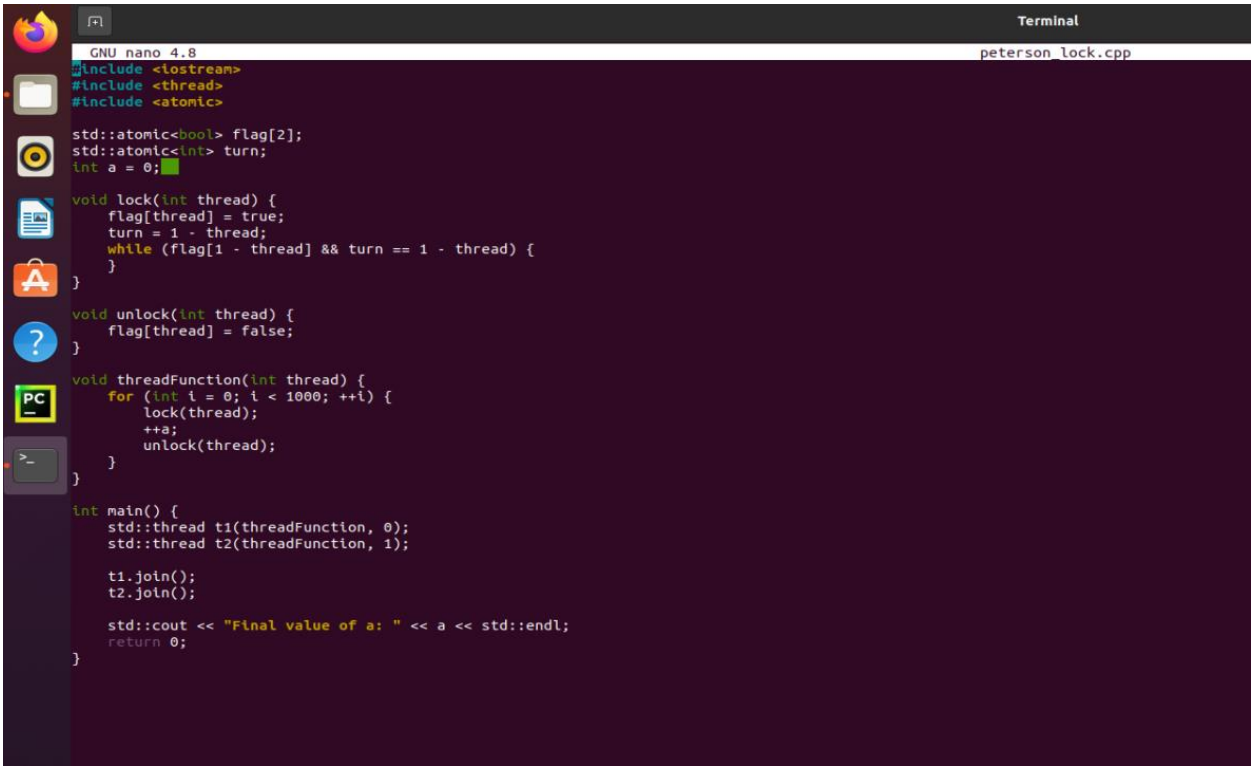
گزارش تمرین دوم سیستم عامل پیشرفته **محمدباربدامیرمزلقانی -۸۱۰۱۰۲۳۴۸**

نسخه ubuntu:

```
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.2 LTS
Release:        20.04
Codename:       focal
daii-VirtualBox :) >
```

سوال ۱:

در قسمت اول این سوال لازم بود تا الگوریتم قفل پترسون که سودوکد آن داده شده بود را در زبان C++ پیاده سازی کنیم در همین راستا ابتدا کد را میبینیم و سپس قسمت های مختلف آن را توضیح میدهم.



پس از اضافه کردن لایبرری های ورودی خروجی، ترد و اتمیک به سراغ تعریف متغیر های گلوبال میرویم در همین راستا ابتدا یک آرایه دوتایی بولین داریم که مشخص میکند آیا هر کدام از تردها میخواهند وارد حالت

بحرانی شوند یا خیر سپس یک متغیر اینتیجر داریم که نوبت ورود را مشخص میکند و در نهایت یک متغیر اشتراک گذاری شده که قرار است توسط ترد ها افزایش یابد.

تابع بعدی قفل است که ابتدا فلگی که در بالا تعریف کردیم را برای آن ترد صحیح میکنیم سپس نوبت را به بعدی میدهیم و وارد یک بیزی ویت میشویم که تا زمانی ادامه میابد که یا فلگ ترد دیگر فالس شود و یا نوبت به این ترد برسد.

در تابع unlock هم فلگ ترد حال حاضر را فالس میکنیم که یعنی از حالت بحرانی در میاید.

در بخش بعدی که threadFunction میباشد یک حلقه ۱۰۰۰ مرتبه ای تعریف کردیم، ابتدا لاک میکند سپس متغیر اشتراکی را یکی زیاد میکند و سپس آنلاک میکند.

