

The Wayback Machine - <https://web.archive.org/web/20071011225814/http://www.linux-tutorial.info/modules.php?name=MConten...>


[Create an Account](#)
[Home](#) | [Submit News](#) | [Your Account](#)

October 11, 2007

Tutorial Menu

[Linux Tutorial Home](#)
[Table of Contents](#)
[Up to -->](#)
[Introduction to Operating Systems](#)

[Processes](#)
[Virtual Memory Basics](#)

[Glossary](#)
[MoreInfo](#)
[Man Pages](#)
[Linux Topics](#)
[Test Your Knowledge](#)

Site Menu

[Site Map](#)
[FAQ](#)
[Copyright Info](#)
[Terms of Use](#)
[Privacy Info](#)
[Disclaimer](#)
[WorkBoard](#)
[Thanks](#)
[Donations](#)
[Advertising](#)
[Your Account](#)

Communication

[Feedback](#)
[Forums](#)
[Private Messages](#)
[Recommend Us](#)
[Surveys](#)

Features

[HOWTOs](#)
[News](#)
[News Archive](#)
[NukeSentinel](#)
[Reviews](#)
[Submit News](#)
[Topics](#)
[User Articles](#)
[Web Links](#)

Google



☐ The Web

Linux Tutorial - Introduction to Operating Systems - Processes

Processes

One basic concept of an operating system is the process. If we think of the program as the file stored on the hard disk or floppy and the process as that program in memory, we can better understand the difference between a program and a process. Although these two terms are often interchanged or even misused in "casual" conversation, the difference is very important for issues that we talk about later. Often one refers to an *instance* of that command or program.

A process is more than just a program. Especially in a multi-user, multi-tasking operating system such as UNIX, there is much more to consider. Each program has a set of data that it uses to do what it needs. Often, this data is not part of the program. For example, if you are using a text editor, the file you are editing is not part of the program on disk, but is part of the process in memory. If someone else were to be using the same editor, both of you would be using the same program. However, each of you would have a different process in memory. See the figure below to see how this looks graphically.

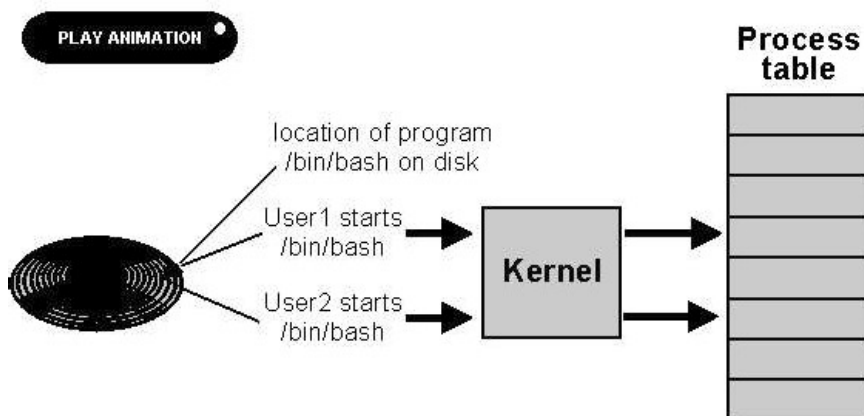


Image - Reading programs from the hard disk to create processes. (interactive)

Under UNIX, many different users can be on the system at the same time. In other words, they have processes that are in memory all at the same time. The system needs to keep track of what user is running what process, which terminal the process is running on, and what other resources the process has (such as open files). All of this is part of the process.

With the exception of the init process (PID 1) every proc another process. In general, every process has the poter of another process. Perhaps the program is coded in such never start another process. However, this is a limitation and not the operating system.

When you log onto a UNIX system, you usually get acces interpreter, or shell. This takes your input and runs progr are familiar with DOS, you already have used a common COMMAND.COM program. Under DOS, your shell gives y something similar). Under UNIX, the prompt is usually sc

Login



Don't have an account yet? You can create one. As a registered user you have some advantages like theme manager, comments configuration and post comments with your name.

Help if you can!



Amazon Wish List



Did You Know?

You can get all the latest Site and Linux news by checking out our news page.

