Project Title

Dynamic Memory Allocator .NET Desktop Application

Submitted by

Ayesha Islam Qadri (2020-CE-36)

Fatima Sohail Shaukat (2020-CE-37)

Ayesha Shafique (2020-CE-39)

Submitted to

Ma'am Darakhshan Abdul Ghaffar

Course

CMPE-331L Operating Systems

Semester

5th

Date

22nd December 2022

Department of Computer of Engineering

University of Engineering and Technology, Lahore

Table of Contents

Problem Statement

Fixed memory allocation is therefore defined as the system of dividing memory into non-overlapping sizes that are fixed, unmovable, and static. A process may be loaded into a partition of equal or greater size and is confined to its allocated partition. If we have comparatively small processes with respect to the fixed partition sizes, this poses a big problem. [1] This results in occupying all partitions with lots of unoccupied space left. Within the fixed partition context, this is known as internal fragmentation. An alternate solution to address these problems is dynamic memory allocation. Still, dynamic memory allocation can cause internal fragmentation. There will be external fragmentation despite the absence of internal fragmentation. So, our main concern is to remove internal and external fragmentation by staying under the umbrella of contiguous memory management techniques.

Abstract

Besides the responsibility of managing processes, the operating system must efficiently manage the primary memory of the computer. The part of the operating system which handles this responsibility is called the memory manager. Since every process must have some amount of primary memory to execute, the performance of the memory manager is crucial to the performance of the entire system. The memory manager is responsible for allocating primary memory to processes and for assisting the programmer in loading and storing the contents of the primary memory. Managing the sharing of primary memory and minimizing memory access time are the basic goals of the memory manager. The real challenge of efficiently managing memory is seen in the case of a system that has multiple processes running at the same time. [2]

Objective

Memory Management Techniques are basic techniques that are used in managing the memory in the operating system. Memory Management Techniques are classified into two categories; Contiguous and Non-contiguous. The contiguous technique is further divided into fixed and dynamic memory allocation. In our project, our main focus is dynamic memory allocation and handling of external fragmentation. For this purpose, we will implement First fit, Best fit, and Worst fit. To overcome the problem of external fragmentation, compaction technique or non-contiguous memory management techniques are used. We will implement the compaction technique as our coalescing policy. Compaction here means moving all the processes toward the top or toward the bottom to make free available memory in a single continuous place. In a nutshell, we will make a desktop application for better visualization of dynamic memory allocation.

Scope

Dynamic Memory Allocation is a part of the Contiguous allocation technique. It is used to alleviate the problem faced by Fixed Memory allocation. In contrast with fixed memory allocation, partitions are not made before the execution or during system configuration. Initially, RAM is empty, and partitions are made during the run-time according to the process's need instead of partitioning during system configuration. The size of the partition will be equal to the incoming process. The partition size varies according to the need of the process so that internal fragmentation can be avoided to ensure efficient utilization of RAM. [3]

Features

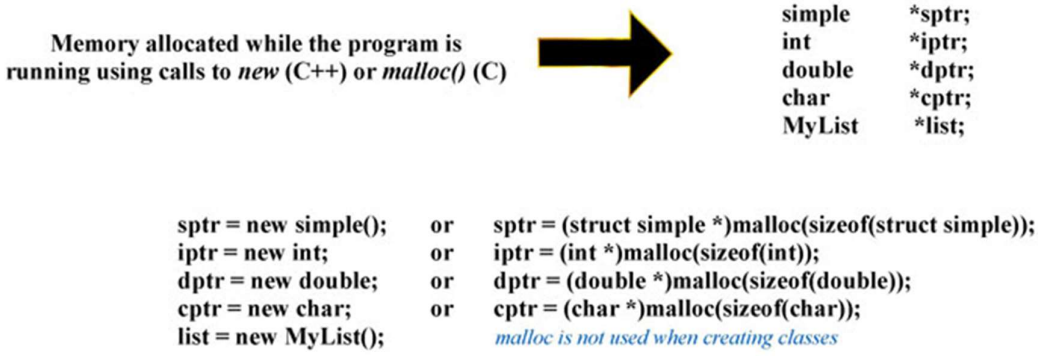
Our desktop-based dynamic memory allocation application has the following features.

- Users can add holes and specify their sizes and starting addresses in memory.
- Users can add and remove processes.
- Users can select a memory allocation technique for each process allocation in memory.
- Users can visualize the memory on each update and deletion.
- Users can handle external fragmentation using coalescing policy.
- All exceptions will be handled in a good manner.
- Users can specify Memory Size.

