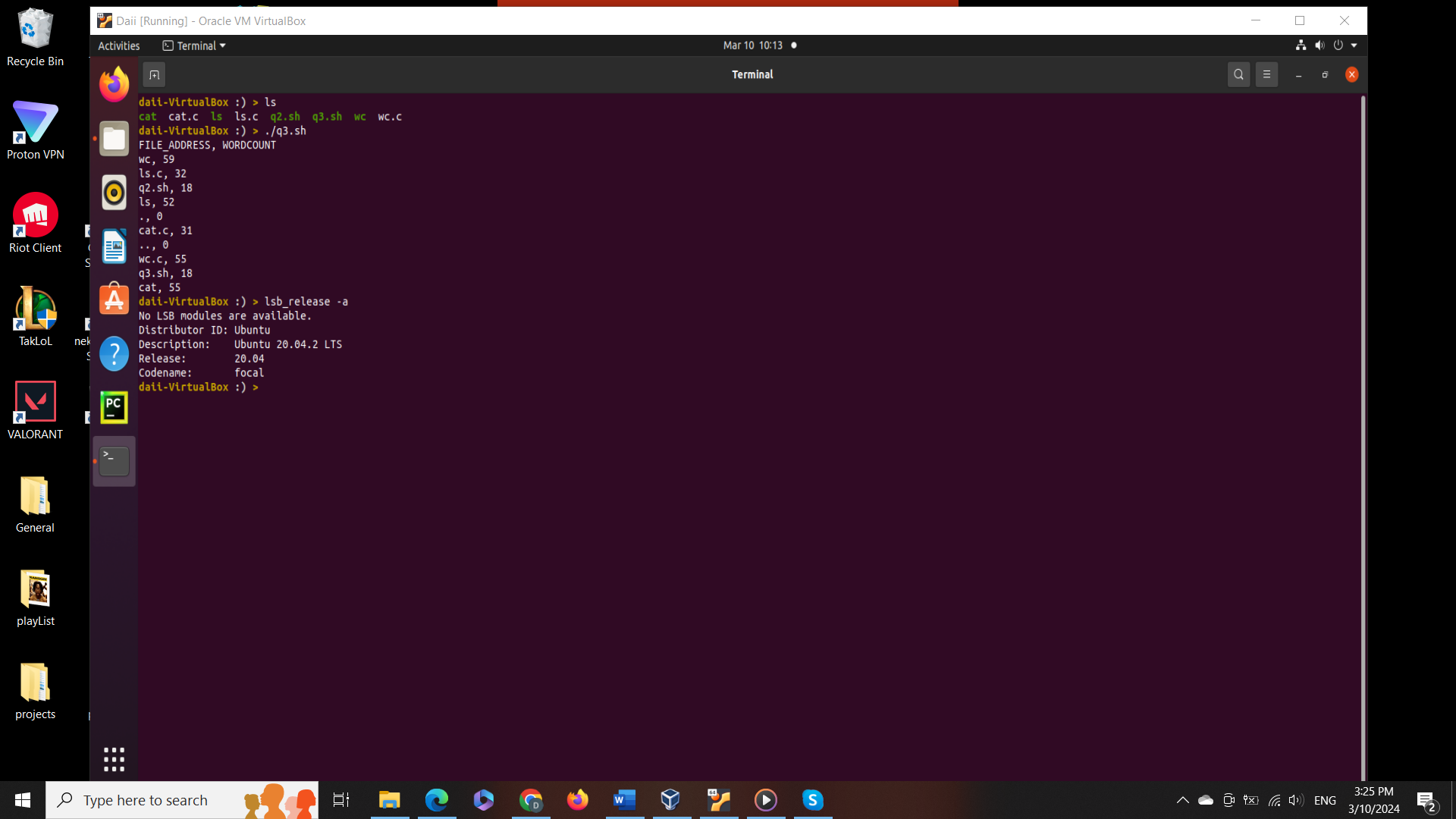
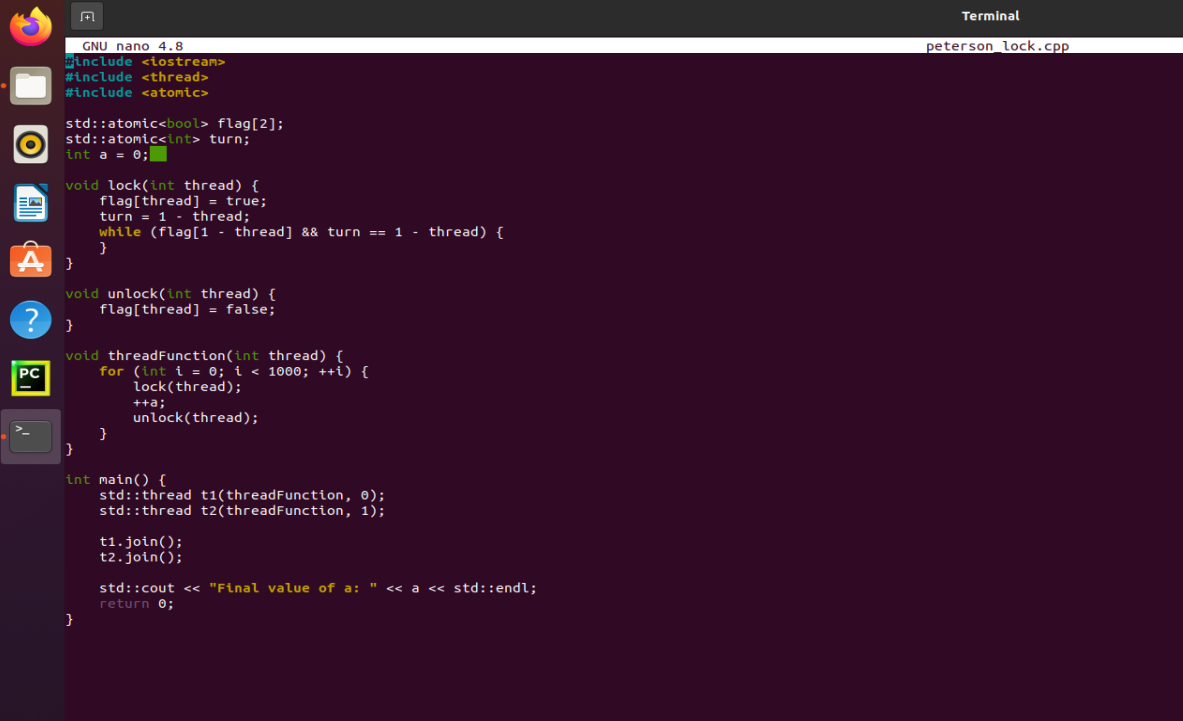
**گزارش تمرین دوم سیستم عامل پیشرفته محمدباربدامیرمزلقانی -810102348**