The Wayback Machine - <https://web.archive.org/web/20071011225814/http://www.linux-tutorial.info/modules.php?name=MConten...>


[Create an Account](#)
[Home](#) | [Submit News](#) | [Your Account](#)

October 11, 2007

Tutorial Menu

[Linux Tutorial Home](#)
[Table of Contents](#)
[Up to -->](#)
[Introduction to Operating Systems](#)

[Processes](#)
[Virtual Memory](#)
[Basics](#)

[Glossary](#)
[MoreInfo](#)
[Man Pages](#)
[Linux Topics](#)
[Test Your Knowledge](#)

Site Menu

[Site Map](#)
[FAQ](#)
[Copyright Info](#)
[Terms of Use](#)
[Privacy Info](#)
[Disclaimer](#)
[WorkBoard](#)
[Thanks](#)
[Donations](#)
[Advertising](#)
[Your Account](#)

Communication

[Feedback](#)
[Forums](#)
[Private Messages](#)
[Recommend Us](#)
[Surveys](#)

Features

[HOWTOs](#)
[News](#)
[News Archive](#)
[NukeSentinel](#)
[Reviews](#)
[Submit News](#)
[Topics](#)
[User Articles](#)
[Web Links](#)

Google



☐ The Web

Linux Tutorial - Introduction to Operating Systems - Processes

Processes

One basic concept of an operating system is the process. If we think of the program as the file stored on the hard disk or floppy and the process as that program in memory, we can better understand the difference between a program and a process. Although these two terms are often interchanged or even misused in "casual" conversation, the difference is very important for issues that we talk about later. Often one refers to an *instance* of that command or program.

A process is more than just a program. Especially in a multi-user, multi-tasking operating system such as UNIX, there is much more to consider. Each program has a set of data that it uses to do what it needs. Often, this data is not part of the program. For example, if you are using a text editor, the file you are editing is not part of the program on disk, but is part of the process in memory. If someone else were to be using the same editor, both of you would be using the same program. However, each of you would have a different process in memory. See the figure below to see how this looks graphically.

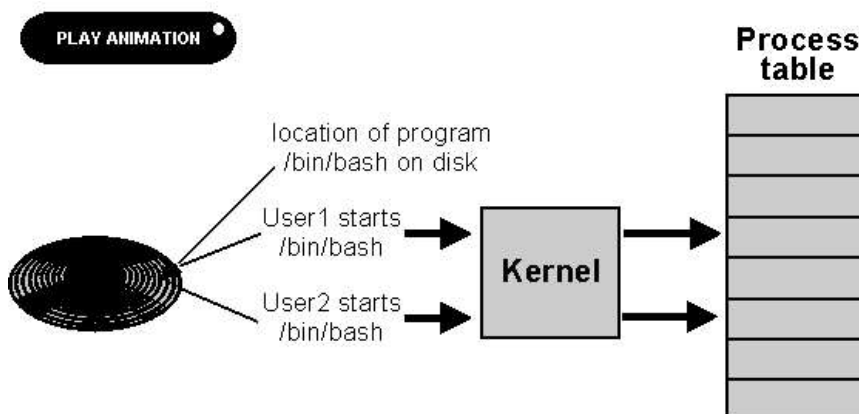


Image - Reading programs from the hard disk to create processes. (interactive)

Under UNIX, many different users can be on the system at the same time. In other words, they have processes that are in memory all at the same time. The system needs to keep track of what user is running what process, which terminal the process is running on, and what other resources the process has (such as open files). All of this is part of the process.

With the exception of the init process (PID 1) every process is the child of another process. In general, every process has the potential to be the parent of another process. Perhaps the program is coded in such a way that it will never start another process. However, this is a limitation of that program and not the operating system.

When you log onto a UNIX system, you usually get access to a command line interpreter, or shell. This takes your input and runs programs for you. If you are familiar with DOS, you already have used a command line interpreter: the COMMAND.COM program. Under DOS, your shell gives you the C:> prompt (or something similar). Under UNIX, the prompt is usually something like \$, #, or

Login



Don't have an account yet? You can create one. As a registered user you have some advantages like theme manager, comments configuration and post comments with your name.

Help if you can!



Amazon Wish List



Did You Know?

You can get all the latest Site and Linux news by checking out our news page.

Friends



● linux-tutorial.info

Who's Online

There are currently, 228 guest(s) and 2 member(s) that are online.

