**Georgia Southern University**

**Allen E. Paulson College of Engineering and Computing**

**Information Technology Department**

**Dr. Atef Shalan**

**Course: IT2530-Operating Systems**

**Module 4-**Main and Virtual Memory Management– reading material

Contents and Reading Material

[**4** Module 4- Main and Virtual Memory Management 1](#_Toc96419709)

[4.1 Main memory 2](#_Toc96419710)

[4.1.1 Memory Management Requirements 2](#_Toc96419711)

[4.1.2 Memory Addresses spaces 4](#_Toc96419712)

[4.1.3 Memory address binding 4](#_Toc96419713)

[4.1.4 Process swapping 5](#_Toc96419714)

[4.1.5 Memory Management Techniques 5](#_Toc96419715)

[4.1.6 Contiguous memory allocation 6](#_Toc96419716)

[4.1.7 Segmentation 9](#_Toc96419717)

[4.1.8 Paging 9](#_Toc96419718)

[4.2 Virtual Memory 10](#_Toc96419719)

[4.2.1 Virtual Memory address space 10](#_Toc96419720)

[4.2.2 Demand Paging 11](#_Toc96419721)

[4.2.3 Demand Segmentation and combined approach 11](#_Toc96419722)

# Module 4- Main and Virtual Memory Management

Processors depend on their limited size register memory to execute programs. According to Von Neumann’s computer model, programs are stored in memory while the computer processor executes instructions one by one from memory through the fetch and execute cycle. Therefore, the active processes must be stored in the computer memory during the program run. This module teaches about the computer’s main memory and virtual memory components. This module will discuss the memory management requirements, memory address binding, process swapping, and memory management schemes. We will learn about contiguous memory allocation, segmentation, and paging. We will also learn about virtual memory concepts management schemes, including demand segmentation and demand paging.

## Main memory

The [Main memory](https://www.geeksforgeeks.org/memory-management-in-operating-system/) is the closest nonvolatile storage to the computer microprocessor. The main memory can be viewed as a stream of memory cells, addressed using: 4 bytes in the 32-bit computer systems and 8 bytes in the 64-bit computer systems. Each of these cells has a memory address (i.e., index). The memory includes hardware interface functions that allow two operations: read and write. Every access request to the main memory is constituted of

* Access operation type: read/write
* Memory address reference
* Data to write [if write operation]

Due to the criticality of memory operations to process execution in operating systems, we start by learning about the main memory management requirements and then learn about management schemes and discuss how they meet or fulfill the main memory management requirements.

### Memory Management Requirements

Memory management is the method of organizing data storage in memory and accessing this data while supporting the operating system execution to user and system processes stored in memory. Memory management methods are intended to satisfy the following requirements: Allocation and Relocation, protection and sharing, and logical and physical organization.

#### Allocation and Relocation

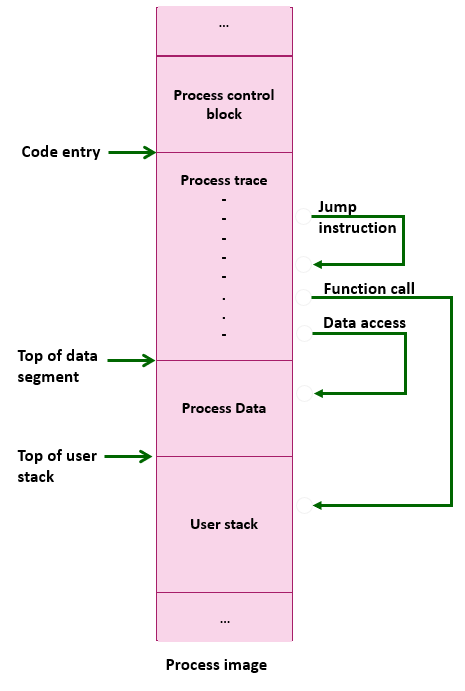


Figure 1- Process image and memory access operations

Allocation is the method of assigning a memory space to new processes. If a process is swapped out (i.e., suspended), it may need to be reallocated back into memory. Reallocating a process back to the same memory space it used to occupy is not feasible in operating systems as it contradicts the main reasons it was suspended at first. For example, if a process was suspended to free some memory space, preserving this particular space for the process while it is suspended is limiting. Therefore, reallocation of the process into memory may choose a memory space that is entirely different from the previously allocated space.