**نسخه ubuntu:**



**سوال 1:**

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.

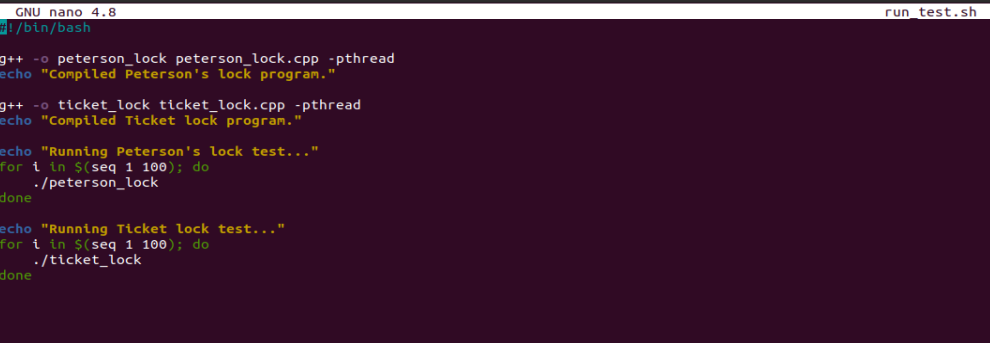
تابع بعدی قفل است که ابتدا فلگی که در بالا تعریف کردیم را برای آن ترد صحیح میکنیم سپس نوبت را به بعدی میدهیم و وارد یک بیزی ویت میشویم که تا زمانی ادامه میابد که یا فلگ ترد دیگر فالس شود و یا نوبت به این ترد برسد.

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 1000 مرتبه ای تعریف کردیم، ابتدا لاک میکند سپس متغیر اشتراکی را یکی زیاد میکند و سپس آنلاک میکند.

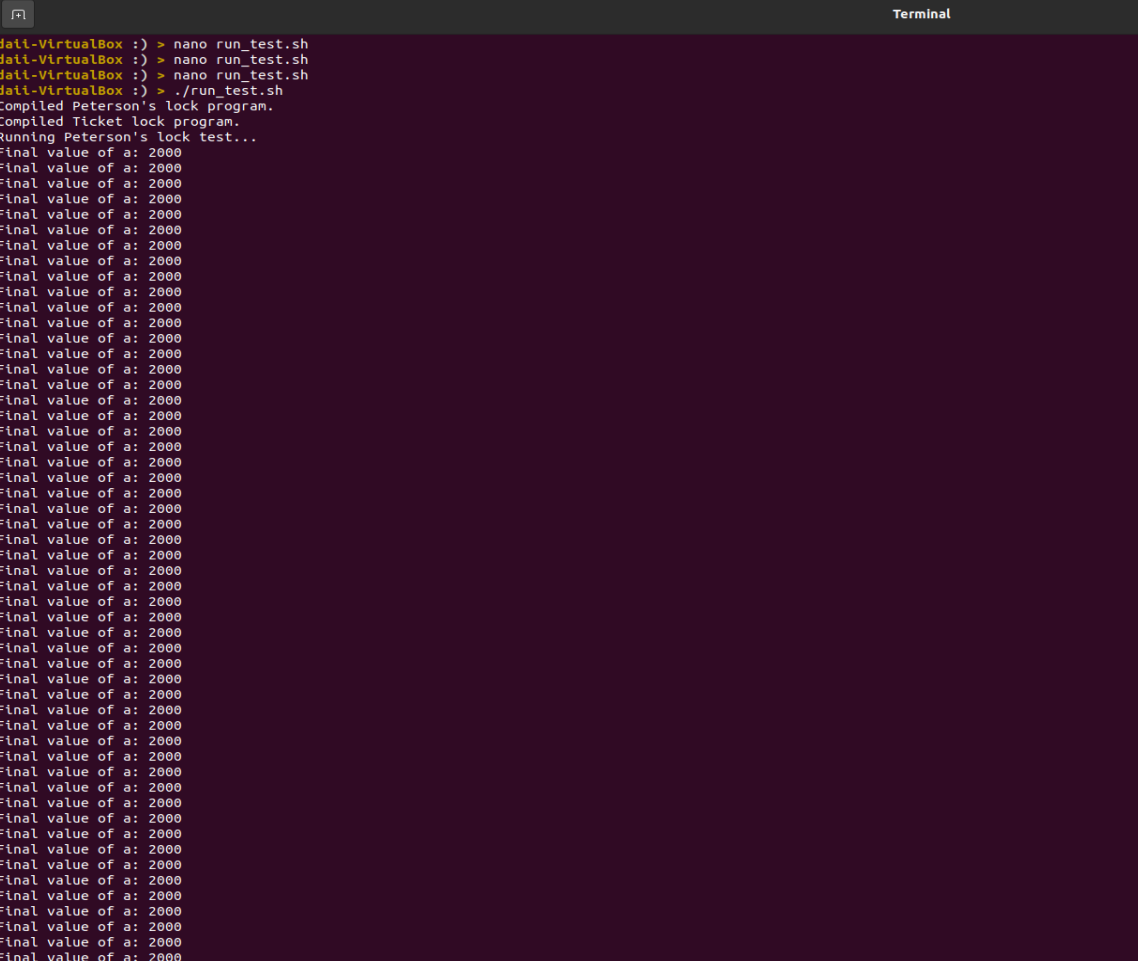
برای اجرا و در تابع main هم ابتدا دو ترد میسازیم، که 0 و 1 را به عنوام آرگومنت به threadFunction میدهیم سپس منتظر میمانیم که تمام شوند و در نهایت مقدار نهایی a را چاپ میکنیم.

