

test1

8th gen Intel i5 4-cores/8-threads

os: Manjaro xfce 20.0.3

Времето се взима с Unix.time()

направени са по 10 теста за всяка големина на буфер с който четем

и и имаме 10 такива, и от 1 до 16 нишки

Решението ми използва coroutines функционалността на го, които се държат горе долу като нишки но според официалната документация са по леки от истински нишки. (за момента предположението ми е че няколко инстанции на могат да са на една процесорна нишка с някакъв мениджър който ги управлява)

В задачата проблема е следния четене от файл паралелно. Това само по себе си не е кой знае колко трудно, нищо че е бавна операция (ще се адресира след малко) четенето без промяна е threadsafe и след като не пишем никъде във файла който четем го прави още по safe.

Специално за го нямам представа как става четенето от файл, потенциално с дескриптори или File* (ще използвам C терминология, защото с нея съм най запознат) .

Специално за моята имплементация до колкото разбирам използваме дескриптори, което значи че имаме тази оптимизация, макар и file* да буферират (което може да окаже ефект при четене, в някои случаи можеда е по оптимално.) като съм взел и това предвид при моята програма и е имплементираната променлива дължина на буфера за четене(има ефект в някои случаи, почваме от 512 и удвояваме и след 4096 инкрементиране с 4096 и последния е 32к след като страничките в паметта са толкова големи). която се задава чрез параметър -m/-meth ,която приема параметър от 0 до 9.

