Q1. Using the framework code (stdafx.h, helper.h, helper.cpp, and sharing.cpp) as a starting point, write code to test the performance of the following locks when used to protect the non-atomic increment of a global variable (except case 1).

1. Atomic increment (provides a performance base line)
2. Bakery or Black and White Bakery lock (NB: you may have to add fence instruction(s))
3. TestAndSet lock
4. TestAndTestAndSet lock
5. MCS lock

Plot, on a single graph, the number of increments per second (Y axis) against the number of threads (X axis) for each type of lock. The maximum number of threads should be twice the number of logical CPUs. Add code to check that the number of increments is correct.

Why is the throughput of the MCS lock so poor when the number of threads is greater than the number of logical CPUs? Discuss how you might estimate (mathematically) the expected throughput in this case?

Write a short report summarising your results (2 to 3 pages) and include a code listing.

Hint: the following is an example implementation of a ticket lock embedded in the framework (NB: OPTYP has been replaced with LOCKTYP).

```c
#elif LOCKTYP == ...

class TicketLock {

public:

    volatile long ticket;
    volatile long nowServing;

    inline void acquire()
    {
        int myTicket = InterlockedExchangeAdd(&ticket, 1);
        while (myTicket != nowServing)
            SLEEP_OR_PAUSE();
    }

    inline void release()
    {
        nowServing++;
    }

};

TicketLock lock;

#define LOCKSTR            "ticket lock"
#define INIT()             lock.ticket = lock.nowServing = 0
#define ACQUIRE()          lock.acquire()
#define RELEASE()          lock.release();

#elif LOCKTYP == ...
```