نحوه ران کردن و شل اسکریپت هم در ادامه میبینیم :



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 100 بار ران کنیم که خروجی آن به صورت زیر میشود.



با تعداد ران بیشتر مثل 1000 و 10000 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 5 مدل زیر تعریف کرد:

1.memory\_order\_relaxed: تنها atomicity را گارانتی میکند و به ترتیب خیلی کاری ندارد.

2.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)

3.memory\_order\_release: دقیقا برعکس حالت قبل است(برای نوشتن در متغیر در جهت آزاد شدن قفل)

4.memory\_order\_acq\_rel: که ترتیب دوتای قبلی است برای عملیات های read-modify-write مورد استفاده قرار میگیرد.

5.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. :

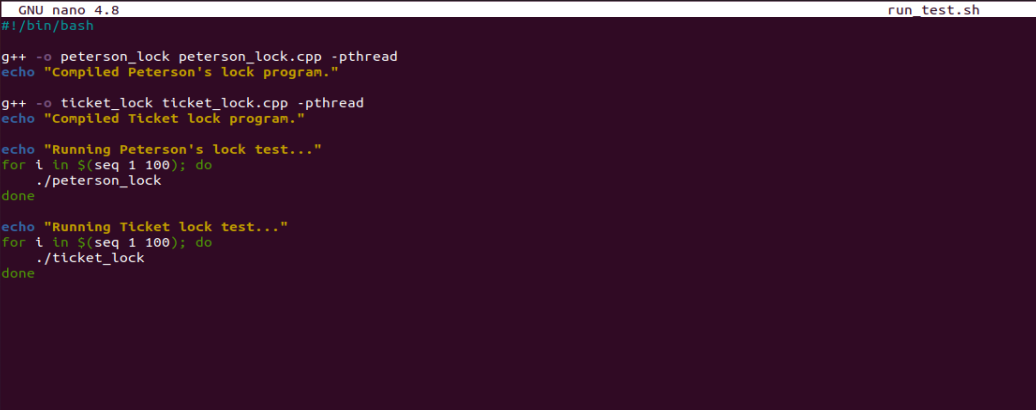


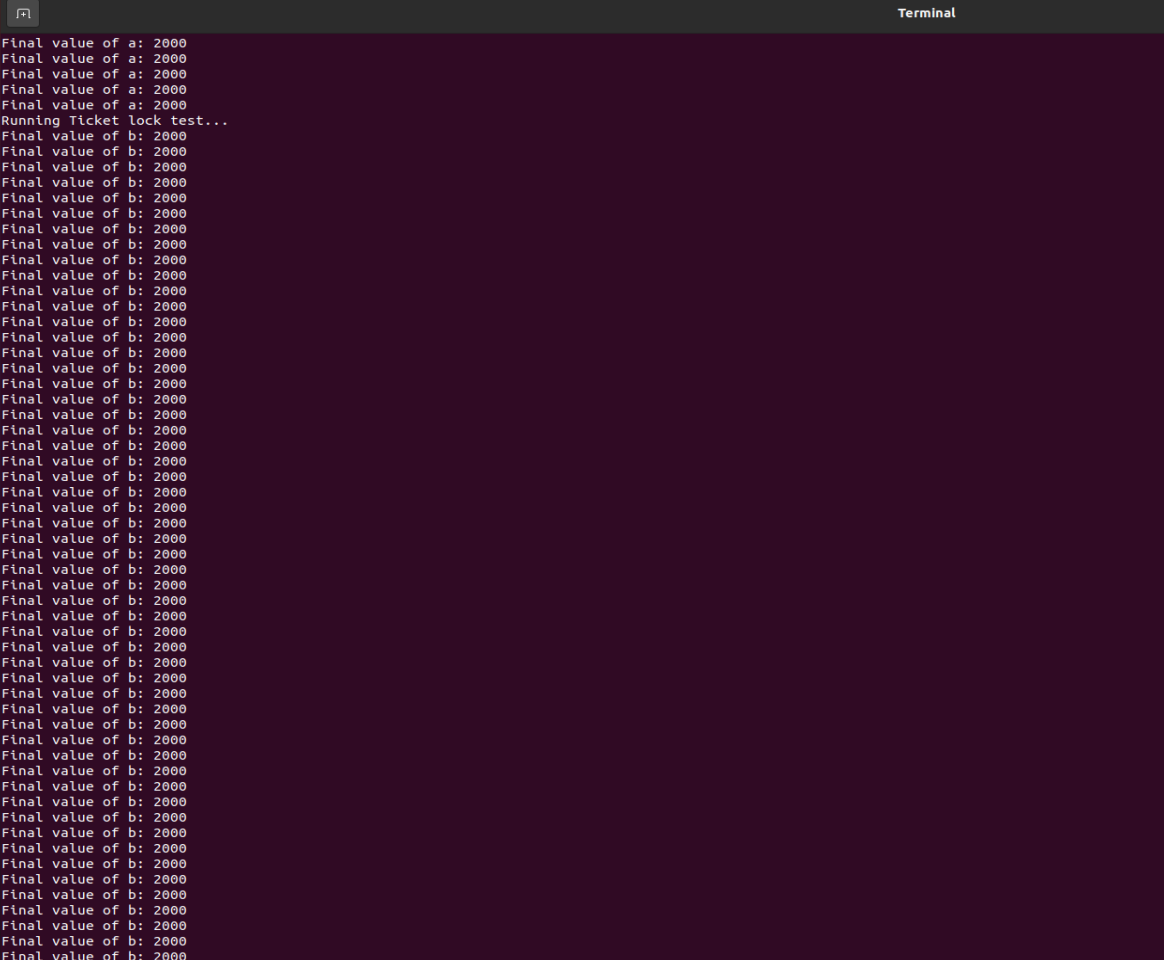
اگر بخواهیم آن را به صورت خلاصه توضیح بدهیم ابتدا متغیرهای جهانی را تعریف میکنیم که اولی شماره تیکت بعدی را نگه میدارد، دومی تیکت نامبری که در حال سرویس دهی است و سومی هم متغیری است که قرار است یکی یکی زیاد شود(توسط تردها)