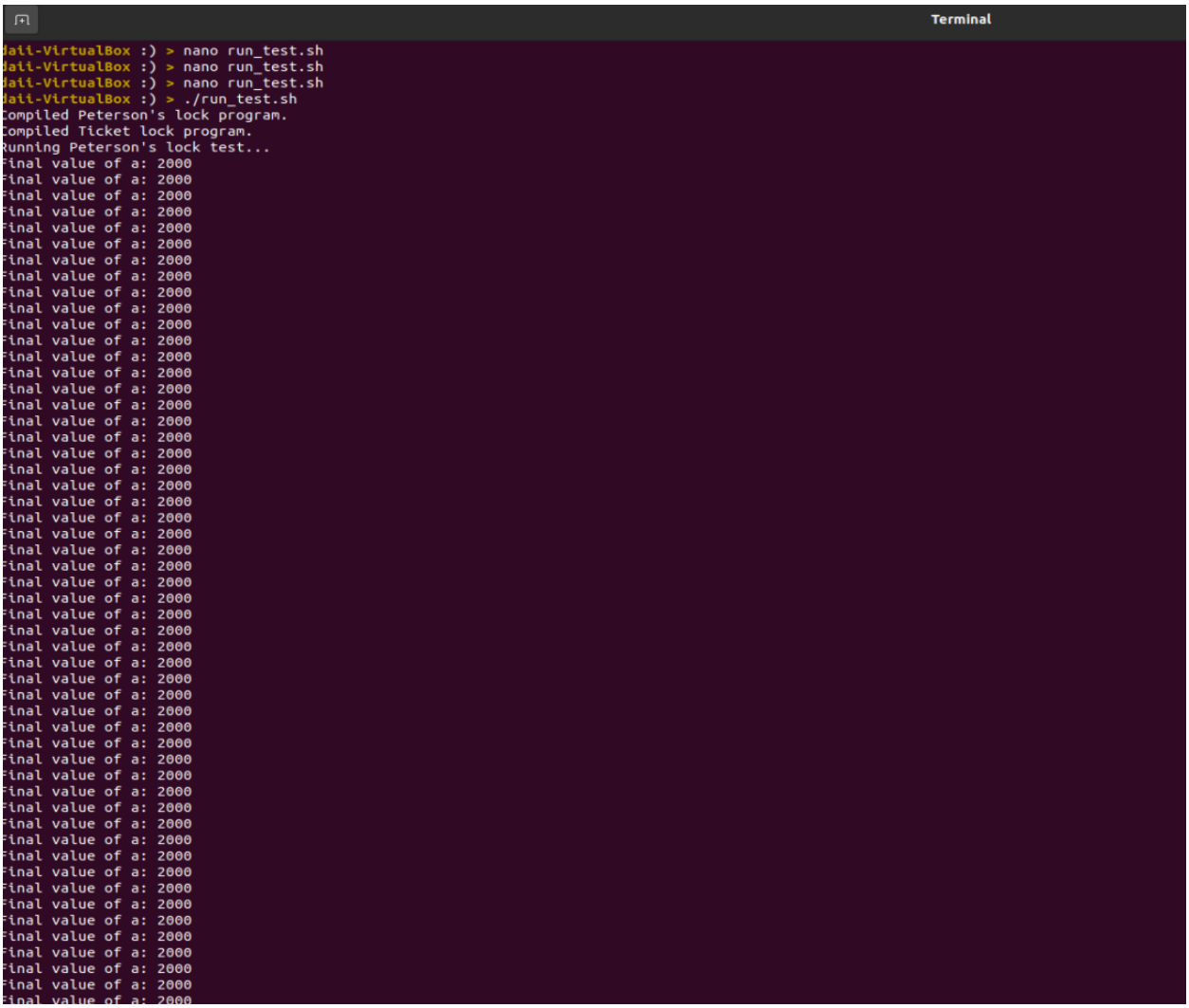
برای اجرا و در تابع main هم ابتدا دو ترد میسازیم، که ۰ و ۱ را به عنوان آرگومنت به threadFunction میدهیم سپس منتظر میمانیم که تمام شوند و در نهایت مقدار نهایی a را چاپ میکنیم.

نحوه ران کردن و شل اسکریپت هم در ادامه میبینیم :

```
 GNU nano 4.8                                                    run_test.sh
#!/bin/bash

g++ -o peterson_lock peterson_lock.cpp -pthread
echo "Compiled Peterson's lock program."

g++ -o ticket_lock ticket_lock.cpp -pthread
echo "Compiled Ticket lock program."

echo "Running Peterson's lock test..."
for i in $(seq 1 100); do
    ./peterson_lock
done

echo "Running Ticket lock test..."
for i in $(seq 1 100); do
    ./ticket_lock
done
```

لازم است هنگامی که قرار است از ترد استفاده کنیم از آرگومنت -pthread هم موقع کامپایل استفاده کنیم.

برای قسمت دوم هم پیش نیاز ها انجام شده است حال دو ترد داریم و قرار است برنامه تستی که نوشتیم را ۱۰۰ بار ران کنیم که خروجی آن به صورت زیر میشود.



با تعداد ران بیشتر مثل ۱۰۰۰ و ۱۰۰۰۰ هم تست کردم ولی جواب همینطور بود صرفا تعداد خروجی بیشتر بود و در واقع اتفاق درست و جواب درست را داشتیم.(ممکن است به خاطر تک هسته ای بودن VM باشد)

در بخش سوم قرار است راجب مموری اردرینک صحبت کنیم، با توجه به سرچ های صورت گرفته نقش اساسی در برنامه سازی موازی دارد که تعدادی ترد به یک دیتای اشتراکی دسترسی دارند و میخواهند آن را تغییر دهند و ضروری است که این کار با ترتیب خواسته شده صورت گیرد پیچیدگی و سختی کار هنگام هندل کردن همزمان آپتیمایزشن های مروبط به کامپایلر، آپتیمایزشن های مربوط به اجرای CPU مثل reordering و میزان visibility آن برای دیگر تردهاست.

در C++ مموری اردرینگ به کمک کتابخانه std::atomic هندل میشود که تعدادی ترتیب برای ما فراهم میکند، این حالت ها به کامپایلر و CPU میگویند که چگونه reordering مربوط به عملیات های read/write صورت بگیرد و حالت های اصلی آن در C++ را میتوان به صورت ۵ مدل زیر تعریف کرد:

۱.memory_order_relaxed: تنها atomicity را گارانتی میکند و به ترتیب خیلی کاری ندارد.

۲.memory_order_acquire: معمولا هنگامی استفاده میشود که یک متغیری را میخوانیم تا لاک را آزاد کنیم (در واقع از عملیات هایی که بعد یک برنامه رخ میدهد جلوگیری میکند و نمیگذارد reorder شوند به نحوی که پیش از آن انجام شوند)

۳.memory_order_release: دقیقا برعکس حالت قبل است(برای نوشتن در متغیر در جهت آزاد شدن قفل)

۴.memory_order_acq_rel: که ترتیب دوتای قبلی است برای عملیات های read-modify-write مورد استفاده قرار میگیرد.

۵.memory_order_seq_cst: سخت گیرانه ترین بین موارد بالاست و تضمین میکند که تمامی عملیات ها در یک ترتیب فیکس ران میشوند که دیفالت هم همین است.

در بخش چهارم هم به سراغ ticket_lock و با توجه به قطعه کد راهنمایی آن را به صورت زیر پیاده سازی میکنیم. :

```
  GNU nano 4.8                                                    ticket_lock.cpp
#include <iostream>
#include <thread>
#include <atomic>

std::atomic_int next_ticket(0);
std::atomic_int now_serving(0);
int b = 0;

void ticketLock_acquire() {
    int my_ticket = next_ticket.fetch_add(1, std::memory_order_seq_cst);
    while (now_serving.load(std::memory_order_seq_cst) != my_ticket) {
    }
}

void ticketLock_release() {
    now_serving.fetch_add(1, std::memory_order_seq_cst);
}

void ticketThreadFunction() {
    for (int i = 0; i < 1000; ++i) {
        ticketLock_acquire();
        ++b;
        ticketLock_release();
    }
}

int main() {
    std::thread t1(ticketThreadFunction);
    std::thread t2(ticketThreadFunction);

    t1.join();
    t2.join();

    std::cout << "Final value of b: " << b << std::endl;
    return 0;
}
```

اگر بخواهیم آن را به صورت خلاصه توضیح بدهیم ابتدا متغیرهای جهانی را تعریف میکنیم که اولی شماره تیکت بعدی را نگه میدارد، دومی تیکت نامبری که در حال سرویس دهی است و سومی هم متغیری است که قرار است یکی یکی زیاد شود(توسط تردها)

در ticketLock_acquire ابتدا نکست را یکی زیاد میکنیم و همزمان به کمک memory_order_seq_cst از ترتیب اجرای درست اطمینان پیدا میکنیم و در نهایت وارد بیزی ویت میشویم.

