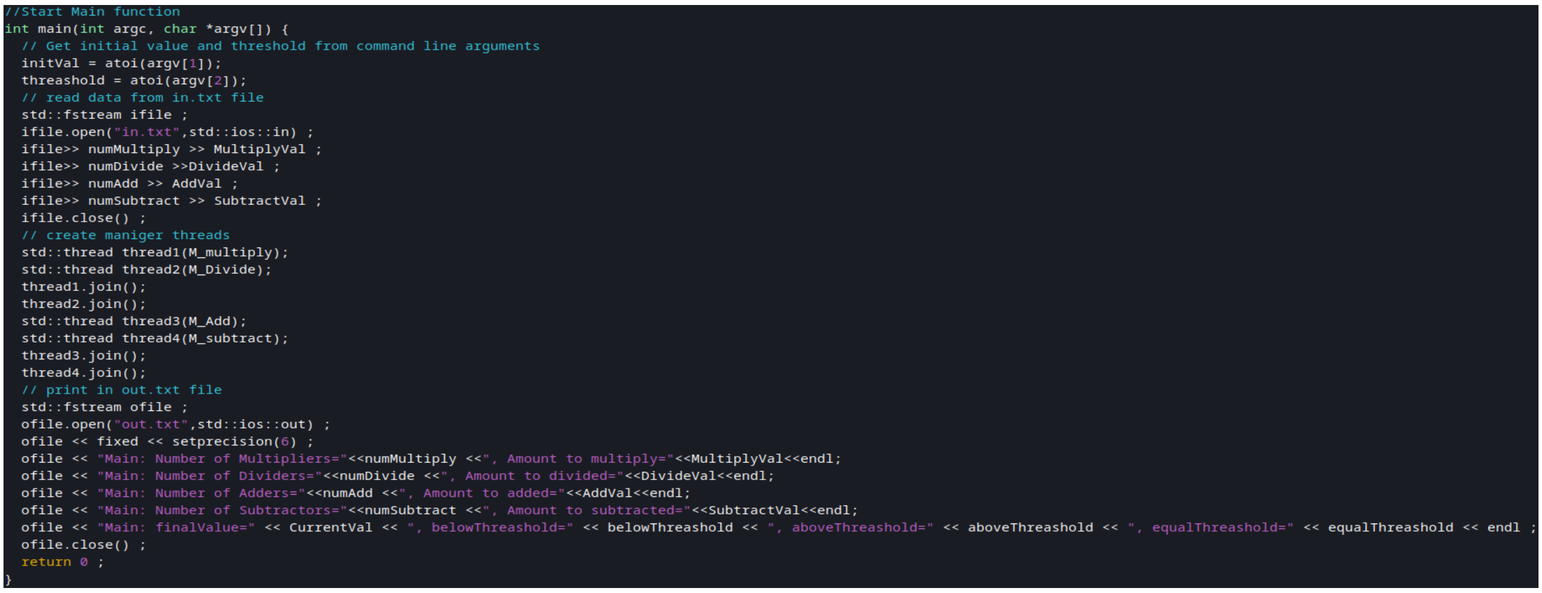
In this report, we will showcase two iterations of a multi-threaded code, one of which is thread-unsafe while the other version uses binary semaphores to fix the issue and make it thread-safe.

The outputs and performance of the two programs will be compared in the report that follows, along with screen pictures of the two codes.

Fig.1 Thread-Unsafe version



```
//Start Main function
int main(int argc, char *argv[]) {
  // Get initial value and threshold from command line arguments
  initVal = atoi(argv[1]);
  threashold = atoi(argv[2]);
  // read data from in.txt file
  std::fstream ifile ;
  ifile.open("in.txt",std::ios::in) ;
  ifile>> numMultiply >> MultiplyVal ;
  ifile>> numDivide >>DivideVal ;
  ifile>> numAdd >> AddVal ;
  ifile>> numSubtract >> SubtractVal ;
  ifile.close() ;
  // create maniger threads
  std::thread thread1(M_multiply);
  std::thread thread2(M_Divide);
  thread1.join();
  thread2.join();
  std::thread thread3(M_Add);
  std::thread thread4(M_subtract);
  thread3.join();
  thread4.join();
  // print in out.txt file
  std::fstream ofile ;
  ofile.open("out.txt",std::ios::out) ;
  ofile << fixed << setprecision(6) ;
  ofile << "Main: Number of Multipliers="<<numMultiply <<", Amount to multiply="<<MultiplyVal<<endl;
  ofile << "Main: Number of Dividers="<<numDivide <<", Amount to divided="<<DivideVal<<endl;
  ofile << "Main: Number of Adders="<<numAdd <<", Amount to added="<<AddVal<<endl;
  ofile << "Main: Number of Subtractors="<<numSubtract <<", Amount to subtracted="<<SubtractVal<<endl;
  ofile << "Main: finalValue=" << CurrentVal << ", belowThreashold=" << belowThreashold << ", aboveThreashold=" << aboveThreashold << ", equalThreashold=" << equalThreashold << endl ;
  ofile.close() ;
  return 0 ;
}
```

Fig. 1.1 main function

This is the main function of the Thread-unsafe code.

The main function will get two inputs from the user via the command line, indicating the initial value and the threshold, respectively, and will then receive the number of threads for each type and the amount for each operation via an input file "in.txt."

The function creates two threads for multiplication and division operations, then uses (join) to wait until the threads finish their work, then creates two threads for addition and subtraction operations, followed by uses (join) to wait until the threads finish their work and continues the function work.

The multiplication and division threads will run in parallel, and after that, the addition and subtraction threads will run in parallel as well, so that the preceding threads are not interrupted.

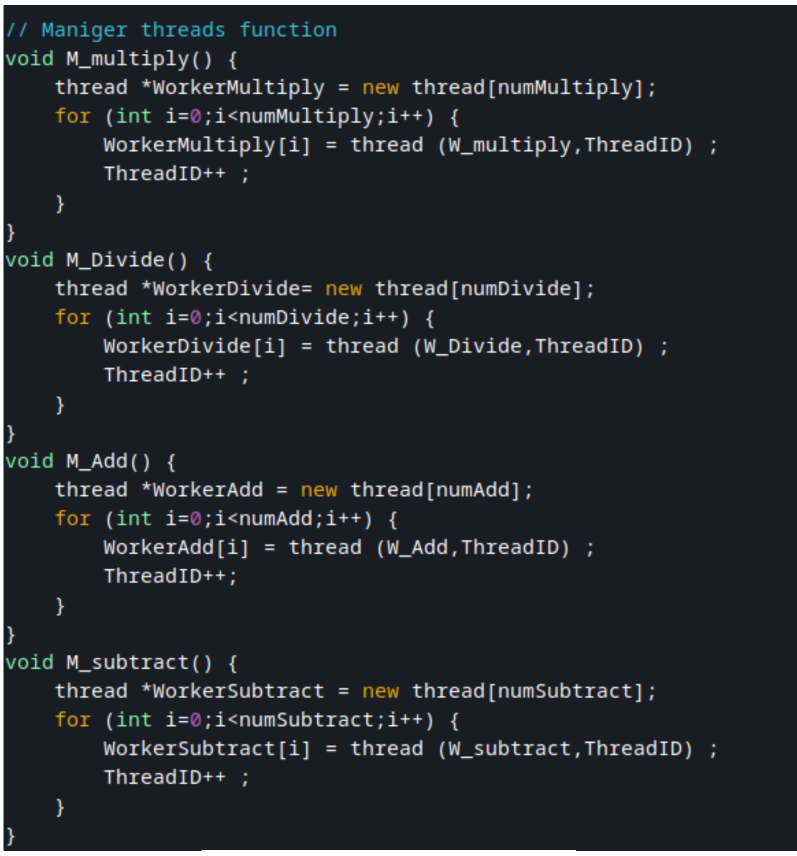
Finally, the function will print the results on the "out.txt" file.