You are an Anonymous user. You can register for free by clicking [here](#)

%. This shell is a process and it belongs to you. That is, the in-memory (or in-core) copy of the shell program belongs to you.

If you were to start up an editor, your file would be loaded and you could edit your file. The interesting thing is that the shell has not gone away. It is still in memory. Unlike what operating systems like DOS do with some programs, the shell remains in memory. The editor is simply another process that belongs to you. Because it was started by the shell, the editor is considered a "child" process of the shell. The shell is *the parent* process of the editor. (A process has only one parent, but may have many children.)

As you continue to edit, you delete words, insert new lines, sort your text and write it out occasionally to the disk. During this time, the backup is continuing. Someone else on the system may be adding figures to a spreadsheet, while a fourth person may be inputting orders into a database. No one seems to notice that there are other people on the system. For them, it appears as though the processor is working for them alone.

Another example we see in the next figure. When you login, you normally have a single process, which is your login shell (bash). If you start the X Windowing System, your shell starts another process, xinit. At this point, both your shell and xinit are running, but the shell is waiting for xinit to complete. Once X starts, you may want a terminal in which you can enter commands, so you start xterm.

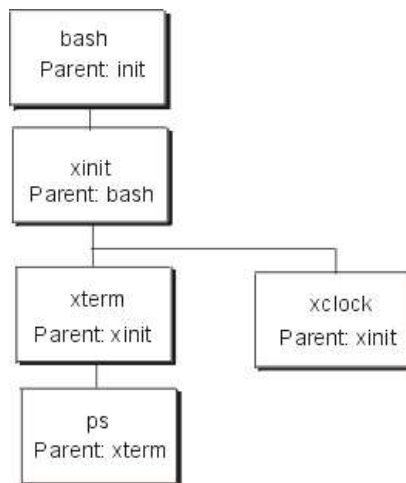


Image - Relationship between parent and child processes. **(interactive)**

From the xterm, you might then start the ps command, to see what other processes are running. In addition, you might have something like I do, where a clock is automatically started when X starts. At this point, your process tree might look like the figure above.

The nice thing about UNIX is that while the administrator is backing up the system, you could be continuing to edit your file. This is because UNIX knows how to take advantage of the hardware to have more than one process in memory at a time. (Note: It is not a good idea to do a backup with people on the system as data may become inconsistent. This was only used as an illustration.)

As I write this sentence, the operating system needs to know whether the characters I press are part of the text or commands I want to pass to the editor. Each key that I press needs to be interpreted. Despite the fact that I can clip along at about thirty words per minute, the Central Processing Unit(CPU) is spending approximately 99 percent of its time doing nothing.

The reason for this is that for a computer, the time between successive keystrokes is an eternity. Let's take my Intel Pentium running at a clock speed of 1.7 GHz as an example. The clock speed of 1.7 GHz means that there are 1.7 billion(!) clock cycles per second. Because the Pentium gets close to one instruction per clock cycle, this means that within one second, the CPU can get close to executing 1.7 billion instructions! No wonder it is spending most of its time idle. (Note: This is an oversimplification of what is going on.)

A single computer instruction doesn't really do much. However, being able to do 1.7 billion little things in one second allows the CPU to give the user an impression of being the only one on the system. It is simply switching between the different processes so fast that no one is aware of it.

Each user, that is, each process, gets complete access to the CPU for an incredibly short period of time. This period of time (referred to as a *time slice*) is typically 1/100th of a second. That means that at the end of that 1/100th of a second, it's someone else's turn and the current process is *forced* to give up the CPU. (In reality, it is much more complicated than this. We'll get into more details [later](#).)

Compare this to an operating system like standard Windows (not Windows NT/2000). The program will hang onto the CPU until it decides to give it up. An ill-behaved program can hold onto the CPU forever. This is the cause of a system hanging because nothing, not even the operating system itself, can gain control of the CPU. Linux uses the concept of *pre-emptive multi-tasking*. Here, the system can pre-empt one process or another, to let another have a turn. Older versions of Windows, use *co-operative* multi-tasking. This means the process must be "cooperative" and give up control of the CPU.

Depending on the load of the system (how busy it is), a process may get several time slices per second. However, after it has run for its time slice, the operating system checks to see if some other process needs a turn. If so, that process gets to run for a time slice and then it's someone else's turn: maybe the first process, maybe a new one.

