Assignment #2

Osama Sh. Almomani

Safe code (GitHub link)

```
#include <iostream>
#include <string>
#include <pthread.h>
#include <fstream>


#define ll long long
void *mult(void *p);
using namespace std;
ll numOfOdd = 0, numOfEven = 0, totalCells = 0,n;

string output="";

ll *A, *B, *C;
pthread_mutex_t lock1, lock2, lock3, lock4;
struct Args
{
  ll start;
  ll len;
  ll id;
} * args;
```

```
int main(int argc, char **argv)
{
  ofstream out("output.txt");

  freopen("in.txt", "r", stdin);
  cin >> n;

  A = new ll[n * n];
  B = new ll[n * n];
  C = new ll[n * n];

  ll t = stoll(argv[1]);
  if (t > n * n)
    t = n * n;

  pthread_t *threads = new pthread_t[t];

  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      cin >> A[i * n + j];

  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      cin >> B[i * n + j];

  ll cells = n * n / t, e = n * n % t;
  for (int i = 0; i < t; i++)
  {
    args = new struct Args;
    args->id = i;
    if (i == 0)
    {
      args->start = 0;
      args->len = cells + e;
    }
    else
    {
      args->start = i * cells + e;
```

```
        args->len = cells;
    }

    pthread_create(&threads[i], NULL, mult, (void *)args);
  }
  for (int i = 0; i < t; i++)
  {
    pthread_join(threads[i], NULL);
  }

  output+= "numOfEven=" + to_string(numOfEven) + ", numOfOdd=" +  to_string(numOfOdd) +
              ", totalCells=" + to_string(totalCells)+"\n";
  out << output;
}
```

```
void *mult(void *p)
{
  ofstream out("output.txt");

  struct Args *r = (struct Args *)p;
  ll start = r->start;
  ll len = r->len;
  pthread_mutex_lock(&lock4);

  output+= "ThreadID=" +to_string (r->id) + ", startLoop="+ to_string(start) + ", endLoop=" +to_string
(start+len)  +  "\n";
  pthread_mutex_unlock(&lock4);

  for (int i = start; i < start + len; ++i)
  {
     ll tmp = 0;
     ll current_row = i / n * n;
     ll current_clm = i % n;
     for (int j = 0; j < n; j++)
        tmp += A[current_row + j] * B[current_clm + j * n];
     C[i] = tmp;
     if (tmp % 2 == 0)
     {
        pthread_mutex_lock(&lock1);
        numOfEven++;
        pthread_mutex_unlock(&lock1);
     }
     else
     {
        pthread_mutex_lock(&lock2);
        numOfOdd++;
        pthread_mutex_unlock(&lock2);
     }
     pthread_mutex_lock(&lock3);
     totalCells++;
     pthread_mutex_unlock(&lock3);
  }
}
```