مانند قسمت قبل است با این تفاوت که now_serving را داریم یکی زیاد میکنیم و بعدی وارد منطقه بحرانی میشود دوباره برای ترتیب اجرای استریکت و درست از memory_order_seq_cst استفاده میکنیم.

در بخش اجرا و خروجی هم از شل اسکریپی که نوشته بودیم استفاده میکنیم و خروجی را در ادامه میبینیم:

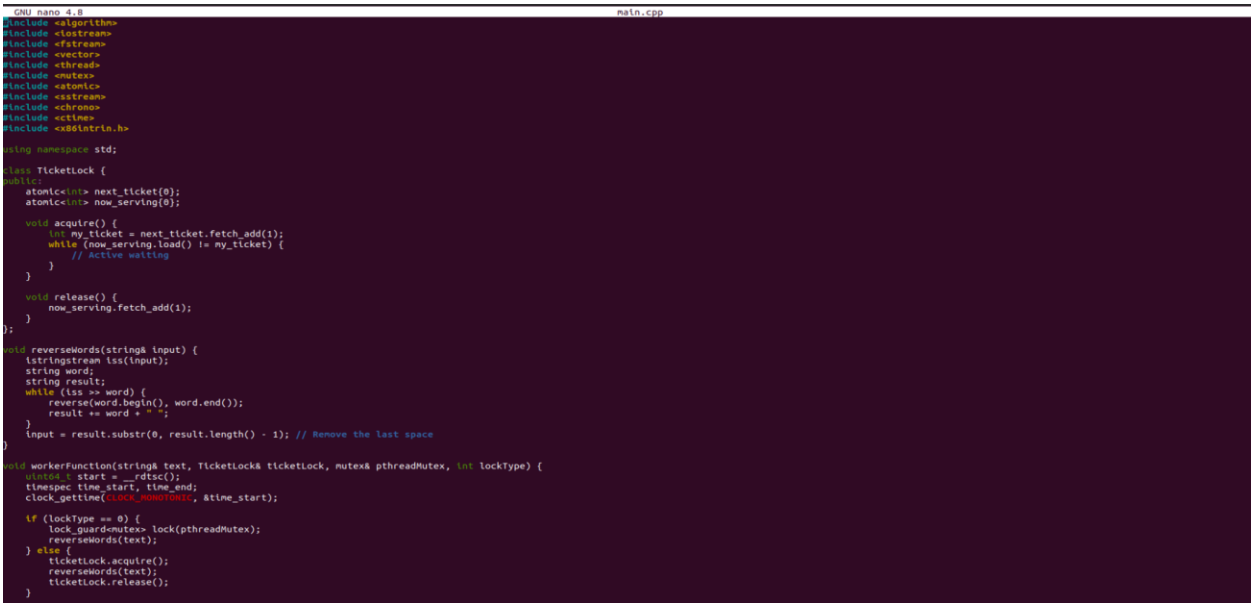
```
  GNU nano 4.8                                                                          run_test.sh
#!/bin/bash

g++ -o peterson_lock peterson_lock.cpp -pthread
echo "Compiled Peterson's lock program."

g++ -o ticket_lock ticket_lock.cpp -pthread
echo "Compiled Ticket lock program."

echo "Running Peterson's lock test..."
for i in $(seq 1 100); do
    ./peterson_lock
done

echo "Running Ticket lock test..."
for i in $(seq 1 100); do
    ./ticket_lock
done
```

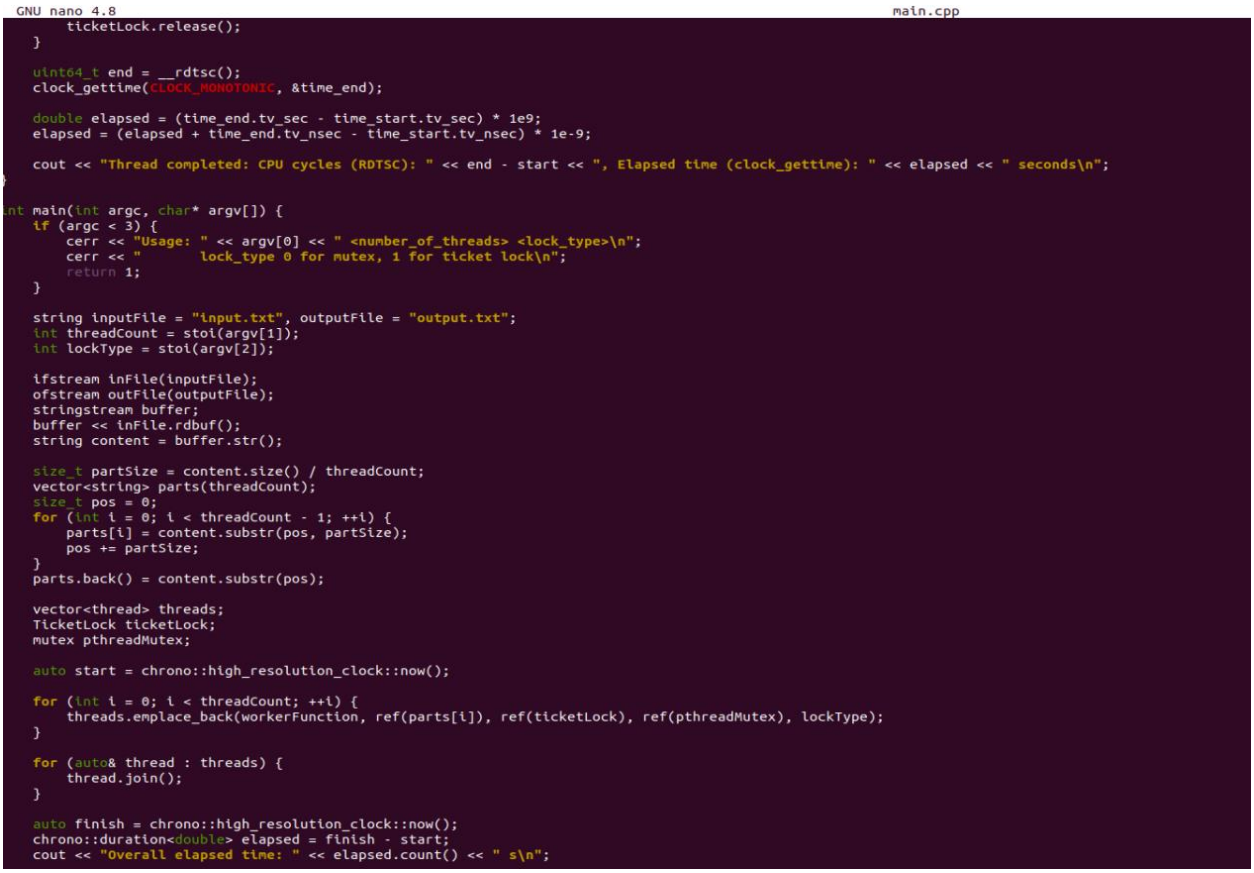
```
Terminal
Final value of a: 2000
Final value of a: 2000
Final value of a: 2000
Final value of a: 2000
Final value of a: 2000
Running Ticket lock test...
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
Final value of b: 2000
```

