Linux Notes

Tommy Bui

10-10-2022

1 High-level View of Unix Environment

- To best understand how an operating system works, keep in mind the concept of abstraction.
  - Abstraction focuses on the basic purpose and operation of an object.
- There are many times for an abstracted subdivision in software. In these notes, the term **component**
- This chapter provides a high-level overview of the components that make up a Linux system.

1.1 Levels and Layers of Abstraction in Linux

- Abstraction helps break down the Linux operating system into easy-to-understand components.
- We arrange components into layers or levels, classifications or groupings of components according to where the components lay between the user and hardware.
  - i.e. Web browsers, games, etc. are at the top layer; the bottom layer consists of the memory in hardware which is composed of 0's and 1's.
- A Linux OS consists of 3 main levels:
  - The base consists of hardware:
    - Hardware includes the memory as well as the Central Processing Unit(s) to perform computation or RD/WR to memory.
    - Devices such as disks and network interfaces are also part of the hardware.
    - Examples of Hardware: CPU, main memory (RAM), Disks, Network ports, etc.
  - The next level up is the kernel:
    - The kernal is consider the software within memory that tells the CPU where to look for its next task.
    - As a mediator, the kernal manages hardware (i.e. main memory) and is the primary interface between hardware and any running program.
    - Linux Kernal contains: System calls, Process Management, Memory Management, and Device Drivers
  - Processes:
    - Running programs that are managed by the kernal, make up the system's upper level known as **user space** (i.e. all web servers run as **user processes**).

    - User Processes include GUI, Servers and Shell
  - The main difference between how the kernal and the user processes run is that the kernal runs in kernal mode and user processes run in user mode
  - Code running in kernal mode has unrestricted access to the processor and main memory. This can be powerful but is a dangerous privilege that can cause the kernal to easily corrupt and crash the entire system.
  - Memory area that the only the kernal can access is **kernal space**
  - Unlike kernal mode, user mode is restricted to a subset of memory and safe CPU operations
    - The Linux kernal can run kernal threads, which are similar to processes but have access to kernal space (i.e. kthreadd and kblockd)
  - *User space* refers to the parts of the main memory that user processes can access. If a process were to crash, the consequences are limited and can be repaired by the kernel
    - i.e. If your web browser crashes, it won't stop the scientific computation background processes that has been running for days
    - In theory, a user process gone haywire can't damage the majority of the system. However, user processes may affect other parts of your system
    - i.e. With the correct permissions, a user process can damage data on a disk

1.2 Hardware: Understanding Main Memory

- In it's rawest form, main memory is a giant storage of bits.
- All input & output from peripheral devices flows through main memory is also in form of bits.
- The CPU operates on memory; it reads instructions & data from the memory and writes data back to the memory.
- The term state in reference to memory, processoes, the kernal, etc. refers to the particular arrangement of bits.

1.3 The Kernel

Nearly everything the kernel does revolves around the main memory. One of the kernel's task is to partition memory into subdivisions and it must maintain certain state information about those subdivisions at all times. Each process gets its own share of memory & the kernel manages each process' memory.

The kernal is in charge of managing tasks in four general system areas:

- **Processes:** The kernel is responsible for determining which proccess can use the CPU.
- **Memory:** The kernel needs to keep track of all memory; Memory can be shared between processes even if allocated to a particular process.
- **Device drivers:** The kernel acts as an interface between hardware & processes. The kernel usually operates the hardware.
- **System calls & support:** Processes normally use system calls to communicate with the kernel.

1.3.1 Process Management

- Process management describes the starting, pausing, resuming, scheduling, & terminating of processes.
- Explaining the concepts behind starting & terminating processes are straightfoward, but describing how a process uses the CPU in its operation is more complex.
- i.e. On any OS, may processes run simultaneoulsy on a desktop computer at the same time. However, things are not as they appear: the processes behind these applications typically do not run at the exact same time.
- Consider a system with a one-core CPU. Many processes may be able to use the CPU, **but only one process an actually use the CPU at any given time**.
- In practice, each process uses the CPU for a fraction of a second, pauses, then another process uses the CPU for a fraction of time and so on.
- The act of one process giving up control of the CPU to another process is called a **context switch**. Each piece of time (called a time slice) gives a process enough time for significant computation (a process often finishes its current task during a single slice).
- Since the slices are so small, people cannot perceive them and the system seems to be running multiple processes at the same time.
- The kernal is responsible for context switching. Consider a process that is running in user mode but its time slice is up:
    1. The CPU (Hardware) interrupts the current process based on an internal timer, and returns control back to the kernel by switching into kernel mode.
    2. The kernel records the current state of the CPU & memory, which will be essential to resuming the process that was just interrupted.
    3. The kernal performs any tasks that might have come up during the preceding time slice (such as collecting data from IO operations.

4. The kernal is now ready to let another process run. The kernel analyzes the list of processes that are ready to run and chooses one.
5. The kernel prepares the memory for this new process and then prepares the CPU.
6. The kernel tells the CPU how long the time slice for the new process will be.
7. The kernel switches the CPU into user mode and hands control of the CPU to the process.

- Context switch answers the important questions of when the kernel runs. The answer is that it runs between process time slices during a context switch.
- In the case of a mult-CPU system (which most machines are), things become more complicated because the kernel deosn't need to relinquish control of its current CPU in order to allow a process to run on a different CPU, & more than one process may run at a time. However to maximuze the usage of all available CPUs, the kernel typically performs these steps anyway.

1.3.2 Memory Management

- In order for the kernel to manage memory during a context switch (which can be a complex job), The following conditions must hold:
  - The kernel must have its own private area in memory that user processes cannot access
  - Each user process needs its own section of memory
  - One user porcess may not access the private memory of another process
  - User processes can share memory
  - Some memory in user processes can be read-only
  - The system can use more memory that is physically present by using disk space as auxiliary
- Fortunately for the kernel, modern CPUs include a memory management unit (MMU) that enables a memory access scheme called virtual memory.
- When using virtual memory, a process does not directly access the memory by its physical location in hardware, instead the kernel sets up each process to act as if it had an entire machine to itself. When the process accesses some of its memory, the MMU intercepts the access & uses a memory address map to translate the memory location from the process point of view into an actual physical memory location in the machine.

- The kernel must still initialize & continuously maintain & alter this memory address map (i.e. During a context switch, the kernel has to change the map from outgoing process to the incoming process.
- The implementation of memory address maps is called a page table (which is covered in chapter 8).