DOI: https://doi.org/10.33216/2222-3428-2019-17-8

Мовшук Н. А.

РОЗРОБКА ТА АНАЛІЗ ПРОГРАМНОГО РІШЕННЯ ФАЙЛОВОГО КЕШУВАННЯ ДАНИХ В МОВІ C++ НА ОСНОВІ ТЕХНОЛОГІЇ MEMORY MAPPED FILES

В статті розглянуто метод файлового кешування даних оснований на технології memory mapped files для використання на мові програмування C++ в операційних системах сімейства Windows. Надано основний огляд технології memory mapped files. Описані переваги використання даного програмного методу в системах критичним до часу виконання та мінімізації споживання оперативної пам’яті. Проведено тестування з аналізом ефективності виконання та споживання системних ресурсів комп’ютера з наведеними графіками при виконанні ряду тестів.
Ключові слова: Кешування, файл відображення, MMF, RAM, C++.

Вступ. Темпи зростання обчислювальної потужності основних компонентів сучасних мікропроцесорних систем стрімко зростають, що дозволяє розробляти більш масштабне та складне прикладне програмне забезпечення. Незважаючи на зростаючий темп підвищення обчислювальної потужності, вузьким місцем донині залишається обмеження в оперативно запам'ятовуючого пристрою (RAM), чий розмір в середньому становить близько 4 GB.

У сучасних операційних системах, таких як Windows, додатки і деякі системні процеси завжди посилаються до пам'яті за допомогою адресів віртуальної пам'яті, кожному з яких виділено власний приватний віртуальний адресний простір. Ці адреси автоматично перетворюються обладнанням в реальні адреси RAM. Тільки базові частини ядра операційної системи пропускають це перетворення і безпосередньо використовують реальні адреси. На відміну обмеженої пам'яті RAM, віртуальна пам'ять для більшості практичних цілей не обмежена.

Віртуальна пам'ять в ОС Windows застосовується завжди, навіть якщо обсяг пам'яті необхідний всім запущеним процесам не перевищує обсяг RAM. Якщо обсяг пам'яті, що використовується всіма поточними процесами перевищує обсяг RAM, операційна система переміщує сторінки (частинами по 4 КБ) одного або декількох віртуальних адресних просторів на жорсткий диск комп'ютера. Це дозволяє звільнити RAM для інших цілей.

У будь-якій комп'ютерній системі при збільшення кількості працюючих процесів або росту навантаження обсягу роботи продуктивність зменшується, але нелінійно. Збільшення навантаження після певної точки призводить до значного зниження продуктивності роботи - що свідчить про досягнення обмеження архітектури.

Таким чином, для досягнення оптимальної спільної продуктивності між усіма працюючими програмами, споживання пам'яті RAM має бути зведено до мінімуму.

Постановка завдання. Для досягнення високої продуктивності та мінімізації споживання пам'яті розроблюваної програми без шкоди до останньої, задовільним рішенням буде використання файлового кешу для зберігання об'ємних і рідко використовуваних даних у зовнішній пам'яті такий як HDD або SSD, які значно перевершують обсяг пам'яті в порівнянні з RAM. Дане рішення значно знизить навантаження на пам'ять, що позитивно позначиться на продуктивності роботи при спільному використанні програмами операційною системою, а також надасть програмі більше вільного простору пам'яті для використання.

Дане рішення є оптимальним для мінімізації використовуваної пам'яті RAM і розвантаження ОС, проте у випадку запиту програмою даних файлу за допомогою стандартних способів приведе до значних системних затримок на виконання операції: виділення пам'яті RAM для буфера, читання/запис файлу даних, звільнення пам'яті.

Для уникнення подібної ситуації при файловому кешуванні відмінним рішенням буде використання технології файлу відображення в пам'ять (MMF) вхідний до складу ОС Windows.

MMF - це спосіб роботи з файлами на зовнішніх накопичувачах, при якому весь файл або до певної його частини представляється відповідна ділянка віртуальної пам'яті. Причому читання та запис за цією адресою призводить до читання або запису даних які знаходяться безпосередньо на диску. Також за допомогою потужних внутрішніх механізмів файл відображення можна зробити загальнодоступним для спільного звернення процесами до однієї ж ділянки пам’яті.