دلیل اجرای درست هم در توضیحات گفته شد که memory_order_seq_cst میباشد.

سوال ۲:

در بخش اول به سراغ پیاده سازی چند نخی و همچنین الگوریتم ریورس کردن میرویم ابتدا کد را مشاهده میکنیم و سپس وارد جزییات هر بخش میشویم.





```
GNU nano 4.8                                                                main.cpp
        ticketLock.release();
    }

    uint64_t end = __rdtsc();
    clock_gettime(CLOCK_MONOTONIC, &time_end);

    double elapsed = (time_end.tv_sec - time_start.tv_sec) * 1e9;
    elapsed = (elapsed + time_end.tv_nsec - time_start.tv_nsec) * 1e-9;

    cout << "Thread completed: CPU cycles (RDTSC): " << end - start << ", Elapsed time (clock_gettime): " << elapsed << " seconds\n";
}

int main(int argc, char* argv[]) {
    if (argc < 3) {
        cerr << "Usage: " << argv[0] << " <number_of_threads> <lock_type>\n";
        cerr << "       lock_type 0 for mutex, 1 for ticket lock\n";
        return 1;
    }

    string inputFile = "input.txt", outputFile = "output.txt";
    int threadCount = stoi(argv[1]);
    int lockType = stoi(argv[2]);

    ifstream inFile(inputFile);
    ofstream outFile(outputFile);
    stringstream buffer;
    buffer << inFile.rdbuf();
    string content = buffer.str();

    size_t partSize = content.size() / threadCount;
    vector<string> parts(threadCount);
    size_t pos = 0;
    for (int i = 0; i < threadCount - 1; ++i) {
        parts[i] = content.substr(pos, partSize);
        pos += partSize;
    }
    parts.back() = content.substr(pos);

    vector<thread> threads;
    TicketLock ticketLock;
    mutex pthreadMutex;

    auto start = chrono::high_resolution_clock::now();

    for (int i = 0; i < threadCount; ++i) {
        threads.emplace_back(workerFunction, ref(parts[i]), ref(ticketLock), ref(pthreadMutex), lockType);
    }

    for (auto& thread : threads) {
        thread.join();
    }

    auto finish = chrono::high_resolution_clock::now();
    chrono::duration<double> elapsed = finish - start;
    cout << "Overall elapsed time: " << elapsed.count() << " s\n";
```

پس از افزودن لایبرری های مورد نیاز که نسبت به قبل بیشتر است مانند موارد مورد نیاز برای شمارش تعداد سایکل و ... به سراغ تعریف ticket lock که دقیقا مثل سوال قبل است میرویم، برای راحتی و خوانایی راحتتر و اصولی تر آن را در قالب یک کلاس تعریف میکنیم، در تابع بعدی که reverseWords است بعد از خواندن کلمات به کمک لایبرری الگوریتم آنها را معکوس میکنیم، تابع بعدی worker function ماست که چند وظیفه دارد ۱.تایم شروع را طبق سایکل و زمان محاسبه میکند ۲.وارد فاز بحرانی میشود طبق یکی از دو الگوریتم ۳.فراخوانی تابع معکوس کننده ۴.زمان و سایکل نهایی را محاسبه میکند و خروجی میدهد.

در نهایت در تابع main ابتدا دریافت آرگومنت را هندل میکنیم که نوع لاک و تعداد ترد را مشخص میکند، سپس دیتا را میخوانیم و پارتیشن میکنیم برای هر ترد در نهایت تایم کلی را محاسبه میکنیم و خروجی را در فایل جدید مینویسیم.

