Log Book Week 5

Jose Devian Hibono
1706039603
System Programming - C

This week's material is about memory. First I read the slides from SCeLE and did the pre-test using the resources on the internet. Here are the resources I gathered, I tried to write my own understandings in this log book.

Memory Structure of a Process

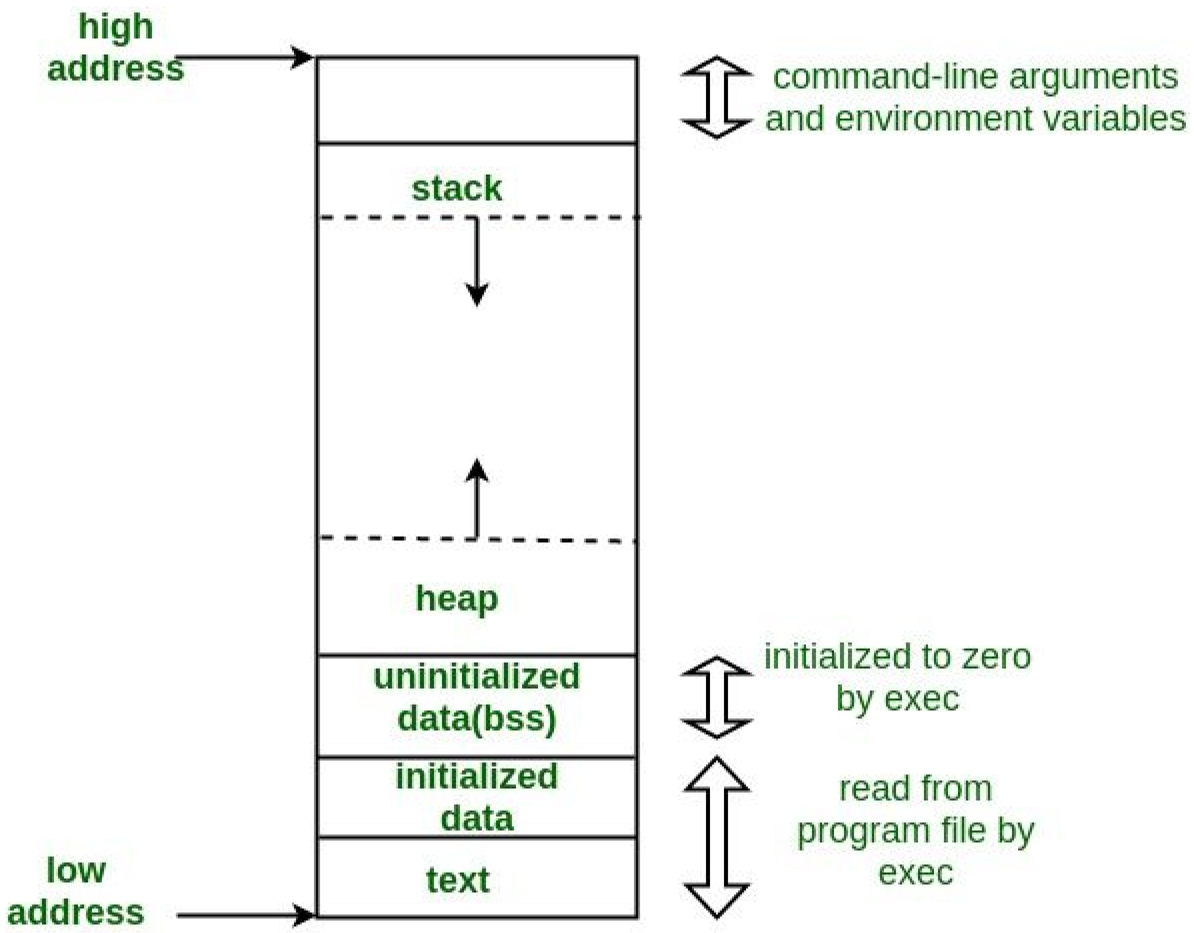
high address
command-line arguments and environment variables
stack
heap
uninitialized data(bss)
initialized to zero by exec
initialized data
text
read from program file by exec
low address

