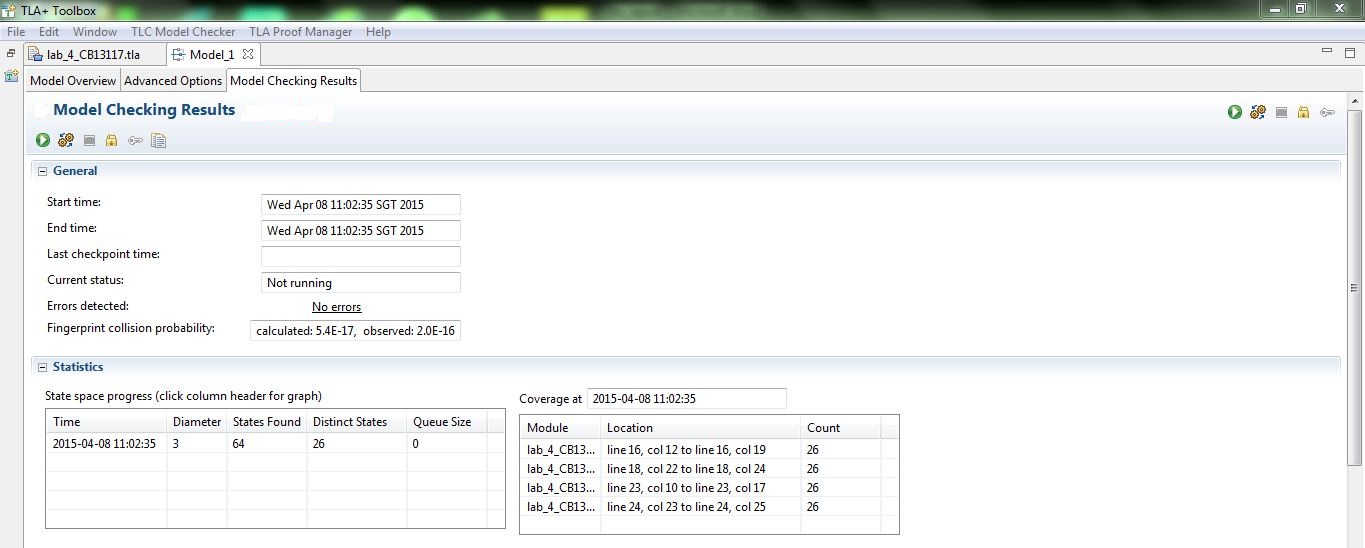
There are four constant which are **Enqueue, Dequeue, HPSize**, and **Dataset**. We assume that the variables start from the initial which is null set or empty set. We need to take note that the **HPSize** is also non-empty set and there is at least one amount of data can be delivering through the buffer. After that we declared **HPbuffer** variable and **WaitingThreads** variable. The **HPbuffer** is a series of data elements while **WaitingThreads** is a set of threads. **AllThreads** is used as the **Enqueue** threads union with the entire **Dequeue** threads and **Running Threads** is utilize as a set of difference between all the threads and the ongoing running threads. For the Alert, it is being used as to choose the thread within the **WaitingThreads** set, if the process need to be continue, else it will remain unchanged for the **WaitingThreads**. Wait process is defined as suspension of the thread and allocate into the **WaitingThreads**. Init is being used to execute the **HPbuffer** as empty sets of order and at the same time the **WaitingThreads** as an empty set. The Method of “Get” suspends the calling thread until the buffer is already “non-empty”. The thread then deletes one object from the **HPbuffer** and make and **“Alert”** to unlock the thread block on **Wait** order. Alert is used whenever the **HPbuffer** experience modification. In order to suspend “**Wait**” is being used. “**Next”** is used for system changeover to its next state. **NoDeadLock** is defined as there is at least one thread running and fulfill the **NoDeadLock** characteristic.

**TLA MODEL OVERVIEW**

****

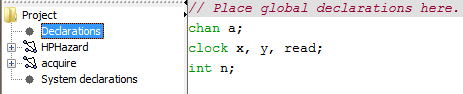
We need to initialize the value in the Enqueue, HPSize, Dequeue, and Dataset before checking is being made. We using HPSize -> 5 for our data initiation during th model checking process. The result shown in the **“MODEL CHECKING RESULT”** diagram.

