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, потенцилано с дескриптори или File* (ще използвам C терминология, защото с нея съм най запознат) .

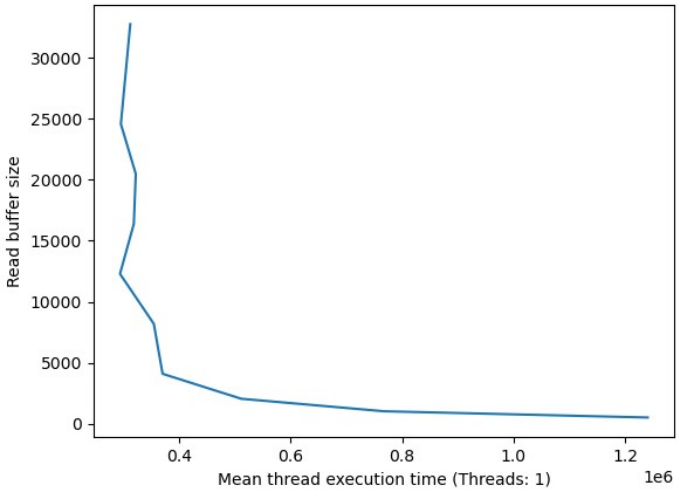
Специално за моята имплементация до колкото разбирам използваме дескриптори, което значи че имаме тази оптимизация, макар и file* да буферират (което може да окаже ефект при четене, в някои случаи можеда е по оптимално.) като съм взел и това предвид при моята програма
и е имплементирата пролменлива дължина на буфера за четене(има ефект в някои случаи, почваме от 512 и удвояваме и след 4096 инкрементираме 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.

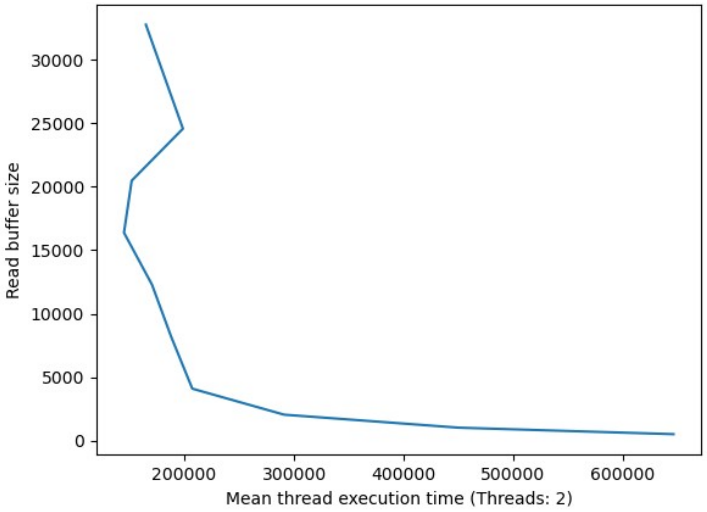
Накрая на програмата независимо от опциите се извежда работното време на всяка функция както и средното аритметично.

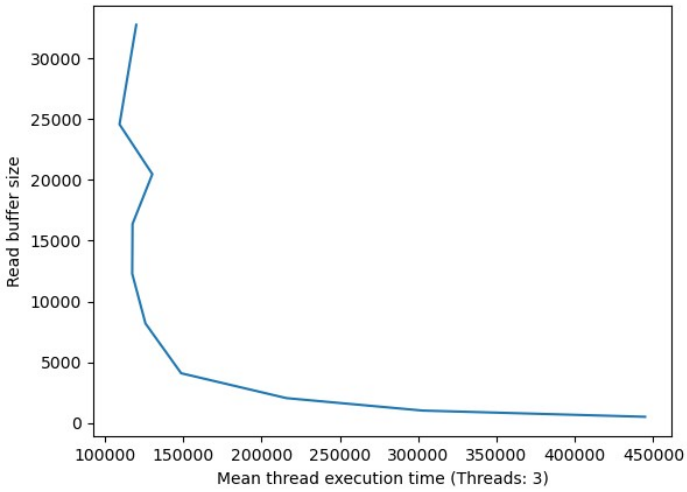
Има голяма вариация между врмето за изпълнение, и то се взима преди да се отпуши мутекса. разбираме че треда е приключил след като се отпуши глобалния мутекс за този тред. и тогава можем съвсем спокойно да четем от тръбата.
(първо от функцията пишем в тръбата резултата, след това освобождаваме мутекса, 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 н+1 нишки в 1 и се налага сериалицаия чрез тези мутекси),
липсва метод trylock в го или поне аз не се сещам за правилна имплемнетация. но това няма да промени мноого четенето