Image Source: GeeksForGeeks

1. Text Segment
   One of the parts of a program in an object file or in memory that contains executable instructions is a text segment, often referred to as a code segment or simply as text. As a memory area, a text segment can be positioned below the heap or stack to avoid overwriting of heaps and stack overflows.
2. Initialized Data Segment
   Initialized Data Segment or generally referred to simply as the Data Segment. A data segment is a part of a program's virtual address space that includes the user's global variables and static variables that have been initialized. Because the values of the variables can be altered at runtime, the data section is not read-only.
3. Uninitialized Data Segment
   Uninitialized data section, also referred to as the "bss" section, named after an old assembler operator that stood for "block started by symbol." The kernel initializes data in this segment to arithmetic 0 until the program begins executing uninitialized data at the end of the data segment and includes all global variables and static variables initialized to zero or not.
4. Stack
   The stack area was typically relatively close to the heap area and grew in the opposite direction; free memory was depleted when the stack pointer met the heap pointer. (They can be positioned almost anywhere with modern wide address spaces and virtual memory techniques, but they still usually expand in opposite directions.) The stack area contains the program stack, a LIFO structure, typically located in the higher memory sections.
5. Heap
   Heap is the segment where it normally takes place to assign dynamic memory. At the end of the BSS segment, the heap area begins and grows from there to greater addresses.

Spatial and Temporal Locality

Spatial Locality ensures that all those commands which are stored nearby to the recently performed command have high chances of execution. It refers to the use of data elements that are comparatively close in storage locations. Temporal Locality ensures that a freshly executed command has a strong probability of execution again. The command is then kept in cache memory such that it can quickly be accessed and does not take time to check for the same command.

Spatial	Temporal
In Spatial Locality, it is possible that neighboring instructions for recently executed	It is likely that a recently executed instruction will very soon be executed again.

instructions will soon be executed.	
This relates to the trend of execution involving a variety of memory locations.	It refers to the trend of execution where there is access to memory locations that have recently been used.
It is also known as locality in space.	It is also known as locality in time.
It applies only to data objects in memory that are closed together.	In a short time period, it refers repeatedly to the same results.
New data comes into effect every time.	Each time the same useful data comes into execution.

Memory Leaks

A memory leak is a type of resource leak that happens when a computer program handles memory allocations inappropriately in a way that does not release memory that is no longer needed. A memory leak can occur when an object is stored in the memory, but the running code can not access it. Memory leak can also occur when programmers create a memory in heap and forget to terminate it.

Physical and Virtual Memory

The advantage of using virtual memory is the program size could be larger than the main memory size. Memory can be used easily and only when it has to be loaded into the CPU is a part of the program loaded. Virtual memory allows unlimited numbers of multi-programming for sharing of code and data. With segmented paging, we can reduce internal fragmentation and suppress external fragmentation. The use of virtual memory has several drawbacks, which have increased overhead for the management of paging interrupts, software complexities and hardware costs. We're still using virtual memory because physical memory is not large enough to compensate for the need for the memory used. Larger physical memory is expensive so we need to cut the cost by using virtual memory. Virtual memory uses hardware and software to allow a device to substitute for physical memory shortages, by temporarily moving data from random access memory (RAM) to disk storage. In essence, virtual memory allows a machine to handle secondary memory as if it were the main memory.

`malloc()`

When a block is malloc-ated, it simply allocates a little more memory than you requested. This extra memory is used to store information such as the allocated block size, and a link in a block chain to the next free/used block, and often some "guard data" that lets the system detect whether you are writing past the end of your allocated block. To free unused memory, it uses the address when you free your pointer to find the specific information it has applied to the starting (usually) of your allocated block. If you pass a different address, it will access the garbage-containing memory and thus its behavior is unknown (but most often results in a crash).