در نهایت خروجی را برای ۵ و ۱۰ و ۲۰ ترد میبینیم :

```
dail-VirtualBox :) > ./main 5 0
Thread completed: CPU cycles (RDTSC): 96696, Elapsed time (clock_gettime): 3.7282e-05 seconds
Thread completed: CPU cycles (RDTSC): 60826, Elapsed time (clock_gettime): 2.339e-05 seconds
Thread completed: CPU cycles (RDTSC): 57748, Elapsed time (clock_gettime): 2.2254e-05 seconds
Thread completed: CPU cycles (RDTSC): 58034, Elapsed time (clock_gettime): 2.2362e-05 seconds
Thread completed: CPU cycles (RDTSC): 54882, Elapsed time (clock_gettime): 2.1134e-05 seconds
Overall elapsed time: 0.000490041 s
dail-VirtualBox :) > ./main 10 0
Thread completed: CPU cycles (RDTSC): 66184, Elapsed time (clock_gettime): 2.5511e-05 seconds
Thread completed: CPU cycles (RDTSC): 55316, Elapsed time (clock_gettime): 2.132e-05 seconds
Thread completed: CPU cycles (RDTSC): 48374, Elapsed time (clock_gettime): 1.8669e-05 seconds
Thread completed: CPU cycles (RDTSC): 43760, Elapsed time (clock_gettime): 1.689e-05 seconds
Thread completed: CPU cycles (RDTSC): 32166, Elapsed time (clock_gettime): 1.2373e-05 seconds
Thread completed: CPU cycles (RDTSC): 29382, Elapsed time (clock_gettime): 1.1299e-05 seconds
Thread completed: CPU cycles (RDTSC): 31134, Elapsed time (clock_gettime): 1.1978e-05 seconds
Thread completed: CPU cycles (RDTSC): 32110, Elapsed time (clock_gettime): 1.2354e-05 seconds
Thread completed: CPU cycles (RDTSC): 31374, Elapsed time (clock_gettime): 1.2072e-05 seconds
Thread completed: CPU cycles (RDTSC): 29758, Elapsed time (clock_gettime): 1.1454e-05 seconds
Overall elapsed time: 0.000643679 s
dail-VirtualBox :) > ./main 20 0
Thread completed: CPU cycles (RDTSC): 50436, Elapsed time (clock_gettime): 1.9426e-05 seconds
Thread completed: CPU cycles (RDTSC): 38802, Elapsed time (clock_gettime): 1.4949e-05 seconds
Thread completed: CPU cycles (RDTSC): 20110, Elapsed time (clock_gettime): 7.729e-06 seconds
Thread completed: CPU cycles (RDTSC): 27172, Elapsed time (clock_gettime): 1.0482e-05 seconds
Thread completed: CPU cycles (RDTSC): 30136, Elapsed time (clock_gettime): 1.1628e-05 seconds
Thread completed: CPU cycles (RDTSC): 18994, Elapsed time (clock_gettime): 7.294e-06 seconds
Thread completed: CPU cycles (RDTSC): 18234, Elapsed time (clock_gettime): 6.988e-06 seconds
Thread completed: CPU cycles (RDTSC): 17526, Elapsed time (clock_gettime): 6.735e-06 seconds
Thread completed: CPU cycles (RDTSC): 18034, Elapsed time (clock_gettime): 6.933e-06 seconds
Thread completed: CPU cycles (RDTSC): 17954, Elapsed time (clock_gettime): 6.914e-06 seconds
Thread completed: CPU cycles (RDTSC): 21780, Elapsed time (clock_gettime): 8.371e-06 seconds
Thread completed: CPU cycles (RDTSC): 18284, Elapsed time (clock_gettime): 7.012e-06 seconds
Thread completed: CPU cycles (RDTSC): 18410, Elapsed time (clock_gettime): 7.074e-06 seconds
Thread completed: CPU cycles (RDTSC): 20992, Elapsed time (clock_gettime): 8.086e-06 seconds
Thread completed: CPU cycles (RDTSC): 19680, Elapsed time (clock_gettime): 7.571e-06 seconds
Thread completed: CPU cycles (RDTSC): 18700, Elapsed time (clock_gettime): 7.186e-06 seconds
Thread completed: CPU cycles (RDTSC): 18696, Elapsed time (clock_gettime): 7.189e-06 seconds
Thread completed: CPU cycles (RDTSC): 18368, Elapsed time (clock_gettime): 7.062e-06 seconds
Thread completed: CPU cycles (RDTSC): 17004, Elapsed time (clock_gettime): 6.53e-06 seconds
Thread completed: CPU cycles (RDTSC): 17418, Elapsed time (clock_gettime): 6.688e-06 seconds
Overall elapsed time: 0.000930286 s
dail-VirtualBox :) >
```

در پیاده سازی ما بهترین حالت چهار ترد بود که خروجی را مشاهده میکنیم :

```
dail-VirtualBox :) > ./main 3 0
Thread completed: CPU cycles (RDTSC): 122854, Elapsed time (clock_gettime): 4.7371e-05 seconds
Thread completed: CPU cycles (RDTSC): 87204, Elapsed time (clock_gettime): 3.3567e-05 seconds
Thread completed: CPU cycles (RDTSC): 205722, Elapsed time (clock_gettime): 7.9362e-05 seconds
Overall elapsed time: 0.000429144 s
dail-VirtualBox :) > ./main 5 0
Thread completed: CPU cycles (RDTSC): 89718, Elapsed time (clock_gettime): 3.4583e-05 seconds
Thread completed: CPU cycles (RDTSC): 78724, Elapsed time (clock_gettime): 3.0358e-05 seconds
Thread completed: CPU cycles (RDTSC): 69292, Elapsed time (clock_gettime): 2.6739e-05 seconds
Thread completed: CPU cycles (RDTSC): 60688, Elapsed time (clock_gettime): 2.3304e-05 seconds
Thread completed: CPU cycles (RDTSC): 55130, Elapsed time (clock_gettime): 2.1231e-05 seconds
Overall elapsed time: 0.000484742 s
```

```
dail-VirtualBox :) > ./main 4 0
Thread completed: CPU cycles (RDTSC): 100326, Elapsed time (clock_gettime): 3.8682e-05 seconds
Thread completed: CPU cycles (RDTSC): 70756, Elapsed time (clock_gettime): 2.7252e-05 seconds
Thread completed: CPU cycles (RDTSC): 65032, Elapsed time (clock_gettime): 2.5054e-05 seconds
Thread completed: CPU cycles (RDTSC): 65756, Elapsed time (clock_gettime): 2.5336e-05 seconds
Overall elapsed time: 0.000377668 s
```

سوال ۳:

در گام اول ورژن را چک میکنیم و ابزار مربوطه را نصب کنیم که در ادامه مشاهده میکنیم :

