10.1.2 Process Heredity

Click one of the buttons to take you to that part of the video.

Process Heredity 0:00-2:22

In this lesson, we are going to discuss the heredity of Linux processes. Now, here's the key thing I want you to take away from this lesson: all Linux processes are directly or indirectly loaded by one single process on your Linux system. Depending upon which distribution you are using, it could be either the init process or the systemd process.

This process, either init or systemd, is started by the Linux kernel when the system first boots up. Understand that some distributions do still use init, whereas most newer Linux distributions have migrated to systemd. It's actually been a bit of a holy war going on in the Linux community over which one is best.

At this time, systemd is starting to win out over init in terms of adoption. It's got a lot of benefits to it that once you get used to using systemd, you realize it's much better than init.

I suspect that init's days are numbered, but there are still a lot of systems out there that have been deployed with Linux distributions based on init. You need to be familiar with init as well as systemd.

The thing that you need to understand is that any process running on a Linux system is allowed to launch additional processes. The process that launched the new process is called the parent, while the new process itself is called the child.

Now this parent-child relationship constitutes the heredity, if you will, of Linux processes, because any process, including new child processes, can launch additional processes. It's therefore possible to have many generations of processes running on the system.

In this example, we have a parent process up here, and it spawned three child processes. Then each of these three child processes spawned child processes of their own. This one launched one, this one launched two, this one launched one.

This process makes the first process up here, the original parent process a grandparent. Do you see now why we call it a heredity of processes? Therefore, for any process running on a Linux system, we need to be able to uniquely identify it, as well as the parent process that launched it.

Process Identifies 2:20-7:12

Whenever a process is created on a Linux System it will be assigned two different resources. First, we have the process ID number, or the PID. This is the number that's assigned to each process that is unique. This is a number that's uniquely assigned to each process.

In other words, no two processes running on the system at the same time will have the same PID number. In addition to the PID, we also have the parent process ID. This is the PID number of the process that launched that process.

It's the process ID of that process's parent. It's the process that spawned it. Now, by assigning these two numbers to every running process, we can track the heredity of that process through the system.

The kernel itself uses something called a process table to keep track of all the processes running on the system. And this process table is maintained in memory by the operating system to facilitate switching between processes, scheduling processes, and prioritizing those processes.

Each entry in the table contains information about the specific process itself, such as its name, its state, the priority that it has on the system, as well as the memory addresses used by that process.

Be aware that there really is a grandparent process on a Linux system that spawns all the other processes that are running on that system. Depending upon which distribution you're using, it could be either systemd as we talked about earlier, or it could be init. Most of your newer distributions are going to use systemd, older distributions will probably still be using init.

When your Linux system first boots up, the kernel process is initially created. This kernel process loads either systemd, as shown in this example, or init, if you are using an older Linux distribution, when the system first boots up. Then this process, either systemd or init, will launch all other child processes.

These child processes will launch other processes. An example of that is shown here. In this case, systemd launches a login shell. Then within that login shell we launch a bash shell, in which runs the vi process to provide a text editor for the end-user to use to edit files.

Whenever you launch a process on a Linux system, its process ID number is going to be assigned randomly from the operating system's table of available PID numbers. However, the systemd process or the init process, depending upon your distribution, will always be assigned a PID process ID number of one.

This brings up an interesting point. If the systemd or init process is the first process on the system from which all other processes descend as shown here, then what is systemd or init's parent process ID? Does it even have one? Actually, it does.

Because the systemd or init process is launched directly by the Linux kernel, which by the way has a PID always of zero. Then the PPID of systemd or init will always be zero, the kernel process.

Now the systemd or init process is responsible for launching all system processes that are configured to automatically start on boot. As shown here, it also creates a login shell that users can use to log into the system.

This brings up an important point. Notice that I have a login shell that I use to login, and then we've placed a second shell right here, the bash shell, in which the vi editor is being run.

You might be asking, "Why didn't you just run vi right here, right out of the login shell? Should we just put vi over here and not have a shell in between the two?"

Actually, in this figure, vi was launched from within this login shell. If we did that, then why do we have this bash shell right down here between the login shell and the vi process?

Because anytime you run a command from within any shell--it doesn't matter if it's a login shell or if it's a standard shell session or terminal shell session, it doesn't matter--a second subshell is created and the process for the command you entered is run within it.

In this case, I ran vi so I had to create a subshell from the login shell and run vi in it. If I were to run top from the login shell, the same thing would happen. I would create another bash shell and then top would run within it.

Process of Forking 7:11-9:12

The important thing to remember is that the subshell that we create is a separate process in and of itself. It has its own process ID assigned, and the parent process ID of the subshell is the process ID of the original shell where the command was entered. In this example, I'm going to run the vi program from within this bash shell session.

This bash shell session has a process ID of 567, and its parent ID--it's parent process ID--is 234. Within this bash shell I run the vi command. When this happens, a second bash subshell is created, and it's assigned whatever process ID number is the next one that's available.

In this case, let's assume that it was 591. Well because this subshell was created by the process for this shell, the parent process ID number of the subshell is the process ID number of the original shell, 567.

Then within that subshell, the actual process for the vi text editor is created, and it's assigned its own process ID number--in this case, 598. Because it was created by the bash subshell process, its parent process ID is the process ID number of the subshell, 591.

This whole entire process is called forking. If you hear the term forking used in the Linux world with relation to processes, this is what it's talking about.

In this situation, we've run the vi command in the original bash shell, we've got our bash subshell created, and then the vi process created by it. We used the vi editor to do whatever it is we want to do, and then when we're done we exit out of the vi text editor. When the vi command is exited, then this subshell is terminated and control is returned back to the original shell session.

Summary 9:11-9:29

That's it for this lesson. This lesson we talked about how Linux handles processes. We first talked about the heredity of Linux processes. We talked about the process ID numbers, both the PID and the PPID. Then we ended this lesson by talking about the process of forking whenever we create a new process within a shell session.