#### Protection and sharing

OS systems are concerned with protecting memory address spaces of processes from illegal access to different processes. Therefore, OSs enforce permission requirements to reference memory locations before granting any reading or writing operations. These access permissions are checked for every access operation by processes at runtime. This protection mechanism must be aware of and must consider the process relocation possibility and therefore allow access to the process wherever it resides in memory during its lifetime.

The protection mechanism must also consider the possibility of the process being allowed to share some resources with another process (e.g., dining philosophers). Therefore, memory management can provide controlled and protected access to shared memory areas among multiple processes without compromising its protection. Figure 1 illustrates an example process accessing different locations in its address space. Process trace instructions in the code segments issue memory access request to references in the code segment, data segment, user stack, and other locations.

#### Logical and physical organizations

The logical organization of memory is how the programmer and the operating system view memory space. Therefore, logical memory address space is viewed as a linear tape/media where memory cells are listed in order of the address reference. I.e., the memory address reference increases from the beginning to the end of the memory tape. Logical organizations make it easy for the OS and the application programmers by avoiding the complexity of the memory hardware and physical address space. Therefore, software programs can be written in separate modules (i.e., segmentation) that can be compiled and bound to physical memory separately without any complexity. Protection mechanisms also work seamlessly with logical memory organization and can provide data protection at different levels of software architectures.

The physical organization is the hardware level of memory access. This organization level is protected by hiding it from programmers and operating systems. A hardware address mapping component is responsible for mapping the logical address to its corresponding physical address and vice versa. The operating systems interact with the physical memory through this hardware mapping by providing logical address information along with the access operation details.

### Memory Addresses spaces

Three main memory reference types are used to access memory during a process runtime: Logical address, relative address, and physical address. Logical memory addresses are generated by the system CPU. The group of logical memory address references or memory chunks allocated to a process is known as the Logical Address space of a process. Physical memory address (aka real memory address) is the actual data location in the main memory as seen by the memory unit. The relative memory address is an intermediate and unbound addressing method that supports process relocation. The relative memory reference is expressed as a location relative to some known Logical memory address. During runtime, all relative memory addresses will be bound to logical memory addresses after process allocation or relocation and then mapped to physical addresses during the memory access operation.

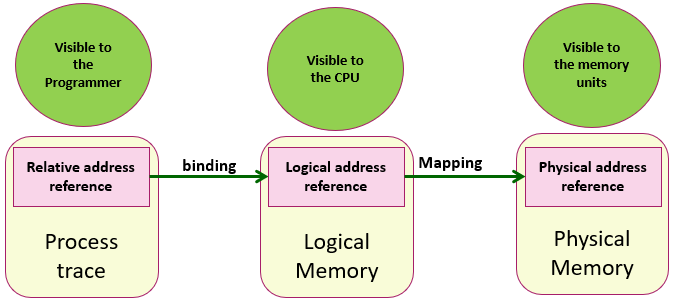


Figure 2- Memory address spaces

Figure 2 describes the relationships between memory address spaces. Relative addresses are visible to application programmers and are part of the process trace. Logical addresses are controlled at the CPU level and are visible to the operating system. Physical addresses are entirely hidden from the OS and programmer and are visible only within the memory unit. [Memory-Management Unit (MMU)](https://en.wikipedia.org/wiki/Memory_management_unit) is a hardware device responsible for mapping a logical address to the corresponding physical address in memory at the process runtime.

### Memory address binding

Software programs on the secondary storage device become ready to be loaded in memory and be executed on the CPU when the user or system starts the program. Memory addresses in the program code are specified symbolically as variable names of specific data types. Depending on the varying size and data type, the CPU can calculate the size of the variable in memory. Depending on the location of the variable inside a program module, the CPU can calculate a relocatable address of this variable relative to the module memory address space. [Memory address Binding](https://www.javatpoint.com/address-binding-in-operating-system) refers to the mapping of program instructions and data into the logical address space of the process. Address binding takes place to happen at three different stages of software system development and execution:

* **Compile-time**
  + Symbolic addresses (e.g., varX) are mapped into relocatable addresses (e.g., 4 bytes from the base register of the module).
* **Load time**
  + Relocatable addresses are mapped into logical addresses (e.g., 7401465)
* **Execution time**
  + Logical addresses are mapped into a physical address (real memory address)

### Process swapping

[Process swapping](https://www.javatpoint.com/swapping-in-operating-system) is the method of suspending a process (e.g., changing the state from ready to ready suspended) to help the CPU improve the main memory utilization. In this case, the process image is removed from the main memory and kept in the virtual memory (i.e., on the secondary storage device). A medium-term Operating system scheduler is part of the memory management called process swapper. Processes that are swapped out of the main memory are later brought back into memory for continued execution. There are two main primitives for process swapping: roll out and roll in. Processes of lower priorities are swapped out, and higher priority processes are brought in for faster execution. The swap time is the time taken by the swapper to invalidate a process in memory and repopulate the process in the virtual memory. The swap time is directly proportional to the size of the swapped processes.

### Memory Management Techniques

Memory management allows operating systems to follow a systematic methodology for memory partitioning, process allocation, and reallocation in memory, protection, and sharing of process data, among other processes. In this part, we study some different memory management schemes and illustrate the advantages and disadvantages of each. Main Memory Management schemes include fixed, dynamic, simple segmentation, and simple paging. As we will see later, demand segmentation and paging are two additional schemes that serve the virtual memory address space.

* **Fixed partitioning** 
  + In this partitioning scheme, the memory is divided into fixed-size chunks pre-generated statically at the system before process existence.
  + Fixed partitions with different sizes are created to accommodate different-sized processes.
  + A process can be allocated to an equal or a more significant memory chunk
* **Dynamic partitioning** 
  + In this scheme, partitions are created dynamically during process scheduling
  + Processes are loaded into partitions of exact similar sizes
* **Simple segmentation**
  + Each process is allocated as a set of segments representing process modules.
  + Segments are loaded into separate partitions that don’t have to be contiguous to each other.
* **Simple paging**
  + Memory partitions are relatively small and equal-sized chunks called pages.
  + Similar to segments, pages are loaded into separate partitions that don’t have to be contiguous to each other.
* **Demand segmentation** 
  + Similar to simple segmentation except that it is used with Virtual memory address spaces
  + A segment can be loaded on-demand only if the process code trace requires it.
* **Demand paging**
  + Similar to simple paging except that it is used with Virtual memory address spaces
  + A page can be loaded on-demand only if required by the process code trace.

In the following subsections, we study [contiguous memory allocation](https://binaryterms.com/contiguous-memory-allocation-in-operating-system.html) (i.e., static and dynamic partitioning), [Segmentation](https://www.javatpoint.com/os-segmentation), and [paging](https://www.geeksforgeeks.org/paging-in-operating-system/). Later, we will also learn the details of demand segmentation and demand paging schemes.

### Contiguous memory allocation

Operating systems aim to allocate the system resources efficiently. Main memory resources are the closest device to computer microprocessor, and it helps temporarily store executing user and system processes. Contiguous allocation is a traditional method for memory management. Contiguous memory allocation is a memory management scheme in which a single adjacent section of memory is allocated to a process. Contiguous memory allocation can be static or dynamic, as explained earlier. In the following part, we will discuss the details of contiguous allocation and learn about its main advantages and disadvantages.

In contiguous allocation, a pair of registers are used to specify the process address space: base register and limit register. The base register references the start of the process in the main memory. The base register is also known as the relocation register. It can be used to relocate a process to another memory location and rebind all relative memory references to the newly allocated space. The limit registers include a value of the size of the process allocated. The limit register is used to identify the legal limit of the process space in memory and assures that it will not reach the allocated space.