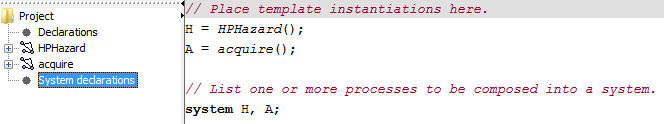
**TLA MODEL CHECKING RESULT**

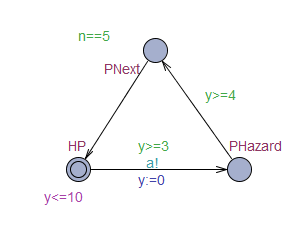
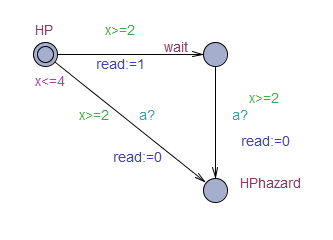
****

**UPPAL MODEL**

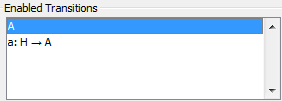
**Hazard Pointer Declaration**





**Hazard Pointer Diagram**

HPHazard acquire

Figure above shows Hazard Pointer which is involve two templates. First one called HPHazard used to run the basic flow and for the second one called acquire used to check and process if there is reader or not at that time.

There is two enable transitions that can be apply while running this tools.

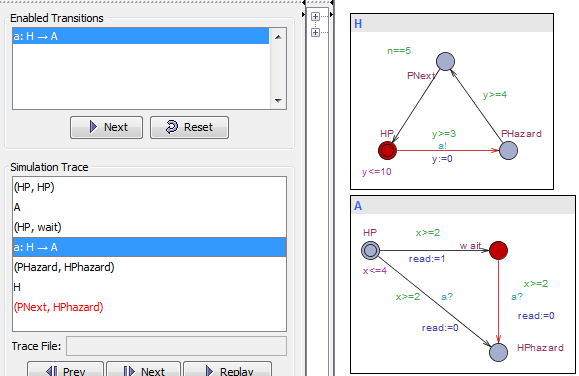
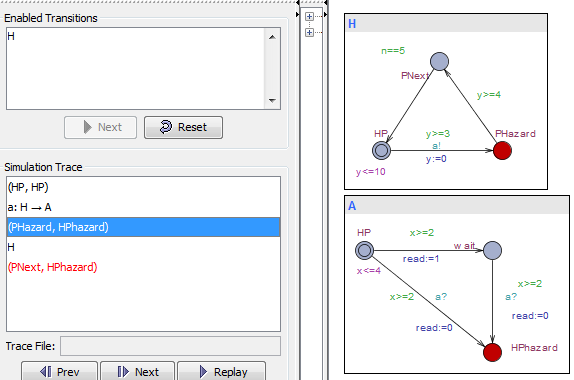
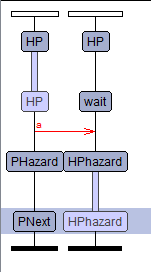
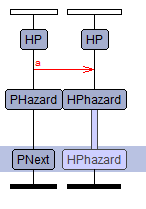


Figure above shows the transaction between HP ->wait->HPhazard and PHazard->PNext. These flows occur if there is at least one reader still reading and have to wait before delete it.



In other transaction, these flows occur when there is no reader still reading and the flow is HP->HPhazard->PNext. It just skips the node waiting and the book can be release without affect anything.



Read>=1

Read=0

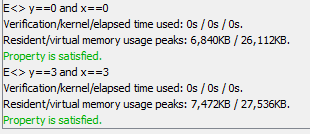
Figure above show the flowchart between one nodes to another nodes.

Figure above shows the verifier that had been test. It was satisfied when x and y are equal to 3 or equal to 0. This is follow the rule in HPHazard allowing x<=4 and y<=10.

**CONCLUSION**

As a conclusion, after a lot of struggling and effort finally we had done our Hazard Pointer project successfully. Along our journey of this project we had gained more experiences, important of TLA and Uppal languages and moral values too. At the same time we also had a great chance to know well about the important of the subject Formal Methods (BCS2213) for computer students. This subject allows us to improve our knowledge in computer programmed development. We also learned some vital value of system such as the accuracy, correctness, robustness of the problem using a specific specification. We also had experiences on the memory which allows to use by consumed nodes to be reclaimed by the OS. It performs at least as efficiently as memory management methods that lack this feature. Hazard pointers only require basic a basic CAS or LL/SC instruction. Hazard pointers can be a bit tricky to use correctly. The correctness “proofs” are far from formal, but still useful. At last not least we also thank our lecturer Prof. Dr. Vitaliy Mezhuyev who shared his ideas regarding our task Producer- Consumer Problem. He also gave motivations to us and helps in complete our task successfully.

**REFERENCES**

* <https://www.research.ibm.com/people/m/michael/ieeetpds-2004.pdf>
* <http://webcourse.cs.technion.ac.il/236755/Spring2011/hw/WCFiles/KoyfmanShubin.pdf>
* <http://web.cecs.pdx.edu/~walpole/class/cs510/fall2011/slides/07.pdf>
* <https://en.wikipedia.org/wiki/Hazard_pointer>
* <https://www.informatik.uniaugsburg.de/de/lehrstuehle/swt/se/staff/formerStaff/tofan/Downloads/ictac11-hazard.pdf>