Introduction

Finding differences between two terms may imply that we are aware of what each individual term stands for yet the purpose of comparison is just seeking a better way of understanding. I think that is true to some extent however I also noticed that there are people who ask the “difference between x and y question” simply because they have no clue what x and y are in the first place. I certainly believe – and repeated that in earlier articles – the lack of proper context and using terms that are not crystal clean in our minds leads to all kinds of confusion when studying any technical topic. For that reason, my approach in clarifying the difference between paging and segmentation is to provide brief description of some relevant concepts then use that to set the right context. Sounds good ? let us proceed.

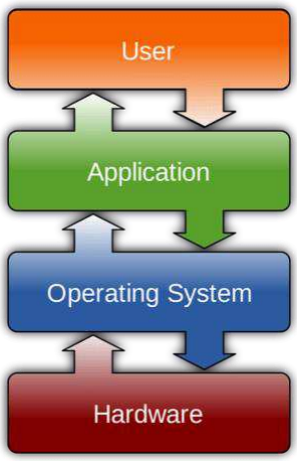
Basic computer organization

Keep in mind that the bare minimum computer system consists of 3 main units:

- **IO**: enters data and instructions before any computation happens and sends results to the outside world.
- **Storage**: permanent or secondary storage (disk) while primary or temporary storage is the physical memory or RAM, stores programs, data and results.
- **CPU**: mainly two units, the ALU which performs arithmetic (ex. addition) and logic operations (Boolean) and control unit that orchestrate all computational activities (ex. directs an input device to feed data into storage)

That is the view when we look at the computing system bottom up or the hardware side. Looking at the same system top down leads us to operating systems and user programs.

Program OS and CPU



As a programmer, you write a program and enter it into the secondary storage using IO. The operating system loads your program from secondary storage into the main memory for execution by the CPU. Recall that a process is nothing but a running user program from the operating system perspective. Let us take an example, image you have the following program with only one line of code written in a high level language:

```
1 x = x + 5
```

The corresponding assembly language code may look like:

```
1 #Load data at memory location x into CPU register R1
2 LOAD R1, x
3
```

```
4 # Add 5 to the content of register R1
5 ADD R1, 5
6
7 # Store content of register R1 back to memory location x
8 STORE x, R1
```

As you can see, we have 3 actors behind the scenes:

- **Program**: a sequence of instructions targeting a specific computer architecture instruction set (ex. LOAD, ADD, STORE)
- **OS**: special purpose software that sits between user programs and the underlying computer architecture. The operating system loads user program into main memory for execution by the CPU. The way the OS loads a user program into main memory for execution is directly related to our topic (paging and segmentation) as we will see later.
- **CPU**: mainly an ALU (Arithmetic and logic unit to do basic operations + set of registers to store instruction operands and intermediate results). CPU fetches instructions from main memory into registers, decodes instructions to figure out what operation needs to be done, retrieves operands (input values or arguments) from memory and finally executes instructions. The way the CPU fetches instructions and operands from main memory is also related to our topic (paging and segmentation) specifically providing hardware support to perform address translation (will mention that later).

So what is the deal here ? nothing special, I just reminded you to put in mind the two views of a typical computer system before discussing topics like paging and segmentation. We could have jumped directly into our main topic without even mentioning computer organization and architecture however doing so puts things into perspective as we proceed to the next sections.

Memory management

Take a second look at the assembly program mentioned earlier. Notice that fetching an instruction or operand from main memory or storing back a result implies that data or instructions are present at the main memory the time of execution. What if that instruction or data is not present ? It has to be brought from secondary storage into main memory. Can we load the entire program along with all other running programs into the main memory ? Is it possible to fit them all at the same time ? What if running programs start to interfere with each other ? as you can see, it turns out that managing memory and running programs is such a complex task that needs to be carefully implemented. With that said, paging and segmentation are nothing but operating system (software) and computer architecture (hardware) features to better manage memory usage. Let us see how?

Virtual memory vs physical memory

The operating system loads a computer program into main memory for execution. User program can be two large to fit into the main memory or the memory can be too so small to fit multiple programs running at the same time. What does this tell us ? There are two memory concepts that we need to deal with:

- Actual physical (hardware) memory or RAM (random access memory) which is fixed in size.
- Imaginable or virtual memory implemented by the operating system. From the program perspective, it is large enough to fit all user program addressable space (will talk about that later).