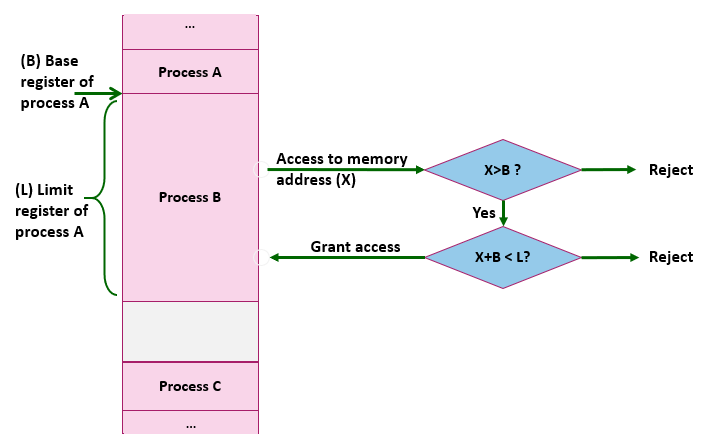


Figure 3- hardware address protection

A [hardware address protection](https://www.geeksforgeeks.org/hardware-protection-and-type-of-hardware-protection/) is part of contemporary microprocessors that use the base and limit registers to protect illegitimate access to a process space in memory. This hardware protection allows CPUs to compare the access request to the process boundaries in memory with every memory access operation. As shown in Figure 3, the hardware address protection checks the memory access request generated in user mode is between base and limit for that user.

Both static and dynamic memory allocation divides the memory into single partition units for each process. Static allocation is easier to implement with little overhead since all partitions are pre-created beforehand. However, there are two main disadvantages of static allocation. The first disadvantage is the limited number of active processes due to the limited number of pre-created process address spaces. Second, the memory resource is consumed too fast due to the internal fragmentation problem of fixed-sized allocated memory spaces. Internal fragmentation is a memory allocation problem where some allocated space is wasted because the allocated processes are smaller than the allocated sizes. The free and unused space allocated causes the **internal** fragmentation problem.

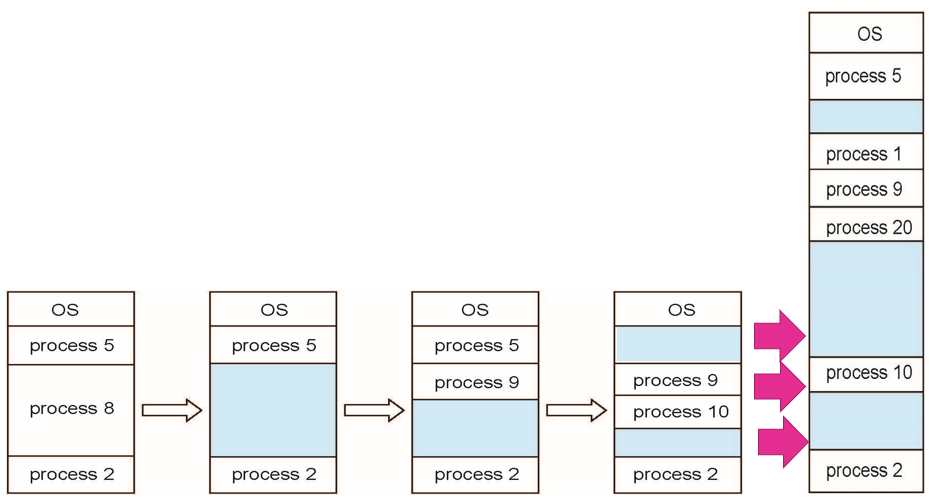


Figure - the increase of memory holes along the time

In dynamic allocation, a process is allocated exact size in memory dynamically during the process scheduling. While this memory management scheme is more costly than static allocation, dynamic allocation overcomes the problem of internal fragmentation since no space is wasted inside the process spaces. However, due to the increased existence of unused spaces (Holes) outside the process spaces scattered along with the memory tape, a new process may not be allocated if these holes are smaller in size. A memory hole is a block of available memory on the memory tape.

Throughout the time, processes are allocated and terminated or reallocated, leaving behind numerous holes of various and small sizes that cannot fit any processes. This problem is called [the external fragmentation](https://www.javatpoint.com/fragmentation-in-operating-system) problem. Figure 4 shows the increase of the memory holes over time. Process termination frees its allocated partition. Any two adjacent holes become a giant hole. The repetition of this scenario causes an external fragmentation problem. The operating system runs a dedicated process for memory space compaction to resolve this problem. Compaction is a technique for overcoming the external fragmentation problem in which the OS shifts or concentrates the processes to avoid the gaps or holes between them. After the compaction, all holes combine into one large hole and become ready for new allocations. However, this compaction process is time-consuming, and it wastes computing resources.