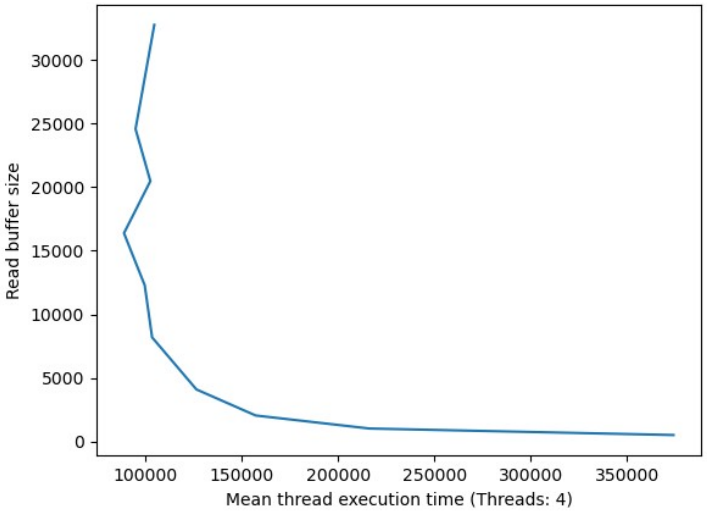
нишката приключва и резултата седи в тръбата.

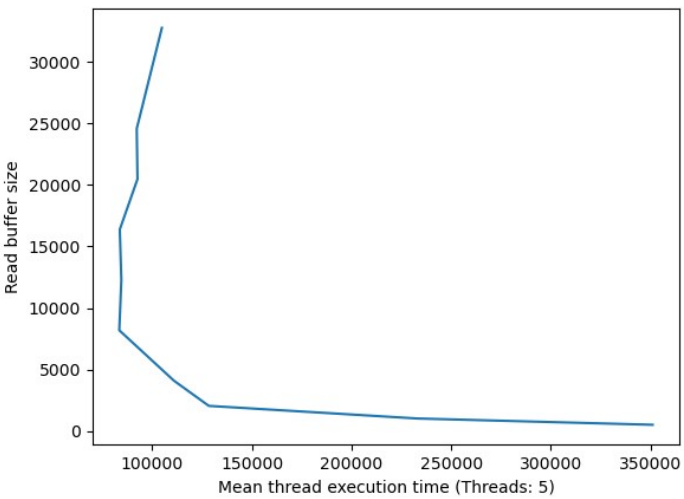
графики:

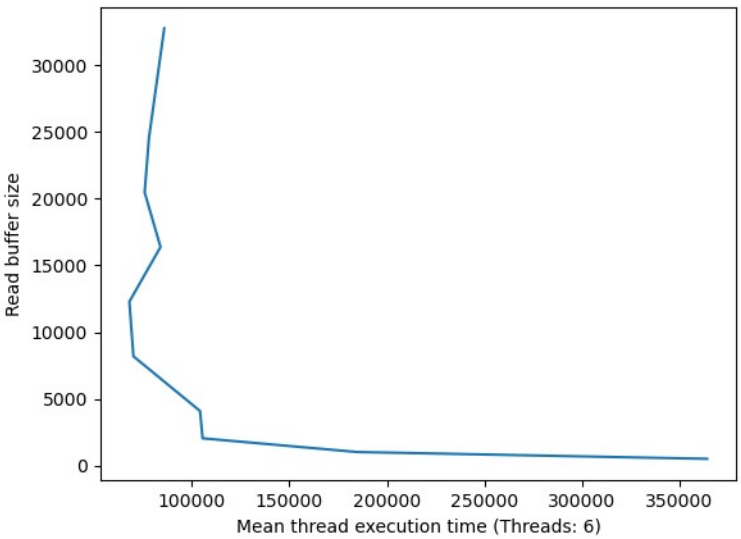

Read buffer size
30000
25000
20000
15000
10000
5000
0
0.4
0.6
0.8
1.0
1.2
1e6
Mean thread execution time (Threads: 1)

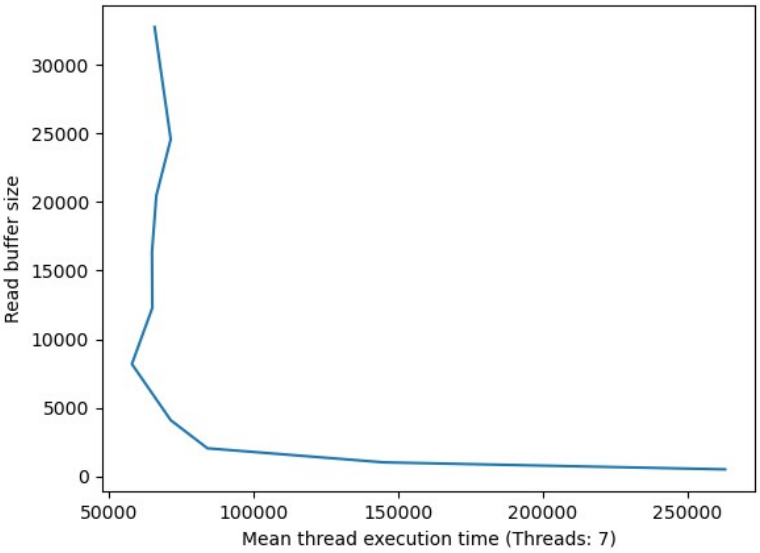

Read buffer size
30000
25000
20000
15000
10000
5000
0
200000
300000
400000
500000
600000
Mean thread execution time (Threads: 2)

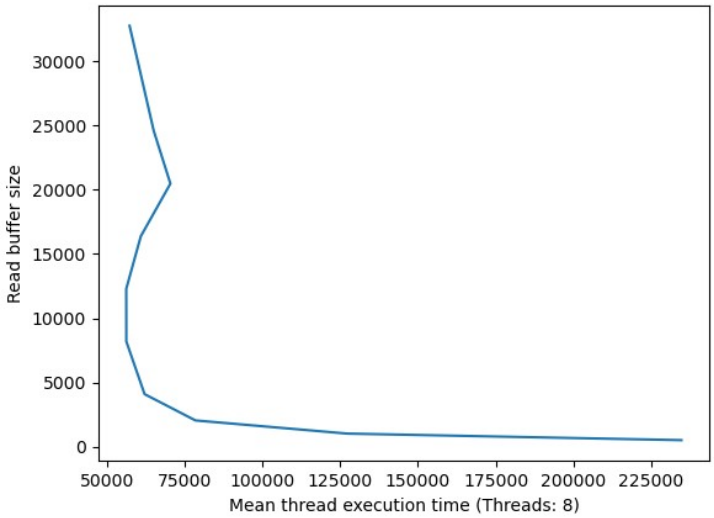

Read buffer size
30000
25000
20000
15000
10000
5000
0
100000
150000
200000
250000
300000
350000
400000
450000
Mean thread execution time (Threads: 3)