In Fig. 1.2 we have the manager threads functions we mentioned above, each of the following functions will do:

- Create a dynamic array of type threads with size equal to value passed from main function that will contain worker threads of each operation type.
- A loop will itterate among the array creating worker threads of that type each calling the worker function and passing the thread ID to that function (more on that later).
- And then increase the thread ID (ThreadID) by one for each worker thread.

Notice that multiple worker threads of type (Multiply and Divide) or (Add and Subtract) will race on the global variables:

- ThreadID



```
// Maniger threads function
void M_multiply() {
    thread *WorkerMultiply = new thread[numMultiply];
    for (int i=0;i<numMultiply;i++) {
        WorkerMultiply[i] = thread (W_multiply,ThreadID) ;
        ThreadID++ ;
    }
}
void M_Divide() {
    thread *WorkerDivide= new thread[numDivide];
    for (int i=0;i<numDivide;i++) {
        WorkerDivide[i] = thread (W_Divide,ThreadID) ;
        ThreadID++ ;
    }
}
void M_Add() {
    thread *WorkerAdd = new thread[numAdd];
    for (int i=0;i<numAdd;i++) {
        WorkerAdd[i] = thread (W_Add,ThreadID) ;
        ThreadID++;
    }
}
void M_subtract() {
    thread *WorkerSubtract = new thread[numSubtract];
    for (int i=0;i<numSubtract;i++) {
        WorkerSubtract[i] = thread (W_subtract,ThreadID) ;
        ThreadID++ ;
    }
}
```

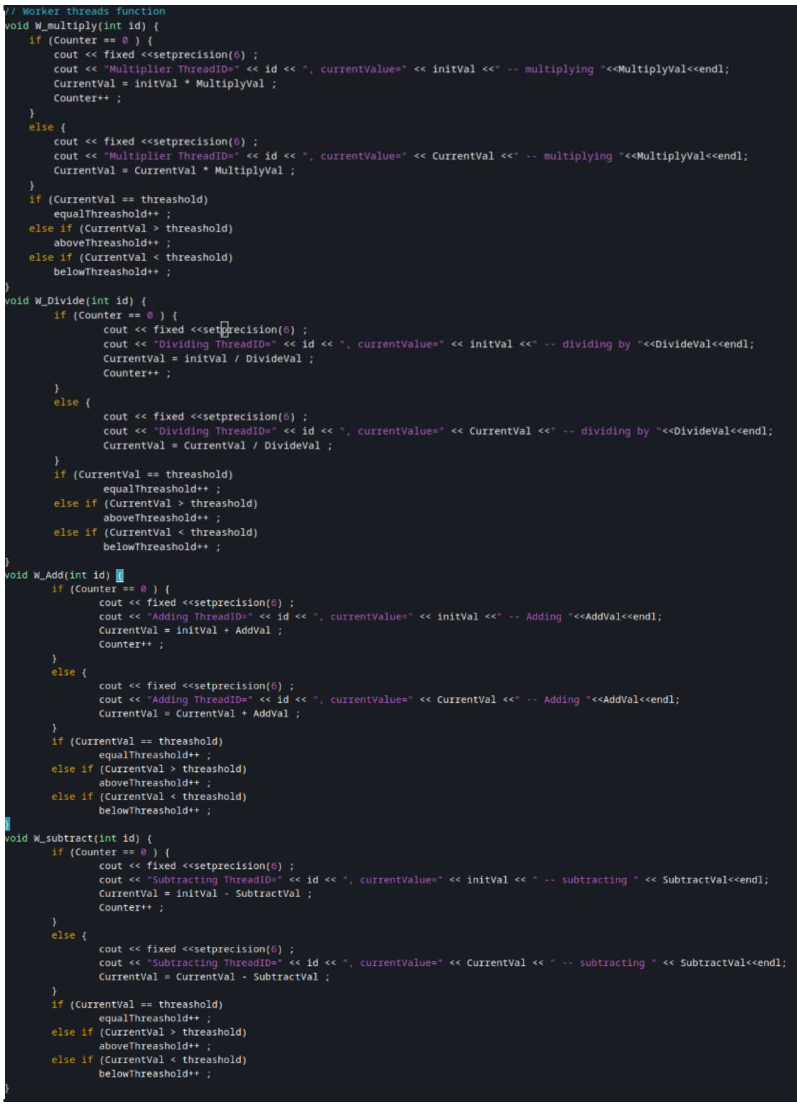
Fig. 1.2 Manager threads

In Fig. 1.3 we have the worker threads functions we mentioned above, each of the following functions will do:

- Check via if statement if "Counter" equal to zero if this is the first thread to operate so it takes the iniatal value. If so Print the message and increment the Counter so that it will never use the initail value again.
- Else it will use the current value after computing to get a new value for "CurrentVal".
- Compare the "CurrentVal" with the given Threshold.

Notice that multiple worker threads of type (Multiply and Divide) or (Add and Subtract) will race on the global variables:

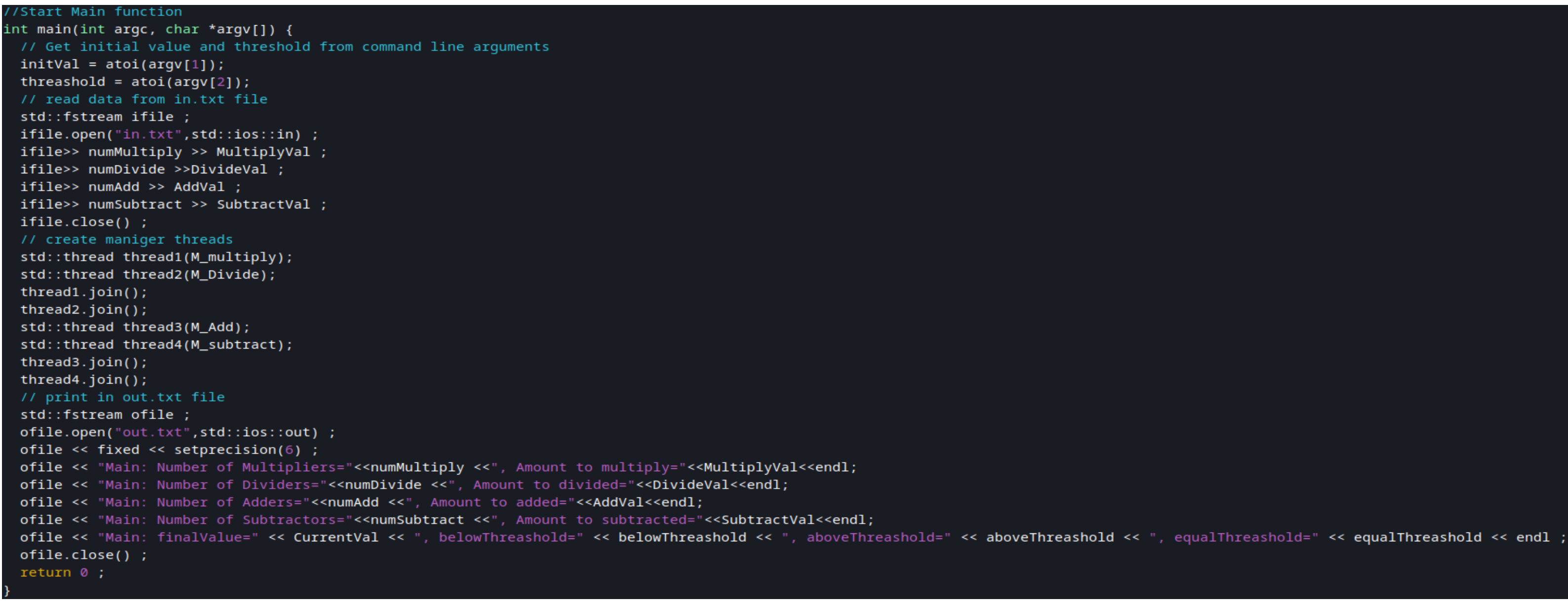
- CurruntVal
- Counter
- belowThreashold
- aboveThreashold
- equalThreashold



```
// Worker threads function
void W_multiply(int id) {
    if (Counter == 0 ) {
        cout << fixed <<setprecision(6) ;
        cout << "Multiplier ThreadID=" << id << ", currentValue=" << initVal <<" -- multiplying "<<MultiplyVal<<endl;
        CurrentVal = initVal * MultiplyVal ;
        Counter++ ;
    }
    else {
        cout << fixed <<setprecision(6) ;
        cout << "Multiplier ThreadID=" << id << ", currentValue=" << CurrentVal <<" -- multiplying "<<MultiplyVal<<endl;
        CurrentVal = CurrentVal * MultiplyVal ;
    }
    if (CurrentVal == threashold)
        equalThreashold++ ;
    else if (CurrentVal > threashold)
        aboveThreashold++ ;
    else if (CurrentVal < threashold)
        belowThreashold++ ;
}
void W_Divide(int id) {
        if (Counter == 0 ) {
                cout << fixed <<setprecision(6) ;
                cout << "Dividing ThreadID=" << id << ", currentValue=" << initVal <<" -- dividing by "<<DivideVal<<endl;
                CurrentVal = initVal / DivideVal ;
                Counter++ ;
        }
        else {
                cout << fixed <<setprecision(6) ;
                cout << "Dividing ThreadID=" << id << ", currentValue=" << CurrentVal <<" -- dividing by "<<DivideVal<<endl;
                CurrentVal = CurrentVal / DivideVal ;
        }
        if (CurrentVal == threashold)
                equalThreashold++ ;
        else if (CurrentVal > threashold)
                aboveThreashold++ ;
        else if (CurrentVal < threashold)
                belowThreashold++ ;
}
void W_Add(int id) {
        if (Counter == 0 ) {
                cout << fixed <<setprecision(6) ;
                cout << "Adding ThreadID=" << id << ", currentValue=" << initVal <<" -- Adding "<<AddVal<<endl;
                CurrentVal = initVal + AddVal ;
                Counter++ ;
        }
        else {
                cout << fixed <<setprecision(6) ;
                cout << "Adding ThreadID=" << id << ", currentValue=" << CurrentVal <<" -- Adding "<<AddVal<<endl;
                CurrentVal = CurrentVal + AddVal ;
        }
        if (CurrentVal == threashold)
                equalThreashold++ ;
        else if (CurrentVal > threashold)
                aboveThreashold++ ;
        else if (CurrentVal < threashold)
                belowThreashold++ ;
}
void W_subtract(int id) {
        if (Counter == 0 ) {
                cout << fixed <<setprecision(6) ;
                cout << "Subtracting ThreadID=" << id << ", currentValue=" << initVal << " -- subtracting " << SubtractVal<<endl;
                CurrentVal = initVal - SubtractVal ;
                Counter++ ;
        }
        else {
                cout << fixed <<setprecision(6) ;
                cout << "Subtracting ThreadID=" << id << ", currentValue=" << CurrentVal << " -- subtracting " << SubtractVal<<endl;
                CurrentVal = CurrentVal - SubtractVal ;
        }
        if (CurrentVal == threashold)
                equalThreashold++ ;
        else if (CurrentVal > threashold)
                aboveThreashold++ ;
        else if (CurrentVal < threashold)
                belowThreashold++ ;
}
```

Fig. 1.3 Worker Threads

Fig. 2: Thread-Safe



```
//Start Main function
int main(int argc, char *argv[]) {
  // Get initial value and threshold from command line arguments
  initVal = atoi(argv[1]);
  threashold = atoi(argv[2]);
  // read data from in.txt file
  std::fstream ifile ;
  ifile.open("in.txt",std::ios::in) ;
  ifile>> numMultiply >> MultiplyVal ;
  ifile>> numDivide >>DivideVal ;
  ifile>> numAdd >> AddVal ;
  ifile>> numSubtract >> SubtractVal ;
  ifile.close() ;
  // create maniger threads
  std::thread thread1(M_multiply);
  std::thread thread2(M_Divide);
  thread1.join();
  thread2.join();
  std::thread thread3(M_Add);
  std::thread thread4(M_subtract);
  thread3.join();
  thread4.join();
  // print in out.txt file
  std::fstream ofile ;
  ofile.open("out.txt",std::ios::out) ;
  ofile << fixed << setprecision(6) ;
  ofile << "Main: Number of Multipliers="<<numMultiply <<", Amount to multiply="<<MultiplyVal<<endl;
  ofile << "Main: Number of Dividers="<<numDivide <<", Amount to divided="<<DivideVal<<endl;
  ofile << "Main: Number of Adders="<<numAdd <<", Amount to added="<<AddVal<<endl;
  ofile << "Main: Number of Subtractors="<<numSubtract <<", Amount to subtracted="<<SubtractVal<<endl;
  ofile << "Main: finalValue=" << CurrentVal << ", belowThreashold=" << belowThreashold << ", aboveThreashold=" << aboveThreashold << ", equalThreashold=" << equalThreashold << endl ;
  ofile.close() ;
  return 0 ;
}
```

Fig. 2.1 main function

This is the main function of the Thread-safe code.

The main is the same of the main function of thread-unsafe code, the code will change in the manager and worker parts.

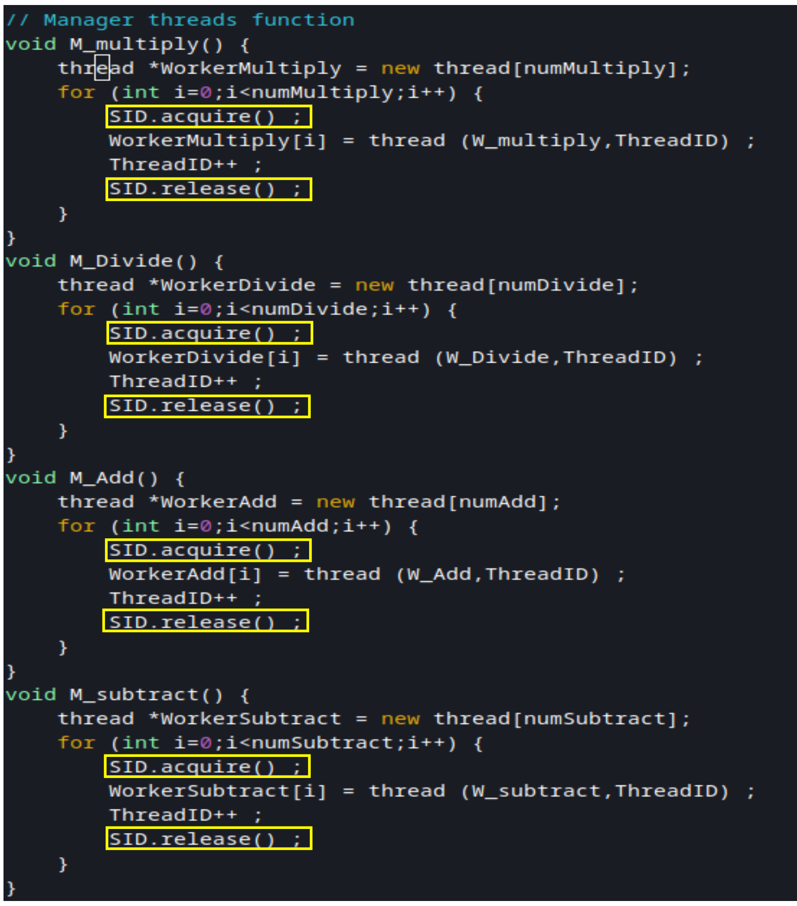
We used binary semaphores to resolve the race condition problem.

