Faculty of Computers & Informatics

Zagazig University

Memory Management

A research project submitted
in partial fulfillment of the requirements for passing
the 2nd semester 2020 evaluation

In

Operating Systems

by

Hassan Ramadan Ibrahim Abdellatif (2123)

Supervised by

Dr. Alaa Alghamry

June 2020

Table of Contents

1. Introduction

Any program that must be executed must be brought into the main memory and placed within a process for it to run. If there is enough space in the main memory, more than one process can be placed in the main memory. To increase system performance, several processes must be kept in memory. The processes must share the memory. There are different ways to manage the main memory. The operating system is responsible for memory management, to avoid different issues related to memory management.

2. Content

Definition:

The main memory consists of a large array of words or bytes. The CPU fetches the instructions from the memory based on the value of the program counter and executes them. In an instruction-execution cycle, an instruction is first fetched from memory. The instruction is decoded. If needed, operands are also fetched. After the execution of the instructions, the results may be stored back in the memory.

Since the main memory is not large enough to accommodate all the active processes at the same time, the processes that are not in the main memory are kept in the backing store or the disk. This collection of processes on the disk that are waiting to be brought into memory to run the program are placed in an input queue.

One of these processes is selected and loaded into the main memory and is executed. After the termination of the process, the main memory space allocated to the process is freed. User programs go under several steps before being run.

The addresses in the source program are represented in symbolic form. The compiler will bind these addresses to relocatable addresses. These addresses will range from 0 to a maximum for that particular process. The loader will turn these addresses into absolute addresses.

Concepts:

1. Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at the following three different stages:

1. *Compile time:* If the memory location in which the process will be placed in the main memory is known an absolute code can be generated. But the code must be recompiled if the starting location of the process changes.
2. *Load time:* If the memory location of the process is not known at compile time, binding is delayed until load time. Here, relocatable code is generated. The actual memory locations are known only when the process is loaded into memory.
3. *Execution time:* If the process can be moved during its execution from one memory segment to another, binding is delayed until run time. This needs hardware support for address maps. This is used by most general-purpose operating systems.