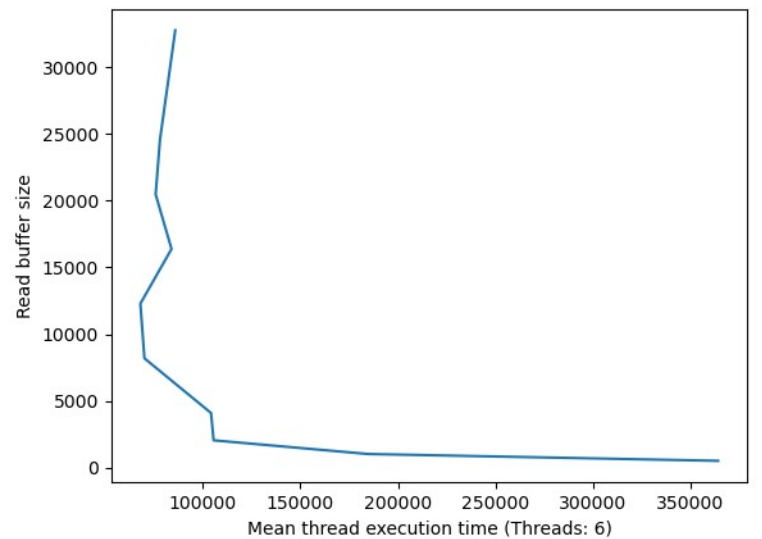
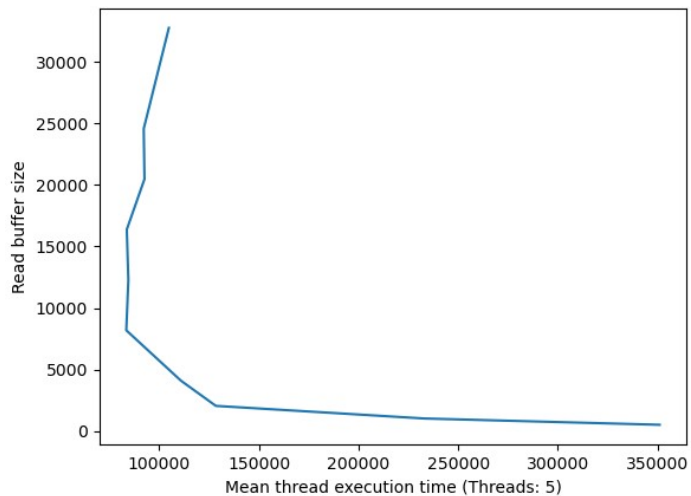
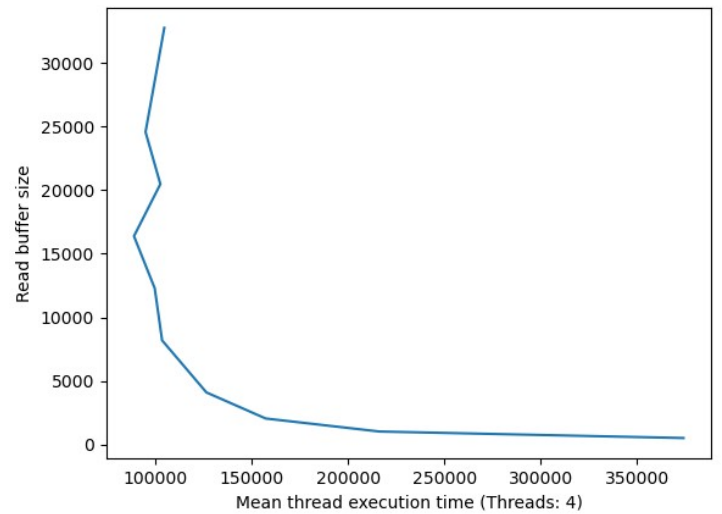
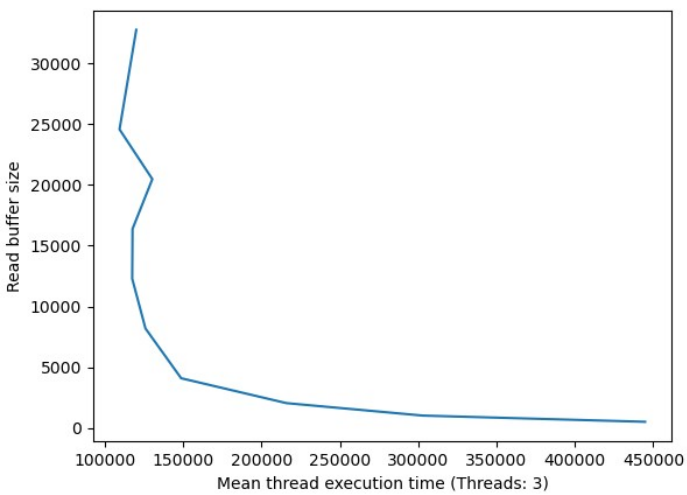
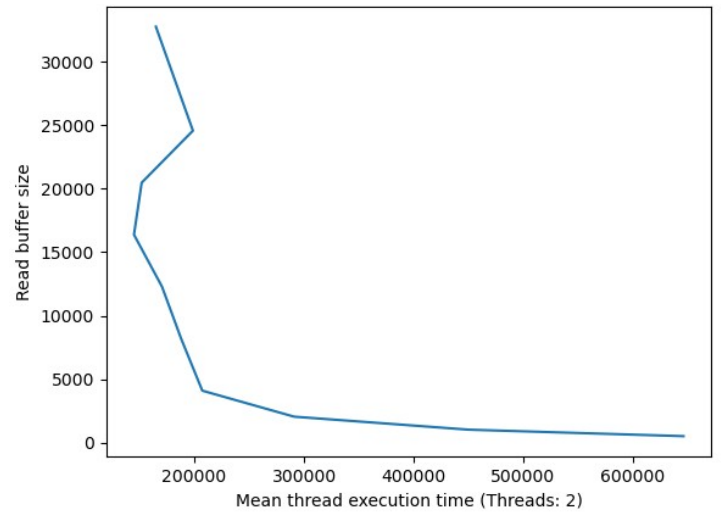
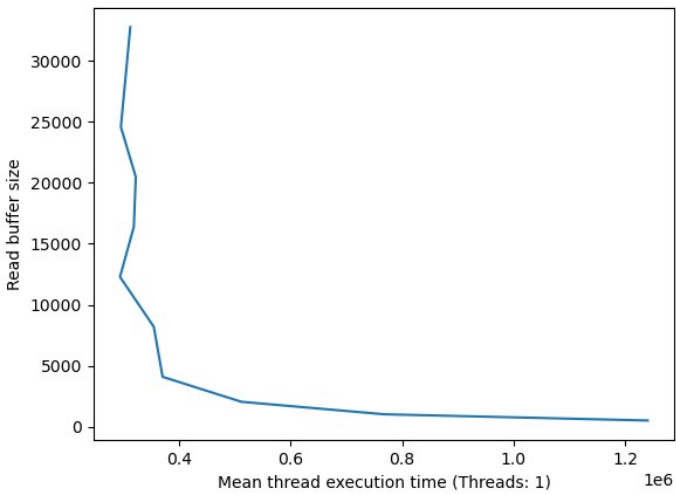
Накрая на програмата независимо от опциите се извежда работното време на всяка функция както и средното аритметично.

