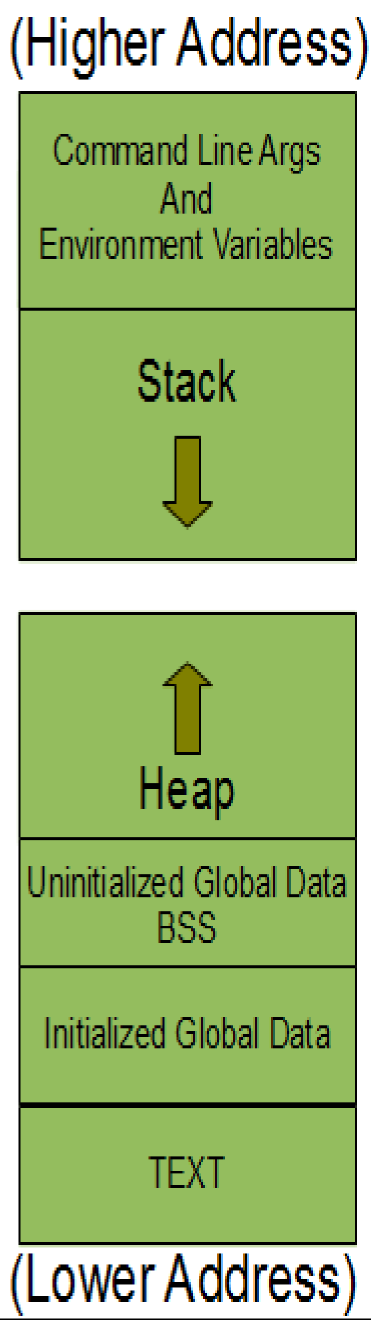
Memory Layout of a Process

The memory layout of a process in Linux can be very complicated if we try to present and describe everything in detail. So, here we will present only the stuff that has significant importance.

If we try to visualize the memory layout of a process, we have something like this:



The **command line arguments and the environment variables** are stored at the top of the process memory layout at the higher addresses.
The **stack segment** is the memory area which is used by the process to store the local variables of function and other information that is saved every time a function is called. This other information includes the return address ie the address from where the function was called, some information on the caller's environment like its machine registers etc are stored on stack. Also worth mentioning here is that each time a recursive function is called a new stack frame is generated so that each set of local variables does not interfere with the any other set.
The **heap segment** is the one which is used for dynamic memory allocation. This segment is not limited to a single process, instead it is shared among all the processes running in the system. Any process could dynamically allocate memory from this segment. Since this segment is shared across the processes so memory from this segment should be used cautiously and should be deallocated as soon as the process is done using that memory.
All the global variable which are not initialized in the program are stored in the **BSS segment**. Upon execution, all the uninitialized global variables are initialized with the value zero. Note that BSS stands for 'Block Started by Symbol'.
All the initialized global variables are stored in the **data segment**.
Finally, the **text segment** is the memory area that contains the machine instructions that CPU executes. Usually, this segment is shared across different instances of the same program being executed. Since there is no point of changing the CPU instructions so this segment has read-only privileges.

As seems from the figure above, the stack grows downwards while the heap grows upwards.

Memory Layout of a Process in Physical memory (RAM)

A virtual memory scheme splits the memory used by each program into small,fixed-size units called **pages**.

Correspondingly, RAM is divided into a series of **page frames** of the same size.

At any one time, only some of the pages of a program need to be resident in physical memory page frames; these pages form the so-called **resident set**.

Copies of the unused pages of a program are maintained in the **swap area**—a reserved area of disk space used to supplement the computer's RAM—and loaded into physical memory only as required.

Processes are isolated from one another and from the kernel, so that one process can't read or modify the memory of another process or the kernel. This is accomplished by having the page-table entries for each process point to distinct sets of physical pages in RAM

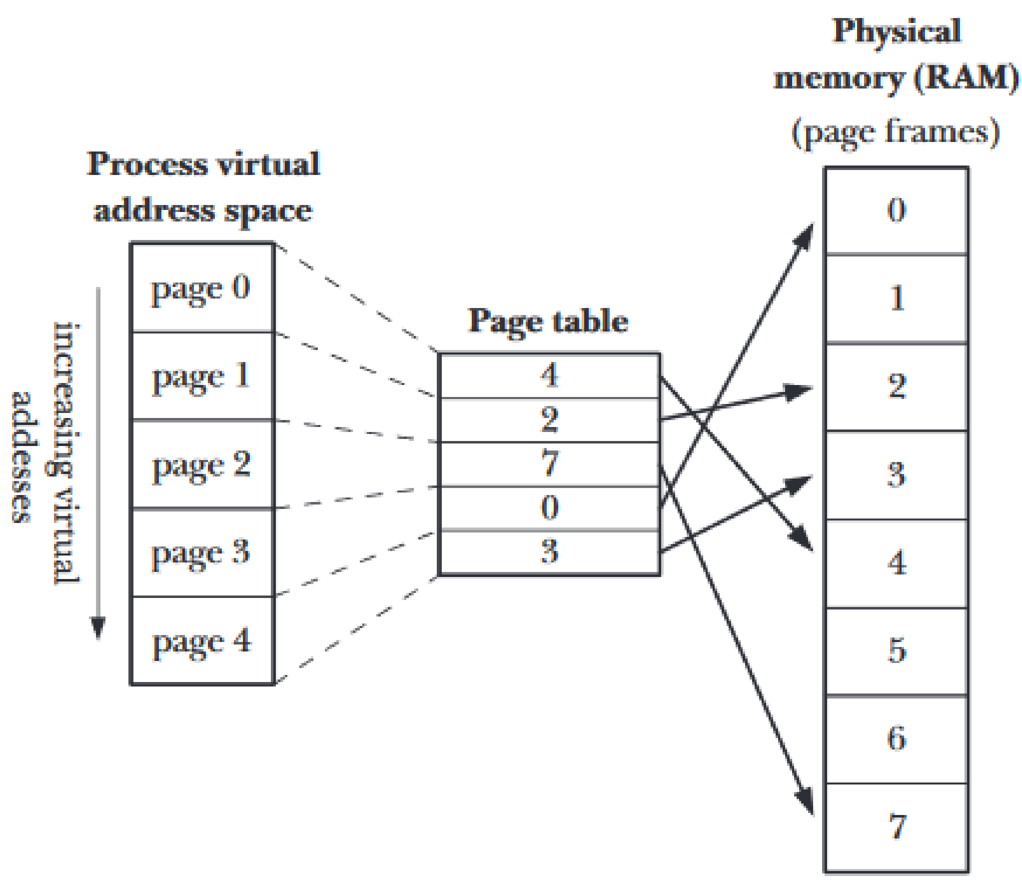


Figure 6-2: Overview of virtual memory