Explanation

Dynamic Memory Allocation

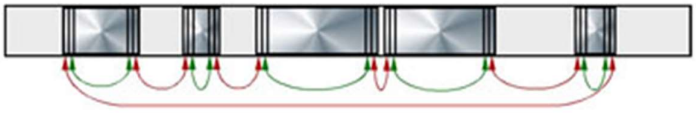
The task of fulfilling an allocation request consists of locating a block of unused memory of sufficient size. Memory requests are satisfied by allocating portions from a large pool of memory called the heap or free store. At any given time, some parts of the heap are in use, while some are "free" (unused) and thus available for future allocations. Several issues complicate the implementation, such as external fragmentation, which arises when there are many small gaps between allocated memory blocks, which invalidate their use for an allocation request. The allocator's metadata can also inflate the size of (individually) small allocations. This is often managed by chunking. The memory management system must track outstanding allocations to ensure that they do not overlap and that no memory is ever "lost" as a memory leak. The lowest average instruction path length required to allocate a single memory slot was 52 (as measured with an instruction level profiler on a variety of software) [4]



Heap

The **Heap** is that portion of computer memory, allocated to a running application, where memory can be allocated for variables, class instances, etc. From a program's heap the OS allocates memory for dynamic use. Given a pointer to any one of the allocated blocks the OS can search in either direction to locate a block big enough to fill a dynamic memory allocation request.

Blocks of memory allocated on the heap are actually a special type of data structure consisting of a pointer to the end of the previous block, a pointer to the end of this block, the allocated block of memory which can vary in size depending on its use, a pointer to the beginning of this block, and a pointer to the next block.



The Heap Layout.

Stack Memory

Stack memory is a region of memory that is allocated to a process in contiguous blocks within RAM. It also serves as a LIFO (last-in-first-out) data or instruction buffer. So, if a variable is the last element on the stack, it will be the first to be removed when memory is deallocated. Local variables, functions, and pointer variables are examples of data stored in the stack.

Fragmentation

Fragmentation Memory fragmentation occurs when a system contains memory that is technically free but that the computer can't utilize. The memory allocator, which assigns needed memory to various tasks, divides and allocates memory blocks as they are required by programs; when data is deleted, more memory blocks are freed up in the system and added back to the pool of available memory. When the allocator's actions or the restoration of previously occupied memory segments leads to blocks or even bytes of memory that are too small or too isolated to be used by the memory pool, fragmentation has occurred. Fragmentation can take a significant bite out of a computer's free memory, and it is often the cause of frustrating out-of-memory error messages. Internal Fragmentation (IF) occurs when the memory allocator leaves extra space empty inside of a block of memory that has been allocated for a client. This usually happens because the processor's design stipulates that memory must be cut into blocks of certain sizes.

For example, blocks may be required to evenly be divided by four, eight or 16 bytes. When this occurs, a client that needs 57 bytes of memory, for example, may be allocated a block that contains 60 bytes, or even 64. The extra bytes that the client doesn't need go to waste, and over time these tiny chunks of unused memory can build up and create large quantities of memory that can't be put to use by the allocator. Because all of these useless bytes are inside larger memory blocks, the fragmentation is considered internal.

External Fragmentation (EF) happens when the memory allocator leaves sections of unused memory blocks between portions of allocated memory. For example, if several memory blocks are allocated in a continuous line but one of the middle blocks in the line is freed (perhaps because the process that was using that block of memory stopped running), the free block is fragmented. The block is still available for use by the allocator later if there's a need for memory that fits in that block, but the block is now unusable for larger memory needs. It cannot be lumped back in with the total free memory available to the system, as total memory must be contiguous for it to