Operating systems use a number of algorithms to satisfy a request of a specific process size from a list of free holes of variable sizes. These algorithms are listed below.

* **the first-fit algorithm** scans the memory from the beginning and allocates the first available hole that is is more significant than the required space.
* **The next-fit algorithm** scans the memory’s current location and up, then allocates the next hole that is is more significant than the required space. This algorithm is faster than the first fit, but it may result in less memory utilization.
* **The best-fit algorithm** starts fromthe beginning, scans the whole memory, and then allocates the fittest hole (i.e., the smallest hole that is large enough to fit the new process). This algorithm is more costly than the others, but it assures the best memory utilization with the smallest leftover holes.

### Segmentation

Segmentation is a memory management scheme that supports multi-modular application systems. It allows the non-contiguous allocation of separate process chunks (called segments) of variable sizes in a different memory location. Therefore, the process does not have to be allocated in one large partition with a segmentation scheme. Instead, it is allocated several segments that can be stored anywhere in the memory. In this case, a memory address must include both segment Id and an offset within this segment.

Given a logical address in the form of segment ID and offset, the steps to translate logical addresses to physical addresses are as follows.

1. Search the index of the process segment table for the segment ID.
2. Extract the segment address in the physical memory (segment base register) and the segment length (segment limit register) from the table row that matched the search.
3. Compare the offset address to the length of the segment and reject the access if the offset is larger than the segment length.
4. Calculate the physical address as the sum of the segment physical address of the offset

The advantages of the segmentation scheme are that it allows allocating the process based on the user view of the process modulation. It also allows the dynamic allocation of variable memory chunks. Moreover, the segmentation allows the non-contiguous allocation of the process in memory. Furthermore, the problem of internal memory segmentation does not exist in memory segmentation. The disadvantages of the segmentation scheme are, first, the varying size chunks provide additional complexity and computation overload to the CPU. Second, the external fragmentation problem is still not resolved. Next, we learn about the paging scheme.

### Paging

Paging is the most recent memory-management scheme. Paging permits the noncontiguous allocation of physical memory. The paging scheme divides the logical memory into fixed-sized blocks called pages and divides the physical memory into blocks of the same size as pages called frames. A program that requires a number of pages in the logical memory will be allocated in the same number of frames in the physical memory. Processes must maintain a page table that maps page id (as table index) to the page base register as data. The process retains two registers for the page table: first, the page-table base register (PTBR) that points to the page table itself, and the Page-table length register (PTLR), which indicates the size of the page table.

Note that there is no need for a limit register since all pages are the same size (usually 512 bytes).

The logical address in the page scheme is two parts, page ID and offset. Given a logical address in the paging scheme, the steps to translate logical addresses to physical addresses are as follows.

1. Compare the page Id to the PTLR, and if the page ID is more extensive, then reject the access; otherwise, continue.
2. Search the index of the process page table for the given page ID.
3. Extract the page address in the physical memory.
4. Compare the offset address to the fixed page length and reject the access if the offset is larger.
5. Calculate the physical address as the sum of the page physical address of the offset.

Due to the fixed and small sizes of pages, the paging scheme solves the problem of external memory fragmentation,and at the same time, it avoids the pain of variable size memory chunks. In most OSs, both Segmentation and Paging work efficiently as a combined approach.

## Virtual Memory

[Virtual memory](https://en.wikipedia.org/wiki/Virtual_memory) extends the main memory on the secondary storage device. The virtual memory scheme allows secondary storage to be addressed and allocated by processes during runtime. This scheme uses a particular hardware structure for memory addressing, and it uses special memory allocation schemes: demand segmentation, demand paging, and a combined scheme. We will learn about virtual memory address space and the memory allocation schemes in the following parts.

### Virtual Memory address space

Contemporary software systems are increasing in size and complexity. Loading a whole application in the main memory during execution may overwhelm the computer resources and consequently decrease the degree of concurrency. When a program runs, some parts of the code are executed, and others may not be needed during the process lifetime, e.g., error-handling code, rare administration code, rarely used program data, and unrequired user choices.Partial program loading allows parts of the program to be loaded in memory while the parts remain in the secondary storage device. [The virtual address](https://docs.microsoft.com/en-us/windows/win32/memory/virtual-address-space) refers to a memory address in the extended memory space (main and virtual memory). Virtual address space is the memory space assigned to a process in the virtual memory. Despite this combined or extended memory model, the virtual memory allocation satisfies the following four main memory management fundamentals.