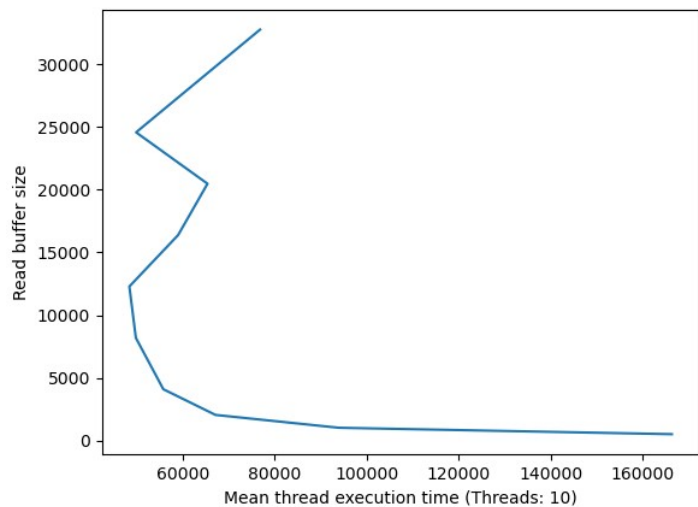
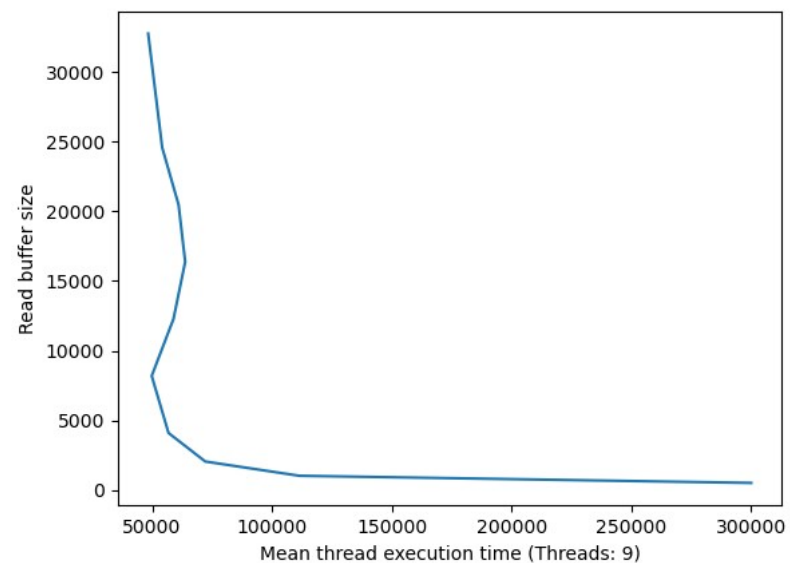
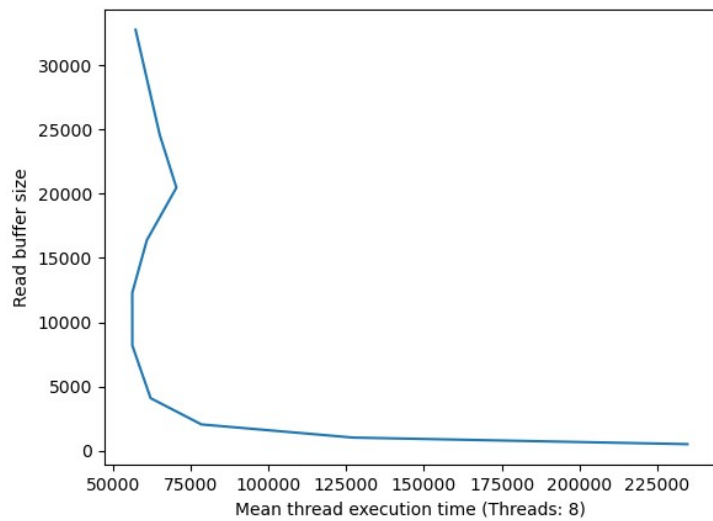
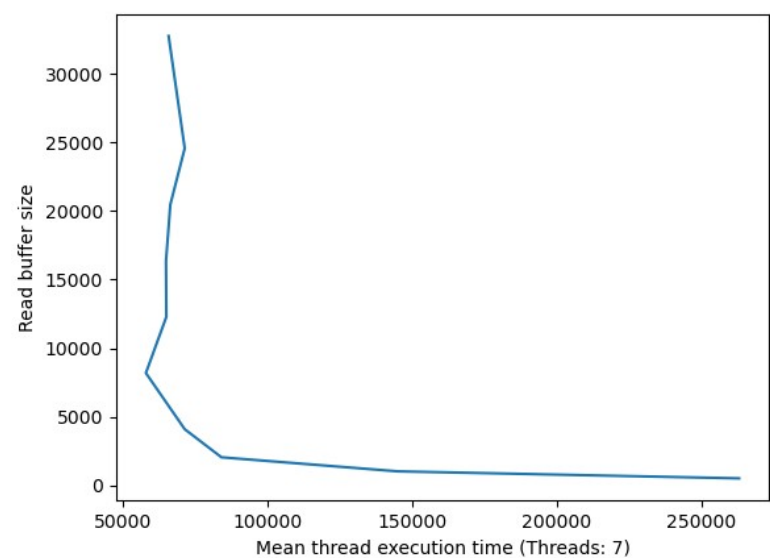
Има голяма вариация между времето за изпълнение, и то се взима преди да се отпусти мутекса. разбираме че тред е приключил след като се отпусти глобалния мутекс за този тред. и тогава можем съвсем спокойно да четем от тръбата.