Крім спрощеної роботи з файлом, основною перевагою є менше навантаження на операційну систему, так як необхідний доступ до відображеного файлу в пам'яті здійснюється через диспетчер пам'яті операційної системи, який автоматично налаштовує процесор так, що сторінки RAM зберігають сусідні фрагменти файлу (як правило, 4 КБ), утворюють безперервний діапазон адрес, надаючи їх програмі по мірі необхідності.

Завдяки чому при наявності невеликої кількості оперативної пам'яті (наприклад 128 мегабайт), можна легко переглянути файл розміром 1 гігабайт або більше, не приходячи до великих накладних витрат для системи. Також виграш в продуктивності відбувається при запису з пам'яті на диск, та при необхідності оновлення великої кількості даних, що знаходяться в пам'яті, вони можуть бути одночасно (за один прохід головки HDD) записані на зовнішній накопичувач.

На рисунку 1 наочно показано як декілька процесів можуть одночасно використовувати спільне представлення одного відображеного в пам'ять файлу.

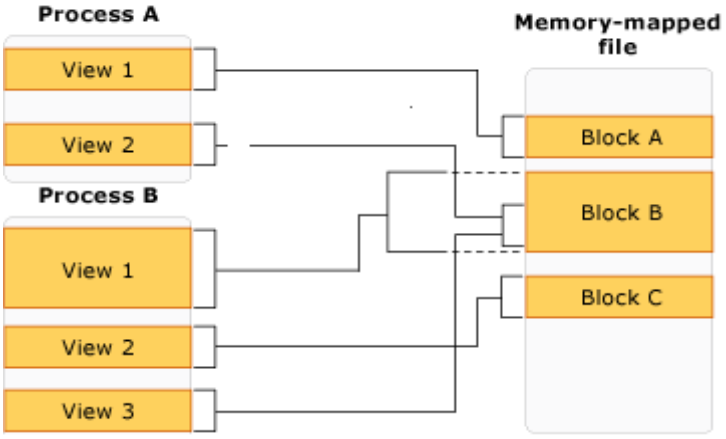

Рисунок 1 - Спільне використовування файлу відображеного в пам'яті

Мета роботи. Розробка програмного рішення для файлового кешування даних в мові C++ на основі технології MMF.

Основна частина. Розробка прикладної програми орієнтована на використання MMF в якості файлового кешування вимагає суворого контролю з боку програміста, а саме:

– управління дескриптором файлу;
– управління дескриптором файлу відображення;
– управління початковою адресою зіставлення даних файлу в пам'яті;
– динамічне управління простором пам'яті для занесення даних в різні ділянки пам'яті;
– контроль зайнятості ділянок пам'яті іншими даними;
– реєстрації звільнених ділянок пам'яті;
– зберігання довжини даних для подальшого читання та запису в ділянки пам'яті;
– контролювання доступу до вмісту при використанні в багатопотоковому режимі;
– синхронізація займаних/звільненних ділянок пам'яті в багатопотоковому режимі;
– коректна поведінка протягом всієї роботи програми.

Невиконання одного з пунктів тягне за собою максимально руйнівні дії, починаючи від аварійного завершення роботи і закінчуючи спотворенням даних. З метою забезпечення надійності, продуктивності та мінімізації накладних витрат пам'яті, автором статті було особисто розроблено програмне рішення, що забезпечує виконання всіх необхідних пунктів.

Програмне рішення розроблялося для мови програмування C++, що виконується на 32/64 розрядній архітектурі процесора в сімействі ОС Windows, з основним ухилом на швидкість виконання і повну свободу дій незалежно від можливих наслідків.

Програмна архітектура являє собою самостійний об'єкт (клас), до складу якої входять дві основні структури даних (рис 2):

1) SHeaderData-структура назви кешованих даних, використовується для зберігання лічильника блокування ділянки даних, розмірності виділення даних в байтах, розташування початкової адреси даних в кешованому файлі відображення в пам'яті.

2) SFreeSpace - структура вільних ділянок пам'яті, що зберігає кількість вільного простору певної ділянки пам'яті з вказівкою на початок вільної пам'яті в кешованої пам'яті.