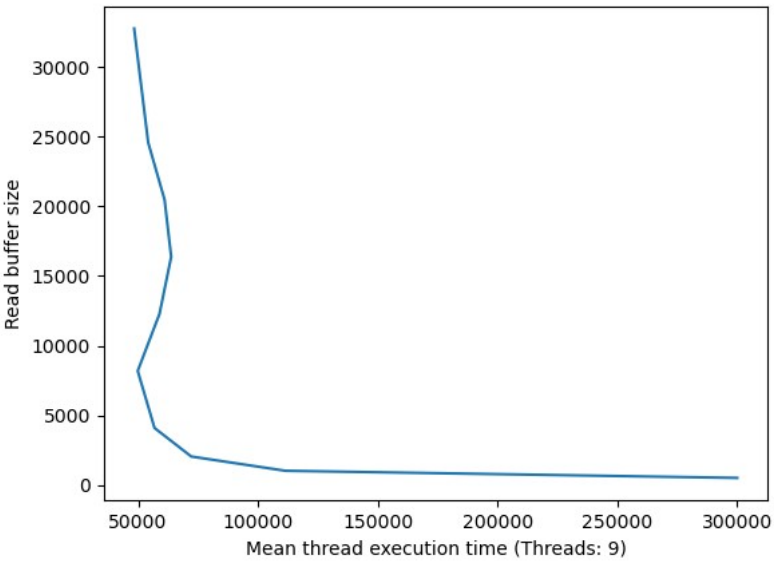

Read buffer size
30000
25000
20000
15000
10000
5000
0
100000
150000
200000
250000
300000
350000
Mean thread execution time (Threads: 4)

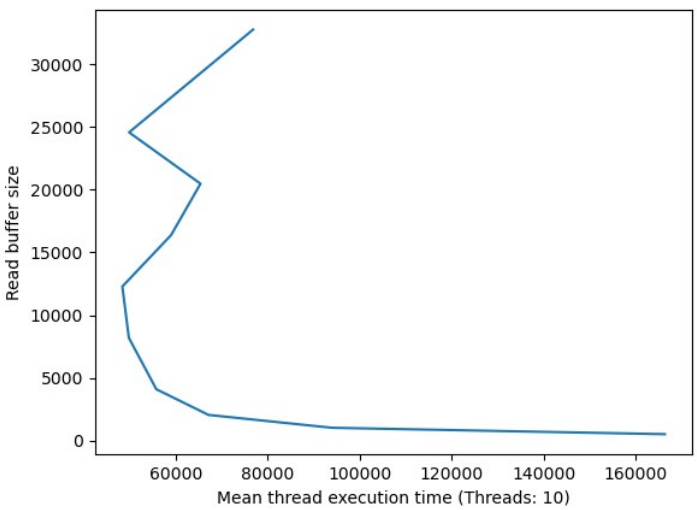

Read buffer size
30000
25000
20000
15000
10000
5000
0
100000
150000
200000
250000
300000
350000
Mean thread execution time (Threads: 5)

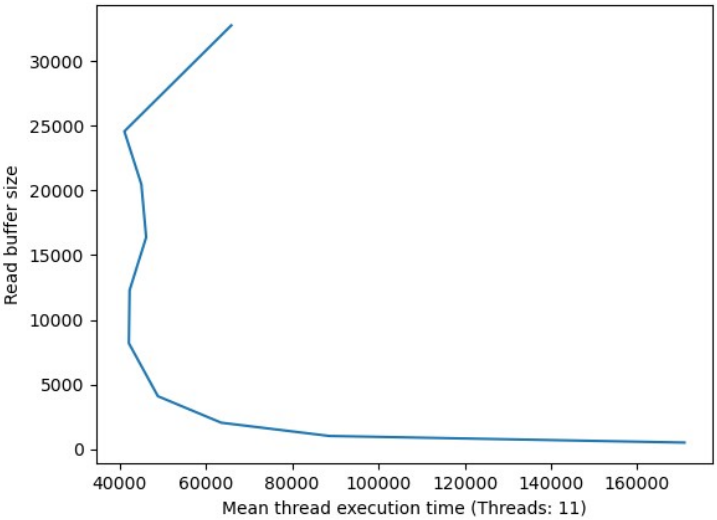

Read buffer size
30000
25000
20000
15000
10000
5000
0
100000
150000
200000
250000
300000
350000
Mean thread execution time (Threads: 6)