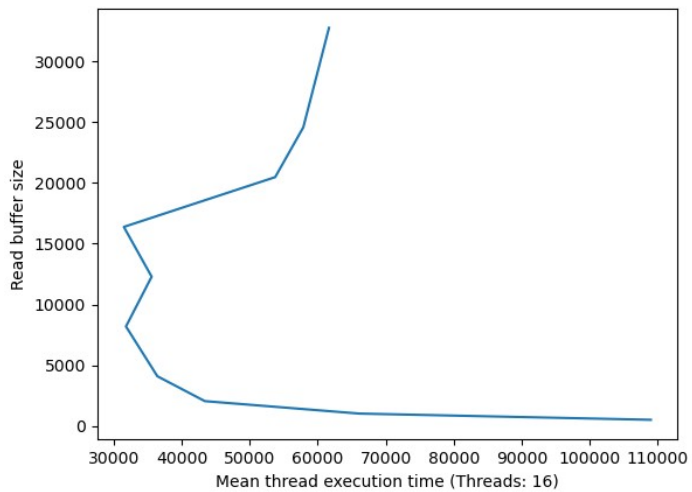
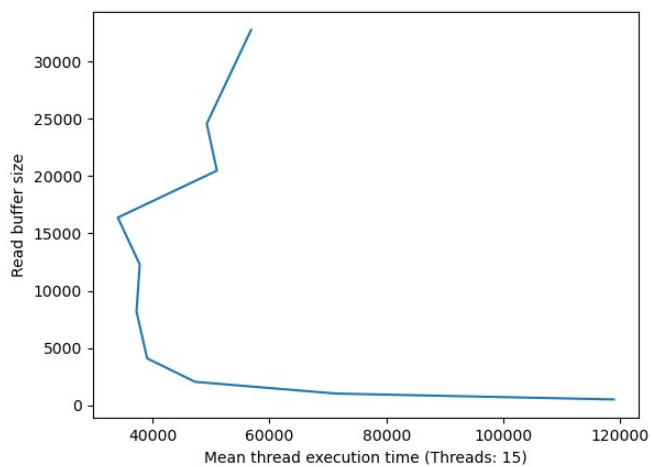
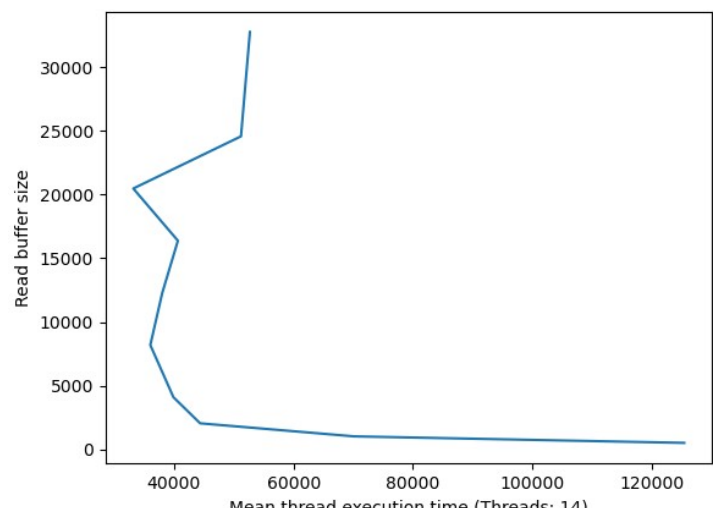
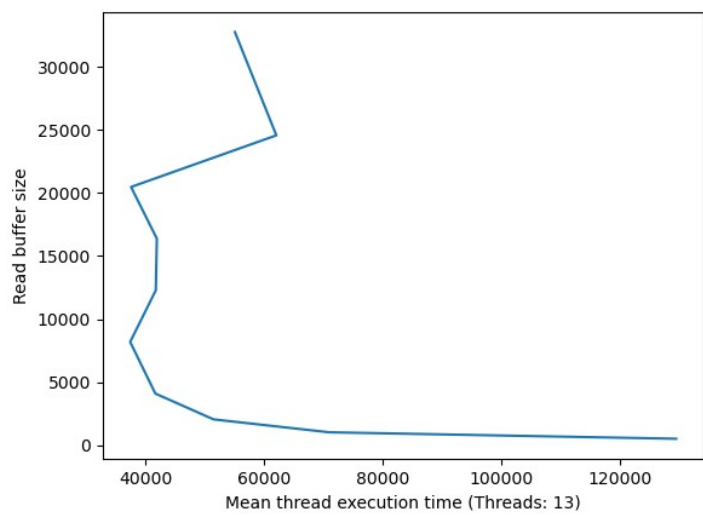
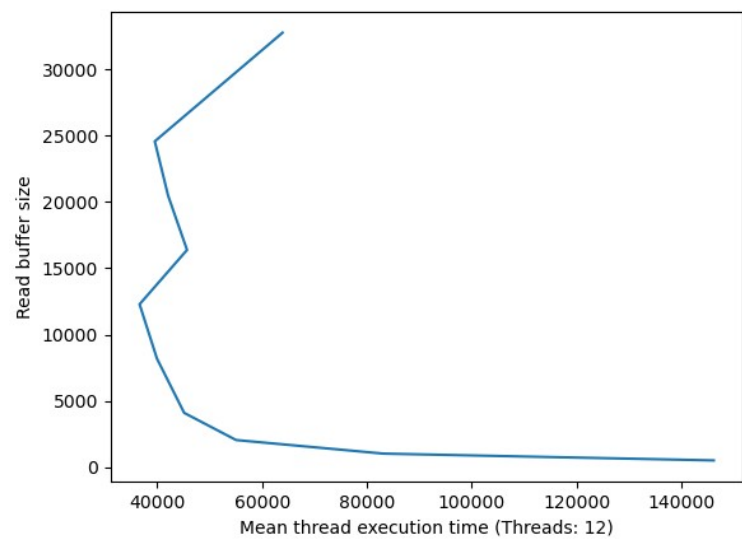
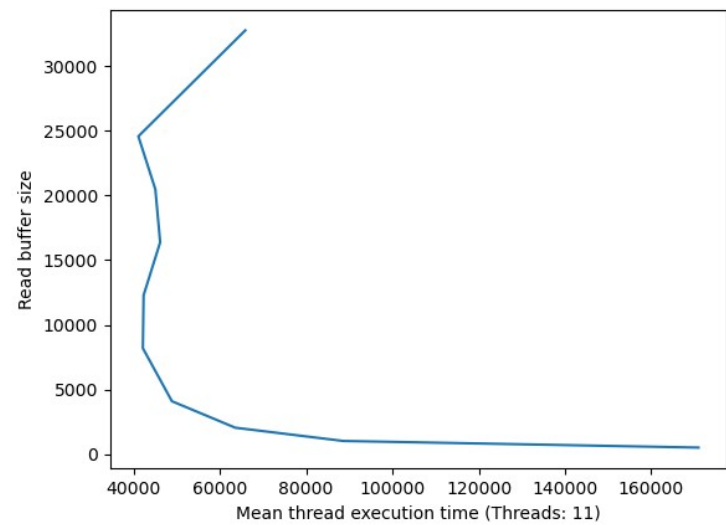
(първо от функцията пишем в тръбата резултата, след това освобождаваме мутекса, докато там където очакваме резултата се опитваме да заключим мутекса, което ни слага на опашката за него в кернела, тоест ще се обработи максимално бързо но не и истински паралелно(ако тред 2 приключи преди тред 1 ,тред 1 ще бъде обработен първи), минаваме от n+1 нишки в 1 и се налага сериализация чрез тези мутекси), липсва метод trylock в го или поне аз не се сещам за правилна имплементация. но това няма да промени много четенето

