Порівняльний аналіз управління пам’яттю в мовах програмування

Виконала Шляхова Олександра Дмитрівна

Керівник – Кандидат фізико-математичних наук,

доцент Бублик Володимир Васильович

Постановка Задачі

- Виявити особливості застосування ручного і автоматичного управління пам'яттю
- З'ясувати накладні витрати, отримані в результаті використання автоматичного управління пам'яттю
- Надати рекомендації до використання мови програмування і методів керування пам'яттю для задач створення інвертованого індексу з підтримкою графічного інтерфейсу
- Класифікувати переваги та недоліки розглянутих способів автоматичного управління пам'яттю

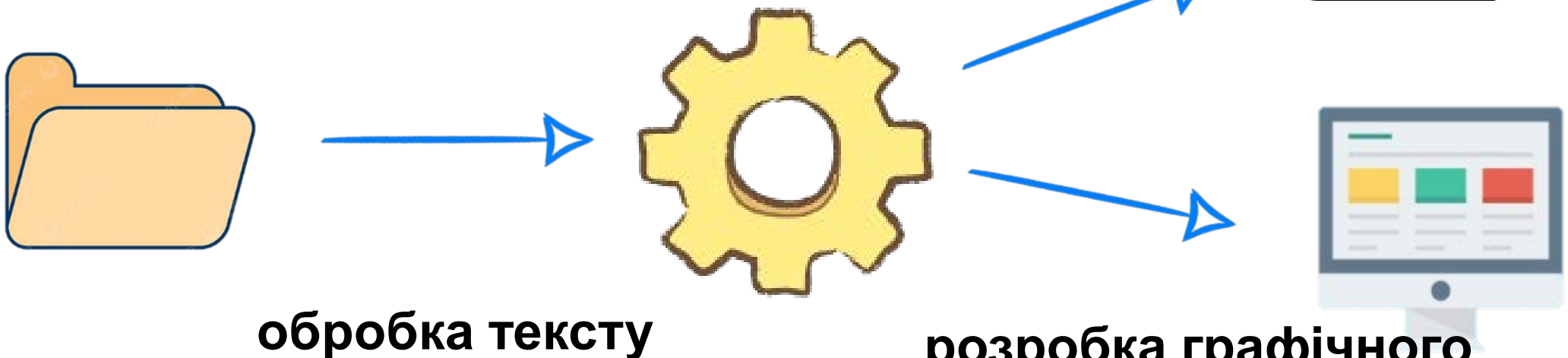
Етапи Роботи

- Ознайомлення з існуючими способами управління пам'яттю
- Розгляд цих способів у контексті існуючих мов програмування
- Написання застосунку мовами Java та C++, порівняння їх ефективності в контексті використання пам’яті
- Підбиття підсумків

Порівняння стратегій використання пам'яті у програмах, написаних мовами C++ та Java

створення словника

Опис Програми



обробка тексту

розробка графічного інтерфейсу

Засоби для Моніторингу Використання Ресурсів

- Вбудована система профілювання Microsoft Visual Studio
- IntelliJ Profiler
- Моніторинг Ресурсів операційної системи Windows
- Інструменти JDK для збору статистики використання власної (native) пам'яті та роботи прибиральника сміття

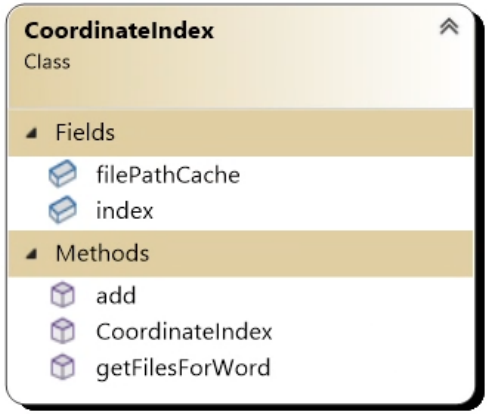
Опис Простої Реалізації

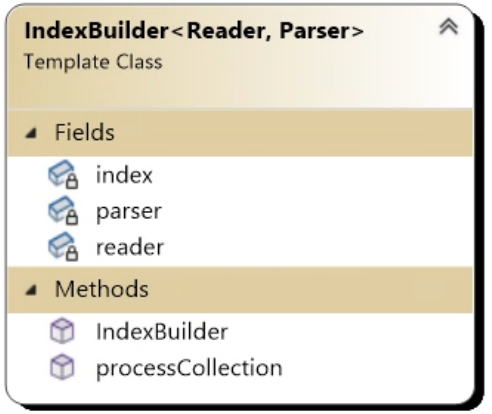
- Рекурсивне зчитування файлів
- Розбиття вмісту на слова
- Збереження пар слово – назва файлу