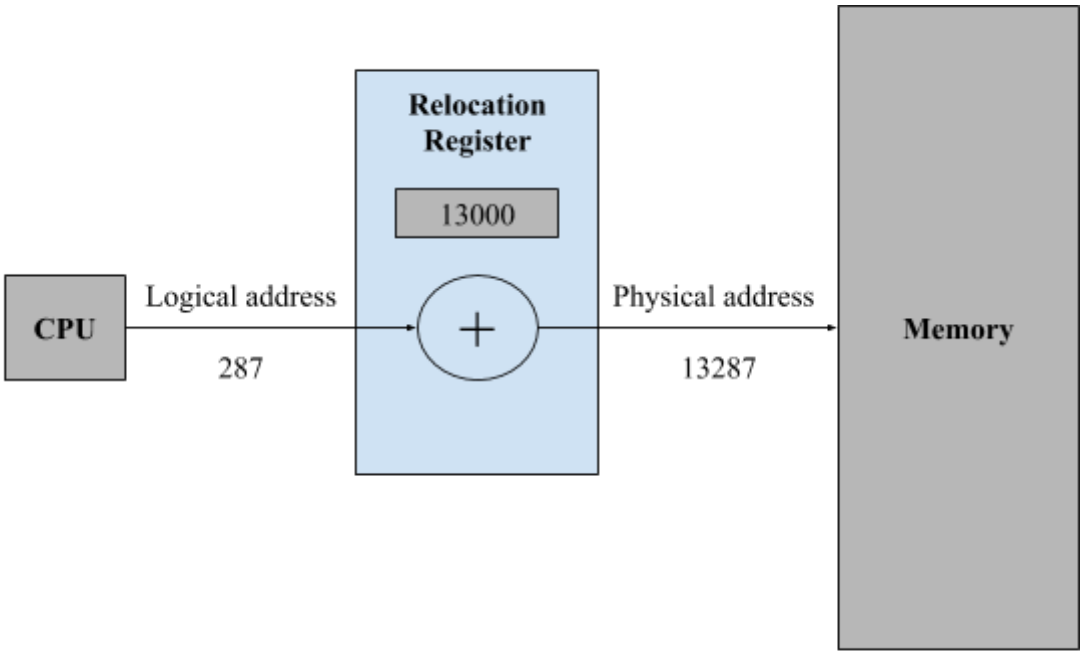
2. Logical versus Physical Addresses

When a program is compiled, relocatable addresses are generated. These addresses range from 0 to maximum. These addresses comprise the logical address space. When the program is placed in the physical memory, it need not be placed in the same address as that of the logical address. The address where the program is placed in the physical memory comprises the physical address space. Thus, the logical address is generated by the CPU and is also referred to as a virtual address. The physical address is that the address seen by the memory unit. The logical and physical addresses are the same in compile time and load time address binding schemes; logical and physical addresses differ in execution-time address-binding.

3. Memory Management Unit

Since the logical address and the corresponding physical address may be different, there is a need for a memory management unit that maps the logical address to physical address. The user program deals with logical addresses; it never sees the important physical addresses. Only the memory-mapping hardware converts the logical addresses to physical addresses. There are many memory management schemes available like paging, segmentation, and page segmentation.

We now look at a simple memory management unit scheme and understand how the mapping is done from the logical address to the physical address. The starting address from where the process is loaded in the main memory is kept in a relocation register. The value within the relocation register is added to each address generated by a user process at the time it's sent to memory. The resultant value is the physical address. Say, a process is loaded within the main memory from address 13000. This value (13000) is stored in the relocation register. If the logical address is 346, then the physical address will be 13000+287 = 13287.



Simple Memory Management Unit

Drawed on google docs drawings by Hassan Ramadan

Thus, logical addresses range from 0 to max and physical addresses range from R + 0 to R + max, where R is the starting address of the process in physical memory. The user thinks that the process runs in locations 0 to max and supplies logical addresses. Logical addresses are converted to physical addresses before they are used.

4. Dynamic Loading

So, the entire program and data of a process must be in physical memory for the process to execute. The size of a process is restricted to the dimensions of physical memory. To obtain better memory-space utilization, we will use dynamic loading. With dynamic loading, a routine isn't loaded until it's called. All routines are kept on disk during a relocatable load format. Only the main program is loaded.

When a routine is called, the calling routine checks if the called routine is already loaded. If not, the relocatable linking loader is called to load the routine. The loader updates the program's address table. Control is passed to the newly loaded routine. This results in better memory-space utilization; unused routine is never loaded. This kind of loading is useful when large amounts of code are needed to handle infrequently occurring cases, say, error routines.

Some types of errors may occur very rarely, but still, the code for handling these types of errors should be a part of the program. If a particular type of error does not occur, the code for handling that error will not be used, and hence, that part of the program need not be loaded into memory at all. No special support from the OS is required and may be implemented through program design. Users should design their programs in a modular way for this to work. The operating system may provide library routines to implement dynamic loading.