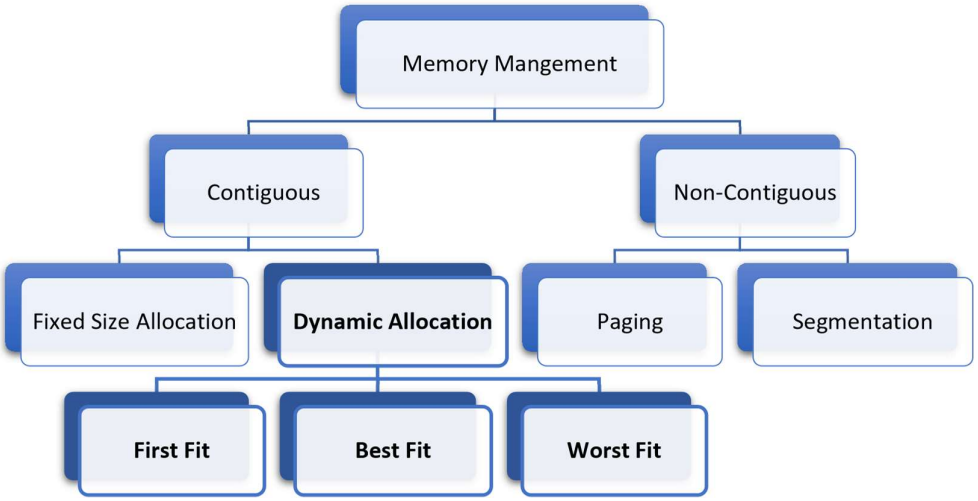
be useable for larger tasks. In this way, entire sections of free memory can end up isolated from the whole that are often too small for significant use, which creates an overall reduction of free memory that over time can lead to a lack of available memory for key tasks. Fragmentation can become an issue because it builds up over time, creating small and useless blocks of memory and limiting the amount of a computer's available free memory. As it progresses, fragmentation can cause system performance to become slow and sluggish in the short term; in the long term, fragmentation can shorten the life of a computer or server by 30 percent on average. Of the two types of fragmentation, internal is more predictable than external because the amount of wasted space is determined by the memory allocator's parameters (how big the allocated blocks must be), which is a constant. In addition, the amount of overall memory lost to internal fragmentation is usually less than what's lost to external fragmentation, although it can gradually accumulate. External fragmentation, on the other hand, is harder to predict because in most cases several processes are regularly starting and stopping in the system and blocks of memory that are used for varying lengths of time are freed up in a different order than they were filled, leaving gaps in the available memory [5]

Segmentation

The segmentation method is very similar to paging. The only distinction between the two is that segments are variable in length, whereas pages in the paging method are always fixed in size.

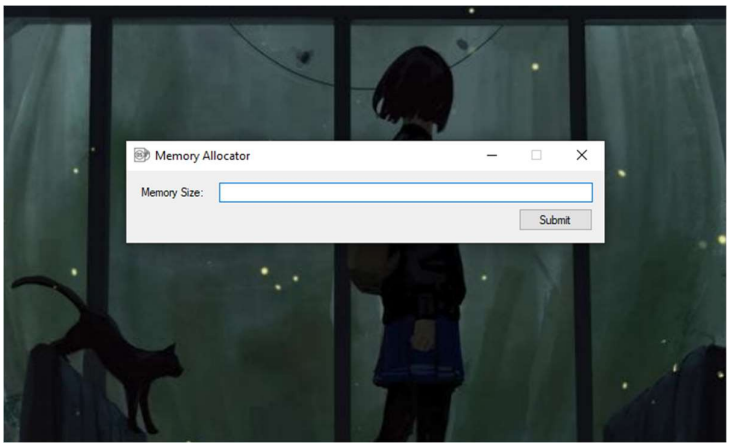
A program segment includes the main function of the program, data structures, utility functions, and so on. For each process, the operating system keeps a segment map table. It also contains a list of free memory blocks, as well as their size, segment numbers, and memory locations in main memory or virtual memory.