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:

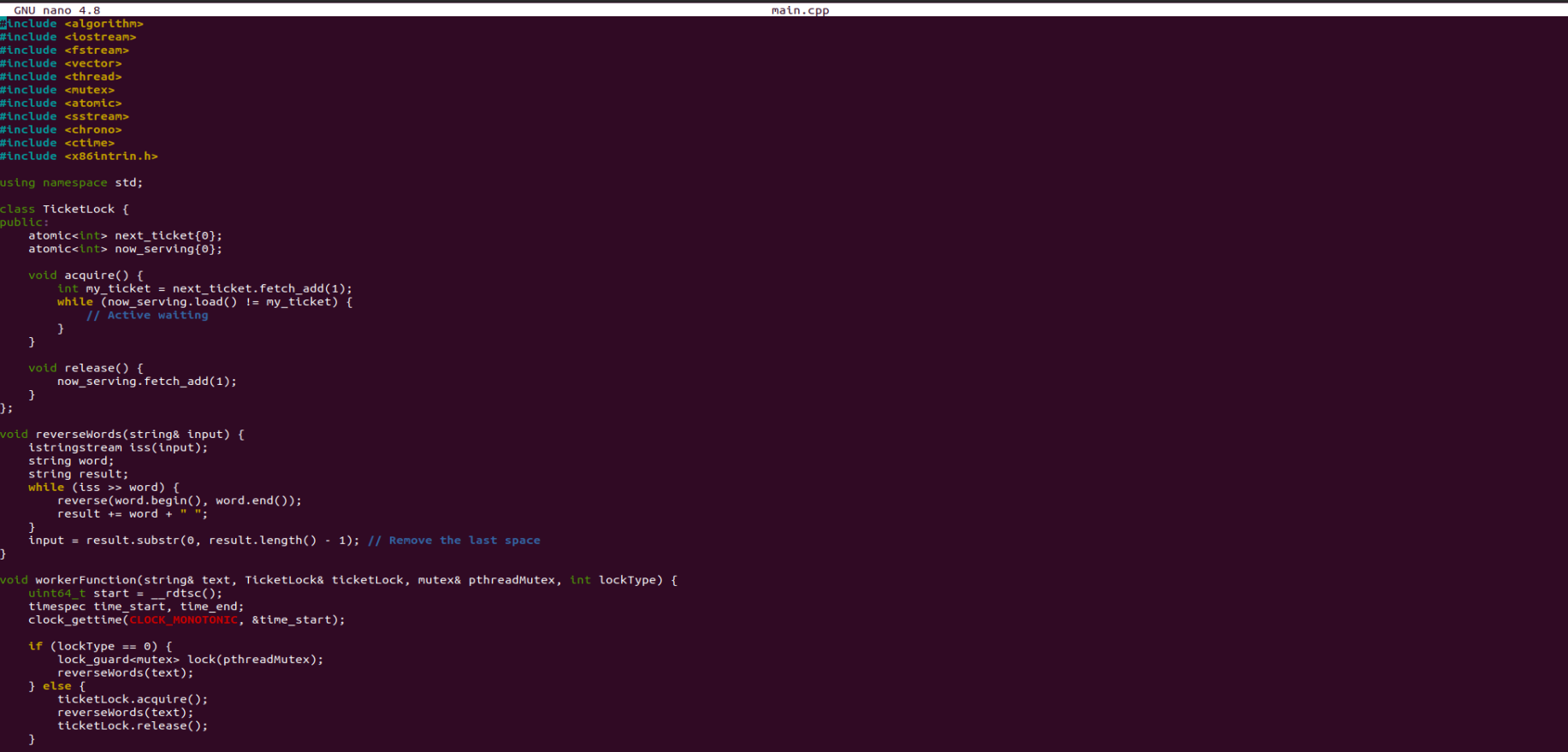




دلیل اجرای درست هم در توضیحات گفته شد که memory\_order\_seq\_cst میباشد.

**سوال 2:**

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.