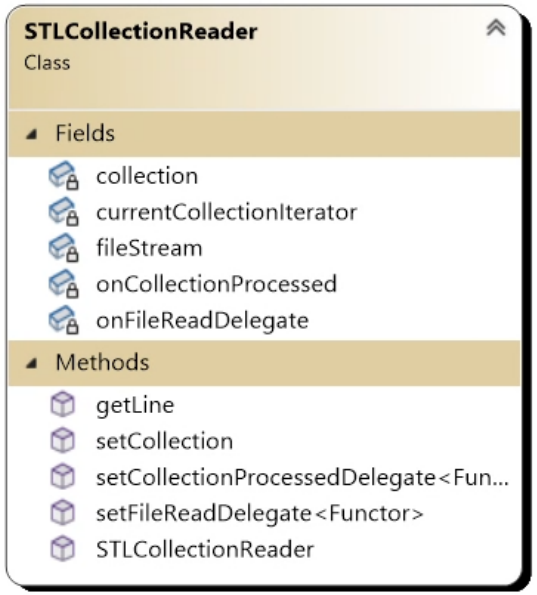
Проста Реалізація Мовою C++

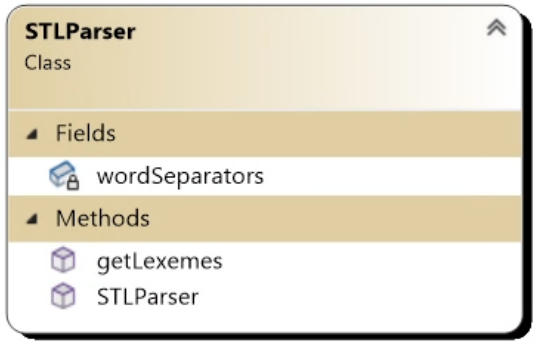
- Збереження пар слово – назва файлу

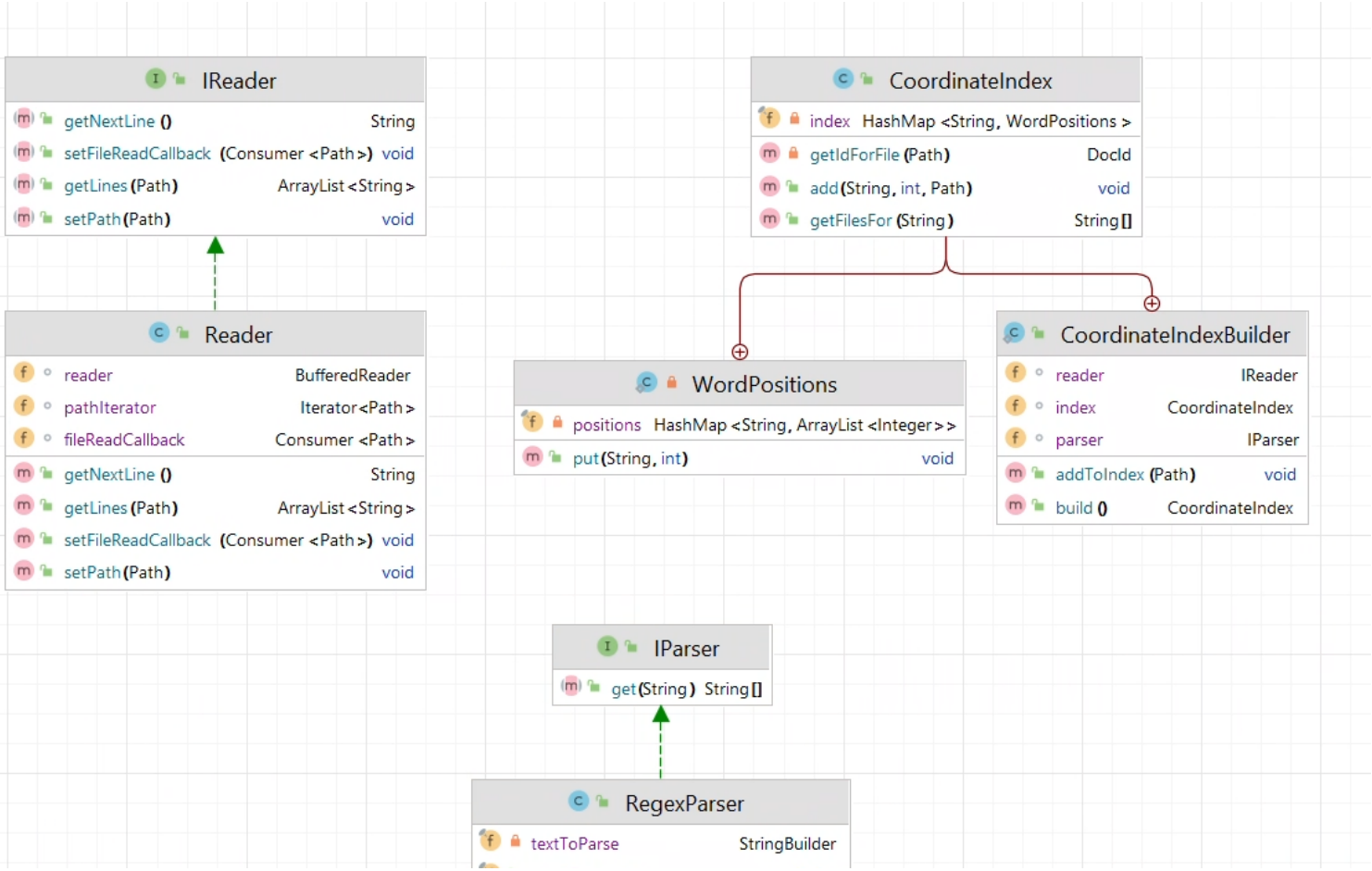
- Використання бібліотеки STL







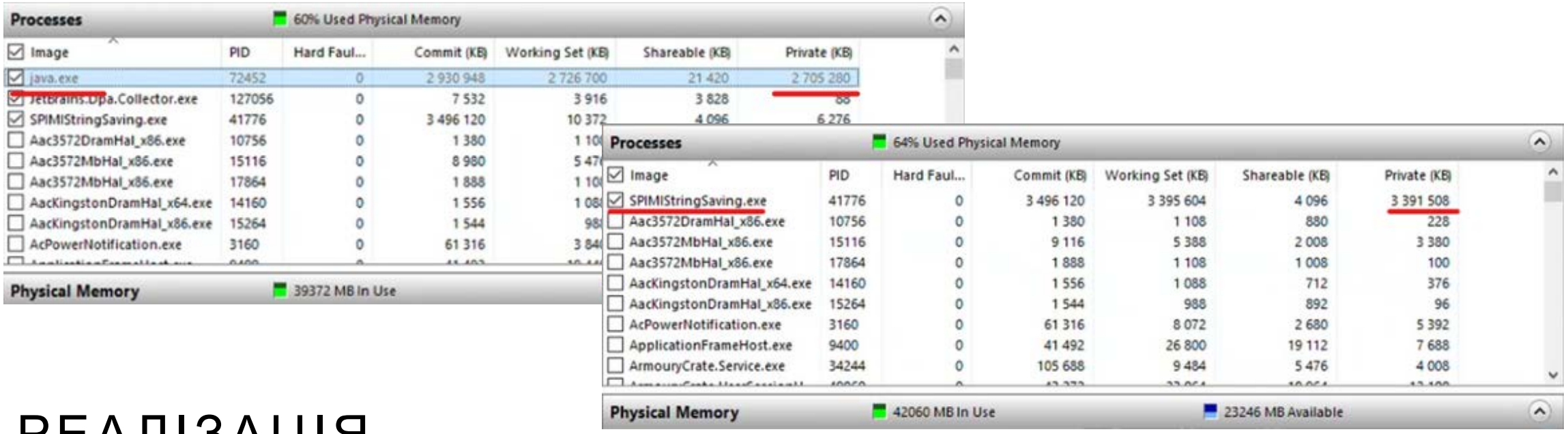




Проста Реалізація Мовою Java

- Збереження пар слово – назва файлу
- Використання стандартної бібліотеки Java

Порівняння Простих Реалізацій



Processes — 60% Used Physical Memory

Image	PID	Hard Faul...	Commit (KB)	Working Set (KB)	Shareable (KB)	Private (KB)
java.exe	72452	0	2 930 948	2 726 700	21 420	2 705 280
JetBrains.Dpa.Collector.exe	127056	0	7 532	3 916	3 828	88
SPIMIStringSaving.exe	41776	0	3 496 120	10 372	4 096	6 276
Aac3572DramHal_x86.exe	10756	0	1 380	1 10		
Aac3572MbHal_x86.exe	15116	0	8 980	5 47		
Aac3572MbHal_x86.exe	17864	0	1 888	1 10		
AacKingstonDramHal_x64.exe	14160	0	1 556	1 08		
AacKingstonDramHal_x86.exe	15264	0	1 544	98		
AcPowerNotification.exe	3160	0	61 316	3 84		

Physical Memory — 39372 MB In Use

Processes — 64% Used Physical Memory

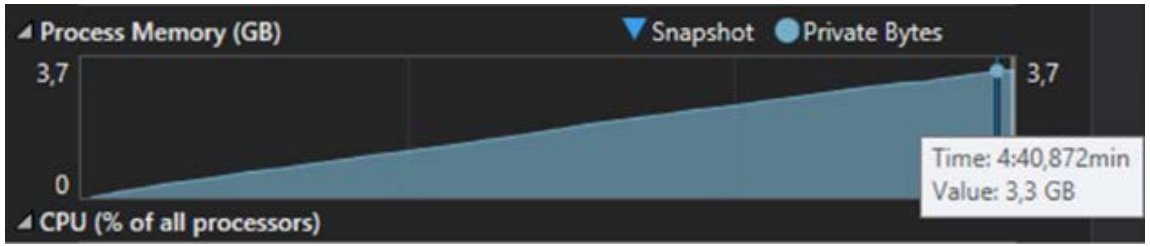