5. **Dynamic Linking**

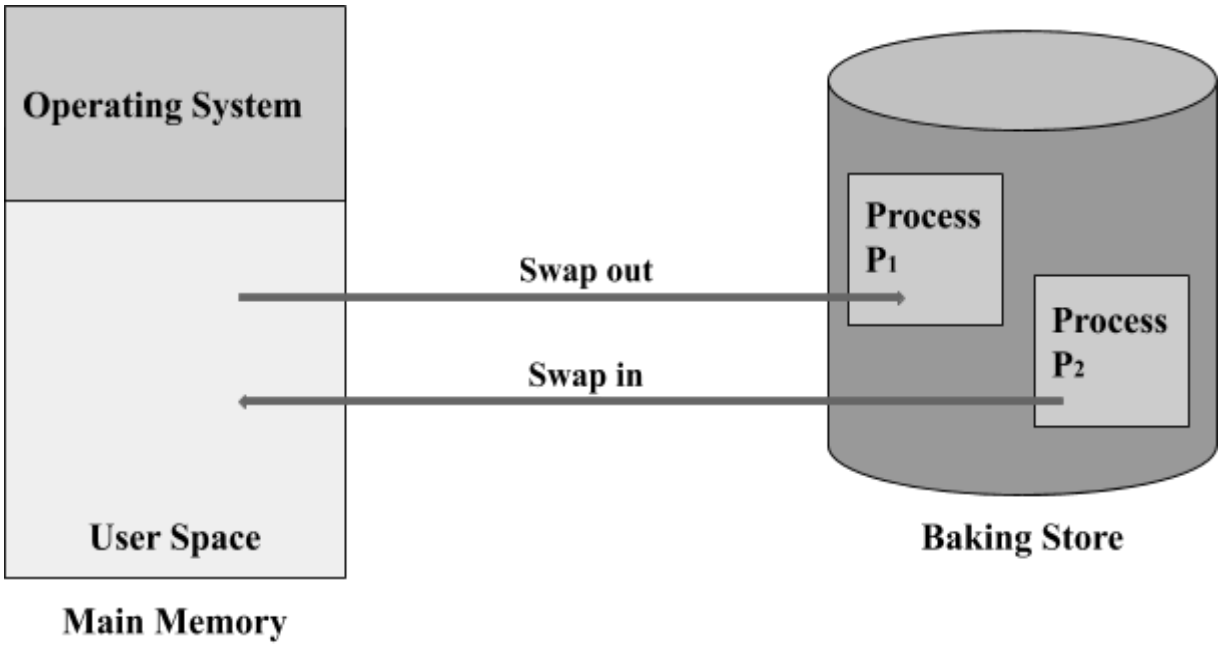
In dynamic linking, linking of library routines is postponed until execution time. For each library-routine reference, a stub is included in the executable image. The stub is a small piece of code that is used to locate the appropriate memory-resident library routine. When the stub is executed, it replaces itself with the address of the routine, and executes routine. Dynamic linking is particularly useful for libraries.

Techniques and mechanisms:

1. **Swapping**

Any process that is currently being executed needs to be kept in the main memory. A process may move between the running state and waiting state, many times during its lifetime. When the process is in the waiting state, the process is not using the CPU and hence need not be kept in the main memory. When one process is in the waiting state, another process can use the CPU and the process that is currently using the CPU is kept in main memory. When there are many active processes, all processes cannot be accommodated in the main memory, since the size of main memory is small.

Therefore, processes that are not executing currently can be swapped temporarily out of main memory to a backing store, and then brought back into memory for continued execution. The backing store is a fast disk large enough to accommodate copies of all memory images for all users. This moving out and moving in of processes is called swapping.



Swapping Process

Drawed on google docs drawings by Hassan Ramadan

Suppose round-robin scheduling is the CPU scheduling algorithm used. A time slice is allotted to each process. Suppose there are 10 processes that are using the CPU in a round-robin manner.

The process whose time slice is completed is swapped out, if there is no enough space in the main memory, and the next process is brought in. If priority-based scheduling is used, a low-priority process is swapped out to bring in a high-priority process. This moving out is called roll out and moving in is called roll in.

In swapping, the major part of the swap time is transfer time, that is, the time required to transfer the process form the main memory to the disk. The total transfer time is directly proportional to the quantity of memory swapped.