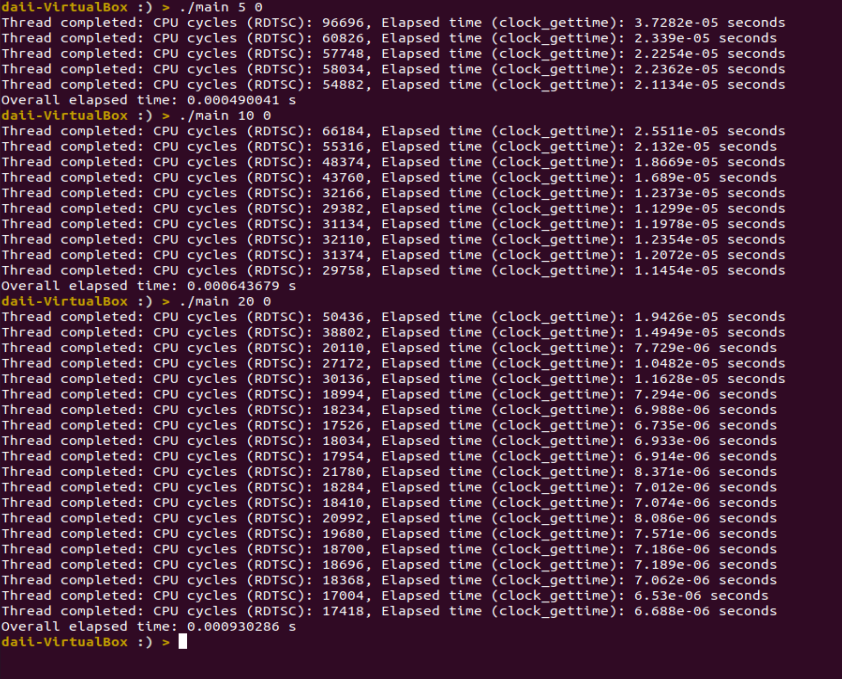




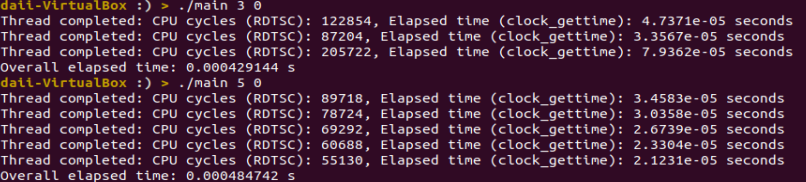
پس از افزودن لایبرری های مورد نیاز که نسبت به قبل بیشتر است مانند موارد مورد نیاز برای شمارش تعداد سایکل و ... به سراغ تعریف ticket lock که دقیقا مثل سوال قبل است میرویم، برای راحتی و خوانایی راحتتر و اصولی تر آن را در قالب یک کلاس تعریف میکنیم، در تابع بعدی که reverseWords است بعد از خواندن کلمات به کمک لایبرری الگوریتم آنها را معکوس میکنیم، تابع بعدی worker function ماست که چند وظیفه دارد 1.تایم شروع را طبق سایکل و زمان محاسبه میکند 2.وارد فاز بحرانی میشود طبق یکی از دو الگوریتم 3.فراخوانی تابع معکوس کننده 4.زمان و سایکل نهایی را محاسبه میکند و خروجی میدهد.

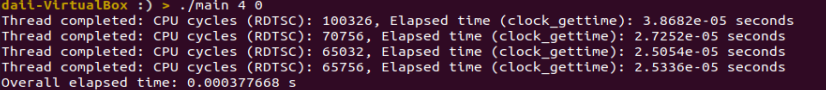
در نهایت در تابع main ابتدا دریافت آرگومنت را هندل میکنیم که نوع لاک و تعداد ترد را مشخص میکند، سپس دیتا را میخوانیم و پارتیشن میکنیم برای هر ترد در نهایت تایم کلی را محاسبه میکنیم و خروجی را در فایل جدید مینویسیم.

در نهایت خروجی را برای 5 و 10 و 20 ترد میبینیم :



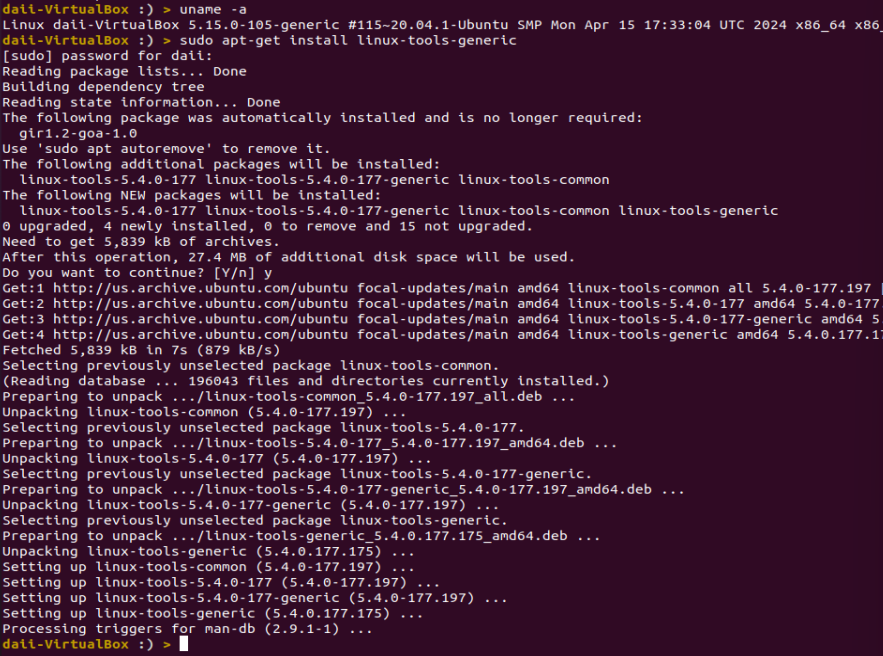
در پیاده سازی ما بهترین حالت چهار ترد بود که خروجی را مشاهده میکنیم :