Image	PID	Hard Faul...	Commit (KB)	Working Set (KB)	Shareable (KB)	Private (KB)
SPIMIStringSaving.exe	41776	0	3 496 120	3 395 604	4 096	3 391 508
Aac3572DramHal_x86.exe	10756	0	1 380	1 108	880	228
Aac3572MbHal_x86.exe	15116	0	9 116	5 388	2 008	3 380
Aac3572MbHal_x86.exe	17864	0	1 888	1 108	1 008	100
AacKingstonDramHal_x64.exe	14160	0	1 556	1 088	712	376
AacKingstonDramHal_x86.exe	15264	0	1 544	988	892	96
AcPowerNotification.exe	3160	0	61 316	8 072	2 680	5 392
ApplicationFrameHost.exe	9400	0	41 492	26 800	19 112	7 688
ArmouryCrate.Service.exe	34244	0	105 688	9 484	5 476	4 008

Physical Memory — 42060 MB In Use — 23246 MB Available

РЕАЛІЗАЦІЯ

Мовою C++ використовувала 3 391 508 Мб

Мовою Java використовувала 2 705 280 Мб



Порівняння Простих Реалізацій (залежність використаної пам’яті від часу виконання)

- Різниця розмірів купи пов’язана з дублюванням стрічок, яке наявне в реалізації мовою C++, але автоматично оптимізується мовою програмування Java

Порівняння Швидкодії Основних Операцій

	File reading	Text Parsing	Saving Words
C++(STL)	5 560 мс	194 148 мс	31 377 мс
Java	5 081 мс	12 875 мс	32 843 мс