2. Memory Allocation

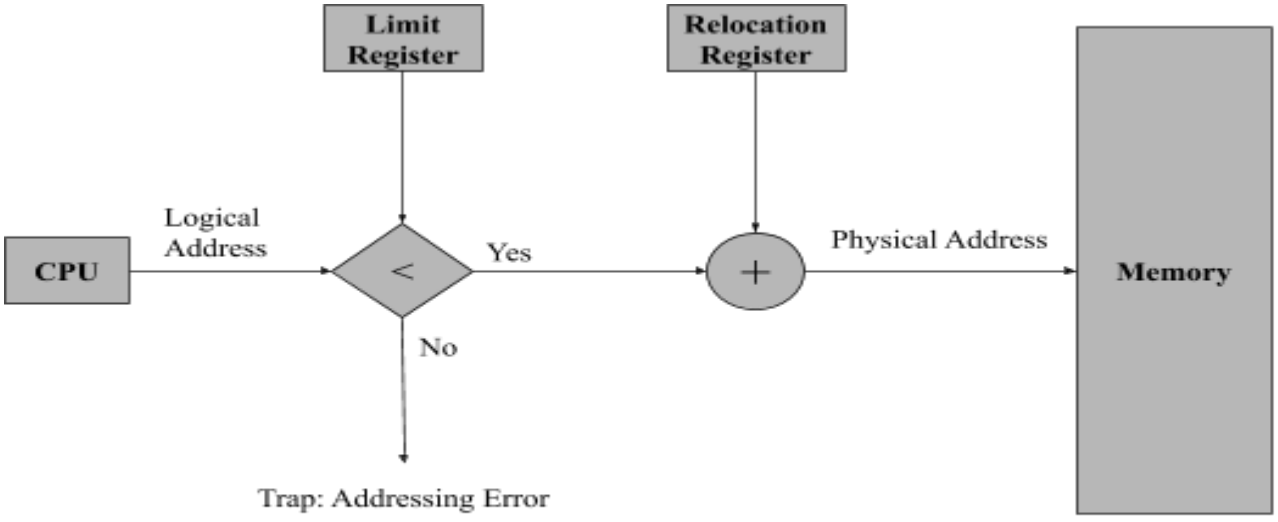
The main memory must accommodate both the OS and therefore the various user processes. We need to allocate different parts of the main memory in the most efficient way possible.

There are two common ways to do so:

1. *Single-partition allocation:*

In this scheme, the main memory is considered as a single big partition. Processes are brought into the main memory and can be placed at any free space in the main memory. The starting address where the process is placed in remembered, say, in a relocation-register. The relocation-register scheme is employed to guard user processes from one another , and from changing operating-system code and data. The relocation register contains the value of the smallest physical address of the process. Another register called the limit register is maintained which contains the range of logical addresses, So each logical address must be less than the limit register.

Any address that is generated by the CPU is checked with the contents of the limit register. The address generated must be less than the contents of the limit register. If it is less, then the address generated is added with the contents of the relocation register to get the physical address. If the logical address generated by the CPU is not less than the value in the limit register, then a trap is given to the operating system, indicating that the address generated is an illegal address.