We defined binary semaphores for the following global variables:

- TID → SID
- CurrentVal → SCurrentVal
- For the operation functions critical section:
  - Sthreashold
  - Sworker

In Fig. 2.2 you can see that we highlighted the semaphores we used to resolve the race conditions that is happening on the manager thread's function.

These semaphores will ensure that each worker thread gets its own ThreadID value.

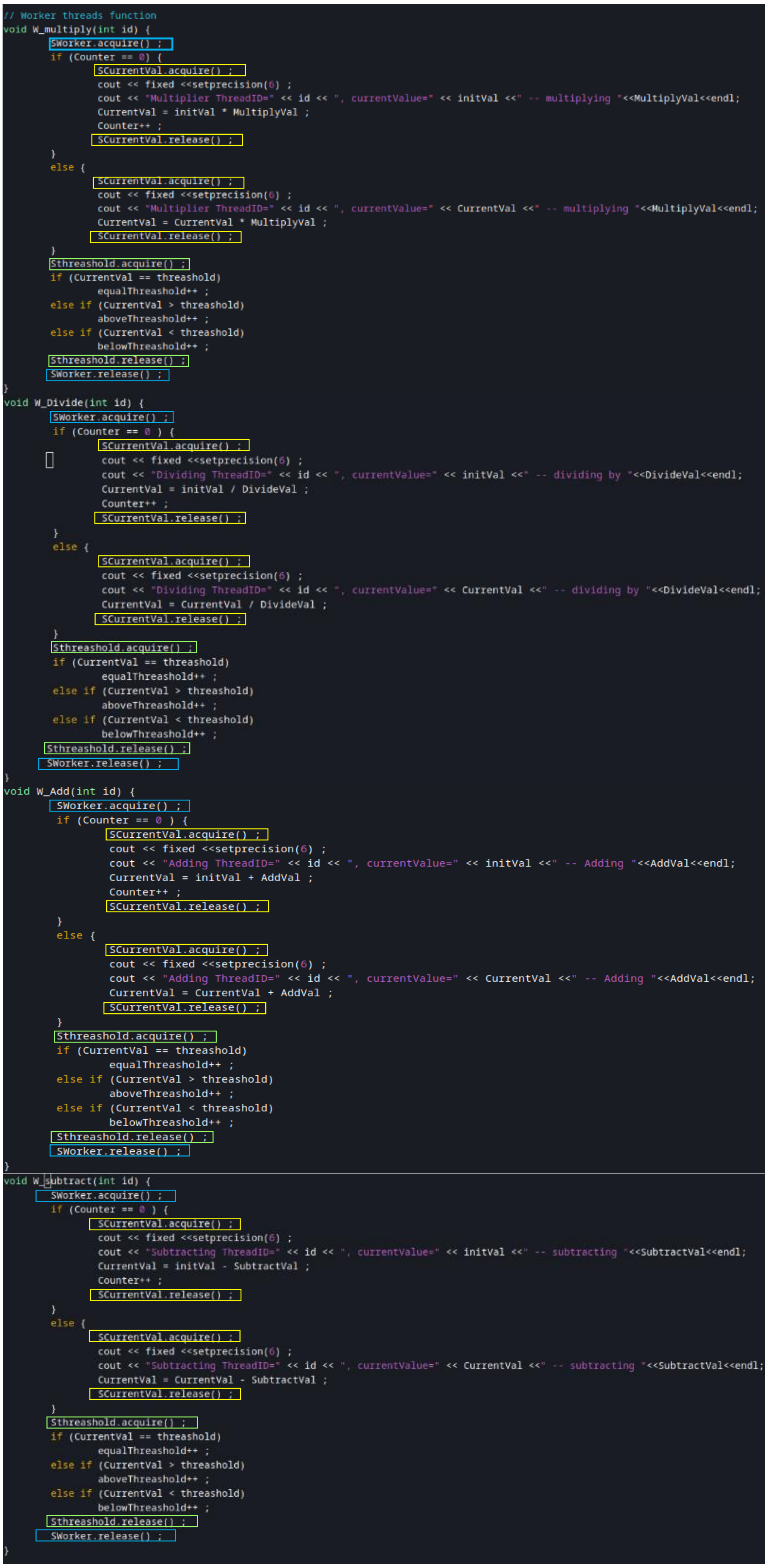


```
// Manager threads function
void M_multiply() {
    thread *WorkerMultiply = new thread[numMultiply];
    for (int i=0;i<numMultiply;i++) {
        SID.acquire() ;
        WorkerMultiply[i] = thread (W_multiply,ThreadID) ;
        ThreadID++ ;
        SID.release() ;
    }
}
void M_Divide() {
    thread *WorkerDivide = new thread[numDivide];
    for (int i=0;i<numDivide;i++) {
        SID.acquire() ;
        WorkerDivide[i] = thread (W_Divide,ThreadID) ;
        ThreadID++ ;
        SID.release() ;
    }
}
void M_Add() {
    thread *WorkerAdd = new thread[numAdd];
    for (int i=0;i<numAdd;i++) {
        SID.acquire() ;
        WorkerAdd[i] = thread (W_Add,ThreadID) ;
        ThreadID++ ;
        SID.release() ;
    }
}
void M_subtract() {
    thread *WorkerSubtract = new thread[numSubtract];
    for (int i=0;i<numSubtract;i++) {
        SID.acquire() ;
        WorkerSubtract[i] = thread (W_subtract,ThreadID) ;
        ThreadID++ ;
        SID.release() ;
    }
}
```

Fig. 2.2 Manager Threads

In Fig. 2.3 you can see that we highlighted the semaphores we used to resolve the race conditions that is happening on the Worker thread's function.

These semaphores will ensure that every worker thread can modify the accumulated value "CurrentVal" variable only when it is the only thread modifying it, it also ensures that no other thread can modify the threshold counters other than the current working worker thread.