Розбиття тексту на слова з використанням STL виявилось дуже неефективним

У вдосконаленій реалізації ця проблема вирішена за допомогою бібліотеки Boost

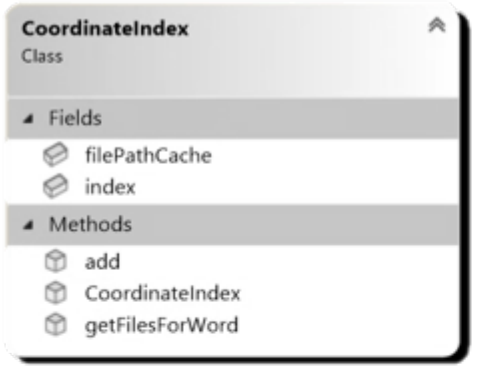
Порівняння Простих Реалізацій

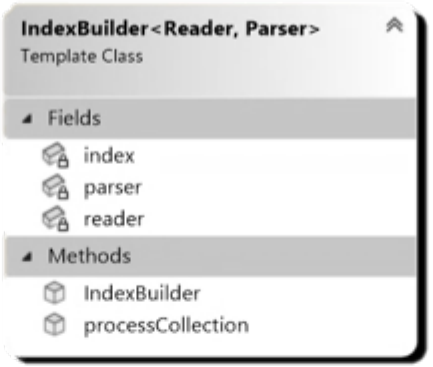
Загалом, проста реалізація мовою Java виявилась більш ефективною з точки зору використання пам’яті через вбудовані оптимізації з незначною перевагою в швидкодії

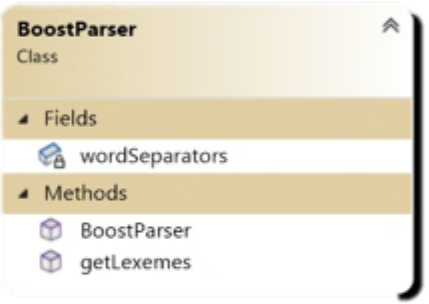
Опис Вдосконаленої Реалізації

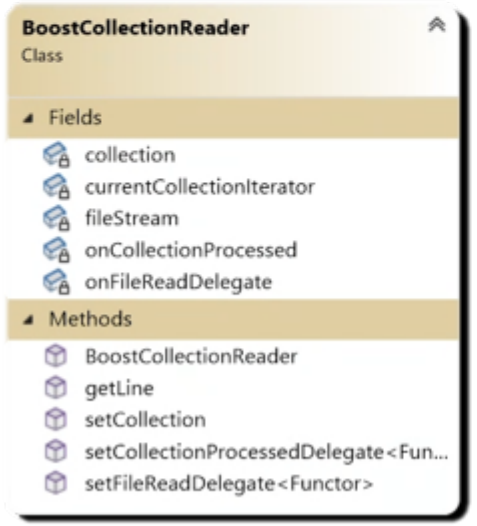
- Збереження ідентифікаторів файлів замість стрічок