SHeaderData																															
Байт	Бит																														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
4	Лічильник блокування				Розмір даних																										
8	Позиція початку даних																														

SFreeSpace																															
Байт	Бит																														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
4	Кількість вільного простору																														
8	Позиція початку вільного простору																														

Рисунок 2 - Структура об'єкта файлового кешування

Дані структури мають вирівняну межу в 8 байт, що мають однаковий розмір на 32/64 розрядних процесорах. Дане вирівнювання даних дозволяє економно використовувати пам'ять, а також оптимізувати доступ до неї на рівні процесора - обробка структури даних кратною розміру машинного слова значно прискорює обробку даних, усуваючи необхідність в додаткових операціях обробки для отримання адресованій частині даних на рівні процесора.

Лічильник відповідає за реєстрацію зайнятих ділянок пам'яті. Постійне блокування та розблокування ділянки кешованих даних позбавлена сенсу, в кращому випадку будуть даремно витрачені такти процесора, якщо лічильник ділянки кешованої пам'яті не дорівнює нулю, в іншому випадку буде викликаний внутрішній механізм розподілу пам'яті об'єкта кешування, та подальше коригування менеджера пам'яті MMF, що в кінцевому підсумку призведе до загального зниження продуктивності роботи програми.

Обмеження на розмір ділянки даних складає 268 435 455 байт, це обмеження викликане відсутністю автора в необхідність кешування особливо великих даних в якості загального об'єкта кешування, що підтверджується результатами графіків тестування, наведених далі.

Загальне обмеження файлу об'єкта кешування становить 4 294 967 295 байт, що в середньому складає понад 16 мільйонів ділянок пам'яті розміром в 268 байт кожен, при використанні близько 128 мб пам'яті RAM.

В якості змінних для зберігання структур виступають контейнери із стандартної бібліотеки мови C++ (STD), а саме unordered_map для SHeaderData і vector для SFeeSpace.

unordered_map - є асоціативним кешованим контейнером, що містить в собі пари ключ-значення з унікальними ключами. Завдяки унікальності ключів, немає необхідності у виділенні додаткової пам'яті для зберігання значень ключів при подальшому доступу до кешованих даних. Основною перевагою контейнера на ряду з унікальністю ключів, є виконання пошуку, вставки і видалення за константивний час, що позитивно позначається на швидкості роботи програми.

vector - є послідовним контейнером, основною перевагою якого є швидкість роботи за рахунок лінійно розташованих даних, виконання операцій прямого доступу, пошуку і сортування які виконуються значно простіше і швидше.

При створенні об'єкта кешування вказується ім'я створюваного файлу кешування і максимальний розмір файлу кешування, після чого для ефективного використання сторінок пам'яті виконується обчислення необхідного розміру файлу кратним гранулярності розподілу пам'яті операційної системи округлений в більшу сторону згідно формули 1.

$$Size = \frac{\text{SizeFile}+(\text{GranMem}-1)}{\text{GranMem}} * \text{GranMem} \quad (1)$$

де SizeFile - необхідний розмір файлу кешування;
FranMem - гранулярність розподілу пам'яті операційної системи;
Size - результат обчислювання розміру файлу з урахуванням гранулярності пам'яті.

Після чого об'єкт кешування бере на себе управління дескрипторами файлу, файлом відображення, адресою зіставлення початку даних, а також подальше коректне закриття їх при запиті видалення об'єкта кешування стандартними засобами C++.

Для виділення необхідної кешованої пам'яті достатньо лише задати бажаний унікальний ключ і необхідну розмірність в байтах, повернуте значення є успішністю виконання. У випадку успішного виконання виділення області пам'яті внутрішній лічильник на ділянку даних встановлюється рівним 1.

Звільнення непотрібних ділянок пам'яті відбувається коли внутрішні значення лічильника блокування стає рівним 0, при цьому вся займана пам'ять повертається в контейнер структури SFeeSpace. Необхідна перевірка на звільнення займаної пам'яті проводиться кожен раз при виклику методу розблокування пам'яті. Відсутність додаткових збирачів не використовуваної пам'яті дозволяє значно ефективно використовувати перевірки на необхідні дії тільки коли це дійсно необхідно.