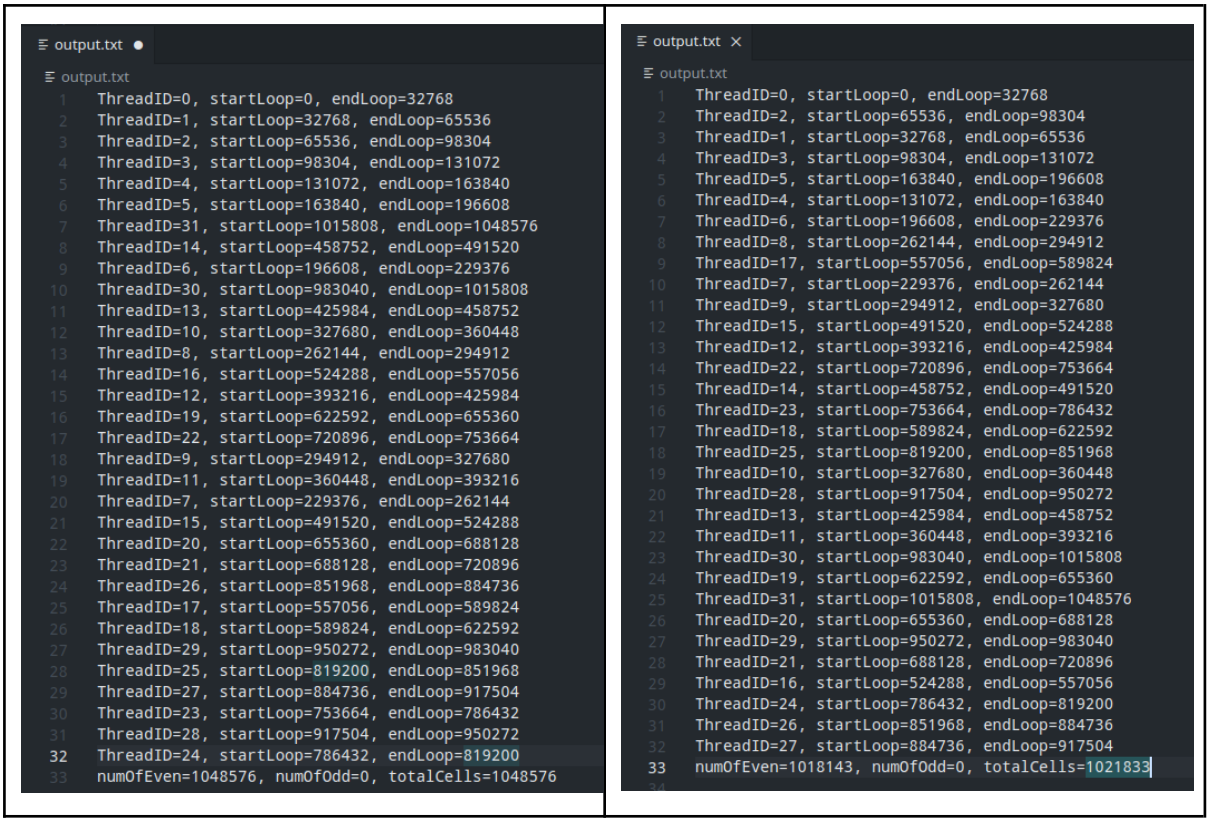
The previous code is safe because of using locks, removing the green-highlighted lines will make it **unsafe**.

Explanation

we are using locks here because some threads might work at the same time, therefore they might try to change the shared variables in the Critical section at the same time (before one of them get finished) which causes a thread to use an old value of the variable, therefore, making incorrect values