- Використання бібліотеки Boost для зменшення часу роботи програми мовою C++



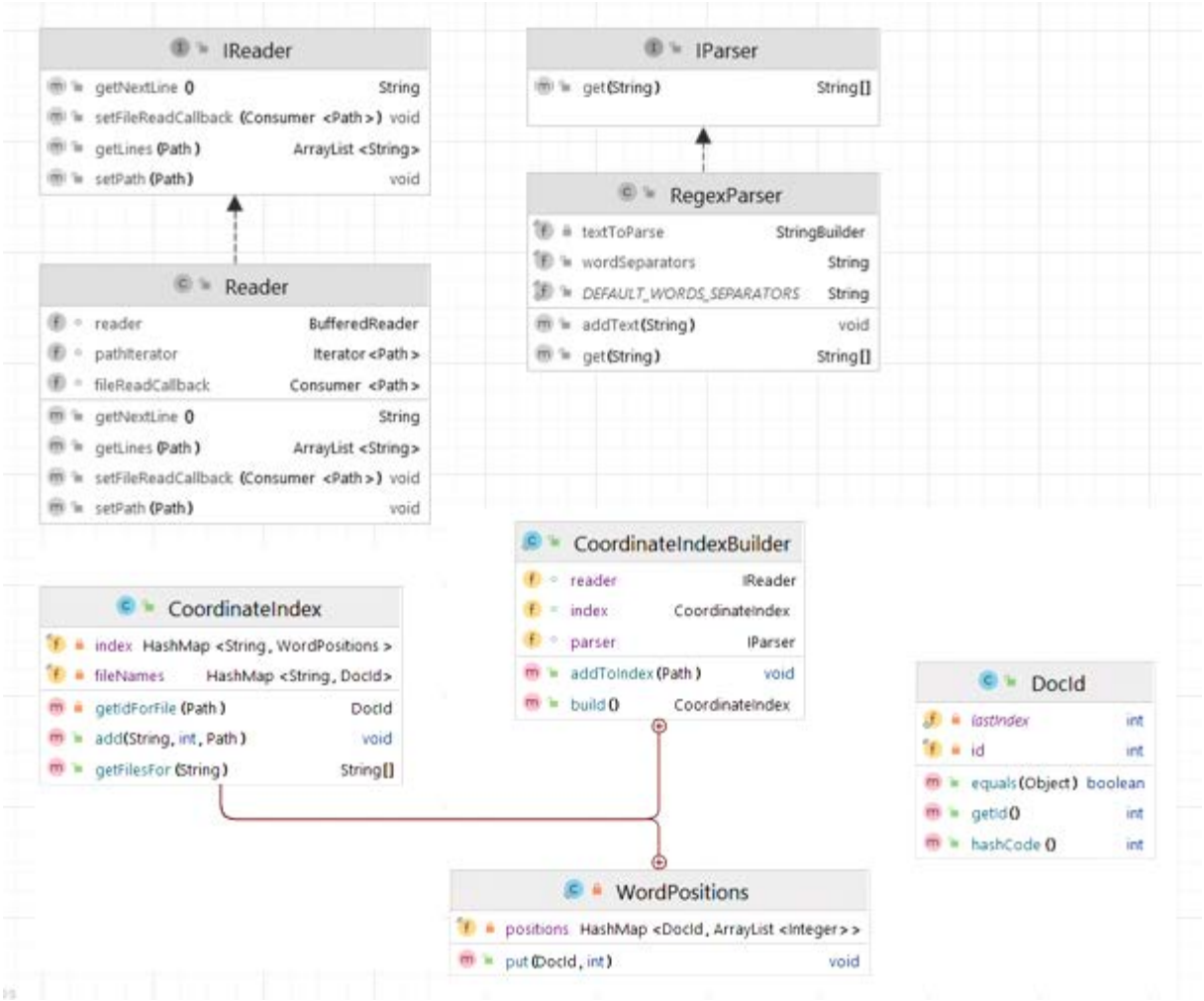




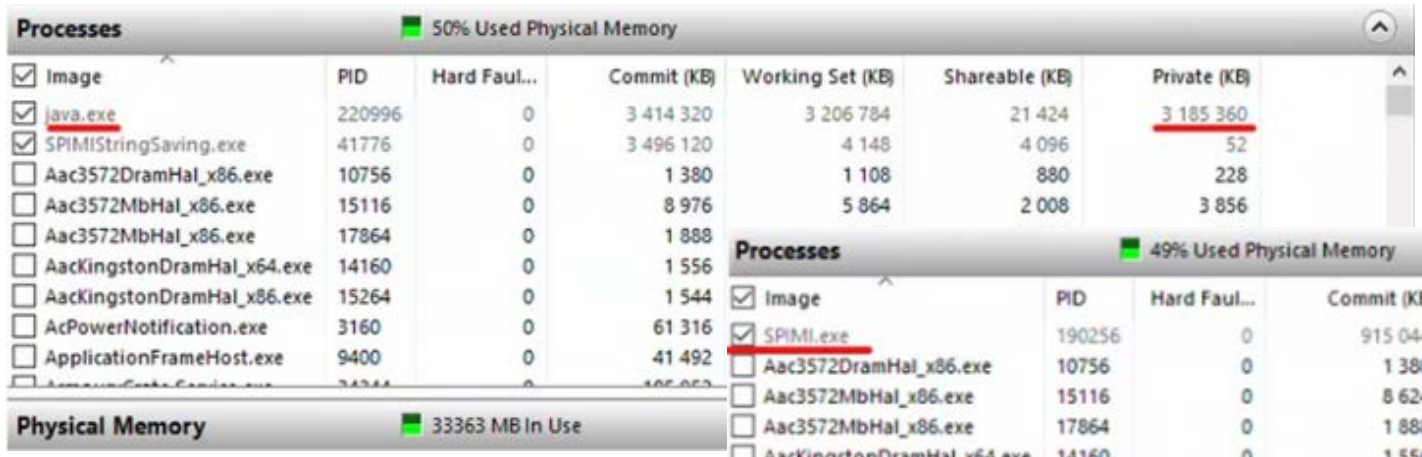


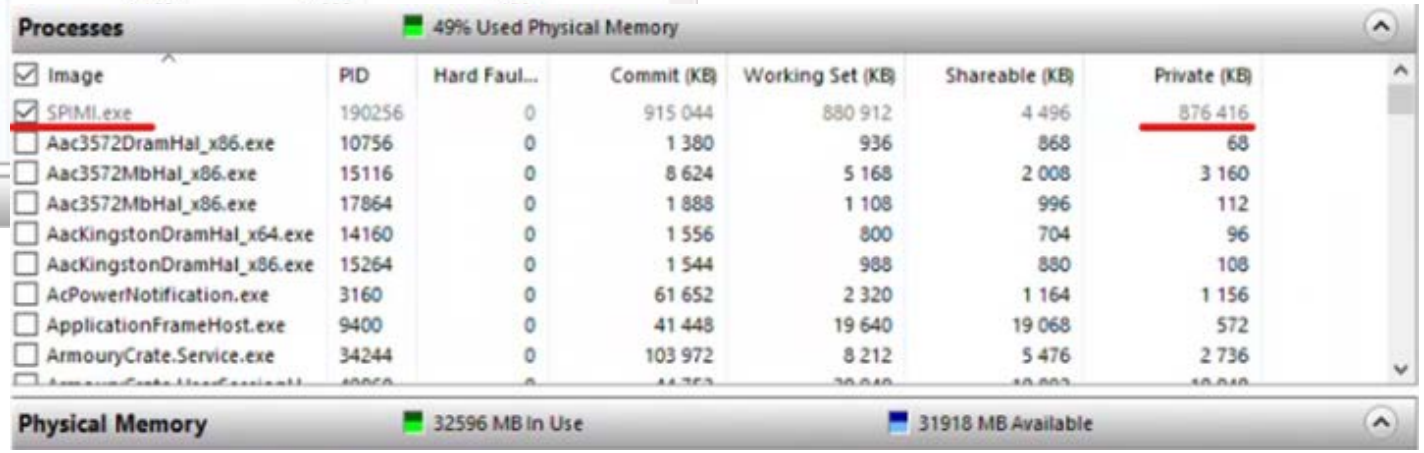
Опис Вдосконаленої Реалізації

Збереження ідентифікаторів файлів замість стрічок



Порівняння Вдосконалених Реалізацій





РЕАЛІЗАЦІЯ

Мовою C++ використовувала 876 416 Мб

Мовою Java використовувала 3 185 360 Мб

Порівняння Швидкодії Використання Регулярних Виразів

Розмір Колекції	755 Мб	235 Мб	41 Мб
STL	194 148 мс	59 764 мс	12 367 мс
Boost	27 051 мс	10 849 мс	1 078 мс