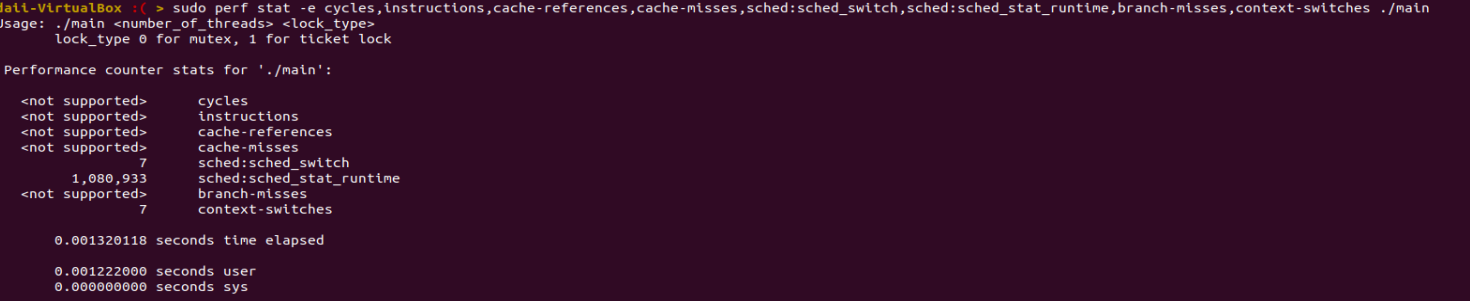


**سوال 3:**

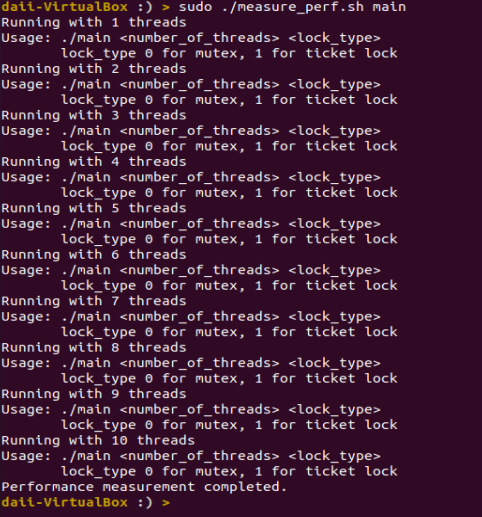
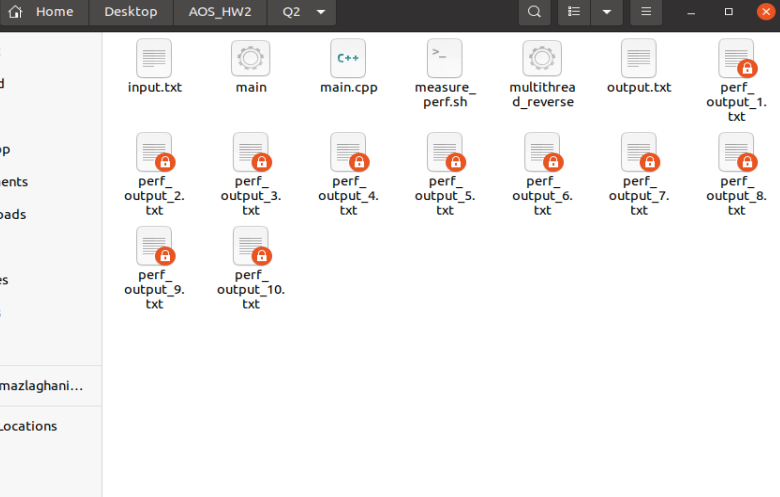
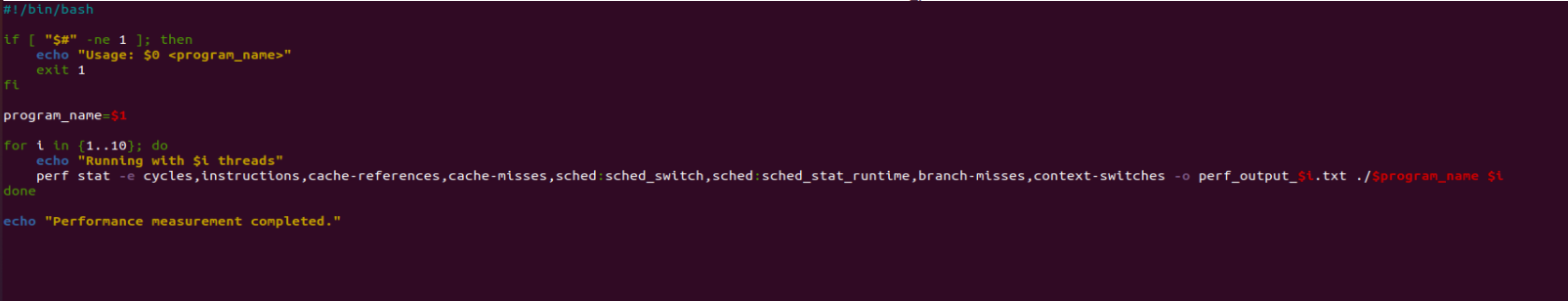
در گام اول ورژن را چک میکنیم و ابزار مربوطه را نصب کنیم که در ادامه مشاهده میکنیم :



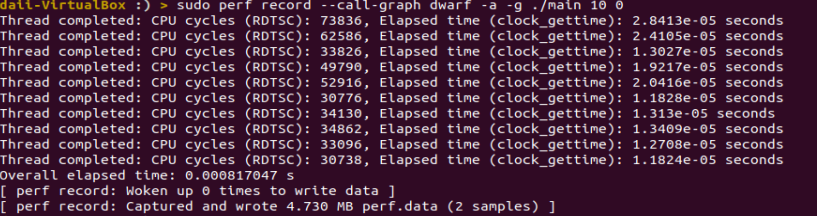
سپس به سراغ اجرای یک بار ساده به کمک perf میرویم :

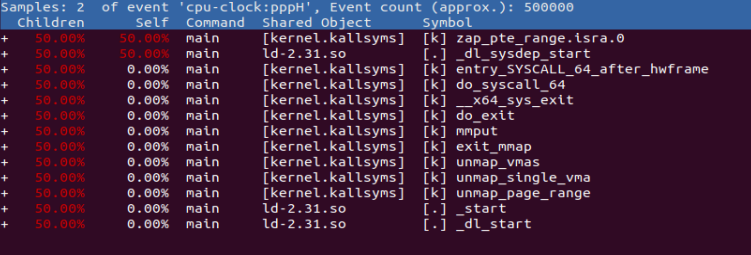


حال به سراغ پیاده سازی 1 تا 10 ترد به صورت اتومات میرویم در همین راستا فایل بش زیر را مینویسیم و خروجی آن طبق تصاویر ذیل میشود جالب اینجاست در این قسمت بهترین عملکرد برای 7 ترد بود :