Отримання прямого адресу на ділянку кешованої пам'яті здійснюється за допомогою виклику відповідної функції, яка надає покажчик на початок кешованій ділянки пам'яті тільки після перевірки унікального ключа на існування, в іншому випадку повертається значення 0.

Однією з відмінною рисою об'єкта кешування є забезпечення незмінності прямої адреси ділянки кешованих даних у файлі, без використання додаткових навантажувальних механізмів. Це дозволяє не турбуватися про отримані раніше покажчики на ділянки пам'яті при додаванні, зміні та видаленні інших ділянок кешованої пам'яті.

Підтримка безпечної багатопоточності забезпечується лише при блокуванні та розблокуванні ділянок кешованої пам'яті, в інших випадках надається вільний незахищений доступ. Наявність лічильника блокування забезпечує основну безпеку від випадкового видалення діючих ділянок даних.

Результати розробки. В якості результату розробки був виконаний певних набір тестів з аналізу продуктивності та використання пам'яті об'єктом файлом кешування, тестування виконується з характеристиками системи наведених у таблиці 1. Всі тести проводилися на компіляторі MS C/C++ Compiler 11.00.50727.1 (VS 2012) з параметром -0x, скомпільований з підтримкою C++11.

Таблиця 1

Характеристики виконуваної системи

Операційна система	Windows 7 Ultimate x64
Процесор	AMD A6-3400M APU with Radeon(tm) HD Graphics 1.40 GHz
Оперативна пам'ять	4 Gb DDR3 1333 МГц
Зовнішня пам'ять	HDD Western Digital Red 1TB 5400rpm WD10EFRX

Тест 1 - Залежність швидкості створення і закриття файлу кешування від розміру останнього (рис 3).

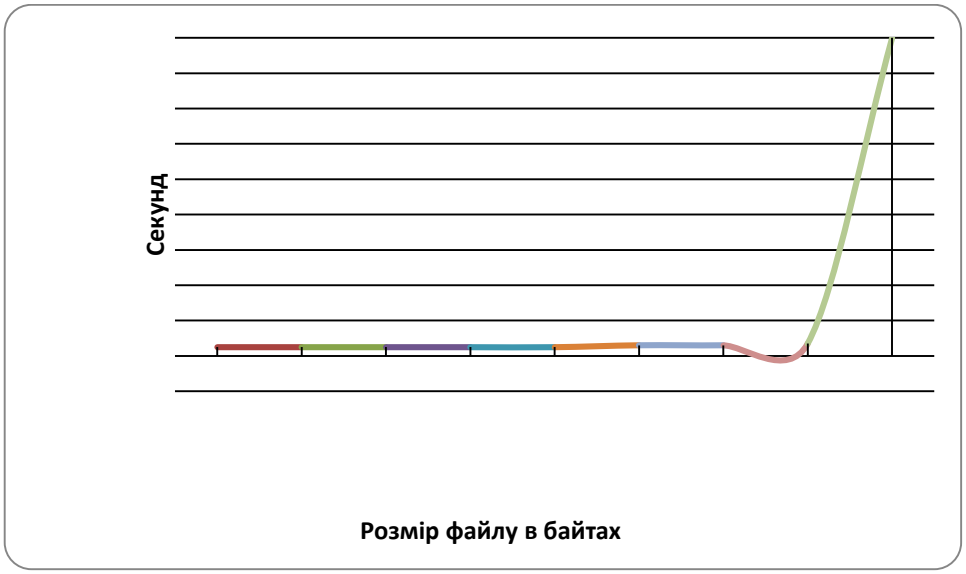

Рисунок 3 - Графік результатів виконання тесту 1

Згідно результатів графіка (рис. 3) тесту, наочно видно переваги використання MMF в якості основи механізму управління файлом кешування на зовнішньому накопичувачі, в теж час згідно з даними диспетчера завдань Windows створення файлу та утримання об'єкта кешування розміром в 268 435 456 байт обходиться додатковими 2 дескрипторами і додатковими 512 кб пам'яті, виділеної пам'яті програми.

Тест 2 - Залежність швидкості виділення і звільнення 64 ділянок пам'яті від розміру останніх (рис 4).