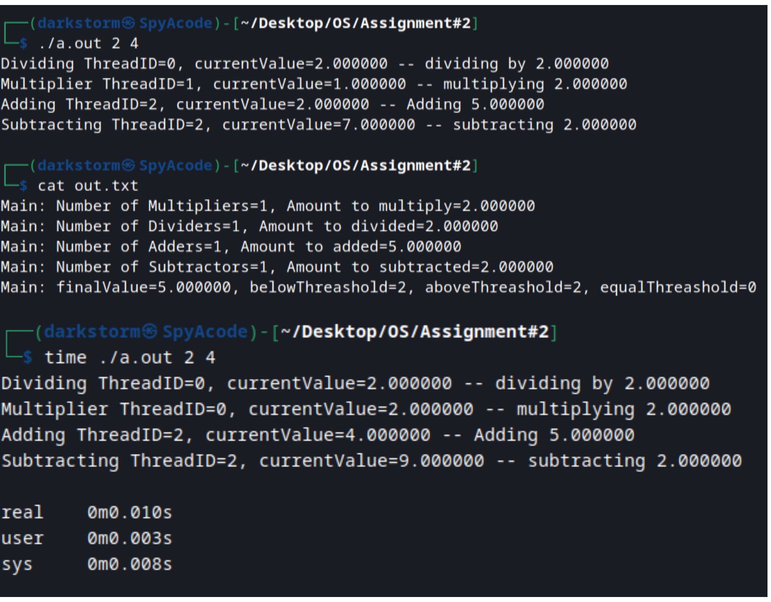


```
// Worker threads function
void W_multiply(int id) {
        SWorker.acquire() ;
        if (Counter == 0) {
                SCurrentVal.acquire() ;
                cout << fixed <<setprecision(6) ;
                cout << "Multiplier ThreadID=" << id << ", currentValue=" << initVal <<" -- multiplying "<<MultiplyVal<<endl;
                CurrentVal = initVal * MultiplyVal ;
                Counter++ ;
                SCurrentVal.release() ;
        }
        else {
                SCurrentVal.acquire() ;
                cout << fixed <<setprecision(6) ;
                cout << "Multiplier ThreadID=" << id << ", currentValue=" << CurrentVal <<" -- multiplying "<<MultiplyVal<<endl;
                CurrentVal = CurrentVal * MultiplyVal ;
                SCurrentVal.release() ;
        }
        Sthreashold.acquire() ;
        if (CurrentVal == threashold)
                equalThreashold++ ;
        else if (CurrentVal > threashold)
                aboveThreashold++ ;
        else if (CurrentVal < threashold)
                belowThreashold++ ;
        Sthreashold.release() ;
        SWorker.release() ;
}
void W_Divide(int id) {
        SWorker.acquire() ;
        if (Counter == 0 ) {
                SCurrentVal.acquire() ;
                cout << fixed <<setprecision(6) ;
                cout << "Dividing ThreadID=" << id << ", currentValue=" << initVal <<" -- dividing by "<<DivideVal<<endl;
                CurrentVal = initVal / DivideVal ;
                Counter++ ;
                SCurrentVal.release() ;
        }
        else {
                SCurrentVal.acquire() ;
                cout << fixed <<setprecision(6) ;
                cout << "Dividing ThreadID=" << id << ", currentValue=" << CurrentVal <<" -- dividing by "<<DivideVal<<endl;
                CurrentVal = CurrentVal / DivideVal ;
                SCurrentVal.release() ;
        }
        Sthreashold.acquire() ;
        if (CurrentVal == threashold)
                equalThreashold++ ;
        else if (CurrentVal > threashold)
                aboveThreashold++ ;
        else if (CurrentVal < threashold)
                belowThreashold++ ;
        Sthreashold.release() ;
        SWorker.release() ;
}
void W_Add(int id) {
        SWorker.acquire() ;
        if (Counter == 0 ) {
                SCurrentVal.acquire() ;
                cout << fixed <<setprecision(6) ;
                cout << "Adding ThreadID=" << id << ", currentValue=" << initVal <<" -- Adding "<<AddVal<<endl;
                CurrentVal = initVal + AddVal ;
                Counter++ ;
                SCurrentVal.release() ;
        }
        else {
                SCurrentVal.acquire() ;
                cout << fixed <<setprecision(6) ;
                cout << "Adding ThreadID=" << id << ", currentValue=" << CurrentVal <<" -- Adding "<<AddVal<<endl;
                CurrentVal = CurrentVal + AddVal ;
                SCurrentVal.release() ;
        }
        Sthreashold.acquire() ;
        if (CurrentVal == threashold)
                equalThreashold++ ;
        else if (CurrentVal > threashold)
                aboveThreashold++ ;
        else if (CurrentVal < threashold)
                belowThreashold++ ;
        Sthreashold.release() ;
        SWorker.release() ;
}
void W_subtract(int id) {
        SWorker.acquire() ;
        if (Counter == 0 ) {
                SCurrentVal.acquire() ;
                cout << fixed <<setprecision(6) ;
                cout << "Subtracting ThreadID=" << id << ", currentValue=" << initVal <<" -- subtracting "<<SubtractVal<<endl;
                CurrentVal = initVal - SubtractVal ;
                Counter++ ;
                SCurrentVal.release() ;
        }
        else {
                SCurrentVal.acquire() ;
                cout << fixed <<setprecision(6) ;
                cout << "Subtracting ThreadID=" << id << ", currentValue=" << CurrentVal <<" -- subtracting "<<SubtractVal<<endl;
                CurrentVal = CurrentVal - SubtractVal ;
                SCurrentVal.release() ;
        }
        Sthreashold.acquire() ;
        if (CurrentVal == threashold)
                equalThreashold++ ;
        else if (CurrentVal > threashold)
                aboveThreashold++ ;
        else if (CurrentVal < threashold)
                belowThreashold++ ;
        Sthreashold.release() ;
        SWorker.release() ;
}
```

Fig. 2.3 Worker Threads

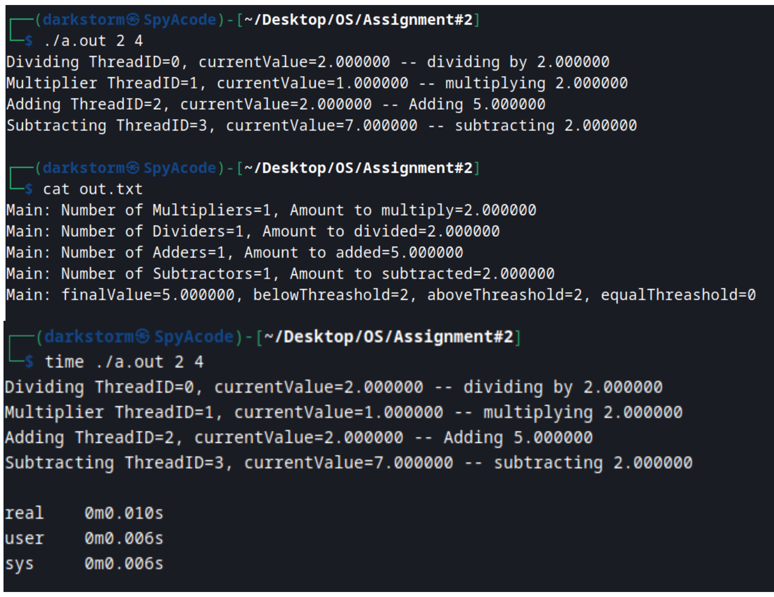
Comparing Both codes

Case 1: Using 4 threads.



```
┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ ./a.out 2 4
Dividing ThreadID=0, currentValue=2.000000 -- dividing by 2.000000
Multiplier ThreadID=1, currentValue=1.000000 -- multiplying 2.000000
Adding ThreadID=2, currentValue=2.000000 -- Adding 5.000000
Subtracting ThreadID=2, currentValue=7.000000 -- subtracting 2.000000

┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ cat out.txt
Main: Number of Multipliers=1, Amount to multiply=2.000000
Main: Number of Dividers=1, Amount to divided=2.000000
Main: Number of Adders=1, Amount to added=5.000000
Main: Number of Subtractors=1, Amount to subtracted=2.000000
Main: finalValue=5.000000, belowThreashold=2, aboveThreashold=2, equalThreashold=0

┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ time ./a.out 2 4
Dividing ThreadID=0, currentValue=2.000000 -- dividing by 2.000000
Multiplier ThreadID=0, currentValue=2.000000 -- multiplying 2.000000
Adding ThreadID=2, currentValue=4.000000 -- Adding 5.000000
Subtracting ThreadID=2, currentValue=9.000000 -- subtracting 2.000000

real    0m0.010s
user    0m0.003s
sys     0m0.008s
```