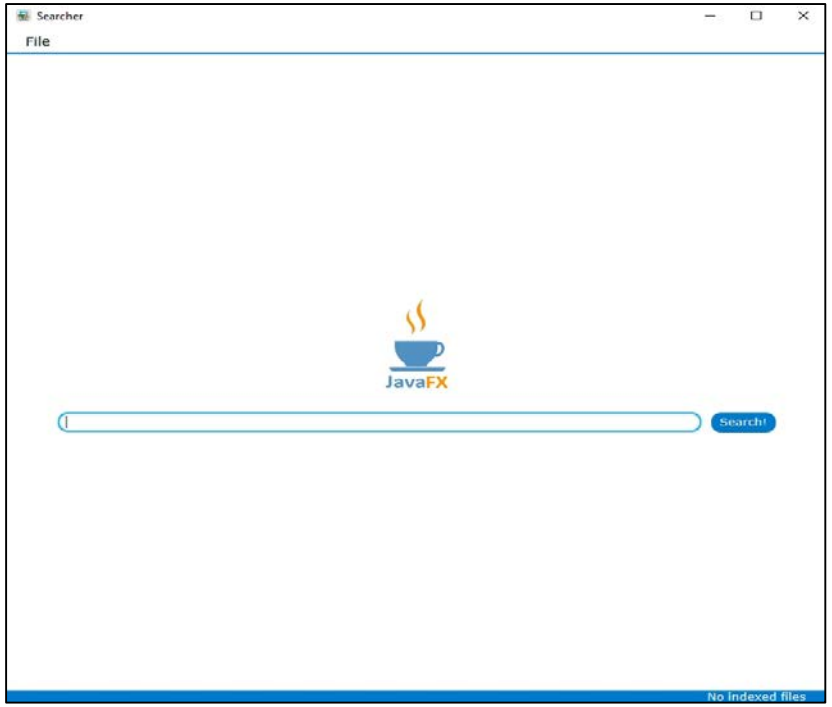
Для пришвидшення роботи програми було прийняте рішення використовувати більш ефективну реалізацію регулярних виразів бібліотеки Boost

Розширення Програми Графічним Інтерфейсом

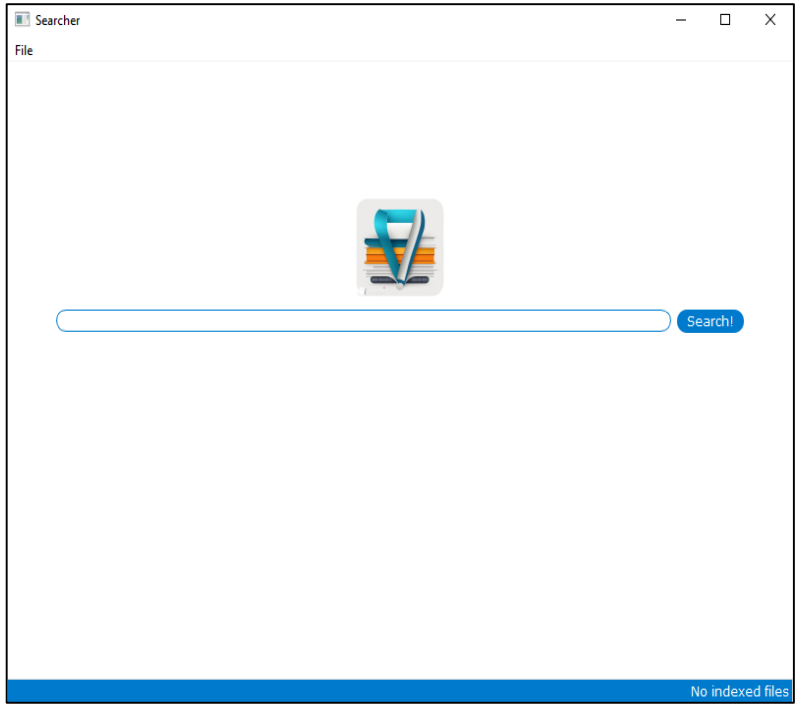
Використання графічного інтерфейсу накладає додаткові вимоги на систему з автоматичним управлінням пам’яттю

Паузи основного потоку, які виникають під час прибирання сміття, не мають бути помітними для користувача