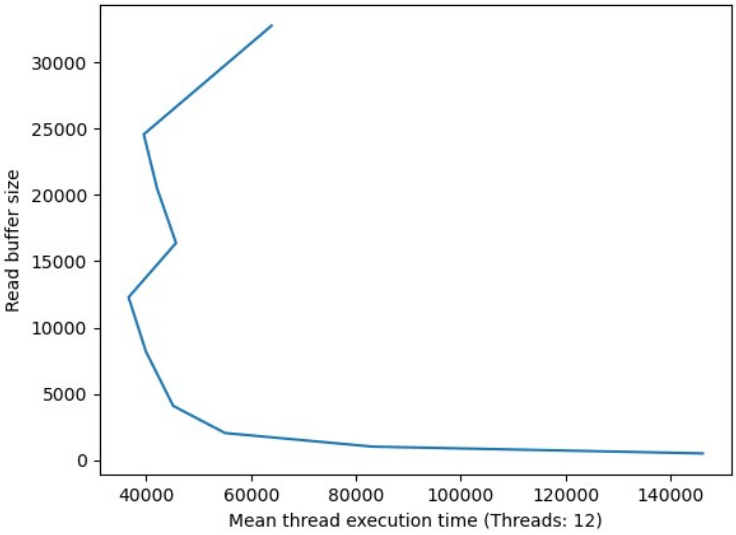

Read buffer size
30000
25000
20000
15000
10000
5000
0
50000
100000
150000
200000
250000
Mean thread execution time (Threads: 7)

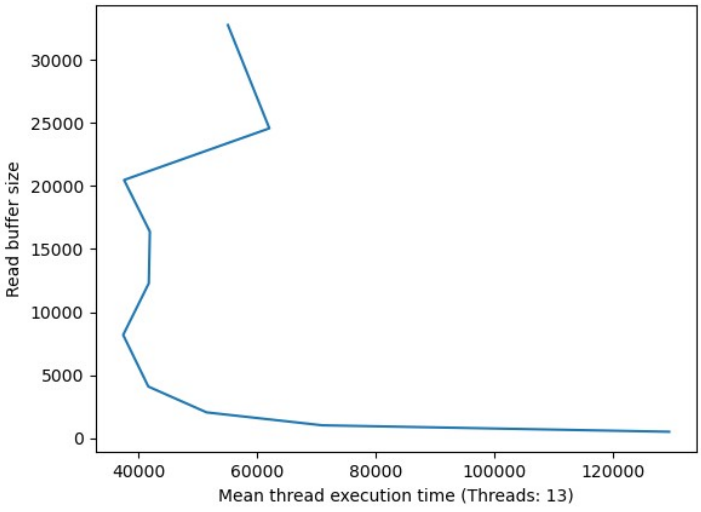

Read buffer size
30000
25000
20000
15000
10000
5000
0
50000
75000
100000
125000
150000
175000
200000
225000
Mean thread execution time (Threads: 8)

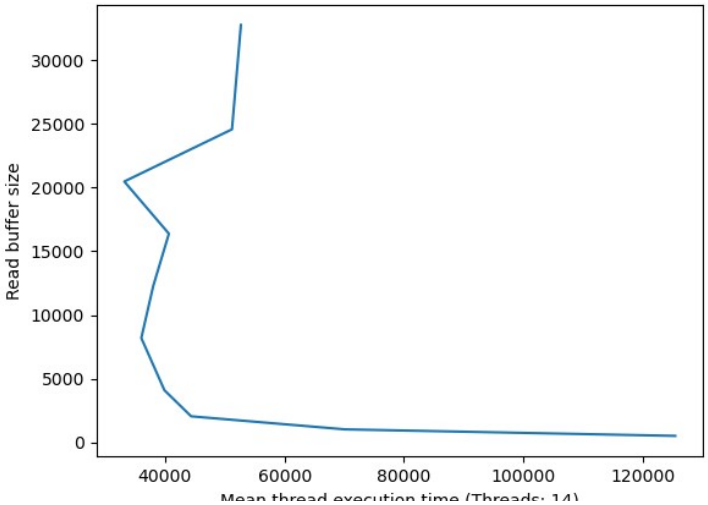

Read buffer size
30000
25000
20000
15000
10000
5000
0
50000
100000
150000
200000
250000
300000
Mean thread execution time (Threads: 9)