Output comparison

Safe Code	Unsafe Code
output.txt ThreadID=0, startLoop=0, endLoop=1048576 numOfEven=1048576, numOfOdd=0, totalCells=1048576	output.txt ThreadID=0, startLoop=0, endLoop=1048576 numOfEven=1048576, numOfOdd=0, totalCells=1048576
output.txt ThreadID=0, startLoop=0, endLoop=524288 ThreadID=1, startLoop=524288, endLoop=1048576 numOfEven=1048576, numOfOdd=0, totalCells=1048576	output.txt ThreadID=0, startLoop=0, endLoop=524288 ThreadID=1, startLoop=524288, endLoop=1048576 numOfEven=1036347, numOfOdd=0, totalCells=1036548
output.txt ThreadID=0, startLoop=0, endLoop=262144 ThreadID=1, startLoop=262144, endLoop=524288 ThreadID=3, startLoop=786432, endLoop=1048576 ThreadID=2, startLoop=524288, endLoop=786432 numOfEven=1048576, numOfOdd=0, totalCells=1048576	output.txt ThreadID=1, startLoop=262144, endLoop=524288 ThreadID=2, startLoop=524288, endLoop=786432 ThreadID=3, startLoop=786432, endLoop=1048576 numOfEven=976157, numOfOdd=0, totalCells=977513
output.txt ThreadID=0, startLoop=0, endLoop=131072 ThreadID=1, startLoop=131072, endLoop=262144 ThreadID=3, startLoop=393216, endLoop=524288 ThreadID=4, startLoop=524288, endLoop=655360 ThreadID=6, startLoop=786432, endLoop=917504 ThreadID=5, startLoop=655360, endLoop=786432 ThreadID=2, startLoop=262144, endLoop=393216 ThreadID=7, startLoop=917504, endLoop=1048576 numOfEven=1048576, numOfOdd=0, totalCells=1048576	output.txt ThreadID=0, startLoop=0, endLoop=131072 ThreadID=1, startLoop=131072, endLoop=262144 ThreadID=2, startLoop=262144, endLoop=393216 ThreadID=3, startLoop=393216, endLoop=524288 ThreadID=5, startLoop=655360, endLoop=786432 ThreadID=7, startLoop=917504, endLoop=1048576 ThreadID=4, startLoop=524288, endLoop=655360 ThreadID=6, startLoop=786432, endLoop=917504 numOfEven=912702, numOfOdd=0, totalCells=922740
output.txt ThreadID=1, startLoop=65536, endLoop=131072 ThreadID=2, startLoop=131072, endLoop=196608 ThreadID=0, startLoop=0, endLoop=65536 ThreadID=7, startLoop=458752, endLoop=524288 ThreadID=5, startLoop=327680, endLoop=393216 ThreadID=15, startLoop=983040, endLoop=1048576 ThreadID=9, startLoop=589824, endLoop=655360 ThreadID=13, startLoop=851968, endLoop=917504 ThreadID=12, startLoop=786432, endLoop=851968 ThreadID=10, startLoop=655360, endLoop=720896 ThreadID=6, startLoop=393216, endLoop=458752 ThreadID=4, startLoop=262144, endLoop=327680 ThreadID=11, startLoop=720896, endLoop=786432 ThreadID=8, startLoop=524288, endLoop=589824 ThreadID=3, startLoop=196608, endLoop=262144 ThreadID=14, startLoop=917504, endLoop=983040 numOfEven=1048576, numOfOdd=0, totalCells=1048576	output.txt ThreadID=1, startLoop=65536, endLoop=131072 ThreadID=4, startLoop=262144, endLoop=327680 ThreadID=5, startLoop=327680, endLoop=393216 ThreadID=13, startLoop=851968, endLoop=917504 ThreadID=8, startLoop=524288, endLoop=589824 ThreadID=9, startLoop=589824, endLoop=655360 ThreadID=11, startLoop=720896, endLoop=786432 ThreadID=12, startLoop=786432, endLoop=851968 ThreadID=6, startLoop=393216, endLoop=458752 ThreadID=10, startLoop=655360, endLoop=720896 ThreadID=14, startLoop=917504, endLoop=983040 ThreadID=3, startLoop=196608, endLoop=262144 ThreadID=7, startLoop=458752, endLoop=524288 ThreadID=15, startLoop=983040, endLoop=1048576 numOfEven=1026027, numOfOdd=0, totalCells=1028784



output.txt

```
ThreadID=0, startLoop=0, endLoop=32768
ThreadID=1, startLoop=32768, endLoop=65536
ThreadID=2, startLoop=65536, endLoop=98304
ThreadID=3, startLoop=98304, endLoop=131072
ThreadID=4, startLoop=131072, endLoop=163840
ThreadID=5, startLoop=163840, endLoop=196608
ThreadID=31, startLoop=1015808, endLoop=1048576
ThreadID=14, startLoop=458752, endLoop=491520
ThreadID=6, startLoop=196608, endLoop=229376
ThreadID=30, startLoop=983040, endLoop=1015808
ThreadID=13, startLoop=425984, endLoop=458752
ThreadID=10, startLoop=327680, endLoop=360448
ThreadID=8, startLoop=262144, endLoop=294912
ThreadID=16, startLoop=524288, endLoop=557056
ThreadID=12, startLoop=393216, endLoop=425984
ThreadID=19, startLoop=622592, endLoop=655360
ThreadID=22, startLoop=720896, endLoop=753664
ThreadID=9, startLoop=294912, endLoop=327680
ThreadID=11, startLoop=360448, endLoop=393216
ThreadID=7, startLoop=229376, endLoop=262144
ThreadID=15, startLoop=491520, endLoop=524288
ThreadID=20, startLoop=655360, endLoop=688128
ThreadID=21, startLoop=688128, endLoop=720896
ThreadID=26, startLoop=851968, endLoop=884736
ThreadID=17, startLoop=557056, endLoop=589824
ThreadID=18, startLoop=589824, endLoop=622592
ThreadID=29, startLoop=950272, endLoop=983040
ThreadID=25, startLoop=819200, endLoop=851968
ThreadID=27, startLoop=884736, endLoop=917504
ThreadID=23, startLoop=753664, endLoop=786432
ThreadID=28, startLoop=917504, endLoop=950272
ThreadID=24, startLoop=786432, endLoop=819200
numOfEven=1048576, numOfOdd=0, totalCells=1048576
```