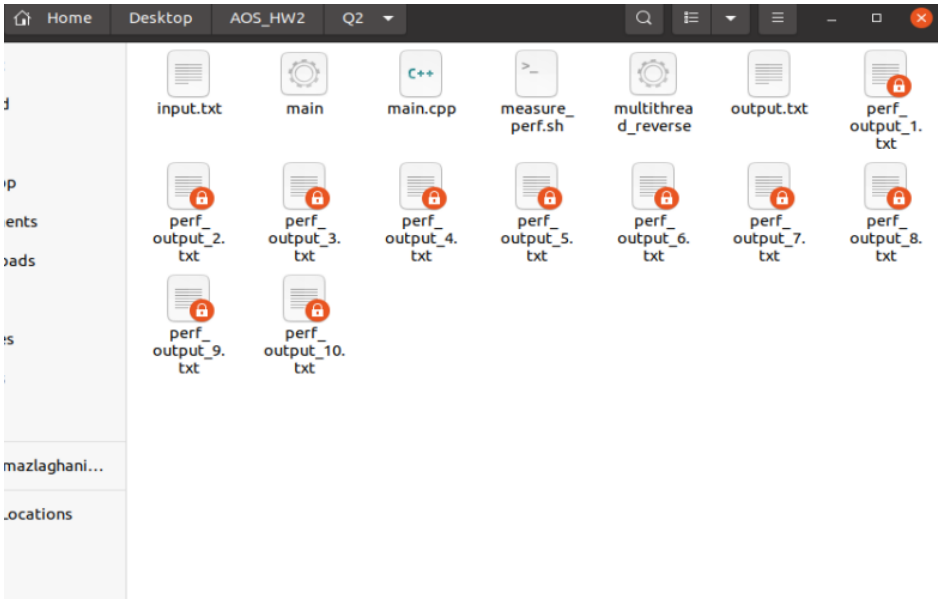
```
dail-VirtualBox :) > uname -a
Linux dail-VirtualBox 5.15.0-105-generic #115~20.04.1-Ubuntu SMP Mon Apr 15 17:33:04 UTC 2024 x86_64 x86_
dail-VirtualBox :) > sudo apt-get install linux-tools-generic
[sudo] password for dail:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  gir1.2-goa-1.0
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  linux-tools-5.4.0-177 linux-tools-5.4.0-177-generic linux-tools-common
The following NEW packages will be installed:
  linux-tools-5.4.0-177 linux-tools-5.4.0-177-generic linux-tools-common linux-tools-generic
0 upgraded, 4 newly installed, 0 to remove and 15 not upgraded.
Need to get 5,839 kB of archives.
After this operation, 27.4 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 linux-tools-common all 5.4.0-177.197
Get:2 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 linux-tools-5.4.0-177 amd64 5.4.0-177
Get:3 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 linux-tools-5.4.0-177-generic amd64 5
Get:4 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 linux-tools-generic amd64 5.4.0.177.1
Fetched 5,839 kB in 7s (879 kB/s)
Selecting previously unselected package linux-tools-common.
(Reading database ... 196043 files and directories currently installed.)
Preparing to unpack .../linux-tools-common_5.4.0-177.197_all.deb ...
Unpacking linux-tools-common (5.4.0-177.197) ...
Selecting previously unselected package linux-tools-5.4.0-177.
Preparing to unpack .../linux-tools-5.4.0-177_5.4.0-177.197_amd64.deb ...
Unpacking linux-tools-5.4.0-177 (5.4.0-177.197) ...
Selecting previously unselected package linux-tools-5.4.0-177-generic.
Preparing to unpack .../linux-tools-5.4.0-177-generic_5.4.0-177.197_amd64.deb ...
Unpacking linux-tools-5.4.0-177-generic (5.4.0-177.197) ...
Selecting previously unselected package linux-tools-generic.
Preparing to unpack .../linux-tools-generic_5.4.0.177.175_amd64.deb ...
Unpacking linux-tools-generic (5.4.0.177.175) ...
Setting up linux-tools-common (5.4.0-177.197) ...
Setting up linux-tools-5.4.0-177 (5.4.0-177.197) ...
Setting up linux-tools-5.4.0-177-generic (5.4.0-177.197) ...
Setting up linux-tools-generic (5.4.0.177.175) ...
Processing triggers for man-db (2.9.1-1) ...
dail-VirtualBox :) >
```

سپس به سراغ اجرای یک بار ساده به کمک perf میرویم :

```
dail-VirtualBox :( > sudo perf stat -e cycles,instructions,cache-references,cache-misses,sched:sched_switch,sched:sched_stat_runtime,branch-misses,context-switches ./main
Usage: ./main <number_of_threads> <lock_type>
      lock_type 0 for mutex, 1 for ticket lock

Performance counter stats for './main':

   <not supported>      cycles
   <not supported>      instructions
   <not supported>      cache-references
   <not supported>      cache-misses
                 7      sched:sched_switch
         1,080,933      sched:sched_stat_runtime
   <not supported>      branch-misses
                 7      context-switches

       0.001320118 seconds time elapsed

       0.001222000 seconds user
       0.000000000 seconds sys
```

حال به سراغ پیاده سازی ۱ تا ۱۰ ترد به صورت اتومات میرویم در همین راستا فایل بش زیر را مینویسیم و خروجی آن طبق تصاویر ذیل میشود جالب اینجاست در این قسمت بهترین عملکرد برای ۷ ترد بود :

```bash
#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <program_name>"
    exit 1
fi

program_name=$1

for i in {1..10}; do
    echo "Running with $i threads"
    perf stat -e cycles,instructions,cache-references,cache-misses,sched:sched_switch,sched:sched_stat_runtime,branch-misses,context-switches -o perf_output_$i.txt ./$program_name $i
done

echo "Performance measurement completed."
```



