Table of Contents

[Introduction 2](#_Toc158300790)

[Theory 2](#_Toc158300791)

[Results 3](#_Toc158300792)

[Analysis 4](#_Toc158300793)

# Introduction

In this assignment, I will deepen my understanding of memory management in modern operating systems. I'll be examining the critical roles of memory and process management and their interconnectedness. My task is to develop a C++ program that allocates memory incrementally until it can no longer do so, revealing the limits of memory under various system configurations.

Using Visual Studio, I'll explore how factors like the system's address space and the use of virtual memory influence a process's capabilities. The practical component involves running tests across different target platforms and configurations, including the use of Large Address Aware settings. I will document the memory allocation limits and analyze system performance data to gain insights into the efficient use of system resources.

# Theory

In the realm of computer architecture and memory management, 32-bit processors can transfer data up to 32 bits per clock cycle, while their 64-bit counterparts expand the potential by utilizing 16 general-purpose registers and harnessing a vastly larger memory space, up to 16 exabytes, by exploiting a 64-bit address space. This address space encapsulates both physical and virtual addresses, allowing processors to access a broad spectrum of data and execute a greater number of calculations per second, which enhances task completion speeds. Multi-core configurations in 64-bit processors further augment computational power for home computing.

Memory management in software is pivotal; a memory leak signifies the failure of a program to release memory no longer in use, often due to unreachable or incorrectly managed memory allocations. Memory compression, as implemented in Windows 10, Linux, and macOS, mitigates the limitations of physical memory by compressing data, allowing more information to be stored in RAM than its uncompressed capacity would typically permit, thus reducing reliance on slower page file storage.

The concept of 'committed' memory in operating systems refers to virtual memory allocated for processes, a portion of which may reside in physical RAM while the rest is stored in the page file. Virtual memory enables systems to use more memory than the available physical RAM by swapping rarely used data to disk storage and retrieving it when needed.

The allocation of memory in systems typically occurs in 4 KB pages. When RAM is full, the operating system may swap less frequently accessed pages to the page file to free up RAM for other processes. Should the swapped-out page be required again, it is retrieved from the disk into a free RAM page.

A 'bad allocation' error in programming denotes a failure in memory allocation, typically due to insufficient available memory, indicating that the system is unable to fulfill a request for memory allocation, which is essential for program execution and data processing.

Results

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Unit | Test bit | LAA | Task manager (GB) committed | Visual studio | Terminal (GB) | sec | Note |
| GiB | 64 | ON | 60.0/61.3 | 54 | 50 | 32.894 | Bad allocation |
| GiB | 32 | ON | 17.1/19.8 | 2 | 2 | 0.843 | Bad allocation |
| GiB | 64 | OFF | 11.2/19.1 | 1 | 1 | 0.045 | Bad allocation |
| GiB | 32 | OFF | 15.3/19.8 | 1 | 1 | 0.082 | Bad alloction |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Unit | Test bit | LAA | Task manager (GB) committed | Visual studio  (GB) | Terminal | sec | Note |
| KiB | 64 | ON | 61.4/61.4 | 60 | 35.00390625 | 68 | Crashes /computer off |
| KiB | 32 | ON | 17.8/19.8 | 4 | 3.650390625 | 27 | Bad allocation |
| KiB | 64 | OFF | 12.2/19.1 | 2.2 | 1.810566902 | 6 | Bad allocation |
| KiB | 32 | OFF | 17.8/19.8 | 2 | 1.81640625 | 2.998 | Bad allocation |

# Analysis

The discussion on memory allocation in Windows operating systems sheds light on several crucial aspects of how memory is managed, reported, and utilized across different platforms and tools. Notably, discrepancies in memory reporting among tools like **Task Manager, Visual Studio, and program-specific printouts highlight the complexity of accurately gauging memory usage**. These discrepancies arise from the differing approaches to including or excluding virtual memory in reported values and the methodologies for memory allocation. **Task Manager focuses solely on physical memory, excluding virtual memory**, whereas **Visual Studio provides a more comprehensive view by considering both.**

The role of Large Address Aware (**LAA**) technology emerges as a pivotal factor in **extending the memory usage capabilities** of **32-bit** applications beyond the conventional **2GB** limit, allowing access up to **4GB** on 32-bit systems and **significantly more on 64-bit** systems. This capability not only **influences the maximum amount of memory** that applications can allocate but also leads to **variations in memory composition during runtime**. The impact of LAA is further illustrated through examples that show how memory allocation outcomes can vary under different conditions, such as with LAA enabled or disabled and across 32-bit versus 64-bit systems.