output.txt

```
ThreadID=0, startLoop=0, endLoop=32768
ThreadID=2, startLoop=65536, endLoop=98304
ThreadID=1, startLoop=32768, endLoop=65536
ThreadID=3, startLoop=98304, endLoop=131072
ThreadID=5, startLoop=163840, endLoop=196608
ThreadID=4, startLoop=131072, endLoop=163840
ThreadID=6, startLoop=196608, endLoop=229376
ThreadID=8, startLoop=262144, endLoop=294912
ThreadID=17, startLoop=557056, endLoop=589824
ThreadID=7, startLoop=229376, endLoop=262144
ThreadID=9, startLoop=294912, endLoop=327680
ThreadID=15, startLoop=491520, endLoop=524288
ThreadID=12, startLoop=393216, endLoop=425984
ThreadID=22, startLoop=720896, endLoop=753664
ThreadID=14, startLoop=458752, endLoop=491520
ThreadID=23, startLoop=753664, endLoop=786432
ThreadID=18, startLoop=589824, endLoop=622592
ThreadID=25, startLoop=819200, endLoop=851968
ThreadID=10, startLoop=327680, endLoop=360448
ThreadID=28, startLoop=917504, endLoop=950272
ThreadID=13, startLoop=425984, endLoop=458752
ThreadID=11, startLoop=360448, endLoop=393216
ThreadID=30, startLoop=983040, endLoop=1015808
ThreadID=19, startLoop=622592, endLoop=655360
ThreadID=31, startLoop=1015808, endLoop=1048576
ThreadID=20, startLoop=655360, endLoop=688128
ThreadID=29, startLoop=950272, endLoop=983040
ThreadID=21, startLoop=688128, endLoop=720896
ThreadID=16, startLoop=524288, endLoop=557056
ThreadID=24, startLoop=786432, endLoop=819200
ThreadID=26, startLoop=851968, endLoop=884736
ThreadID=27, startLoop=884736, endLoop=917504
numOfEven=1018143, numOfOdd=0, totalCells=1021833
```

Observations: we notice here that the unsafe code provides random values, also it sometimes doesn't show all threads ID in the output(because of race-condition on the output variable). The unsafe code works well only when using 1 thread or if you got lucky enough :)

both safe and unsafe doesn't necessarily start running threads in FIFO order -this is handled by the OS scheduler-