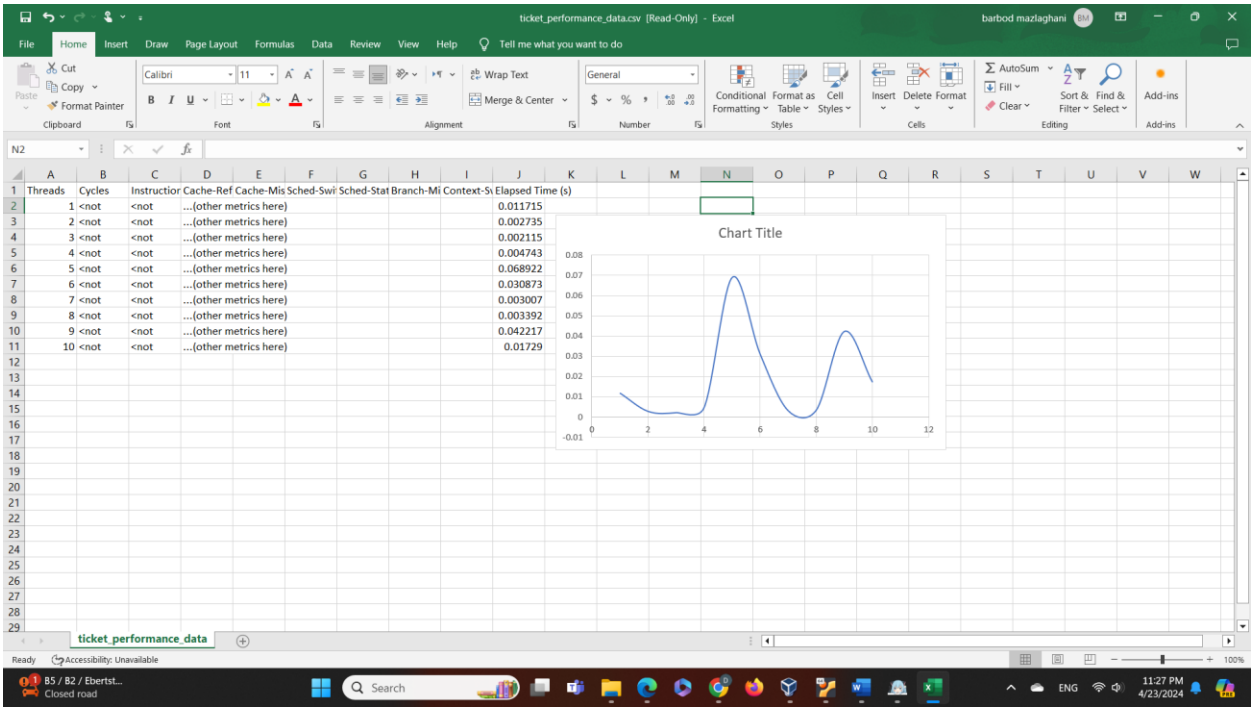

```
daii-VirtualBox :) > sudo ./measure_perf.sh main
Running with 1 threads
Usage: ./main <number_of_threads> <lock_type>
       lock_type 0 for mutex, 1 for ticket lock
Running with 2 threads
Usage: ./main <number_of_threads> <lock_type>
       lock_type 0 for mutex, 1 for ticket lock
Running with 3 threads
Usage: ./main <number_of_threads> <lock_type>
       lock_type 0 for mutex, 1 for ticket lock
Running with 4 threads
Usage: ./main <number_of_threads> <lock_type>
       lock_type 0 for mutex, 1 for ticket lock
Running with 5 threads
Usage: ./main <number_of_threads> <lock_type>
       lock_type 0 for mutex, 1 for ticket lock
Running with 6 threads
Usage: ./main <number_of_threads> <lock_type>
       lock_type 0 for mutex, 1 for ticket lock
Running with 7 threads
Usage: ./main <number_of_threads> <lock_type>
       lock_type 0 for mutex, 1 for ticket lock
Running with 8 threads
Usage: ./main <number_of_threads> <lock_type>
       lock_type 0 for mutex, 1 for ticket lock
Running with 9 threads
Usage: ./main <number_of_threads> <lock_type>
       lock_type 0 for mutex, 1 for ticket lock
Running with 10 threads
Usage: ./main <number_of_threads> <lock_type>
       lock_type 0 for mutex, 1 for ticket lock
Performance measurement completed.
daii-VirtualBox :) >
```

در قسمت بعد به سراغ perf record و report میرویم :

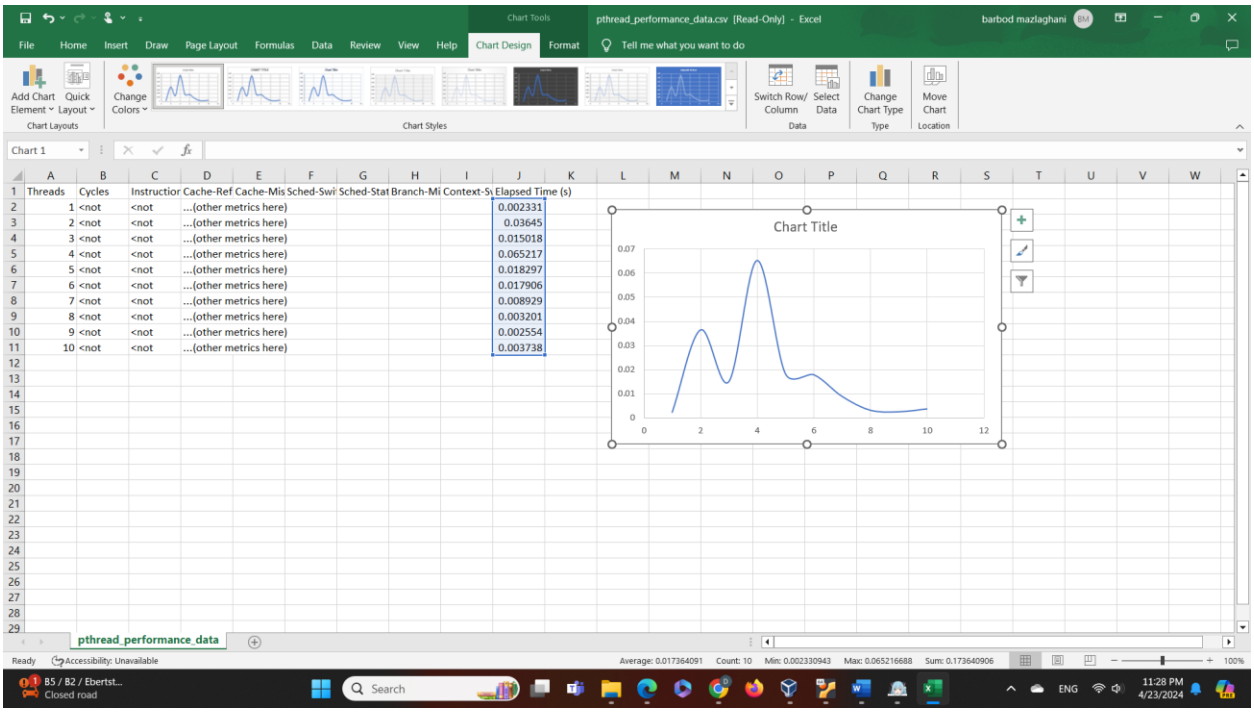
```
dall-VirtualBox :) > sudo perf record --call-graph dwarf -a -g ./main 10 0
Thread completed: CPU cycles (RDTSC): 73836, Elapsed time (clock_gettime): 2.8413e-05 seconds
Thread completed: CPU cycles (RDTSC): 62586, Elapsed time (clock_gettime): 2.4105e-05 seconds
Thread completed: CPU cycles (RDTSC): 33826, Elapsed time (clock_gettime): 1.3027e-05 seconds
Thread completed: CPU cycles (RDTSC): 49790, Elapsed time (clock_gettime): 1.9217e-05 seconds
Thread completed: CPU cycles (RDTSC): 52916, Elapsed time (clock_gettime): 2.0416e-05 seconds
Thread completed: CPU cycles (RDTSC): 30776, Elapsed time (clock_gettime): 1.1828e-05 seconds
Thread completed: CPU cycles (RDTSC): 34130, Elapsed time (clock_gettime): 1.313e-05 seconds
Thread completed: CPU cycles (RDTSC): 34862, Elapsed time (clock_gettime): 1.3409e-05 seconds
Thread completed: CPU cycles (RDTSC): 33096, Elapsed time (clock_gettime): 1.2708e-05 seconds
Thread completed: CPU cycles (RDTSC): 30738, Elapsed time (clock_gettime): 1.1824e-05 seconds
Overall elapsed time: 0.000817047 s
[ perf record: Woken up 0 times to write data ]
[ perf record: Captured and wrote 4.730 MB perf.data (2 samples) ]
```

```
Samples: 2  of event 'cpu-clock:pppH', Event count (approx.): 500000
  Children      Self  Command  Shared Object       Symbol
+   50.00%    50.00%  main     [kernel.kallsyms]   [k] zap_pte_range.isra.0
+   50.00%    50.00%  main     ld-2.31.so          [.] _dl_sysdep_start
+   50.00%     0.00%  main     [kernel.kallsyms]   [k] entry_SYSCALL_64_after_hwframe
+   50.00%     0.00%  main     [kernel.kallsyms]   [k] do_syscall_64
+   50.00%     0.00%  main     [kernel.kallsyms]   [k] __x64_sys_exit
+   50.00%     0.00%  main     [kernel.kallsyms]   [k] do_exit
+   50.00%     0.00%  main     [kernel.kallsyms]   [k] mmput
+   50.00%     0.00%  main     [kernel.kallsyms]   [k] exit_mmap
+   50.00%     0.00%  main     [kernel.kallsyms]   [k] unmap_vmas
+   50.00%     0.00%  main     [kernel.kallsyms]   [k] unmap_single_vma
+   50.00%     0.00%  main     [kernel.kallsyms]   [k] unmap_page_range
+   50.00%     0.00%  main     ld-2.31.so          [.] _start
+   50.00%     0.00%  main     ld-2.31.so          [.] _dl_start
```

