# Races and Synchronization

NOTE: this experiment should be finished on the **granger1** server. NOT **labsrv06** which many of you use in project 1!

## Short Q&A (10%)

* How many threads that can run simultaneously (i.e. thread-level parallelism offered by CPU hardware) on the lab server? If you are using a different machine, state it.
* What does pthread\_join() do in the given code?
* Why concurrent, unsynchronized updates to the\_counter leads to program errors?
* The given counter.c invokes atexit(). What does the function do?

## 1. Zoom in the scene of race condition (20%)

Here is the assembly of function add(long long \*pointer, long long value), as dumped from objdump. Note that without assuming x86 knowledge from you, I showed the ARMv8 version below (compiled with -O2).

640 0000000000001600 <add>:  
641 long long sum = \*pointer + value;  
642 f9400002 ldr x2, [x0]  
643 8b010041 add x1, x2, x1  
644 \*pointer = sum;  
645 f9000001 str x1, [x0]  
646 }  
647 d65f03c0 ret

Read the assembly and answer:

i) How many bits in a long long type of integer?

ii) Point out which instructions (by their line numbers above) constitute the window for race condition.

iii) Will race condition still exist, if we run the program with multiple threads but on a single-core machine?

## 2. Use spinlock & CAS (30%)

Add the following mechanisms to the source code:

* one that protects the add by a spin-lock, enabled by a **--sync=s** option. You will have to implement your own spin-lock operation.
* one that performs the add using compare-and-swap (CAS) primitives to ensure atomic updates to the shared counter, enabled by a **--sync=c** option. Note the name: compare-and-swap is the same as compare-and-exchange.

The provided code can already parse these new options! 😉

### Example output

Before (by the give code)

$./counter --iterations=10000 --threads=10 --sync=s  
test=add-s threadNum=10 iterations=10000 numOperation=200000 runTime(ns)=5640178 avgTime(ns)=28 count=-10113  
$./counter --iterations=10000 --threads=10 --sync=c  
test=add-c threadNum=10 iterations=10000 numOperation=200000 runTime(ns)=4469589 avgTime(ns)=22 count=-7513

After (expected from your code). With with spinlocks and CAS, the final count value is integral, i.e. ==0.

$./counter --iterations=10000 --threads=10 --sync=s  
test=add-s threadNum=10 iterations=10000 numOperation=200000 runTime(ns)=27917650 avgTime(ns)=139 count=0  
$./counter --iterations=10000 --threads=10 --sync=c  
test=add-c threadNum=10 iterations=10000 numOperation=200000 runTime(ns)=20609670 avgTime(ns)=103 count=0

**Implementation hints**: both spinlock and CAS shall be implemented using the GCC's atomic built-ins. Since the built-ins are architecture-independent, you do not have to write any assembly.

* The documentation can be found [here](https://gcc.gnu.org/onlinedocs/gcc/_005f_005fatomic-Builtins.html). Some related [discussion](https://stackoverflow.com/questions/13941385/using-gcc-atomic-builtins).
* Useful functions include \_\_atomic\_compare\_exchange\_n() and \_\_atomic\_store\_n()
* These functions require memory order, for which you may specific \_\_ATOMIC\_SEQ\_CST. (Q: could other memory order work?)
* Note: older GCC offers \_\_sync\_XXX built-ins, which are still supported today for backward compatibility. Avoid them. They are deprecated by the \_\_atomic builtins.

Search for "todo" in the given source code for extra hints.

### Deliverable:

[upload a standalone tarball named as p2-1-2.tar.gz]

## 3. Measure slowdown due to synchronization (40%)

Compare the times taken for parallel updating the shared counter:

* Without any synchronization
* With mutex (--sync=m)
* With spinlock (--sync=s)
* With CAS (--sync=c)

Report the performance with the following arguments. Write a small paragraph to explain your observation. How many repeated runs did you execute? How do you ensure your executions was unaffected by other students who may run the experiments at the same time? Would different thread counts and iteration counts affect your observation?

./counter --iterations=100000 --threads=10 --sync=m  
./counter --iterations=100000 --threads=10 --sync=s  
./counter --iterations=100000 --threads=10 --sync=c