Log Book Week 4

Jose Devian Hibono
1706039603
System Programming - C

This week's material is about processes in 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.

Process Memory

A program can contain a lot of processes, a process contains a lot of threads. A program is an executable file containing a series of instructions written on your machine to perform a specific job. Process is an executing instance of a program. A thread is the smallest executable unit of a process. A program is a file containing a variety of details explaining how a process can be implemented at runtime.

Process ID and Parent Process ID

`getpid()` returns the process ID of the calling process while `getppid()` returns the process ID of the parent of the calling process.

Fork

A `fork()` system call is used to construct a new process, called the child process, which runs simultaneously with the `fork()` calling process (parent process).

Memory Layout of the Process

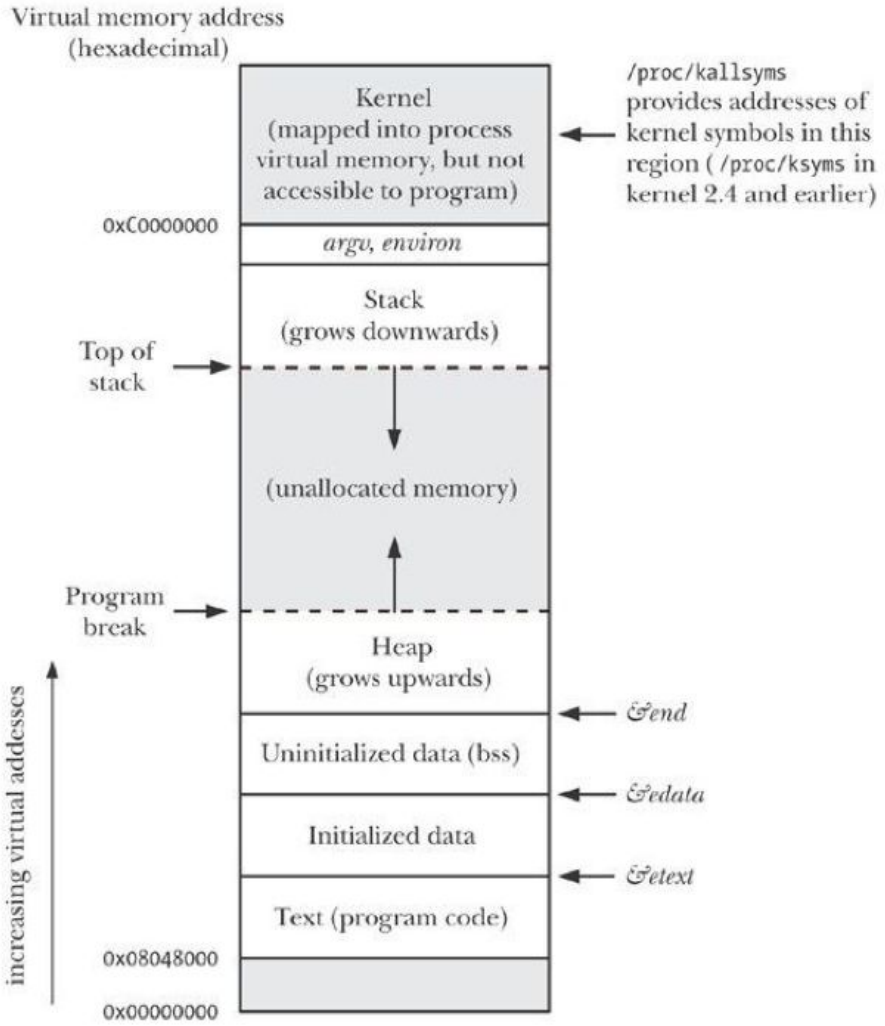

Figure 6-1. Typical memory layout of a process on Linux/x86-32

Taken from Course's Slides

1. At the top of the process memory layout, the command line arguments and the environment variables are stored at the higher addresses.
2. Then comes the segment of the stack. This is the memory region that the process uses to store the function's local variables and other information that is stored each time a function is called. This other information includes the return address, i.e. the address from where the function was called, certain information such as the machine registers, etc. on the caller setting is stored on the stack. It is also worth noting here that a new

stack frame is created each time a recursive function is called so that each set of local variables does not interfere with any other set.

3. The heap section is the one used for the allocation of dynamic memory. This section is not limited to a single process but is shared between all processes running on the system. Any process could allocate memory from this segment dynamically. Since this segment is shared around the processes, a memory from this segment should be used carefully and transferred as soon as that memory is used to complete the process.
4. As it seems from the above figure, while the heap grows upward, the stack grows downwards.
5. In the BSS segment, every global variable that hasn't initialized in the program is stored. All the uninitialized global variables are initialized with a value of zero upon execution. Note that BSS stands for 'Block Started by Symbol'.
6. In the data segment, all the initialized global variables are stored.
7. Finally, the text segment is the memory area containing instructions from the machine executed by the CPU. Typically, this section is spread across multiple instances of execution of the same program. Since the CPU instructions do not need to be modified, this section has read-only privileges.

Virtual Memory

Virtual Memory is utilized in `fork()`. There's a technique called copy-on write. Copy-on-write is a resource management technique to optimize memory usage. The fundamental principle is that you can send them pointers to the same resource if multiple callers ask for resources which are initially indistinguishable. This function can be retained until a caller attempts to modify its "copy" of the resource, at which point a true private copy is generated to prevent the modifications from being accessible to anyone else. Much of this happens to the callers transparently. The primary benefit is that no private copy can ever be produced if a caller never makes any changes. Linux uses this technique when it uses clone() to implement fork(). When calling fork(), it will create a parent and child process, both will run the process on the same page memory.

Memory Allocation

For memory allocation, Linux provides a variety of APIs. You can use `kmalloc` or `kmem_cache_alloc` families to allocate small chunks, use `vmalloc` and its derivatives to allocate wide, practically contiguous areas, or you can directly request pages from the allocation page allocator. More specialized allocators can also be used, such as `cma_alloc` or `zs_malloc`.

Issues

Memory Leaks: This occurs usually when a memory is full but has to take an additional entry. There are other causes such as failing to allocate the memory.