2) class spinlock {

public:

spinlock() {

locked.store(false, std::memory\_order\_relaxed); //

}

void lock() {

while (locked.exchange(true, std::memory\_order\_acquire)) { } //нужно, чтобы возникло SW между записью в locked потока, вышедшего из критической секции и сохранившего false, чтением в другом(во время exchange). Relaxed недостаточно, так как тогда потоки не будут знать о записях в locked других потоков, а именно на этом и строится алгоритм, это точка синхронизации.

}

void unlock() {

locked.store(false, std::memory\_order\_release); // для SW

}

private:

std::atomic<bool> locked;

};

class peterson\_mutex {

public:

peterson\_mutex() {

want[0].store(false, std::memory\_order\_relaxed);

want[1].store(false, std::memory\_order\_relaxed);

victim.store(0, std::memory\_order\_relaxed);

}

void lock(int t) {

want[t].store(true, std::memory\_order\_release); // SW с чтением в цикле(поток должен знать, какое значение записал другой поток, а relaxed этого не обеспечивает)

victim.store(t, std::memory\_order\_release); // SW с чтением в цикле

while (want[1 - t].load(std::memory\_order\_acquire) && victim.load(std::memory\_order\_acquire) == t) { }

}

void unlock(int t) {

want[t].store(false, std::memory\_order\_release);

}

private:

std::array<std::atomic<bool>, 2> want;

std::atomic<int> victim;

};