Основні Екрани Застосунків

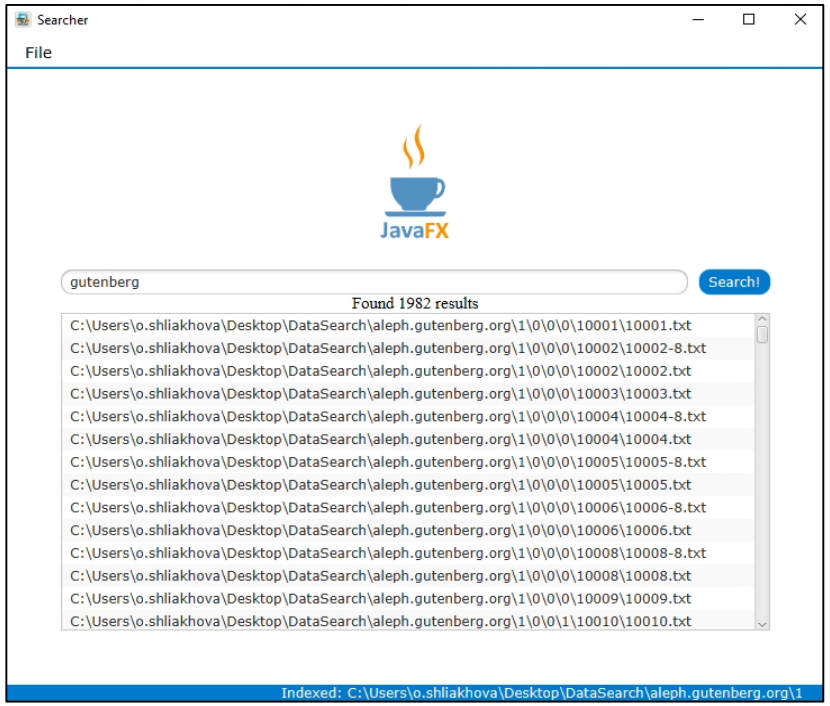


Мовою Java з використанням бібліотеки JavaFX

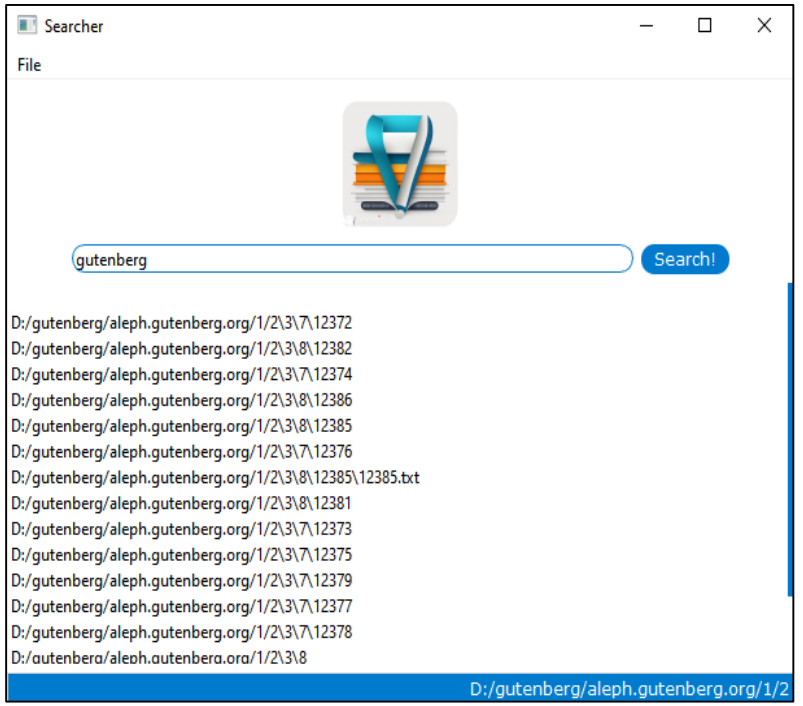


Мовою C++ з використанням бібліотеки Qt

Пошук в Створеній Колекції Файлів



Мовою Java з використанням бібліотеки JavaFX



Мовою C++ з використанням бібліотеки Qt

Порівняння роботи прибиральників сміття у реалізації мовою Java

Serial Garbage Collector

- Тривалі паузи основного потоку
- Використання лише одного потоку
- Невеликий розмір структур GC
- Доля прибиральника сміття залишається стабільною незалежно від розміру колекції

Розмір колекції	Кількість прибирань в молодому поколінні	Кількість прибирань в старому поколінні	Найдовша затримка основного потоку	Загальна затримка основного потоку через прибирання сміття	Час виконання програми
755 Мб	50	5	5095 мс	32424 мс **(48%)**	67348 мс
511 Мб	44	4	3301 мс	20044 мс **(42%)**	47584 мс
235 Мб	30	3	1632 мс	9155 мс **(42%)**	21406 мс
73 Мб	12	1	194 мс	1828 мс **(33%)**	5498 мс
41 Мб	7	1	172 мс	887 мс (29%)	3045 мс
8 Мб	1	1	132 мс	165 мс (27%)	607 мс

Розмір колекції	Розмір купи	Розмір структур GC	Розмір процесу
755 Мб	8804280 Кб	28742 Кб (0,3%)	8930031 Кб
511 Мб	5232896 Кб	17098 Кб (0,3%)	5311404 Кб
235 Мб	2962284 Кб	9722 Кб (0,3%)	3033632 Кб
73 Мб	1048640 Кб	3486 Кб (0,3%)	1150650 Кб
41 Мб	1048640	3486 Кб (0,3%)	1153857 Кб

Parallel Garbage Collector

- Найбільший об’єм пам’яті для підтримки роботи GC
- Відчутна затримка основного потоку
- Доля прибиральника сміття суттєво зменшується зі збільшенням розміру колекції

Розмір колекції	Кількість прибирань в молодому поколінні	Кількість прибирань в старому поколінні	Найдовша затримка основного потоку	Загальна затримка основного потоку через прибирання сміття	Час виконання програми
755 Мб	18	6	956 мс	5 564 мс (12%)	44135 мс
511 Мб	14	4	574 мс	2 626 мс (9%)	28204 мс
235 Мб	11	3	372 мс	1389 мс (10%)	13287 мс
73 Мб	8	1	76 мс	212 мс (5%)	3738 мс
41 Мб	6	1	37 мс	143 мс (6%)	2130 мс
8 Мб	2	1	12 мс	32 мс (6%)	487 мс

Розмір колекції	Розмір купи	Розмір структур GC	Розмір процесу
755 Мб	9938432 Кб	659387 Кб (6%)	10690986 Кб
511 Мб	8477184 Кб	653683 Кб (7%)	9222769 Кб
235 Мб	7091712 Кб	649015 Кб (8%)	7834181 Кб
73 Мб	3254272 Кб	639071 Кб (16%)	3989796 Кб
41 Мб	2071552 Кб	636759 Кб (22%)	2820015 Кб
8 Мб	1048576 Кб	634763 Кб (35%)	1799336 Кб