As your process is running, it will be given full use of the CPU for the entire 1/100th of a second unless one of three things happens. Your process may need to wait for some event. For example, the editor I am using to write this in is waiting for me to type in characters. I said that I type about 30 words per minute, so if we assume an average of six letters per word, that's 180 characters per minute, or three characters per second. That means that on average, a character is pressed once every 1/3 of a second. Because a time slice is 1/100th of a second, more than 30 processes can have a turn on the CPU between each keystroke! Rather than tying everything up, the program waits until the next key is pressed. It puts itself to sleep until it is awoken by some external event, such as the press of a key. Compare this to a "busy loop" where the process keeps checking for a key being pressed.

When I want to write to the disk to save my file, it may appear that it happens instantaneously, but like the "complete-use-of-the-CPU myth," this is only appearance. The system will gather requests to write to or read from the disk and do it in chunks. This is much more efficient than satisfying everyone's request when they ask for it.

Gathering up requests and accessing the disk all at once has another advantage. Often, the data that was just written is needed again, for example, in a database application. If the system wrote everything to the disk immediately, you would have to perform another read to get back that same data. Instead, the system holds that data in a special buffer; in other words, it "caches" that data in the buffer. This is called the *buffer cache*.

If a file is being written to or read from, the system first checks the buffer cache. If on a read it finds what it's looking for in the buffer cache, it has just saved itself a trip to the disk. Because the buffer cache is in memory, it is substantially faster to read from memory than from the disk. Writes are normally written to the buffer cache, which is then written out in larger chunks. If the data being written already exists in the buffer cache, it is overwritten. The flow of things might look like this:

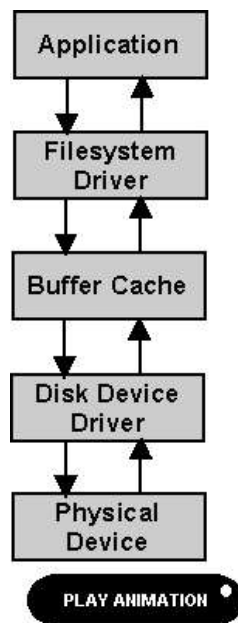


Image - Different layers of file access. (**interactive**)

When your process is running and you make a request to read from the hard disk, you typically cannot do anything until you have completed the write to the disk. If you haven't completed your time slice yet, it would be a waste not to let someone else have a turn. That's exactly what the system does. If you decide you need access to some resource that the system cannot immediately give to you, you are "put to sleep" to wait. It is said that you are put to sleep waiting on an event, the event being the disk access. This is the second case in which you may not get your full time on the CPU.

The third way that you might not get your full time slice is also the result of an external event. If a device (such as a keyboard, the clock, hard disk, etc.) needs to communicate with the operating system, it signals this need through the use of an interrupt. When an interrupt is generated, the CPU itself will stop execution of the process and immediately start executing a routine in the operating system to handle interrupts. Once the operating system has satisfied this interrupt, it returns to its regularly scheduled process. (Note: Things are much more complicated than that. The "priority" of both the interrupt and process are factors here. We will go into more detail in the section on the CPU.)

As I mentioned earlier, there are certain things that the operating system keeps track of as a process is running. The information the operating system is keeping track of is referred to as the process *context*. This might be the terminal you are running on or what files you have open. The context even includes the internal state of the CPU, that is, what the content of each register is.

What happens when a process's time slice has run out or for some other reason another process gets to run? If things go right (and they usually do), eventually that process gets a turn again. However, to do things right, the process must be allowed to return to the exact place where it left off. Any difference could result in disaster.

You may have heard of the classic banking problem concerning deducting from your account. If the process returned to a place *before* it made the deduction, you would deduct twice. If the process hadn't yet made the deduction but started up again at a point after which it would have made the deduction, it appears as though the deduction was made. Good for you, but not so good for the bank. Therefore, everything must be put back the way it was.