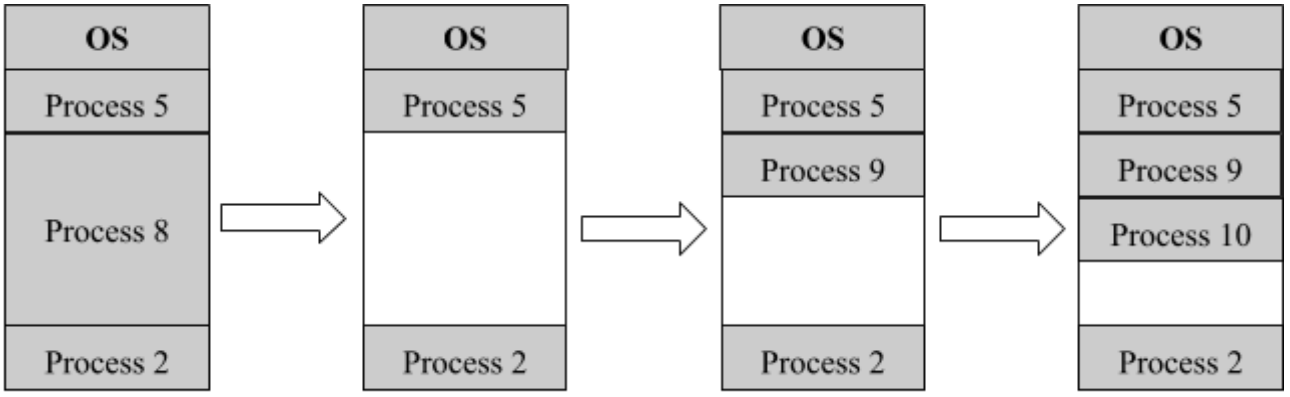


Single-Partition Allocation

Drawed using google docs drawings by Hassan Ramadan

2. *Multiple-partition allocation:*

In multiple-partition allocation, the main memory can be divided into several fixed sized partitions. When a partition is free, a process is selected from the input queue and is loaded to the free partition. It is also possible to have variable-sized partitions. The memory is looked at as one large hole – block of available memory. When a process arrives, it's allocated memory from a hole large enough to accommodate it. When a process terminates, the space is freed and is added to the set of holes. Memory space for processes are allocated and freed, in due course of time, there will be a lot of holes of various sizes scattered throughout the memory.



Multiple-partition allocation

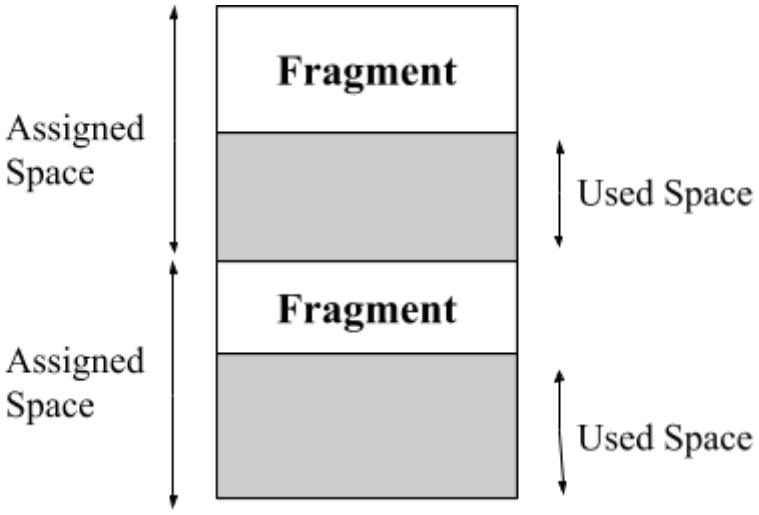
Drawed using google docs drawings by Hassan Ramadan

3. Fragmentation

when processes are loaded and removed from memory, the free space in the memory is broken into small pieces. It happens after processes can't be allocated to memory blocks sometimes considering their small size, and memory blocks remain unused. This is called the Fragmentation problem.

There is two types of Fragmentation:

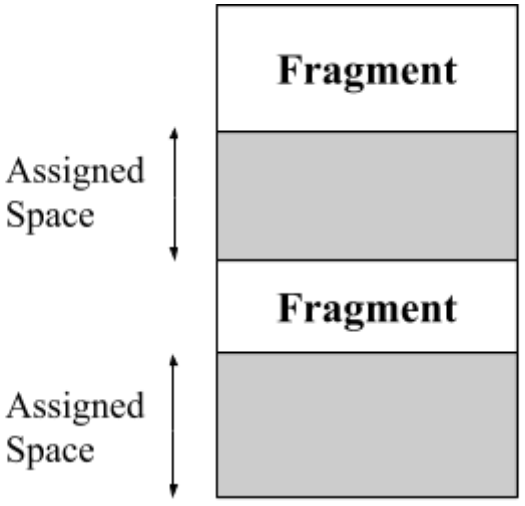
1. *Internal Fragmentation:* This occurs when the memory is broken down into blocks of mounted size. Whenever a memory request for a method is allocated to the method the mounted size block. Just in case the memory allocated to the method is somewhat larger than the memory requested, the difference between the memory allocated and the memory requested is that of the internal fragmentation.



