CS636 Assignment 2 Report

180301 - hrugved wath

1

- Folder: p1
- Commands:
  1. g++ -c p1.cpp LFQueue.cpp
  2. g++ -o main p.1.o LFQueue.o
  3. ./main
- Files:
  - *p1.cpp*: main file
  - *LFQueue.h*: header file containing defination of LFQueue object.
  - *LFQueue.cpp*: source file containing implementations of LFQueue object and relevant structs.
- It implements the algorithm for non-blocking concurrent queue mentioned in MS queue paper. struct *node_t* contains the data and struct *pointer_t* contains a pointer to *node_t* and a tag. *pointer_t* provides an indirect access to *node_t*, while only being 16 bytes(2-word aligned) which can be atomic and be operated on by *CAS*. Apart from *enqueue* and *dequeue* method, there is *dump* method which prints the contents of queue. For *dump* to work correctly, concurrent updates should be disallowed. To achieve this I used a C++ *shared_timed_mutex*, which is a mulitple reader - single writer lock. *dump* will try to acquire it as a writer while other methods can acquire it as readers. Hence the queue is non-blocking except when *dump* is called.

2

2.1

- Folder: p2.1
- Commands:
  1. g++ -c p2.1.cpp hashtable.cpp
  2. g++ -o main p2.1.o hashtable.o
  3. ./main
- Files:

- *p2.1.cpp*: main file
- *hashtable.h*: header file containing defination of hashtable object *CHashTable* and *Bucket*(list at each index of hashtable).
- *hashtable.cpp*: source file containing implementations of objects and relevant structs.

- Hashtable is implemented as an array of fixed size. A hash value of key indexes into the hashtable and list at that index is locked for any operations. For locking, I used a C++ *shared_timed_mutex*, which is a multiple reader - single writer lock. *insert* and *remove* acquires the lock as a writer, while *find* acquires it as reader (since find operations can be concurrent).

2.2

- Folder: p2.2
- Commands:
  1. g++ -c p2.2.cpp hashtable.cpp
  2. g++ -o main p2.2.o hashtable.o
  3. ./main
- Files:
  - *p2.2.cpp*: main file
  - *hashtable.h*: header file containing defination of hashtable object *CHashTable* and other structs and markable-pointer manipulation functions.
  - *hashtable.cpp*: source file containing implementations of hashtable object and relevant structs.
- Hashtable is implemented as an array of fixed size. A hash value of key indexes into the hashtable and list at that index is locked for any operations. The algorithms are based on a paper by Maged M. Michael. To implement the algorithms, I have to make some substantial changes to work with atomics in C++. The atomic operations are done on a 16-byte(2-word aligned) struct called *Meta*. *Meta* has a *Node** pointer, whose MSB is used for setting/unsetting a mark. Relevant functions like *set*, *unset*, *isSet*,etc. regarding manipulating these markable pointers can be found in the header file.