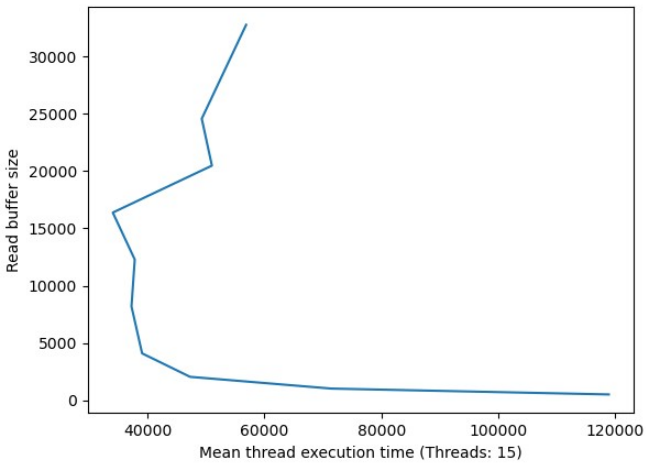

Read buffer size
30000
25000
20000
15000
10000
5000
0
60000
80000
100000
120000
140000
160000
Mean thread execution time (Threads: 10)

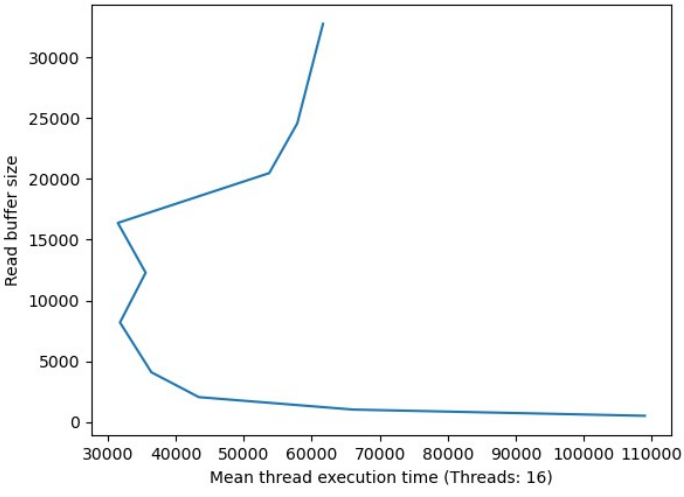

Read buffer size
30000
25000
20000
15000
10000
5000
0
40000
60000
80000
100000
120000
140000
160000
Mean thread execution time (Threads: 11)