Internal Fragmentation

Drawed using google docs drawings by Hassan Ramadan

2. *External Fragmentation:* External fragmentation occurs when there is a sufficient amount of area inside the memory to satisfy a method's memory request. However, the memory request of the process can not be fulfilled since the offered memory is non-contiguous. Either you apply the memory allocation strategy first-fit or best-fit it will cause external fragmentation.



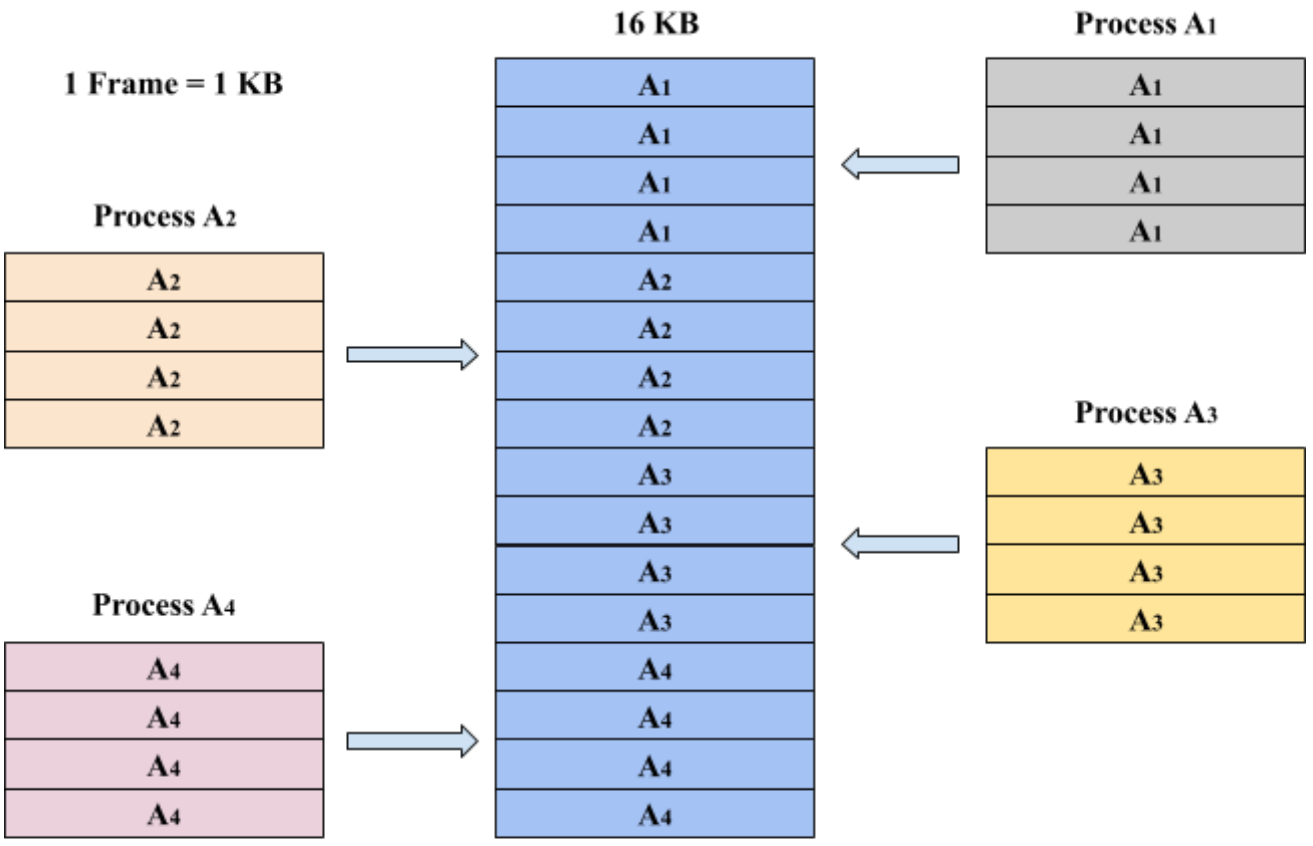
External Fragmentation

Drawed using google docs drawings by Hassan Ramadan

4. Paging

Paging is a storage mechanism which allows the OS to retrieve processes in the form of pages from the secondary storage into the main memory. The main memory in the Paging method is divided into small fixed-size physical memory blocks which are called frames. To maximize the use of the main memory and to avoid external fragmentation, the size of a frame should be kept the same as that of a page. For faster data access, Paging is used, and it is a logical concept.

So, the Paging process should be protected by the insertion concept of an additional bit named Valid / Invalid bit. Paging Memory protection is achieved in paging by associating bits of protection with each page. These bits are associated with the entry of each page table and specify protection on the respective page.