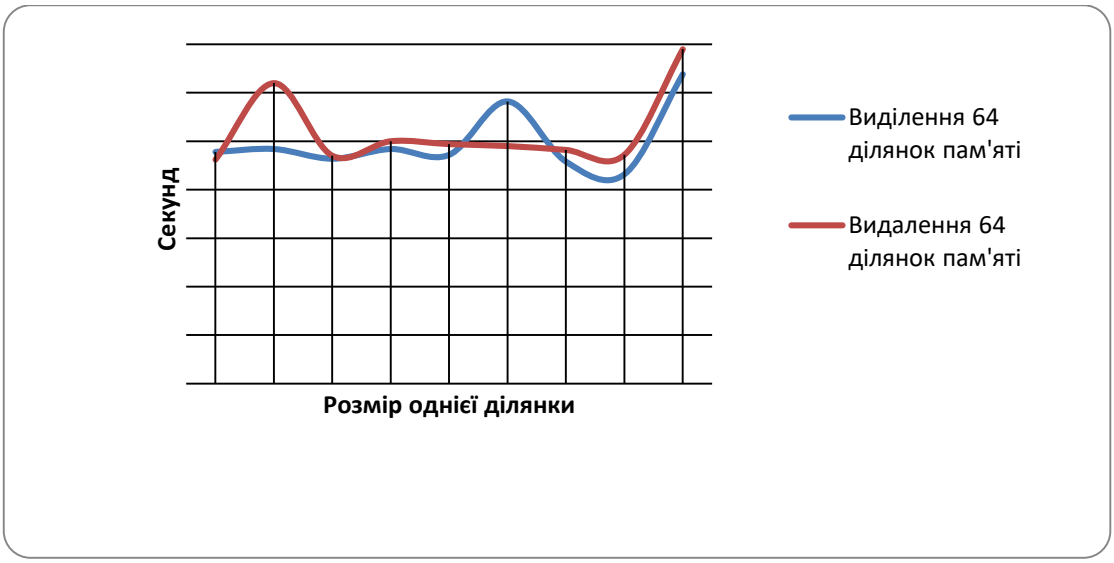

Рисунок 4 - Графік результатів виконання тесту 2

Результати графіка (рис. 4) тесту показують ефективність механізмів виділення і звільнення пам'яті, даний ефект досягається за допомогою використання структури SHeaderData в якості єдиного реєстру пам'яті, завдяки чому не відбувається звернення до зіставлення файлу в пам'яті, що не призводить до додаткових коригувань механізму MMF.

Тест 3 - Залежність швидкості запису і читання 64 ділянок пам'яті від розміру останніх (рис 5).