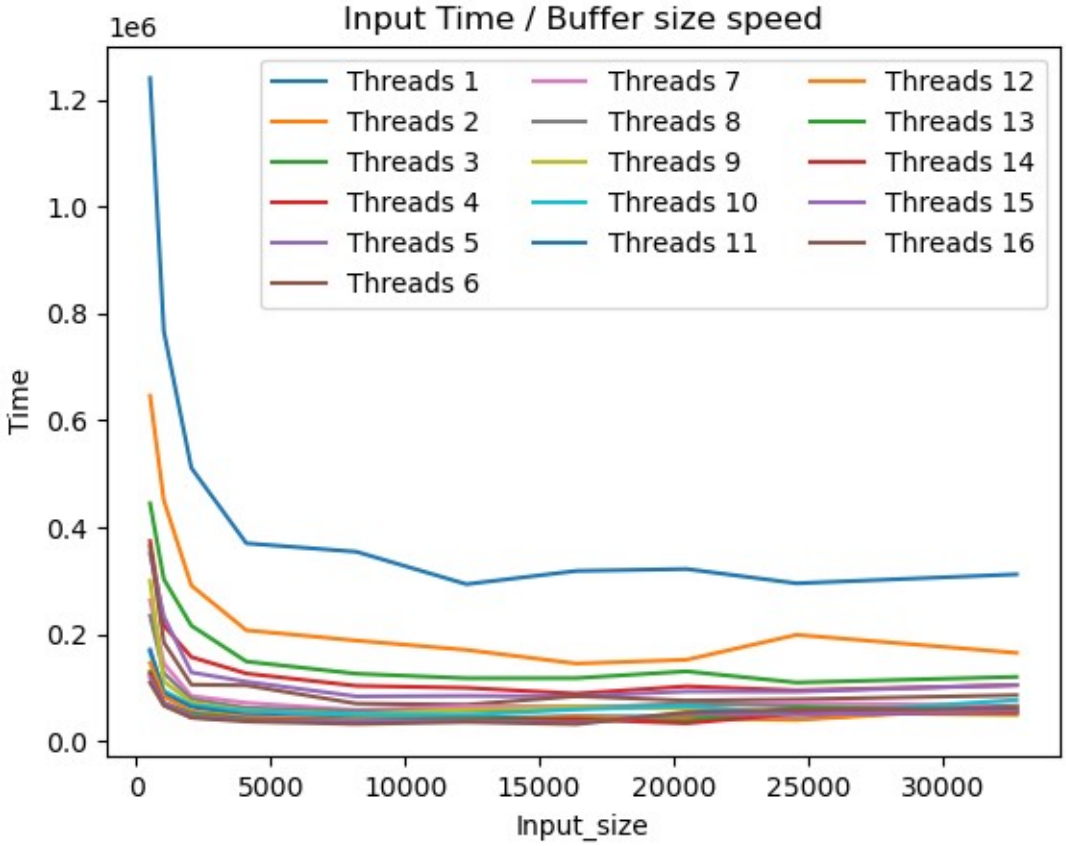

Read buffer size
30000
25000
20000
15000
10000
5000
0
40000
60000
80000
100000
120000
140000
Mean thread execution time (Threads: 12)