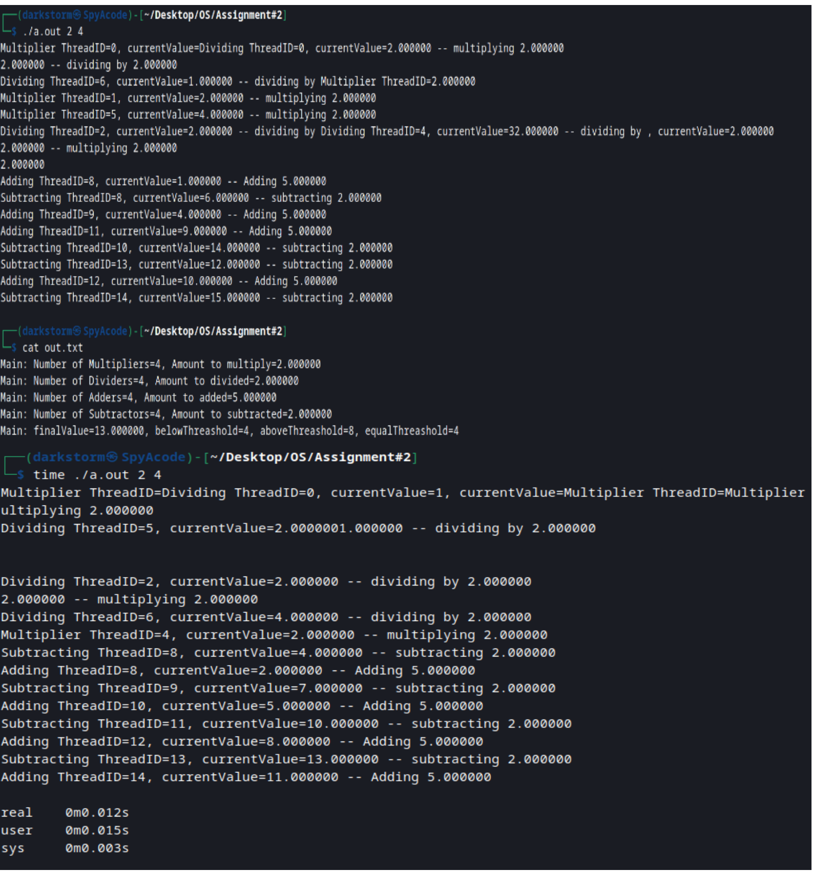
Fig. 3.1 Thread-unsafe output and time



```
┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ ./a.out 2 4
Dividing ThreadID=0, currentValue=2.000000 -- dividing by 2.000000
Multiplier ThreadID=1, currentValue=1.000000 -- multiplying 2.000000
Adding ThreadID=2, currentValue=2.000000 -- Adding 5.000000
Subtracting ThreadID=3, currentValue=7.000000 -- subtracting 2.000000

┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ cat out.txt
Main: Number of Multipliers=1, Amount to multiply=2.000000
Main: Number of Dividers=1, Amount to divided=2.000000
Main: Number of Adders=1, Amount to added=5.000000
Main: Number of Subtractors=1, Amount to subtracted=2.000000
Main: finalValue=5.000000, belowThreashold=2, aboveThreashold=2, equalThreashold=0

┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ time ./a.out 2 4
Dividing ThreadID=0, currentValue=2.000000 -- dividing by 2.000000
Multiplier ThreadID=1, currentValue=1.000000 -- multiplying 2.000000
Adding ThreadID=2, currentValue=2.000000 -- Adding 5.000000
Subtracting ThreadID=3, currentValue=7.000000 -- subtracting 2.000000

real    0m0.010s
user    0m0.006s
sys     0m0.006s
```

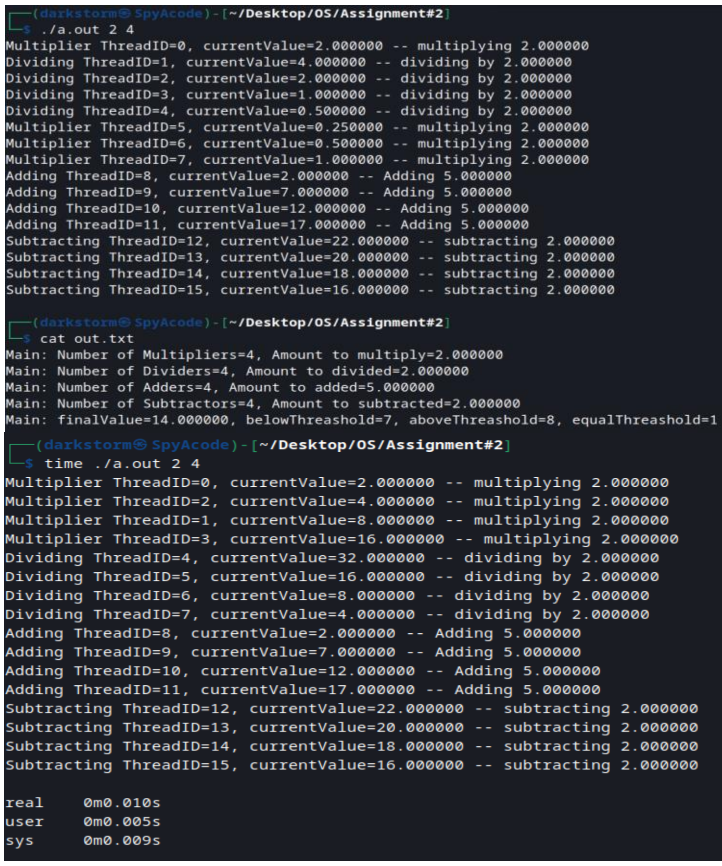
Fig. 3.2 Thread-safe output and time

Case 2: Using 16 threads.



```
┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ ./a.out 2 4
Multiplier ThreadID=0, currentValue=Dividing ThreadID=0, currentValue=2.000000 -- multiplying 2.000000
2.000000 -- dividing by 2.000000
Dividing ThreadID=6, currentValue=1.000000 -- dividing by Multiplier ThreadID=2.000000
Multiplier ThreadID=1, currentValue=2.000000 -- multiplying 2.000000
Multiplier ThreadID=5, currentValue=4.000000 -- multiplying 2.000000
Dividing ThreadID=2, currentValue=2.000000 -- dividing by Dividing ThreadID=4, currentValue=32.000000 -- dividing by , currentValue=2.000000
2.000000 -- multiplying 2.000000
2.000000
Adding ThreadID=8, currentValue=1.000000 -- Adding 5.000000
Subtracting ThreadID=8, currentValue=6.000000 -- subtracting 2.000000
Adding ThreadID=9, currentValue=4.000000 -- Adding 5.000000
Adding ThreadID=11, currentValue=9.000000 -- Adding 5.000000
Subtracting ThreadID=10, currentValue=14.000000 -- subtracting 2.000000
Subtracting ThreadID=13, currentValue=12.000000 -- subtracting 2.000000
Adding ThreadID=12, currentValue=10.000000 -- Adding 5.000000
Subtracting ThreadID=14, currentValue=15.000000 -- subtracting 2.000000

┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ cat out.txt
Main: Number of Multipliers=4, Amount to multiply=2.000000
Main: Number of Dividers=4, Amount to divided=2.000000
Main: Number of Adders=4, Amount to added=5.000000
Main: Number of Subtractors=4, Amount to subtracted=2.000000
Main: finalValue=13.000000, belowThreashold=4, aboveThreashold=8, equalThreashold=4

┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ time ./a.out 2 4
Multiplier ThreadID=Dividing ThreadID=0, currentValue=1, currentValue=Multiplier ThreadID=Multiplier
ultiplying 2.000000
Dividing ThreadID=5, currentValue=2.0000001.000000 -- dividing by 2.000000


Dividing ThreadID=2, currentValue=2.000000 -- dividing by 2.000000
2.000000 -- multiplying 2.000000
Dividing ThreadID=6, currentValue=4.000000 -- dividing by 2.000000
Multiplier ThreadID=4, currentValue=2.000000 -- multiplying 2.000000
Subtracting ThreadID=8, currentValue=4.000000 -- subtracting 2.000000
Adding ThreadID=8, currentValue=2.000000 -- Adding 5.000000
Subtracting ThreadID=9, currentValue=7.000000 -- subtracting 2.000000
Adding ThreadID=10, currentValue=5.000000 -- Adding 5.000000
Subtracting ThreadID=11, currentValue=10.000000 -- subtracting 2.000000
Adding ThreadID=12, currentValue=8.000000 -- Adding 5.000000
Subtracting ThreadID=13, currentValue=13.000000 -- subtracting 2.000000
Adding ThreadID=14, currentValue=11.000000 -- Adding 5.000000

real    0m0.012s
user    0m0.015s
sys     0m0.003s
```