دوباره که صورت سوال را خواندم متوجه شدم که به نمودار هم نیاز داریم در همین راستا کد بش قسمت قبل را اندکی تغییر میدهیم و خروجی csv در دو مود لاک را میگیریم و نهایتا به کمک ابزار اکسل نمودار آن را رسم میکنیم و فایل csv آن ها در کنار اسکریپت ها موجود است و در دسترس شما قرار میگیرد.



طبق مشاهده که در بالاتر هم گفتیم ۷ بهترین بود.



در گام آخر به سراغ flameGraph میرویم ،

```
dali-VirtualBox :) > git clone https://github.com/brendangregg/FlameGraph
Cloning into 'FlameGraph'...
remote: Enumerating objects: 1285, done.
remote: Counting objects: 100% (708/708), done.
remote: Compressing objects: 100% (148/148), done.
remote: Total 1285 (delta 584), reused 574 (delta 560), pack-reused 577
Receiving objects: 100% (1285/1285), 1.92 MiB | 770.00 KiB/s, done.
Resolving deltas: 100% (761/761), done.
dali-VirtualBox :) > cd FlameGraph
dali-VirtualBox :) > perf script | ./stackcollapse-perf.pl > out.perf-folded
Failed to open perf.data: No such file or directory  (try 'perf record' first)
dali-VirtualBox :) > cd ..
dali-VirtualBox :) > perf record -g ./your_program
Error:
Access to performance monitoring and observability operations is limited.
Consider adjusting /proc/sys/kernel/perf_event_paranoid setting to open
access to performance monitoring and observability operations for processes
without CAP_PERFMON, CAP_SYS_PTRACE or CAP_SYS_ADMIN Linux capability.
More information can be found at 'Perf events and tool security' document:
https://www.kernel.org/doc/html/latest/admin-guide/perf-security.html
perf_event_paranoid setting is 4:
  -1: Allow use of (almost) all events by all users
      Ignore mlock limit after perf_event_mlock_kb without CAP_IPC_LOCK
>= 0: Disallow raw and ftrace function tracepoint access
>= 1: Disallow CPU event access
>= 2: Disallow kernel profiling
To make the adjusted perf_event_paranoid setting permanent preserve it
in /etc/sysctl.conf (e.g. kernel.perf_event_paranoid = <setting>)
dali-VirtualBox :( > sudo perf record -g ./main 10
Usage: ./main <number_of_threads> <lock_type>
       lock_type 0 for mutex, 1 for ticket lock
[ perf record: Woken up 0 times to write data ]
[ perf record: Captured and wrote 0.013 MB perf.data ]
dali-VirtualBox :( > perf script | ./stackcollapse-perf.pl > out.perf-folded
bash: ./stackcollapse-perf.pl: No such file or directory
dali-VirtualBox :( > ./flamegraph.pl out.perf-folded > perf.svg
bash: ./flamegraph.pl: No such file or directory
dali-VirtualBox :( > cd FlameGraph/
dali-VirtualBox :) > perf script | ./stackcollapse-perf.pl > out.perf-folded
dali-VirtualBox :) > ./flamegraph.pl out.perf-folded > perf.svg
Stack count is low (0). Did something go wrong?
ERROR: No stack counts found
dali-VirtualBox :( > perf record -g ./your_program
Error:
Access to performance monitoring and observability operations is limited.
Consider adjusting /proc/sys/kernel/perf_event_paranoid setting to open
access to performance monitoring and observability operations for processes
without CAP_PERFMON, CAP_SYS_PTRACE or CAP_SYS_ADMIN Linux capability.
More information can be found at 'Perf events and tool security' document:
https://www.kernel.org/doc/html/latest/admin-guide/perf-security.html
perf_event_paranoid setting is 4:
  -1: Allow use of (almost) all events by all users
      Ignore mlock limit after perf_event_mlock_kb without CAP_IPC_LOCK
>= 0: Disallow raw and ftrace function tracepoint access
```

بخش کلون کردن و فراخوانی به درستی انجام شد اما با توجه به خروجی perf script که unknown بود و با توجه به جستجو انگار مشکل در پیاده سازی در شبیه ساز است :

```
Open                                                    perf_script_output.txt
                                                        ~/Desktop/AOS_HW2/Q2/FlameGraph
main  7740  5086.481917:     250000 cpu-clock:pppH:
        ffffffffb48fb2b9 [unknown] ([kernel.kallsyms])
        ffffffffb558b475 [unknown] ([kernel.kallsyms])
        ffffffffb558ba99 [unknown] ([kernel.kallsyms])
        ffffffffb496cf74 [unknown] ([kernel.kallsyms])
        ffffffffb5581d27 [unknown] ([kernel.kallsyms])
        ffffffffb557d529 [unknown] ([kernel.kallsyms])
        ffffffffb56000da [unknown] ([kernel.kallsyms])
            7ffaeaf06297 __GI___libc_write+0x17 (/usr/lib/x86_64-linux-gnu/libc-2.31.so)
        78742e7475706e69 [unknown] ([unknown])
```

خروجی svg خالی میشد :