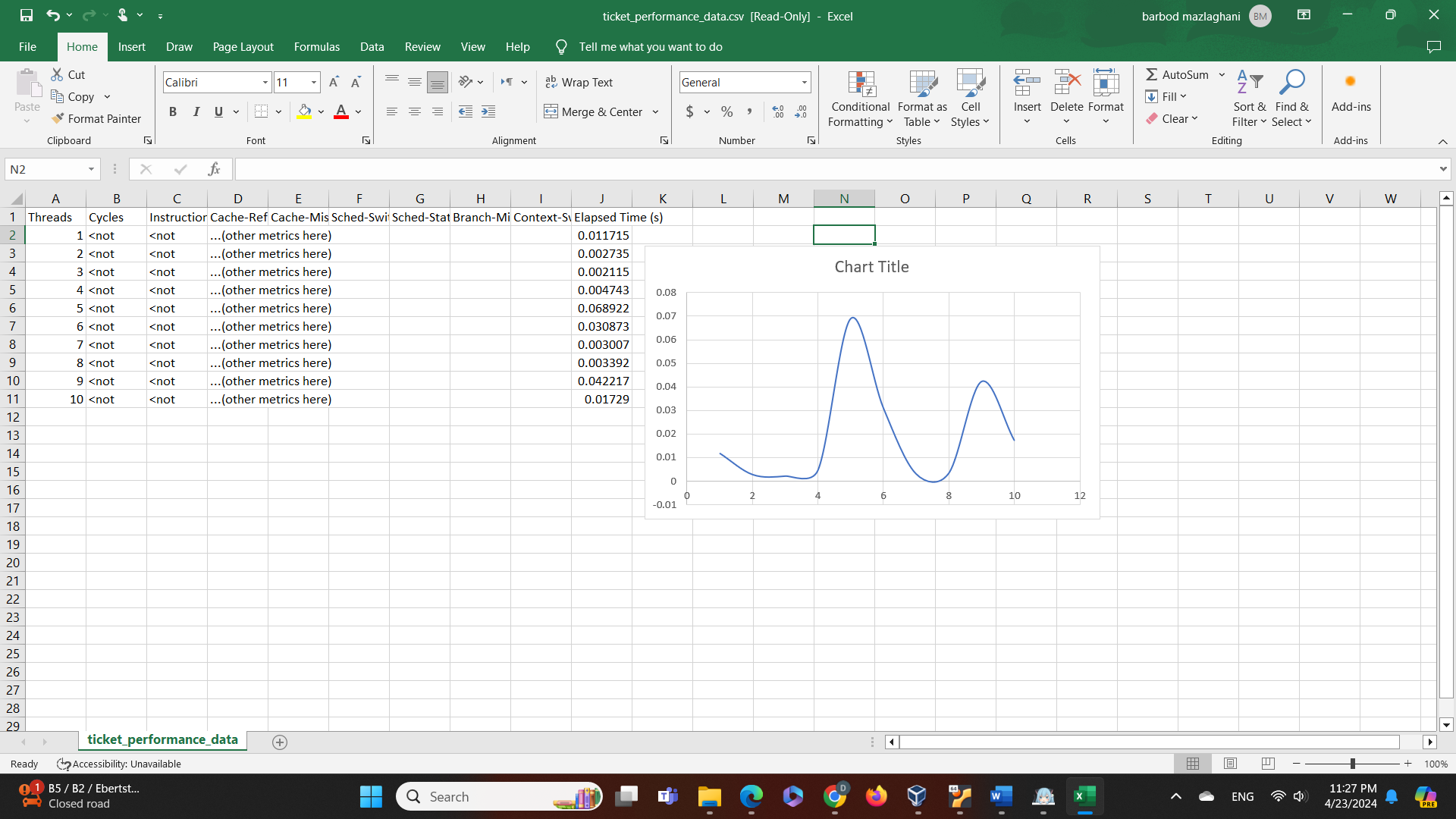


در قسمت بعد به سراغ perf record و report میرویم :