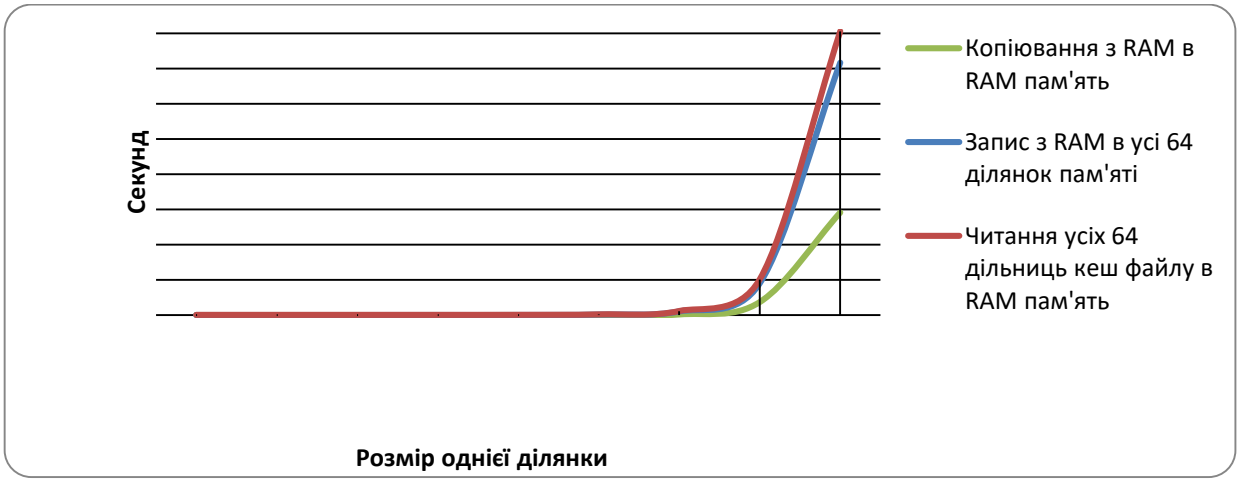

Рисунок 5 - Графік результатів виконання тесту 3

Результати графіка (рис. 5) тесту підтверджують ефективність використання невеликих ділянок кешованої пам'яті без істотного збитку продуктивності програми. Також в графіку наведено результати копіювання виключно у пам'яті RAM для наочного порівняння коефіцієнта зниження продуктивності в залежності від використовуваних розмірів даних.

Висновок. У даній статті запропоновано ефективне програмне рішення файлового кешування даних для використання на C++, що забезпечує надійність, продуктивність і мінімізації накладних витрат пам'яті. В якості основи ефективної взаємодії з зовнішнім файлом даних використовується технологія MMF яка входить до складу OS Windows. Технологія MMF має надійні і високопродуктивні механізми управління зовнішніх файлів, що неодноразово підтверджується графіками в процесі тестування запропонованого програмного рішення. Як актуальність та перспектива подальшого розвитку даний програмний метод буде використовуватися автором при розробці ігрового движка, що вимагає від останнього максимальну продуктивність і мінімізацію споживання пам'яті, що і є основною метою даного програмного рішення.

Література

1. Мельник Д., Курмангалеев Ш., Аветисян А., Белеванцев А., Плотников Д., Варданян Мамикон. Оптимизация приложений для заданных статических компиляторов и целевых архитектур: методы и инструменты // Журнал «Труды Института системного программирования РАН». - 2014;
2. Memory mapped files [Електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/dotnet/standard/io/memory-mapped-files;
3. Memory mapped files [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Memory-mapped_file;
4. Alignment (C++ Declarations) [Електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/cpp/cpp/alignment-cpp-declarations?view=vs-2019.

References

1. Melnyk D., Kurmanhaleev Sh., Avetysian A., Belevantsev A., Plotnykov D., Vardanian Mamykon. Optymyzatsyia prylozhenyi dlia zadannыkh statycheskykh kompyliatorov y tselevыkh arkhytektur: metodы y ynstrumentы // Zhurnal «Trudы Ynstytuta systemnoho prohrammyrovanyia RAN». - 2014;
2. Memory mapped files [Electronic resource]. Rezhym dostupu: https://docs.microsoft.com/en-us/dotnet/standard/io/memory-mapped-files;
3. Memory mapped files [Electronic resource]. Rezhym dostupu: https://en.wikipedia.org/wiki/Memory-mapped_file;
4. Alignment (C++ Declarations) [Electronic resource]. Rezhym dostupu: https://docs.microsoft.com/en-us/cpp/cpp/alignment-cpp-declarations?view=vs-2019.

В статье рассмотрен метод файлового кэширования данных основанный на технологии memory mapped files для использования на языке программирования C++ в операционных системах семейства Windows. Предоставлен основной обзор технологии memory mapped files. Описаны преимущества использования данного программного метода в системах требовательных ко времени выполнения и минимизации потребления оперативной памяти. Проведено тестирование с анализом эффективности выполнения и потребление системных ресурсов компьютера с приведенными графиками при выполнении ряда тестов.
Ключевые слова: Кэширование, файл отображения, MMF, RAM, C++.

The article considers the method of file caching of data based on memory mapped files technology for use in the C++ programming language in Windows operating systems. A basic overview of memory mapped files technology is provided. The advantages of using this software method in time-critical systems and minimizing RAM consumption are described. Testing with the analysis of efficiency of performance and consumption of system resources of the computer with the resulted schedules at performance of a number of tests is carried out.
Keywords: Caching, mapping file, MMF, RAM, C++.

Відомості про авторів:

Мовшук Н.А. - магистр напряму підготовки «Комп'ютерна інженерія», студент факультету інформаційних технологій та електроніки, Східноукраїнський національний університет імені В. Даля.
E-mail: LazyDemonFFF@gmail.com