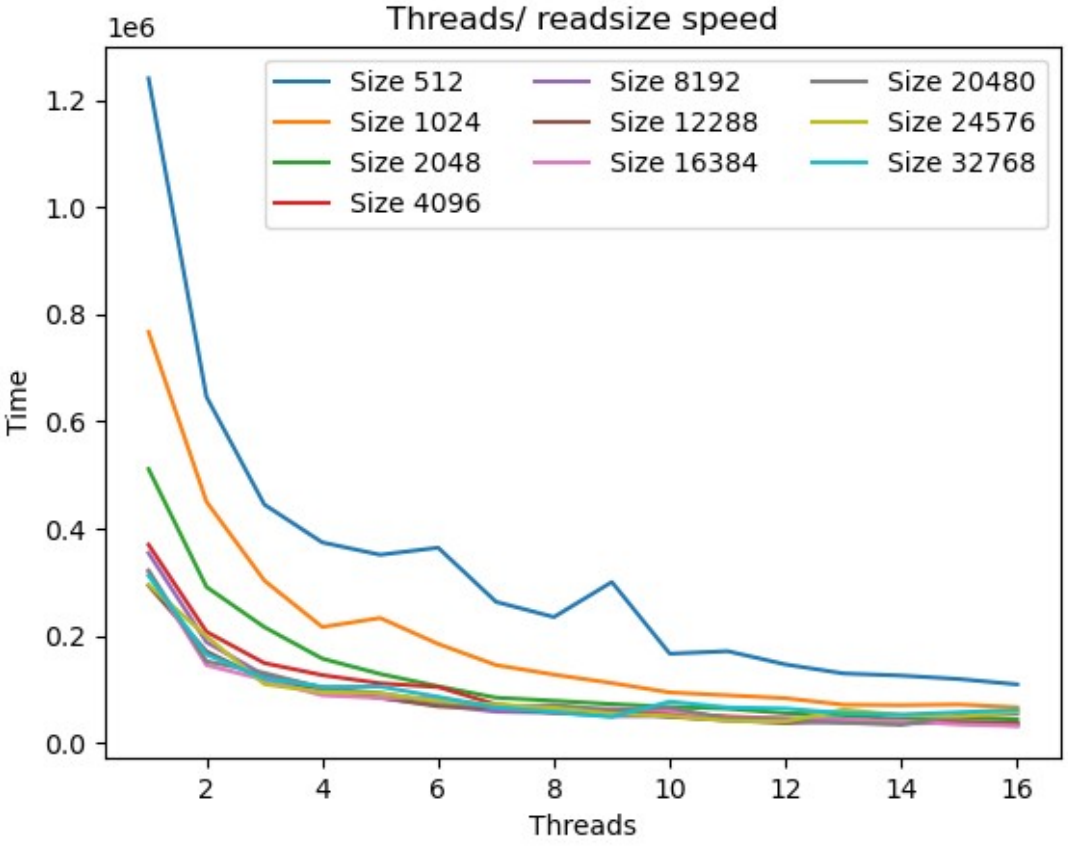

Read buffer size
30000
25000
20000
15000
10000
5000
0
40000
60000
80000
100000
120000
Mean thread execution time (Threads: 13)


Read buffer size
30000
25000
20000
15000
10000
5000
0
40000
60000
80000
100000
120000


Read buffer size
30000
25000
20000
15000
10000
5000
0
40000
60000
80000
100000
120000
Mean thread execution time (Threads: 15)


Read buffer size
30000
25000
20000
15000
10000
5000
0
30000
40000
50000
60000
70000
80000
90000
100000
110000
Mean thread execution time (Threads: 16)


Input Time / Buffer size speed
1e6
Threads 1
Threads 2
Threads 3
Threads 4
Threads 5
Threads 6
Threads 7
Threads 8
Threads 9
Threads 10
Threads 11
Threads 12
Threads 13
Threads 14
Threads 15
Threads 16
Time
Input_size


Threads/ readsize speed
1e6
Size 512
Size 1024
Size 2048
Size 4096
Size 8192
Size 12288
Size 16384
Size 20480
Size 24576
Size 32768
Time
Threads

Това са резултатите от тестване на моята машина.
Началната ни хипотеза е че ще се скалира задачката, кеот ореално не става. И реално след 10+ нишки стават много оеднакви ресултатите 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 отзи 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 както може да се забележи от “threads/readsize speed” графиката, скалирането е добро само ако имаме малък буфер и четем често. Но тъй като не е оптимално да четем голям файл наведнъж за да се обработи наведнъж, и ко просто вдигаме колко четем получаваме доста загубено процесорно време, както се вижда от го. Не получаваме много повече скалиране след 8 нишки (това може да е защото аз на моята машина иамм 8 но според мене е кернела и тва че четем от диска … което е мнгооого бавно, казвам за 10-ти път). Да знам че не беше в заданието да се имплемнтира големина на буфера, но понеже знам че четенето е бавно реших да го метна като опция да пожа колко то може да принесе, и че не всички задачи е хубаво само да се слагат повече нишки и то ще върви по добре(макр че не съм чувал никой да го казва това, но все пак добро предположение е спрямо това какво се очква)

За ваше удобство съм приготвил 2 скрипта
Единия е bench.sh - той прави тестовете и записва времето за тях в csv файлове в папкта res
Другият е graph_gen.py - Който генерира тези изображения
както и файлове lorem* 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. Моля не обръщайте внимание на ужасния ми правопис и грешки. Пунктоацията я нямам за нищо и не ми дреме особнео за правопис.
Смятам че това е ужасна задача за проект, след като диска е ограничението, както и ос-то, което е голямо забавяне и не можед а се покаже забързването толкова. Но тва си е мое мнение де, курса си е ваш.