Time Comparaison

main.cpp | output.txt | statics.py | in.txt

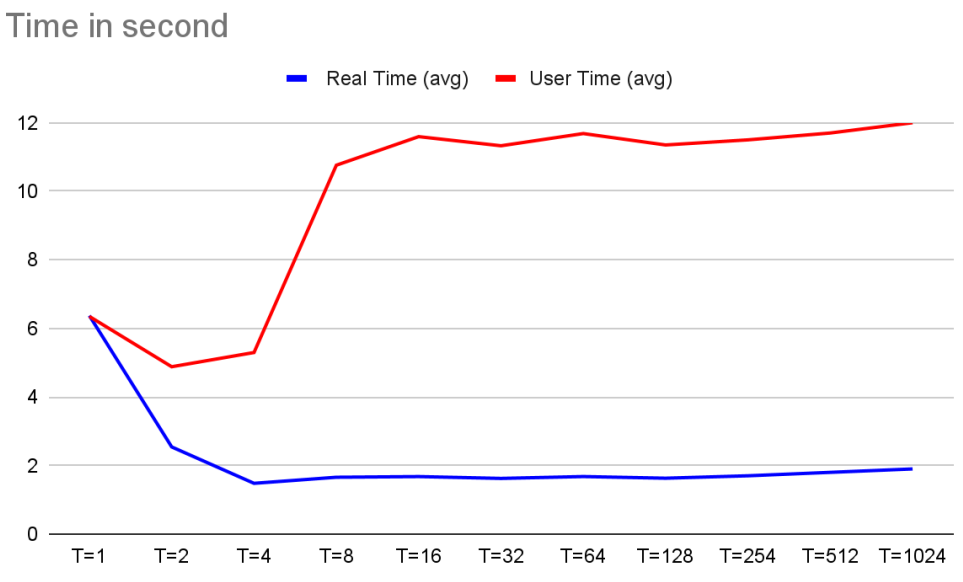
statics.py > ...

```python
num_of_tries=100
for i in [1,2,4,8,16,32,64,128]:
    real_sum[i]=0
    user_sum[i]=0

    for j in range(num_of_tries):
        times=[float(i) for i in re.findall( r'm(.*?)s',subprocess.getoutput( 'time ./a.out {}'.format(i)) )]
        real_sum[i] += times[0]
        user_sum[i] += times[1]
    real_avg[i] = real_sum[i]/num_of_tries
    user_avg[i] = user_sum[i]/num_of_tries

    print('\nThreads=', i,'\n\tReal_Avg: ' + str(real_avg[i]) +'\n\tUser_Avg:' +str(user_avg[i]) + '\n')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
[osama@Lenovo3i assignment]$ python3 statics.py

Threads= 1
        Real_Avg: 6.372240000000001
        User_Avg:6.348550000000001


Threads= 2
        Real_Avg: 2.5420800000000003
        User_Avg:4.881150000000002


Threads= 4
        Real_Avg: 1.48032
        User_Avg:5.2961599999999995


Threads= 8
        Real_Avg: 1.6574
        User_Avg:10.760499999999997


Threads= 16
        Real_Avg: 1.6776600000000004
        User_Avg:11.59419


Threads= 32
        Real_Avg: 1.6234699999999995
        User_Avg:11.327730000000004


Threads= 64
        Real_Avg: 1.6771200000000004
        User_Avg:11.68384


Threads= 128
        Real_Avg: 1.6295899999999996
        User_Avg:11.351229999999997
```

```
[osama@Lenovo3i assignment]$ sudo lshw -C cpu
[sudo] password for osama:
  *-cpu
       description: CPU
       product: Intel(R) Core(TM) i5-10300H CPU @ 2.50G
       vendor: Intel Corp.
       physical id: 4
       bus info: cpu@0
       version: Intel(R) Core(TM) i5-10300H CPU @ 2.50G
       serial: To Be Filled By O.E.M.
       slot: U3E1
       size: 2785MHz
       capacity: 4500MHz
       width: 64 bits
       clock: 100MHz
       capabilities: lm fpu fpu_exception wp vme de pse
rr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse
pe1gb rdtscp x86-64 constant_tsc art arch_perfmon pebs
top_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor d
 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt
 f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb
ibp ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid
 avx2 smep bmi2 erms invpcid mpx rdseed adx smap clflus
etbv1 xsaves dtherm arat pln pts hwp hwp_notify hwp_act
ar flush_l1d arch_capabilities ida cpufreq
       configuration: cores=4 enabledcores=4 threads=8
[osama@Lenovo3i assignment]$
```



Looking at the graph, minimum *real* time when T=4, this means that the CPU is best utilized for this program when running 4 and it is best when these threads were running in at same time in parallel ⇒ **number of cores is 4**

> *real* is the actual elapsed time for a real human clock. However, *user* is The **cumulative** time spent **by all computing units** during the computation in user-mode **-in Hyperthreading, multiple threads can be pipelined in one core with no context switching-**

Notice when T=8 and higher, *user* Time start stabilization with *user*=11.5. This is because the *user* parameter doesn't count context switching, while *real* does. So eventually, *real* will increase with increase of context switching headache(e.g T=8192⇒ real=3.2, user=12)

user ≃ 8 x *real* (and always equal or less, since pipelining saves time)
this mean that we have 8 computing units ⇒ my CPU is hyperthreaded with **8 threads** (2 CPU threads **x** 4 CPU cores)

For architecture designing reasons, usually, the number of HW threads is always multiple of cores number. That is why we assumed before that the number of computing units/threads is 8 and not 6 for example.