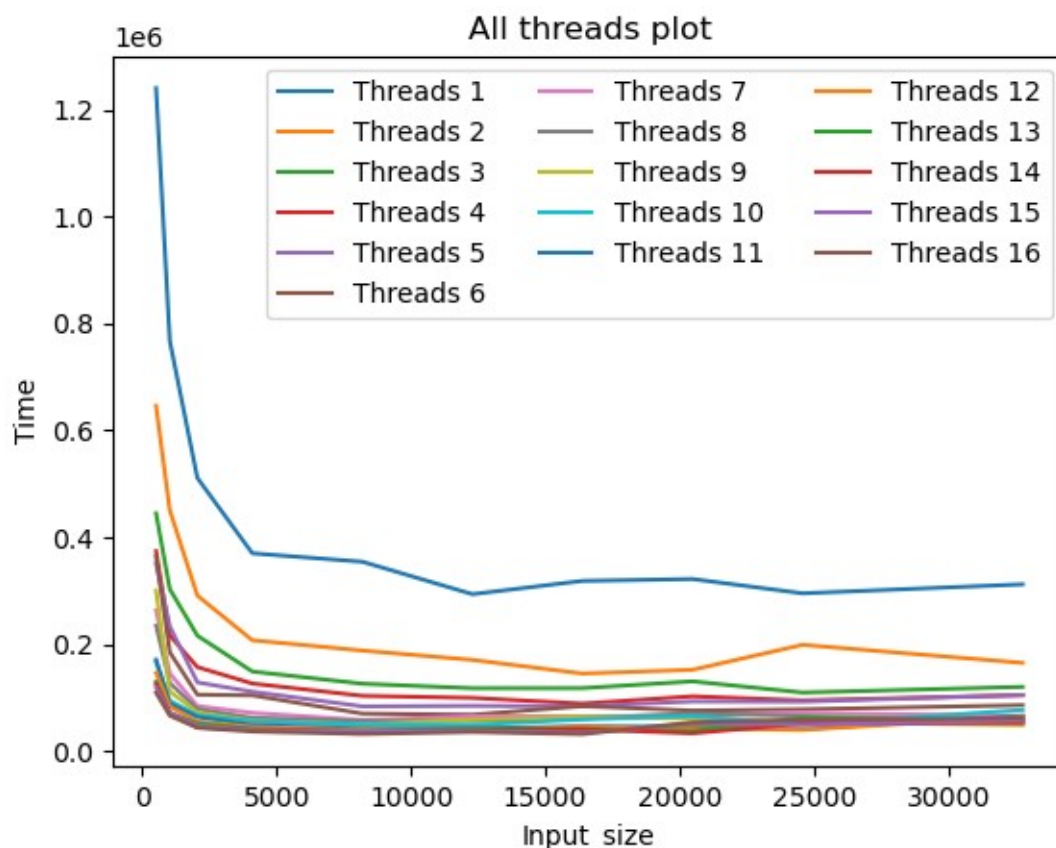
нишката приключва и резултата седи в тръбата.

графики:









Това са резултатите от тестване на моята машина.

Началната ни хипотеза е че ще се скалира задачката, което реално не става. И реално след 10+ нишки стават много еднакви резултатите и както се вижда от последните графики като се увеличи и размера който се чете това даже влияе отрицателно на времето за изпълнение, най вероятно защото се чака на опашки за четене в ядрото. И не процесора е този който ни бави а че ние не можем да четем достатъчно бързо и е по оптимално да четем по малко с по голям брой нишки, което е вярно откъм това че нишките имат по малко памет. И по ефективно решение би било по малко нишки които четат повече и го обработват, тъй като това е относително бързо нещо.

За ваше удобство съм приготвил 2 скрипта

Единият е bench.sh - той прави тестовете и записва времето за тях в csv файлове в папката res

Другият е graph_gen.py - който генерира тези изображения

както и файлове logem* които са тестови, един малък (~300к) и един по голям (~9mb)

но не мисля че има нужда да се вдига по нагоре размера на файла, тъй като и със сегашния (9mb) се бенчва бързо и резултата може да се види