Flowchart

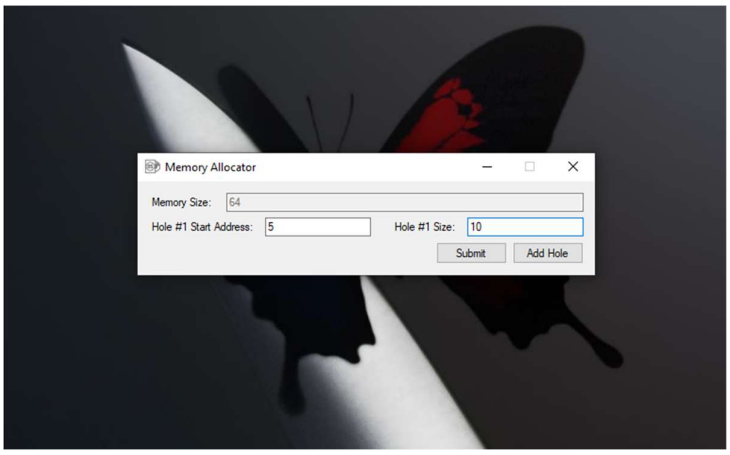


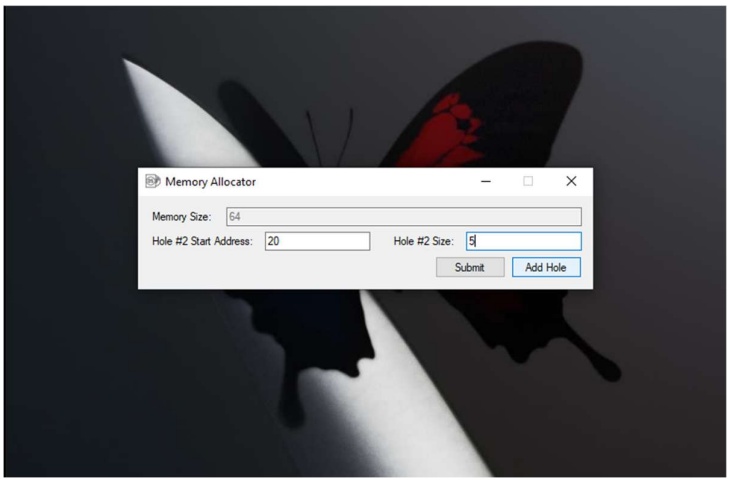
Graphical User Interface (GUI)

This is a GUI based Dynamic Memory Allocator. We run the program and set the memory size as we desire.



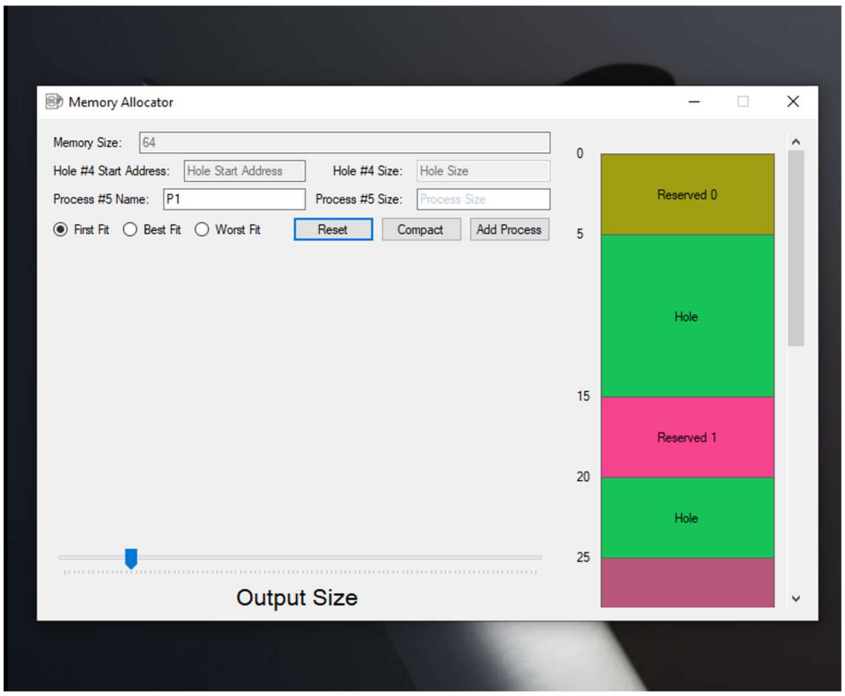
After we have entered the desired memory size, add as many holes as you like.