* Processes use logical addresses as memory references to access the physical memory (either on the main or secondary memory).
* The mapping between the logical addresses and the physical address occurs dynamically at run time through a memory hardware unit.
* A process may be divided into a number of chunks that need not be contiguous to each other.
* A process may keep part of its segments in the secondary storage during its execution.

With the virtual memory scheme, programs can be executed while partially loaded in the main memory. Thus, processes are not limited to the main memory size, which is significantly smaller than the secondary storage. With less loaded programs in memory, more processes can run simultaneously, i.e., a higher degree of concurrency, increased processor utilization, higher throughput, and overall performance. If a process is swapped out of memory, less data resident in memory will need to be swapped out.

The major problem with the virtual memory scheme is the frequent need to swap processes parts in and out during the process’s runtime. If the processor spends more time swapping data in and out than the actual execution of the process itself, this will lead to a thrashing problem.

[Thrashing](https://www.javatpoint.com/what-is-thrash) is when the processor spends more time swapping parts of the process in/out of the main memory rather than actually executing the process. Due to [the principle of locality](https://en.wikipedia.org/wiki/Principle_of_locality), only a few pieces of a process trace are needed to start the process or run it over a short period. Based on this principle, operating systems can predict the parts of the necessary code in memory at any execution time, and which parts can stay in the secondary storage. By doing this mechanism, operating systems can avoid the thrashing problem.

A virtual address of a process is typically a logical address that is mapped as either the main memory location or secondary memory location through a memory mapping device called [Virtual Memory Mapping Unit (VMMU)](https://developer.arm.com/documentation/den0024/a/The-Memory-Management-Unit). In the following parts, we learn about demand paging and demand segmentation.

### Demand Paging

Like a simple paging scheme, [demand paging](https://www.javatpoint.com/os-demand-paging) allows loading processes into fixed-size pages in the logical memory and fixed-size frames in the virtual memory. However, demand pages load pages only if the executing processes need them. Page swapping brings a page from the secondary storage to the main memory. Page swapping technique called lazy swapper swaps pages only if they are required. It first checks if the required page is in the main memory or not. If the page reference is marked in the page table as valid, then it is already loaded in memory; otherwise, if the page reference is invalid, the page is not in memory and needs to be brought from the secondary storage.

The process page table includes a single bit data field that indicates whether a page is valid or invalid: valid means the table is already in the main memory, invalid means the page needs to be swapped from the secondary storage. When the operating system checks the main memory to find the requested page, it reads the valid or invalid field corresponding to the requested page id in the process page table. If the page is invalid, the operating system raises a [page fault](https://techcommunity.microsoft.com/t5/ask-the-performance-team/the-basics-of-page-faults/ba-p/373120).

Page fault is an operating system interrupt raised when a process page is not currently in the main memory. The page fault interrupt handler is a function that rolls the page from the secondary storage into the main memory.

### Demand Segmentation and combined approach

Like simple segmentation, [demand segmentation](https://www.geeksforgeeks.org/difference-between-demand-paging-and-segmentation/) allows the allocating process according to separate modules (i.e., segments) of varying sizes. Like demand paging, demand segmentation will enable segments to remain in the secondary storage until the processor needs them. To implement demand segmentation, the operating systems maintain a segment table for each process indexed by the segment Id. The segment table stores the segment base register, and the segment limit register, in addition to the valid and invalid bit. The operating system also maintains a segment table base register (STBR) and segment table limit register (STLR) to identify table location and validate the segment Id as part of the process.

Most operating systems use a Combined Paging and Segmentation approach for memory management. In the combined memory management approach, the process address space is divided into a number of segments, and each segment is divided into a number of pages. The segment table is part of the process, while each segment maintains its page table. In this case, the memory address reference includes segment id, page id, and offset. The operating system first validates the segment id and extracts the segment address from the segment table of the process. The operating system then validates the page id by comparing it to the specific segment's page table limit register (PTLR). The operating system obtains the page base register if the page Id is valid. The operating system obtains the physical address reference by combining the physical segment address, page physical address, and offset.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_