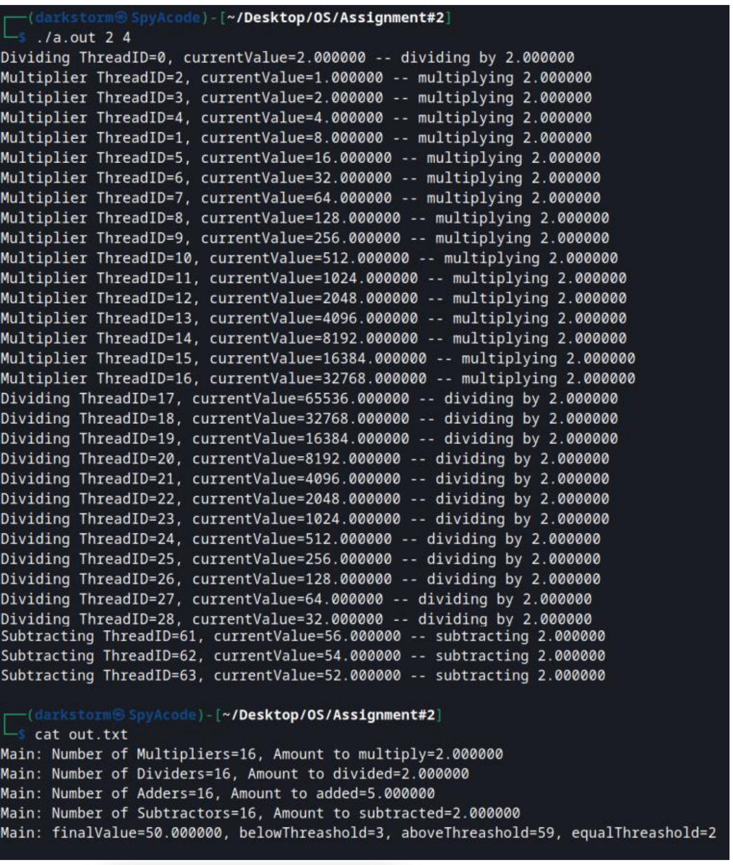
Fig. 3.3 Thread-unsafe output and time



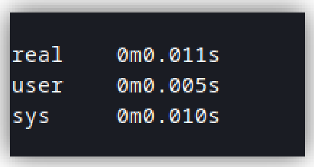
```
┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ ./a.out 2 4
Multiplier ThreadID=0, currentValue=2.000000 -- multiplying 2.000000
Dividing ThreadID=1, currentValue=4.000000 -- dividing by 2.000000
Dividing ThreadID=2, currentValue=2.000000 -- dividing by 2.000000
Dividing ThreadID=3, currentValue=1.000000 -- dividing by 2.000000
Dividing ThreadID=4, currentValue=0.500000 -- dividing by 2.000000
Multiplier ThreadID=5, currentValue=0.250000 -- multiplying 2.000000
Multiplier ThreadID=6, currentValue=0.500000 -- multiplying 2.000000
Multiplier ThreadID=7, currentValue=1.000000 -- multiplying 2.000000
Adding ThreadID=8, currentValue=2.000000 -- Adding 5.000000
Adding ThreadID=9, currentValue=7.000000 -- Adding 5.000000
Adding ThreadID=10, currentValue=12.000000 -- Adding 5.000000
Adding ThreadID=11, currentValue=17.000000 -- Adding 5.000000
Subtracting ThreadID=12, currentValue=22.000000 -- subtracting 2.000000
Subtracting ThreadID=13, currentValue=20.000000 -- subtracting 2.000000
Subtracting ThreadID=14, currentValue=18.000000 -- subtracting 2.000000
Subtracting ThreadID=15, currentValue=16.000000 -- subtracting 2.000000

┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ cat out.txt
Main: Number of Multipliers=4, Amount to multiply=2.000000
Main: Number of Dividers=4, Amount to divided=2.000000
Main: Number of Adders=4, Amount to added=5.000000
Main: Number of Subtractors=4, Amount to subtracted=2.000000
Main: finalValue=14.000000, belowThreashold=7, aboveThreashold=8, equalThreashold=1
┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ time ./a.out 2 4
Multiplier ThreadID=0, currentValue=2.000000 -- multiplying 2.000000
Multiplier ThreadID=2, currentValue=4.000000 -- multiplying 2.000000
Multiplier ThreadID=1, currentValue=8.000000 -- multiplying 2.000000
Multiplier ThreadID=3, currentValue=16.000000 -- multiplying 2.000000
Dividing ThreadID=4, currentValue=32.000000 -- dividing by 2.000000
Dividing ThreadID=5, currentValue=16.000000 -- dividing by 2.000000
Dividing ThreadID=6, currentValue=8.000000 -- dividing by 2.000000
Dividing ThreadID=7, currentValue=4.000000 -- dividing by 2.000000
Adding ThreadID=8, currentValue=2.000000 -- Adding 5.000000
Adding ThreadID=9, currentValue=7.000000 -- Adding 5.000000
Adding ThreadID=10, currentValue=12.000000 -- Adding 5.000000
Adding ThreadID=11, currentValue=17.000000 -- Adding 5.000000
Subtracting ThreadID=12, currentValue=22.000000 -- subtracting 2.000000
Subtracting ThreadID=13, currentValue=20.000000 -- subtracting 2.000000
Subtracting ThreadID=14, currentValue=18.000000 -- subtracting 2.000000
Subtracting ThreadID=15, currentValue=16.000000 -- subtracting 2.000000

real    0m0.010s
user    0m0.005s
sys     0m0.009s
```

Fig. 3.4 Thread-safe output and time

Case 3: Using 64 threads.