Take this example, In playing cards game, at any point in time the table can not fit all cards however a portion of the cards is available on the table while the rest are with players. Similarly, the operating system has to implement a mechanism to facilitate loading program portions that are only needed

not the entire program. Paging and segmentation in OS are two different techniques to implement virtual memory to better manage running programs and utilize the available physical memory.

Memory address space

Memory address space – in plain English – means all possible locations a given instruction or piece of data can be stored. If your program is 100 instructions then the address space spans from 0 to 99. A computer system using 32 bits addressing means any possible instruction or data can be stored at an address between 0 and 2 to the power of 31. This means you need a 2G bytes of virtual memory. An address in a virtual memory is called a logical address while the actual address of an instruction in main memory is called the physical address. For example, using the 100 lines program mentioned earlier, the last instruction at logical address 99 can be stored in the main memory at the physical address of 1048576 (I just invented that number which is a Mega in binary or 1024 x 1024). This translation between a logical address and a physical address is typically supported in the underlying hardware CPU architecture and utilized by the operating system to implement virtual memory management (ex. paging or segmentation). Now it is the right time to dive into our main topic.

Paging vs segmentation example

Imagine you are reading a book. The book is typically divided into chapters, sections and pages, right ? roughly speaking all pages have the same size in terms of number of lines. Pages in a book is just away to divide a story into manageable small pieces however a single page does not necessarily narrate a complete idea. On the other hand dividing the book into chapters regardless of how many pages a chapter spans gives the reader a better idea about the story being told. Book pages versus chapters is very similar to paging versus segmentation in operating system design. Each technique serve a specific design purpose and has its own advantages and disadvantages. Some operating systems implement a combination of both to better manage memory.

What is paging ?

Paging is dividing the available physical memory into equally sized units called frames. The operating system loads user programs into these frames depending on which instruction is currently executing or which data needs to be retrieved. At any point in time – while the program is running – only those needed portions (i.e. pages) are present (or brought) in memory and the rest are kept on disk. This trick allows a limited physical memory to fit multiple (large) programs to execute at the same time.

What is segmentation ?

Segmentation is similar to paging in terms of dividing the memory into manageable smaller units hover the major difference is that in segmentation the division is logical just like chapters of a book. You may have a segment for the main function, a segment for a data table, a segment for a stack, a library and so on. Can you see the difference? a page on the other hand may cut a logical unit into two parts because pages are not aware of any logical boundaries.

Segmentation vs paging

Programs have no choice as paging and segmentation are both operating system features however as a programmer you can design your program in a way to better utilize the operating system and the underlying CPU architecture. For example, an operating system may not support segmentation. I think (please correct me in the comments section below) that Linux does not support segmentation and only supports demand paging. Regarding dedicating segments for code, data and stack, I also think it is implemented using paging. On the architecture side, I think (again correct me if I am

wrong) that ARM architecture does not support segmentation. I mean by “an architecture supporting segmentation or paging” is that the architecture provides the necessary hardware (ex. registers) to compute the required translation from logical addresses to physical addresses.

Paging vs segmentation

I assume at this point we know what paging and segmentation are and the context in which these two techniques are used. Let us now summarize their features and differences in an easy to memorize tabular form:

Feature	Paging	Segmentation
Design goal	Utilize limited physical space	security or protection and sharing
Address space	Sequence of fixed sized units called pages	Number of variable size unity called segments
Logical Address	Single integer number	Segment and offset within
Difficulty	Simple as all processes share one linear address space	Complex involving multiple address spaces
View	System view transparent to the programmer	Programmer has to be aware of it
Fragmentation	Reduces fragmentation because of equal sized pages	Fragmentation because segments are not uniform
Security	Hard to protect individual functions and data	Easy because segments are logically independent
Data sharing	Hard to share data and procedures between users	Easy because segments are logically independent
Page/Segmentation fault	Requested page is not present in memory	Logical address beyond the permitted address space

The goal of this article was to clarify the main difference between paging and segmentation in operating systems. If you are interested in more details I recommend that you research the following topics

- Demand paging vs anticipatory paging
- Page faults and swapping
- Page table and segment map table
- Paged segmentation

Summary

Without being fancy, a one line summary: segmentation is a virtual memory management technique for better code/data isolation and protection while paging is mainly for allocating more memory than what is physically available. That is all, if you liked the article or have questions/comments, please let me know. You can use the comments section below. Thanks for reading.

References

- Virtual memory