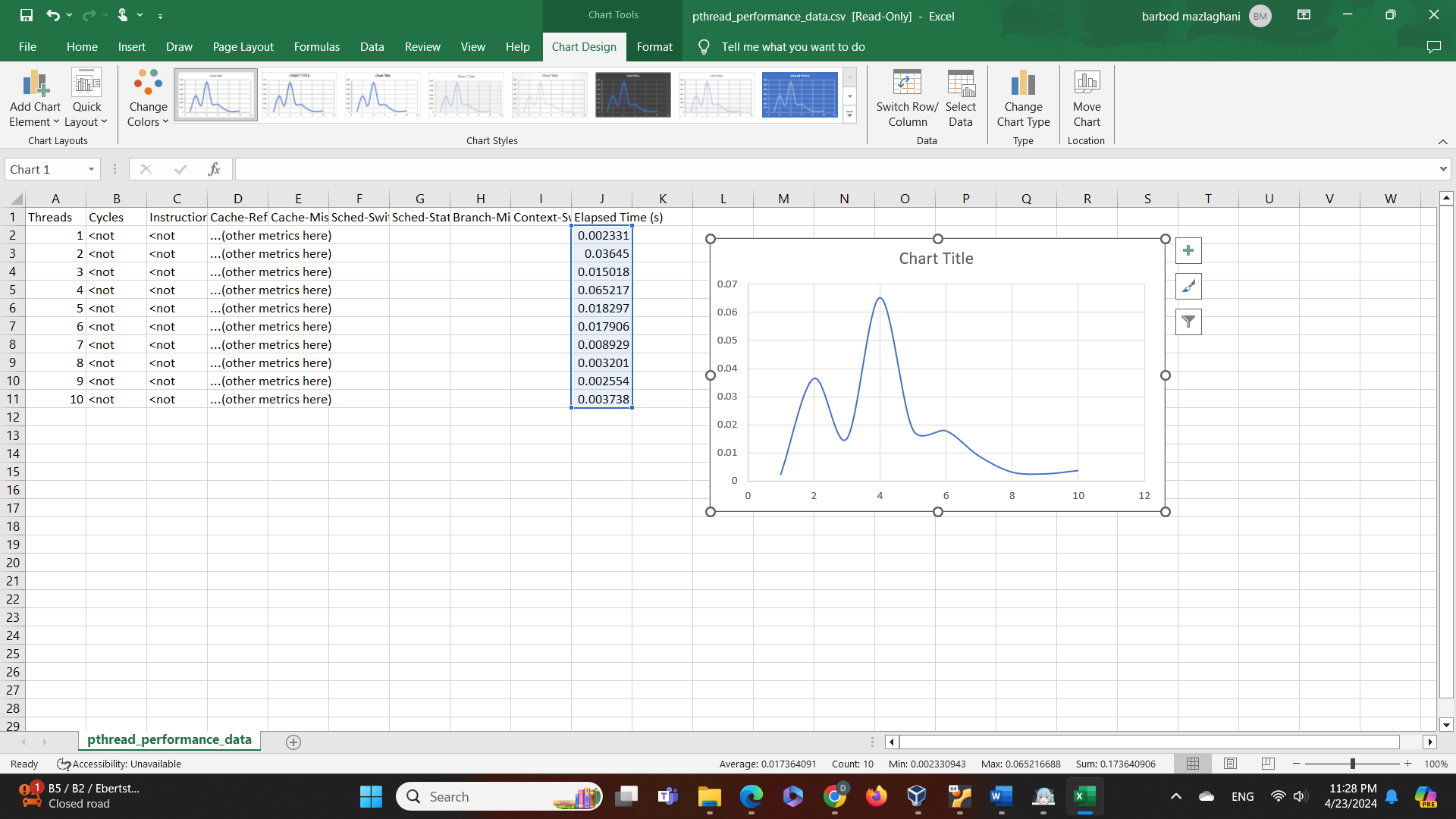




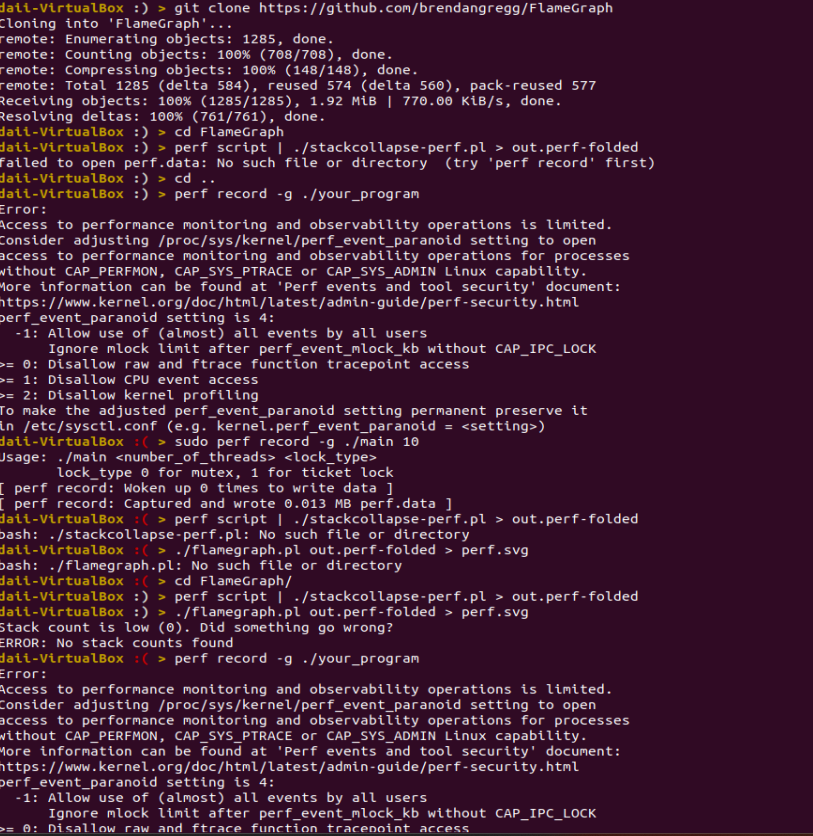
دوباره که صورت سوال را خواندم متوجه شدم که به نمودار هم نیاز داریم در همین راستا کد بش قسمت قبل را اندکی تغییر میدهیم و خروجی csv در دو مود لاک را میگیریم و نهایتا به کمک ابزار اکسل نمودار آن را رسم میکنیم و فایل csv آن ها در کنار اسکریپت ها موجود است و در دسترس شما قرار میگیرد.



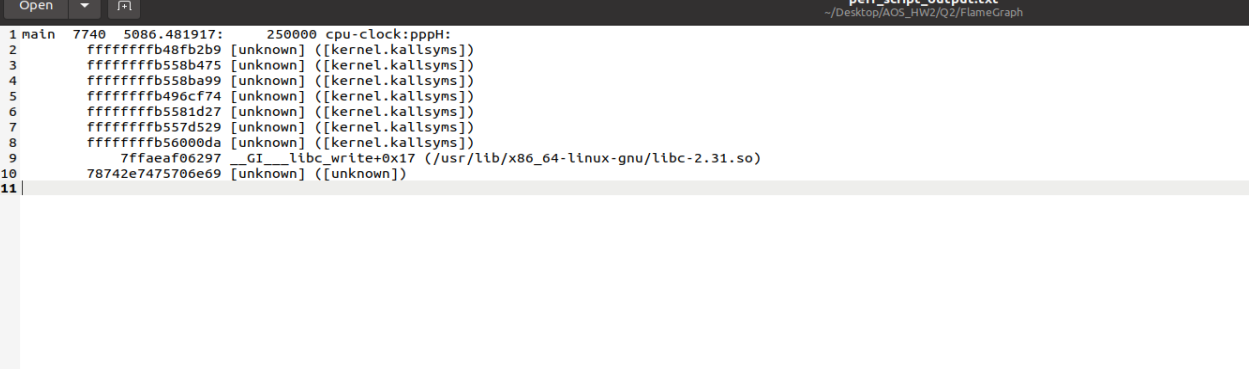
طبق مشاهده که در بالاتر هم گفتیم 7 بهترین بود.



در گام آخر به سراغ flameGraph میرویم ،



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 :



خروجی svg خالی میشد :