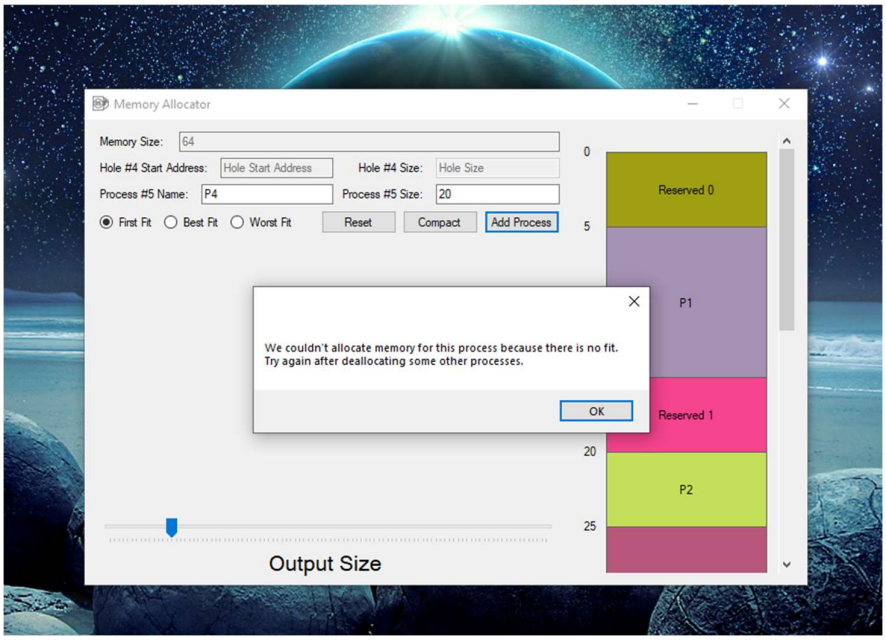


Here we have added 3 holes.

Now you may choose to add as many processes as you like. The size of each process is up to you. We will keep adding processes one by one, until we can't add any more.



We have added four processes and when we tried to add another process, it gives an error message saying that there is no fit.We have to deallocate some memory for the new process to fit. We may choose to compact the memory, and see if we find any space for our new process. Otherwise, simply click on any process you want to deallocate.



Another amazing feature we have implemented here is that we may choose which function to use to allocate memory.

For both fixed and dynamic memory allocation schemes, the operating system must keep a list of each memory location noting which are free and which are busy. Then as new processes come into the system, the free partitions must be allocated. These partitions may be allocated in 3 ways:

1. First-Fit Memory Allocation

This algorithm searches along the list looking for the first segment that is large enough to accommodate the process. The segment is then split into a hole and a process. This method is fast as the first available hole that is large enough to accommodate the process is used.