The processors used by Linux (Intel 80386 and later, as well as the DEC Alpha, and SPARC) have built-in capabilities to manage both multiple users and multiple tasks. We will get into the details of this in later chapters. For now, just be aware of the fact that the CPU assists the operating system in managing users and processes. This shows how multiple processes might look in memory:

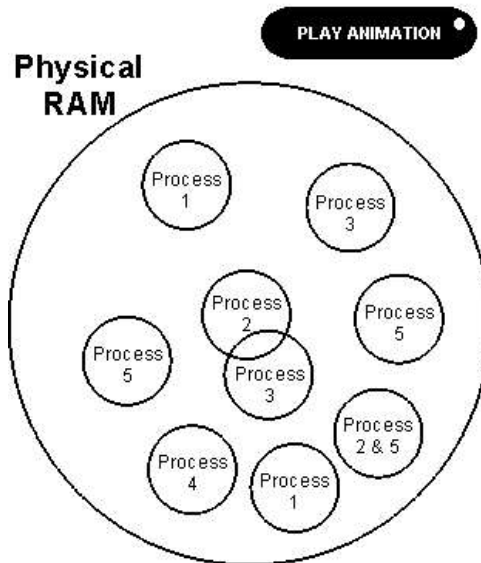


Image - Processes using differing areas of memory. (**interactive**)

In addition to user processes, such as shells, text editors, and databases, there are system processes running. These are processes that were started by the system. Several of these deal with managing memory and scheduling turns on the CPU. Others deal with delivering mail, printing, and other tasks that we take for granted. In principle, both of these kinds of processes are identical. However, system processes can run at much higher priorities and therefore run more often than user processes.

Typically a system process of this kind is referred to as a daemon process or background process because they run behind the scenes (i.e. in the background) without user intervention. It is also possible for a user to put one of his or her processes in the background. This is done by using the ampersand (&) metacharacter at the end of the command line. (I'll talk more about metacharacters in the section on shells.)

What normally happens when you enter a command is that the shell will wait for that command to finish before it accepts a new command. By putting a command in the background, the shell does not wait, but rather is ready immediately for the next command. If you wanted, you could put the next command in the background as well.

I have talked to customers who have complained about their systems grinding to a halt after they put dozens of processes in the background. The misconception is that because they didn't see the process running, it must not be taking up any resources. (Out of sight, out of mind.) The issue here is that even though the process is running in the background and you can't see it, it still behaves like any other process.

[Previous Page](#)
[What Is an Operating System](#)

[Back to Top](#)
[Table of Contents](#)

[Next Page](#)
[Virtual Memory Basics](#)

[MoreInfo](#)

[Test Your Knowledge](#)

User Comments:

Posted by pinklady on August 20, 2005 12:45pm:

I woluld like to say that creating this website is a really good idea. It contains a lot of information. I´ve just started and I´m stuck. Infact, I´m spreading the word. Well done, James.

Posted by checkmate444 on February 20, 2006 01:27pm:

Like the tutorial says, I have never given much thought as to the how's, why's and where's my os operates. It's nice to have this knowledge. AI
Checkmate444

You can only add comments if you are [logged in](#).

Copyright 2002-2004 by James Mohr. Licensed under modified GNU Free Documentation License (Portions of this material originally published by Prentice Hall, Pearson Education, Inc). See [here](#) for details. All rights reserved.

[MContent](#) ©



Is this information useful? How about a donation to the [Jessie Arbogast Medical Fund](#)? At the very least you can help by spreading the word to your favorite newsgroups, mailing lists and forums.

All logos and trademarks in this site are property of their respective owner. The comments are property of their posters. Articles are the property of their respective owners. Unless otherwise stated in the body of the article, article content © 1994-2006 by James Mohr. All rights reserved. The stylized page/paper, as well as the terms "The Linux Tutorial", "The Linux Server Tutorial", "The Linux Knowledge Base and Tutorial" and "The place where you learn Linux" are service marks of James Mohr. All rights reserved.

The Linux Knowledge Base and Tutorial may contain links to sites on the Internet, which are owned and operated by third parties. The Linux Tutorial is not responsible for the content of any such third-party site. By viewing/utilizing this web site, you have agreed to our [disclaimer](#), [terms of use](#) and [privacy policy](#). Use of automated download software ("harvesters") such as wget, httrack, etc. causes the site to quickly exceed its bandwidth limitation and are therefore expressly prohibited. For more details on this, take a look [here](#)

PHP-Nuke Copyright © 2004 by Francisco Burzi. This is free software, and you may redistribute it under the [GPL](#). PHP-Nuke comes with absolutely no warranty, for details, see the [license](#).

Page Generation: 0.96 Seconds