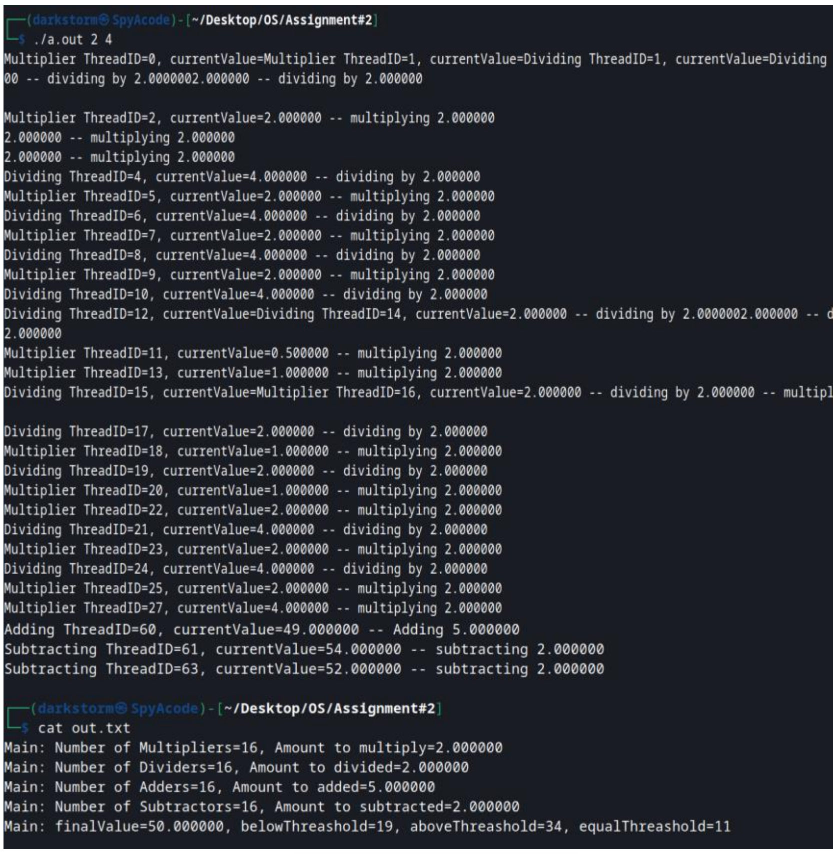


```
┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ ./a.out 2 4
Dividing ThreadID=0, currentValue=2.000000 -- dividing by 2.000000
Multiplier ThreadID=2, currentValue=1.000000 -- multiplying 2.000000
Multiplier ThreadID=3, currentValue=2.000000 -- multiplying 2.000000
Multiplier ThreadID=4, currentValue=4.000000 -- multiplying 2.000000
Multiplier ThreadID=1, currentValue=8.000000 -- multiplying 2.000000
Multiplier ThreadID=5, currentValue=16.000000 -- multiplying 2.000000
Multiplier ThreadID=6, currentValue=32.000000 -- multiplying 2.000000
Multiplier ThreadID=7, currentValue=64.000000 -- multiplying 2.000000
Multiplier ThreadID=8, currentValue=128.000000 -- multiplying 2.000000
Multiplier ThreadID=9, currentValue=256.000000 -- multiplying 2.000000
Multiplier ThreadID=10, currentValue=512.000000 -- multiplying 2.000000
Multiplier ThreadID=11, currentValue=1024.000000 -- multiplying 2.000000
Multiplier ThreadID=12, currentValue=2048.000000 -- multiplying 2.000000
Multiplier ThreadID=13, currentValue=4096.000000 -- multiplying 2.000000
Multiplier ThreadID=14, currentValue=8192.000000 -- multiplying 2.000000
Multiplier ThreadID=15, currentValue=16384.000000 -- multiplying 2.000000
Multiplier ThreadID=16, currentValue=32768.000000 -- multiplying 2.000000
Dividing ThreadID=17, currentValue=65536.000000 -- dividing by 2.000000
Dividing ThreadID=18, currentValue=32768.000000 -- dividing by 2.000000
Dividing ThreadID=19, currentValue=16384.000000 -- dividing by 2.000000
Dividing ThreadID=20, currentValue=8192.000000 -- dividing by 2.000000
Dividing ThreadID=21, currentValue=4096.000000 -- dividing by 2.000000
Dividing ThreadID=22, currentValue=2048.000000 -- dividing by 2.000000
Dividing ThreadID=23, currentValue=1024.000000 -- dividing by 2.000000
Dividing ThreadID=24, currentValue=512.000000 -- dividing by 2.000000
Dividing ThreadID=25, currentValue=256.000000 -- dividing by 2.000000
Dividing ThreadID=26, currentValue=128.000000 -- dividing by 2.000000
Dividing ThreadID=27, currentValue=64.000000 -- dividing by 2.000000
Dividing ThreadID=28, currentValue=32.000000 -- dividing by 2.000000
Subtracting ThreadID=61, currentValue=56.000000 -- subtracting 2.000000
Subtracting ThreadID=62, currentValue=54.000000 -- subtracting 2.000000
Subtracting ThreadID=63, currentValue=52.000000 -- subtracting 2.000000

┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ cat out.txt
Main: Number of Multipliers=16, Amount to multiply=2.000000
Main: Number of Dividers=16, Amount to divided=2.000000
Main: Number of Adders=16, Amount to added=5.000000
Main: Number of Subtractors=16, Amount to subtracted=2.000000
Main: finalValue=50.000000, belowThreashold=3, aboveThreashold=59, equalThreashold=2
```

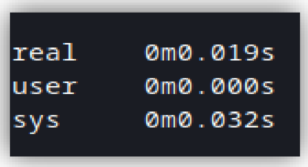


```
real    0m0.011s
user    0m0.005s
sys     0m0.010s
```

Fig. 3.5 Thread-safe output and time.



```
┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ ./a.out 2 4
Multiplier ThreadID=0, currentValue=Multiplier ThreadID=1, currentValue=Dividing ThreadID=1, currentValue=Dividing
00 -- dividing by 2.0000002.000000 -- dividing by 2.000000

Multiplier ThreadID=2, currentValue=2.000000 -- multiplying 2.000000
2.000000 -- multiplying 2.000000
2.000000 -- multiplying 2.000000
Dividing ThreadID=4, currentValue=4.000000 -- dividing by 2.000000
Multiplier ThreadID=5, currentValue=2.000000 -- multiplying 2.000000
Dividing ThreadID=6, currentValue=4.000000 -- dividing by 2.000000
Multiplier ThreadID=7, currentValue=2.000000 -- multiplying 2.000000
Dividing ThreadID=8, currentValue=4.000000 -- dividing by 2.000000
Multiplier ThreadID=9, currentValue=2.000000 -- multiplying 2.000000
Dividing ThreadID=10, currentValue=4.000000 -- dividing by 2.000000
Dividing ThreadID=12, currentValue=Dividing ThreadID=14, currentValue=2.000000 -- dividing by 2.0000002.000000 -- d
2.000000
Multiplier ThreadID=11, currentValue=0.500000 -- multiplying 2.000000
Multiplier ThreadID=13, currentValue=1.000000 -- multiplying 2.000000
Dividing ThreadID=15, currentValue=Multiplier ThreadID=16, currentValue=2.000000 -- dividing by 2.000000 -- multipl

Dividing ThreadID=17, currentValue=2.000000 -- dividing by 2.000000
Multiplier ThreadID=18, currentValue=1.000000 -- multiplying 2.000000
Dividing ThreadID=19, currentValue=2.000000 -- dividing by 2.000000
Multiplier ThreadID=20, currentValue=1.000000 -- multiplying 2.000000
Multiplier ThreadID=22, currentValue=2.000000 -- multiplying 2.000000
Dividing ThreadID=21, currentValue=4.000000 -- dividing by 2.000000
Multiplier ThreadID=23, currentValue=2.000000 -- multiplying 2.000000
Dividing ThreadID=24, currentValue=4.000000 -- dividing by 2.000000
Multiplier ThreadID=25, currentValue=2.000000 -- multiplying 2.000000
Multiplier ThreadID=27, currentValue=4.000000 -- multiplying 2.000000
Adding ThreadID=60, currentValue=49.000000 -- Adding 5.000000
Subtracting ThreadID=61, currentValue=54.000000 -- subtracting 2.000000
Subtracting ThreadID=63, currentValue=52.000000 -- subtracting 2.000000

┌──(darkstorm㉿SpyAcode)-[~/Desktop/OS/Assignment#2]
└─$ cat out.txt
Main: Number of Multipliers=16, Amount to multiply=2.000000
Main: Number of Dividers=16, Amount to divided=2.000000
Main: Number of Adders=16, Amount to added=5.000000
Main: Number of Subtractors=16, Amount to subtracted=2.000000
Main: finalValue=50.000000, belowThreashold=19, aboveThreashold=34, equalThreashold=11
```



```
real    0m0.019s
user    0m0.000s
sys     0m0.032s
```

Fig. 3.6 Thread-unsafe output and time.

In conclusion, we can see that the Thread-safe version gave a logical output while the Thread-unsafe version did not.

We can also conclude from the comparison earlier that the Thread-safe version is better in overall performance.

So, the answer to the question which is faster…

Surely it is the Thread-Safe version.