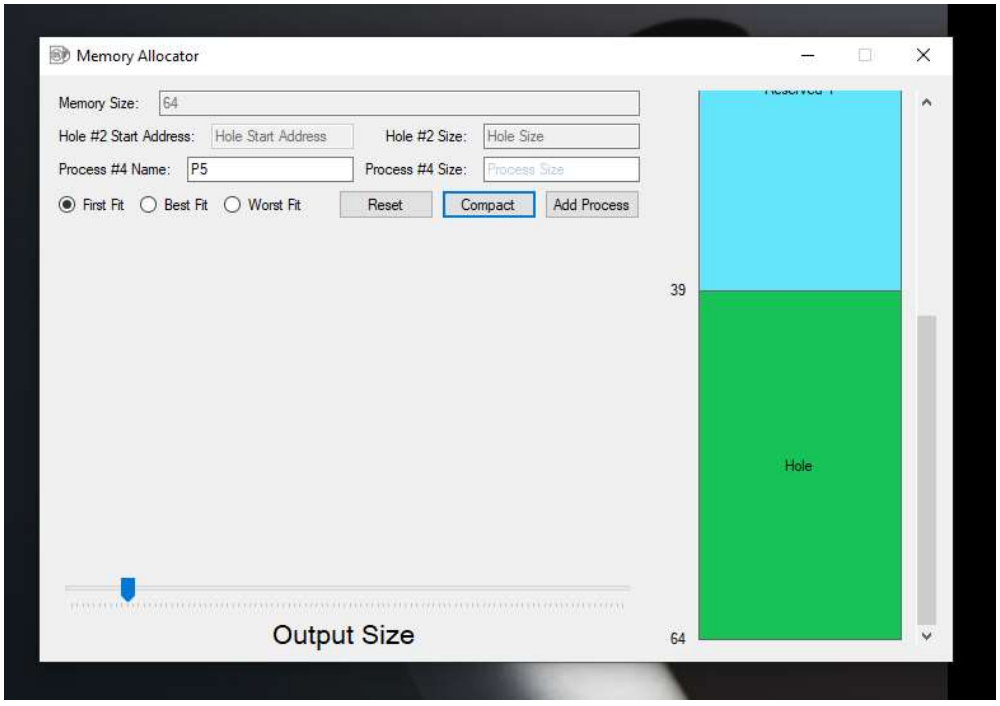
2. Best-Fit Memory Allocation

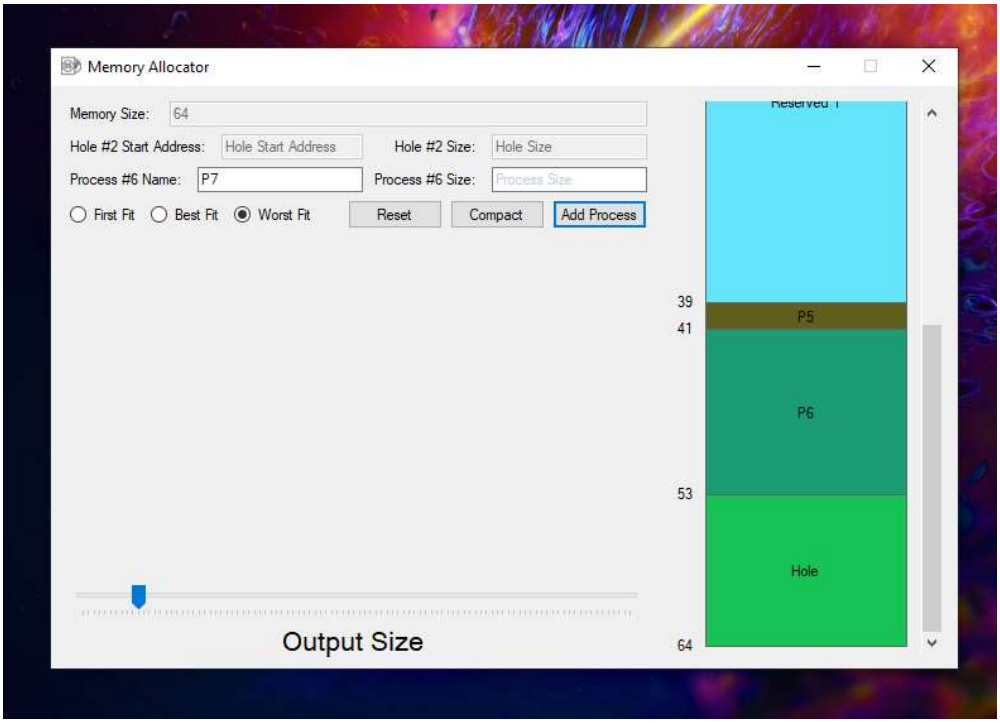
Best fit searches the entire list and uses the smallest hole that is large enough to accommodate the process. The idea is that it is better not to split up a larger hole that might be needed later. The best fit is slower than the first fit as it must search the entire list every time. It has also been shown that the best fit performs worse than the first fit as it tends to leave lots of small gaps.

3. Worst-Fit Memory Allocation

As the best fit leaves many small, useless holes it might be a good idea to always use the largest hole available. The idea is that splitting a large hole into two will leave a large enough hole to be useful. It has been shown that this algorithm is not very good either.

We have added all the process using Best Fit. Let us deallocate some memory and add a new process using worst fit.





Languages and Framework

We're going to implement our project in .NET Framework 4.6 using C#. All implementation is done using Visual Studio 2022. [6]

The original implementation of .NET is the **.NET Framework**. It allows you to run websites, services, desktop apps, and other applications on Windows. .NET is a cross-platform programming language that can run websites, services, and console apps on Windows, Linux, and macOS. .NET is available for free on GitHub. .NET was previously known as .NET Core.

C# is an object-oriented and component-oriented programming language. C# includes language constructs that directly support these concepts, transforming it into a natural language for developing and deploying software components. Since its inception, C# has evolved to support new workloads and emerging software design practises. At its core, C# is an object-oriented programming language. You define the types and their behaviour. C# emphasizes versioning to ensure that programs and libraries can evolve in a compatible way over time. Versioning considerations directly influenced the separate virtual and override modifiers, the rules for method overload resolution, and support for explicit interface member declarations in C#.

Dotnet API

We will use the following .NET APIs in our project:

- System Namespace
- System.Drawing Namespace
- System.Windows.Forms Namespace
- System.Collections.Generic Namespace [7]

Memory Management Uses

- It allows you to check how much memory needs to be allocated to processes that decide which processor should get memory at what time.
- Tracks whenever inventory gets freed or unallocated. According to it will update the status.
- It allocates the space to application routines.
- It also makes sure that these applications do not interfere with each other.
- Helps protect different processes from each other
- It places the programs in memory so that memory is utilized to its full extent.

References

[1] https://study.com/academy/lesson/what-is-memory-partitioning-definition-concept.html

[2] International Journal of Emerging Engineering Research and Technology Volume 3, Issue 9

[3] https://www.geeksforgeeks.org/variable-or-dynamic-partitioning-in-operating-system/

[4] https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380240602

[5] https://itstillworks.com/internal-external-memory-fragmentation-28851.html

[6] https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/

[7] https://learn.microsoft.com/en-us/dotnet/api/?view=netframework-4.6&preserve-view=true