The text delves into both the theoretical and practical limits of memory allocation, noting that while 32-bit systems have a **theoretical limit of 4GB**, **actual** usable memory may be **less due to various system constraints**. Conversely, **64-bit** systems can potentially allocate the system's maximum memory capacity, influenced by the specific Windows edition and system specifications. The use of LAA consistently leads to increased memory allocation by enabling processes to access a larger pool of memory addresses, **which can significantly reduce the amount of free memory available**.

**Virtual** memory is highlighted as a critical technique for managing **physical** memory **shortages**, enabling the **swapping** of data between **RAM** and **disk** storage to allow more applications to run than would be possible with just physical memory. This swapping mechanism enhances computer utilization and efficiency, albeit with potential performance impacts. Virtual memory's role is crucial in **expanding** the effective memory space available to applications through hardware and software coordination.

Furthermore, the discussion touches on **memory leaks** and system resource management, suggesting setting **limits on memory usage** by processes as a strategy to mitigate leaks. This approach allows for system intervention when predefined thresholds are reached, demonstrating the complexities of memory management in ensuring system stability and performance.

The distinction between application and system memory is pivotal. System memory covers the operating system and kernel operations, while application memory pertains to individual application allocations. This differentiation is key to understanding diverse memory usage patterns across applications, influenced by their design and interaction with system resources. For instance, background services versus interactive applications demonstrate varied impacts on system performance due to their distinct memory demands.

Fragmentation affects memory allocation efficiency, splitting free memory into small, scattered blocks, complicating the allocation of large memory segments. This phenomenon, occurring in both physical and virtual memory realms, can lead to performance degradation due to increased paging. Strategies like memory compaction and fragmentation-minimizing allocation algorithms are employed to mitigate its impact, aiming to optimize memory usage and application performance. [2]

Address spaces, delineating the memory range accessible by a process, are fundamental in differentiating between physical and virtual memory. The 32-bit system limit of 4GB, traditionally split between kernel and user space, contrasts sharply with the expanded capacities in 64-bit systems. This expansion, facilitated by technologies like Large Address Aware, not only eases memory constraints but also influences application development by necessitating considerations for larger memory access patterns and data structure designs. [9]

Overall, the exploration of memory management across Windows platforms underscores the importance of technologies like **LAA**, the challenges and **solutions associated with memory allocation**, and the critical **role of virtual memory** in optimizing system performance and application functionality.

**References:**

**[1]** M. Goralewicz, "x64 vs. x86 - Which Architecture is Better & Why?" phoenixNAP, 2020. [Online]. Available: <https://phoenixnap.com/kb/x64-vs-x86> [Accessed: 17- Mar- 2024].

**[2]** M. Staimer, "What is Address Space?" TechTarget, July 2020. [Online]. Available: <https://www.techtarget.com/searchstorage/definition/address-space> [Accessed: 17- Mar- 2024].

**[3] "**A Comparison of Parallel Data Transmission and Serial Data Transmission," VERSITRON, 28-May-2020. [On-line]. Available: <https://www.versitron.com/blog/a-comparison-of-parallel-data-transmission-and-serial-data-transmission> [Accessed: 17- Mar- 2024].

**[4]** "Difference between 32-bit and 64-bit operating systems," GeeksforGeeks, 27-Dec-2019. [On-line]. Available: <https://www.geeksforgeeks.org/difference-32-bit-64-bit-operating-systems/> [Accessed: 17- Mar- 2024].

**[5]** "Memory leak," Wikipedia, 2024. [Online]. Available: <https://en.wikipedia.org/wiki/Memory_leak> [Accessed: 17- Mar- 2024].

**[6]** C. Hoffman, "What Is Memory Compression in Windows 10?" How-To Geek, Sept. 2017. [Online]. Available: <https://www.howtogeek.com/319933/what-is-memory-compression-in-windows-10/> [Accessed: 17- Mar- 2024].

**[7]** "Why is my committed memory so much higher than my actual RAM space?" SuperUser, 2015. [Online]. Available: <https://superuser.com/questions/971967/why-is-my-committed-memory-so-much-higher-than-my-actual-ram-space> [Accessed: 17- Mar- 2024].

**[8]** "Memory Limits for Windows Releases," Microsoft, [Online]. Available: <https://learn.microsoft.com/en-us/windows/win32/memory/memory-limits-for-windows-releases> [Accessed: 17-Mar-2024].

[9] H. Nasr, "Memory Fragmentation," Medium, 22-Oct-2018. [Online]. Available: <https://medium.com/@hnasr/memory-fragmentation-945b88b143c8> [Accessed: 17-Mar-2024].