Q1. Using the framework code (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). You may use a Windows or Linux system.

1. Atomic increment (to provide a performance baseline)
2. Bakery lock (NB: you may have to add fence instruction(s))
3. TestAndTestAndSet lock
4. 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 Bakery and MCS locks so poor when the number of threads is greater than the number of logical CPUs?

Write a short report summarising your results (2 to 3 pages), excluding the code listing.

Hint: the following is an example implementation of a ticket lock embedded in the sample framework where OPTYP has been replaced with LOCKTYP, and the generic lock operations are specified by the MACROS LOCKSTR, INIT, ACQUIRE and RELEASE. The USEPMS code can be removed as well as the loop for cycling through the percentage sharing.

```cpp
#define LOCKTYP == 5
class TicketLock {
public:
    volatile long ticket;
    volatile long nowServing;

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

    void release() {
        nowServing++;
    }
};
TicketLock lock;
#define LOCKSTR "ticket lock"
#define INIT() lock.ticket = lock.nowServing = 0
#define ACQUIRE() lock.acquire()
#define RELEASE() lock.release();
#define LOCKTYP == 6
```