Docu - Project 2

about my templates:

use [Ctrl]+[Shift]+[1-6] to use fonts:

font 1

font 2

# font 3

## font 4

### font 5

font 6

# Main

do\_some\_work()

WIP

test\_mutex()

write about random workload

this will test mutual exclusion of a passed DW\_Lock object with given parameters (more detail)

The idea is to log the events of threads entering the critical section and threads leaving the CS. Mutual exclusion holds true, if every entering event is followed by a leaving event. I.e. no entering event is immediately followed by another entering event.

test\_fcfs()

write about random workload

this will test first-come-first-served of a passed DW\_Lock object with given parameters. (more detail)

At the moment this is done by making the doorway section mutually exclusive. But this will have to change.

The idea is to write an array, that keeps track of the sequence in which threads complete the doorway and another array that keeps track of threads acquiring the lock. FCFS holds true if these arrays are identical.

main()

write about initialization

An instance of one of the lock classes is created.

The lock object is then passed to one or more testing functions.

# Locks

class Lock

this pure abstract class is the Lock interface form the Herlihy book.

class DW\_Lock

this pure abstract class is an interface for "doorway locks". These are locks where the lock() function can be written like this

public: void lock(){  
 doorway();  
 wait();  
}

This is used so that FCFS can be tested from the outside (i.e. without putting logging functions into the lock class).

class Reference\_Lock

this uses these statements

* create an instance of the lock with this statement  
  std::atomic\_flag lock\_stream = ATOMIC\_FLAG\_INIT;
* execute lock() (i.e. doorway and waiting section) with this statement  
  while ( lock\_stream.test\_and\_set() ) {}
* execute unlock() with this statement  
  lock\_stream.clear();

class Lamport\_Lecture

This is the naive straight forward implementation of Lamport Bakery lock according to the lecture notes. It is very bad.

draw\_ticket

wait

class Lamport\_Lecture\_fix

This sub-class of Lamport\_Lecture uses an atomic register latest\_ticket to keep track of the ticket that was last issued. This is done by overriding the draw\_ticket() function like this:

private: virtual int draw\_ticket() override {  
 return ++latest\_ticket;  
}

Like this mutual exclusion holds. It is questionable however, if we can still speak of a wait-free doorway section now. Consider this case: Thread tries to draw a ticket, an atomic read and an atomic write of latest\_ticket are required. If latest\_ticket it is accessed by thread at the time, and thread is stalled, this could mean that there is no maximum number of required can be given in which thread completes the doorway.

class Lamport\_Original

this is the naive straight forward implementation of the Lamport Bakery algorithm from 1974. The variable names and token values are from the paper Jayati 2004.

It is very bad

implementation of wait

# Makefile

Compiler flags

they do weird things

O1 and O3 will break mutex for Lamport\_Original

O1 and O3 will improve mutex for Lamport\_Lecture (lol)

O2 seems to do nothing (test some more?)