Garbage First Garbage Collector

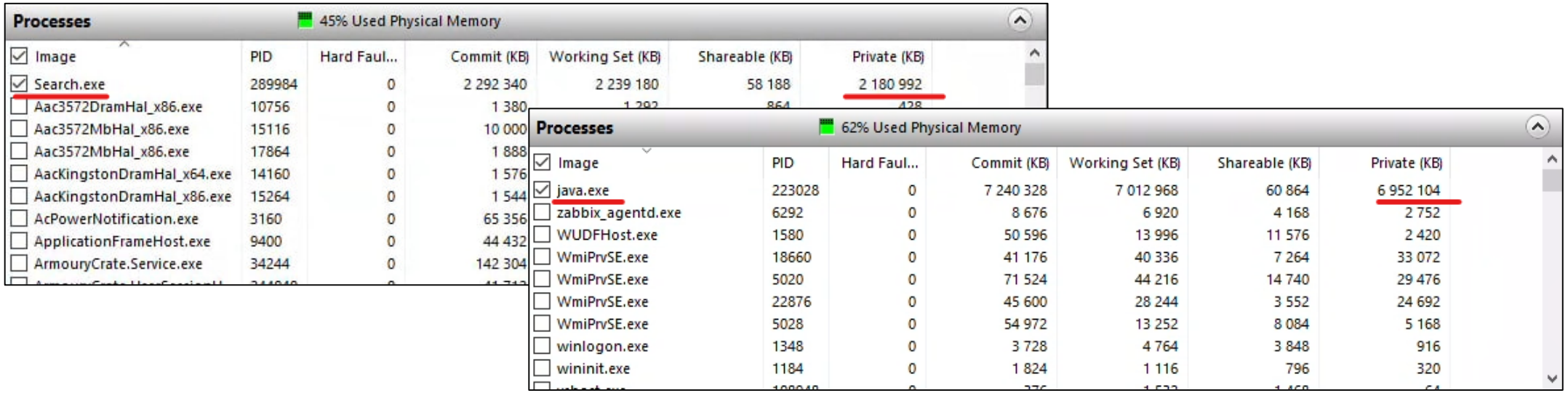
- Найкоротші паузи основного потоку
- Помірне використання пам’яті

Розмір колекції	Кількість прибирань в молодому поколінні	Кількість прибирань в старому поколінні	Найдовша затримка основного потоку	Загальна затримка основного потоку через прибирання сміття	Час виконання програми
755 Мб	41	4	746 мс	2166 мс (4%)	45380 мс
511 Мб	37	4	725 мс	1405 мс (4%)	31595 мс
235 Мб	25	1	76 мс	606 мс (4%)	13576 мс
73 Мб	16	1	31 мс	231 мс (5%)	4279 мс
41 Мб	13	1	27 мс	114 мс (4%)	2403 мс
8 Мб	8	1	8 мс	41 мс (7%)	536 мс

Розмір колекції	Розмір купи	Розмір структур GC	Розмір процесу
755 Мб	9510912 Кб	444592 Кб (4%)	10055796 Кб
511 Мб	8413184 Кб	402217 Кб (4%)	8924842 Кб
235 Мб	3629056 Кб	220636 Кб (5%)	3946648 Кб
73 Мб	1728512 Кб	148442 Кб (7%)	1983853 Кб
41 Мб	1130496 Кб	126109 Кб (9%)	1357615 Кб
8 Мб	1048576 Кб	122223 Кб(9%)	1251778 Кб

Ручне Управління Пам’яттю

- Відсутність пауз основного потоку, пов’язаних з прибиранням сміття
- Менший об’єм використаної пам’яті



Processes — 45% Used Physical Memory

Image	PID	Hard Faul...	Commit (KB)	Working Set (KB)	Shareable (KB)	Private (KB)
Search.exe	289984	0	2 292 340	2 239 180	58 188	2 180 992
Aac3572DramHal_x86.exe	10756	0	1 380	1 292	864	428
Aac3572MbHal_x86.exe	15116	0	10 000			
Aac3572MbHal_x86.exe	17864	0	1 888			
AacKingstonDramHal_x64.exe	14160	0	1 576			
AacKingstonDramHal_x86.exe	15264	0	1 544			
AcPowerNotification.exe	3160	0	65 356			
ApplicationFrameHost.exe	9400	0	44 432			
ArmouryCrate.Service.exe	34244	0	142 304			

Processes — 62% Used Physical Memory

Image	PID	Hard Faul...	Commit (KB)	Working Set (KB)	Shareable (KB)	Private (KB)
java.exe	223028	0	7 240 328	7 012 968	60 864	6 952 104
zabbix_agentd.exe	6292	0	8 676	6 920	4 168	2 752
WUDFHost.exe	1580	0	50 596	13 996	11 576	2 420
WmiPrvSE.exe	18660	0	41 176	40 336	7 264	33 072
WmiPrvSE.exe	5020	0	71 524	44 216	14 740	29 476
WmiPrvSE.exe	22876	0	45 600	28 244	3 552	24 692
WmiPrvSE.exe	5028	0	54 972	13 252	8 084	5 168
winlogon.exe	1348	0	3 728	4 764	3 848	916
wininit.exe	1184	0	1 824	1 116	796	320

Висновки

- Для застосунків з графічним інтерфейсом, написаних мовою Java, краще використовувати G1 Garbage Collector, оскільки він додає найкоротші паузи у роботу застосунку в порівнянні з Serial GC та Parallel GC

- Для реалізації частини програми, яка відповідає за розбиття тексту на слова мовою C++, краще використовувати регулярні вирази з бібліотеки Boost, ніж з STL. Різниця у часі роботи цих двох алгоритмів склала 86%.

- Накладні витрати, пов’язані з роботою віртуальної машини Java, складають 6-8%. Для задач, які оперують на купах великого розміру цими витратами можна нехтувати

- Використання мов програмування з ручним управлінням пам’яттю для розробки програм з графічним інтерфейсом має перевагу відсутності пауз основного потоку, які викликала робота прибиральника сміття в мовах з автоматичним управлінням пам’яттю

Дякую за Увагу!