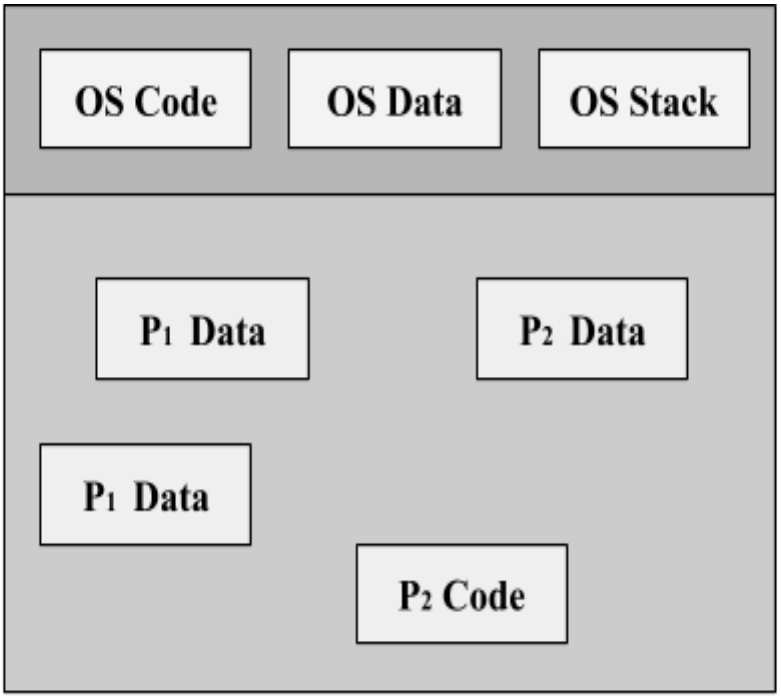
Main Memory Paging

Drawed using google docs drawings by Hassan Ramadan

5. Segmentation

The mechanism of segmentation works almost like paging, the only difference between the two is that the segments are of variable length while the pages are always of fixed size in the paging mechanism.

A segment of the program includes the main feature of the program, data structures, utility functions and many more. For all processes, the OS maintains a segment map table. It also contains a list of free blocks of memory along with its size, segment numbers, and memory locations in the main memory or virtual memory.



Segmentation

Drawed using google docs drawings by Hassan Ramadan

3. Conclusion

So, we have explored the definition of memory management and the concepts that really matter for everyone to understand the memory management mechanisms and eventually explored the memory management mechanisms and techniques. How the OS can manage memory and what are the different techniques to do so is a very important topic, which makes the OS so powerful core component of our modern computers.

References

[1] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. 2012. Operating System Concepts (9th. ed.). Wiley Publishing.

[2] Tanenbaum, A. S., & Woodhull, A. S. (1997). Operating systems: design and implementation (Vol. 68). Englewood Cliffs: Prentice Hall.