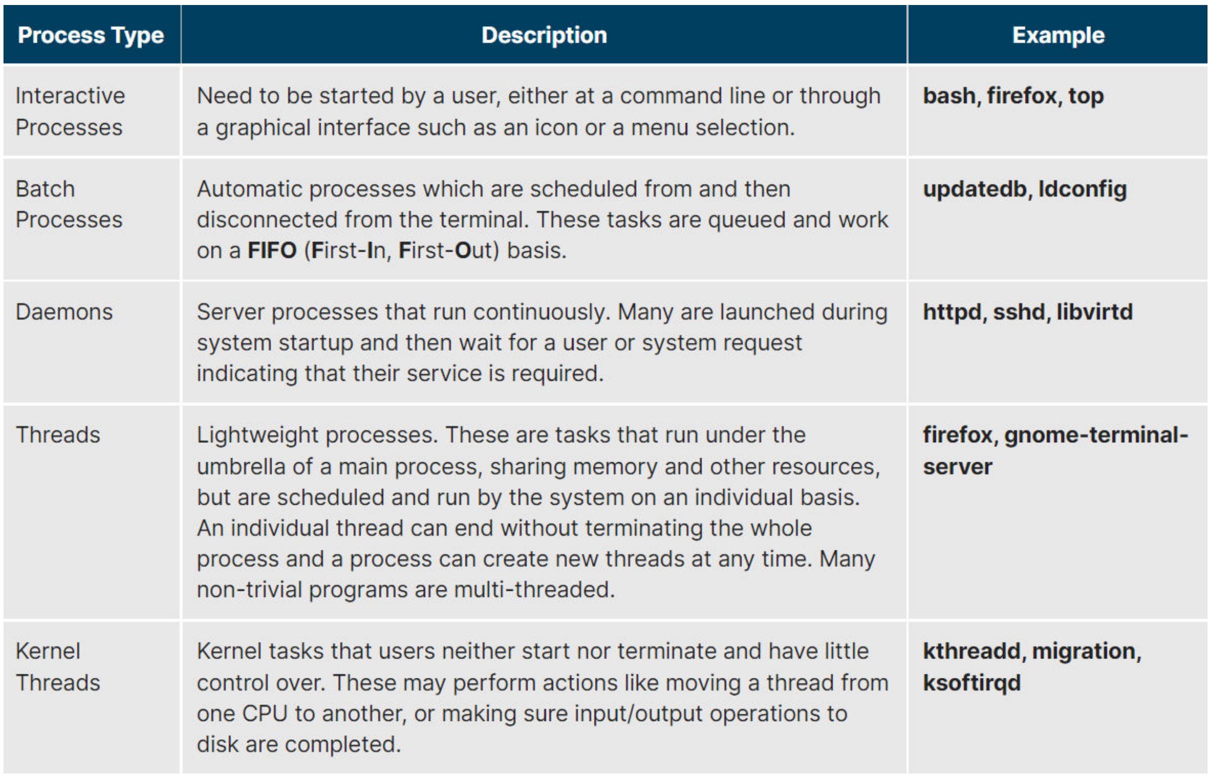
Processes

What Is a Process?

1. A process is simply an instance of one or more related tasks (threads) executing on your computer. It is not the same as a program or a command.
2. A single command may actually start several processes simultaneously.
3. Some processes are independent of each other and others are related.
4. A failure of one process may or may not affect the others running on the system.
5. Processes use many system resources, such as memory, CPU (central processing unit) cycles, and peripheral devices, such as network cards, hard drives, printers and displays.
6. The operating system (especially the kernel) is responsible for allocating a proper share of these resources to each process and ensuring overall optimized system utilization.

Process Types:

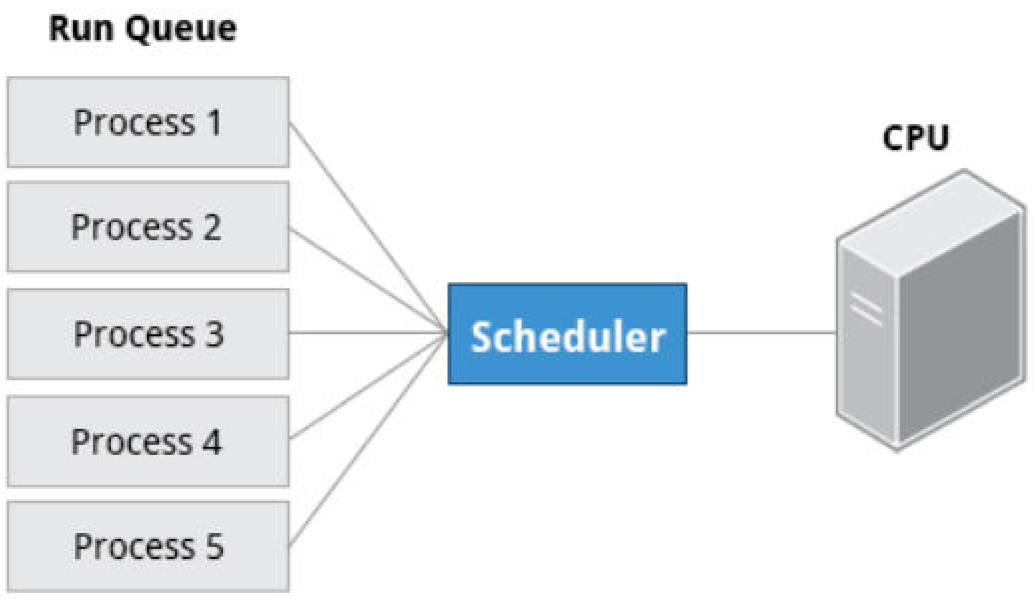
1. A terminal window (one kind of command shell) is a process that runs as long as needed. It allows users to execute programs and access resources in an interactive environment.
2. You can also run programs in the background, which means they become detached from the shell.
3. Processes can be of different types according to the task being performed.
4. The below are some different process types, along with their descriptions and examples:



Process Type	Description	Example
Interactive Processes	Need to be started by a user, either at a command line or through a graphical interface such as an icon or a menu selection.	**bash, firefox, top**
Batch Processes	Automatic processes which are scheduled from and then disconnected from the terminal. These tasks are queued and work on a **FIFO** (**F**irst-**I**n, **F**irst-**O**ut) basis.	**updatedb, ldconfig**
Daemons	Server processes that run continuously. Many are launched during system startup and then wait for a user or system request indicating that their service is required.	**httpd, sshd, libvirtd**
Threads	Lightweight processes. These are tasks that run under the umbrella of a main process, sharing memory and other resources, but are scheduled and run by the system on an individual basis. An individual thread can end without terminating the whole process and a process can create new threads at any time. Many non-trivial programs are multi-threaded.	**firefox, gnome-terminal-server**
Kernel Threads	Kernel tasks that users neither start nor terminate and have little control over. These may perform actions like moving a thread from one CPU to another, or making sure input/output operations to disk are completed.	**kthreadd, migration, ksoftirqd**

Process Scheduling and States:

1. A critical kernel function called the **scheduler** constantly shifts processes on and off the CPU, sharing time according to relative priority, how much time is needed and how much has already been granted to a task.
2. When a process is in a so-called **running** state, it means it is either currently executing instructions on a CPU, or is waiting to be granted a share of time (a time slice) so it can execute.
3. All processes in this state reside on what is called a run queue and on a computer with multiple CPUs, or cores, there is a run queue on each.
4. However, sometimes processes go into what is called a **sleep** state, generally when they are waiting for something to happen before they can resume. In this condition, a process is said to be sitting on a wait queue.
5. There are some other less frequent process states, especially when a process is terminating. Sometimes, a child process completes, but its parent process has not asked about its state. Amusingly, such a process is said to be in a zombie state; it is not really alive, but still shows up in the system's list of processes.



Process Scheduling and States

Process and Thread IDs:

1. At any given time, there are always multiple processes being executed.
2. The operating system keeps track of them by assigning each a unique process ID (PID) number.
3. The PID is used to track process state, CPU usage, memory use, precisely where resources are located in memory, and other characteristics.
4. New PIDs are usually assigned in ascending order as processes are born.

5. Thus, PID 1 denotes the init process (initialization process), and succeeding processes are gradually assigned higher numbers.
6. The below table explains the PID types and their descriptions

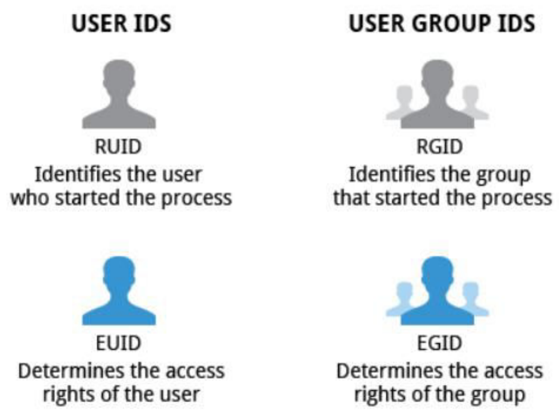
ID Type	Description
Process ID (PID)	Unique Process ID number
Parent Process ID (PPID)	Process (Parent) that started this process. If the parent dies, the PPID will refer to an adoptive parent; on recent kernels, this is kthreadd which has PPID=2.
Thread ID (TID)	Thread ID number. This is the same as the PID for single-threaded processes. For a multi-threaded process, each thread shares the same PID, but has a unique TID.

Terminating a Process:

1. At some point, one of your applications may stop working properly. We will eliminate it by terminating the process.
2. To terminate a process, you can type **kill -SIGKILL <pid>** or **kill -9 <pid>**

User and Group IDs:

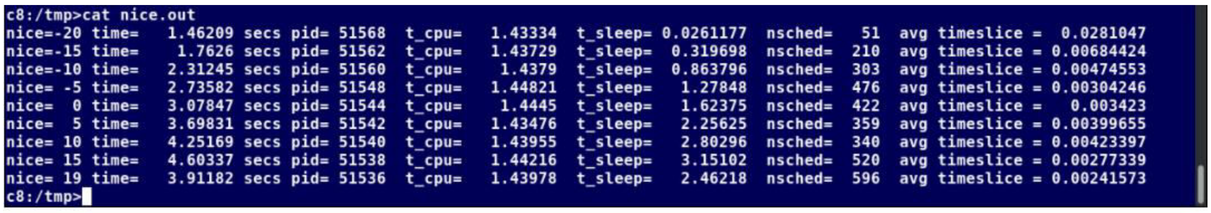
1. Many users can access a system simultaneously, and each user can run multiple processes.
2. The operating system identifies the user who starts the process by the Real User ID (**RUID**) assigned to the user.
3. The user who determines the access rights for the users is identified by the Effective UID (**EUID**).
4. The **EUID** may or may not be the same as the RUID.
5. Users can be categorized into various groups. Each group is identified by the Real Group ID (**RGID**).
6. The access rights of the group are determined by the Effective Group ID (**EGID**).
7. Each user can be a member of one or more groups.
8. Most of the time we ignore these details and just talk about the User ID (**UID**) and Group ID (**GID**).


USER IDS
USER GROUP IDS
RUID
Identifies the user who started the process
RGID
Identifies the group that started the process
EUID
Determines the access rights of the user
EGID
Determines the access rights of the group

User and Group IDs

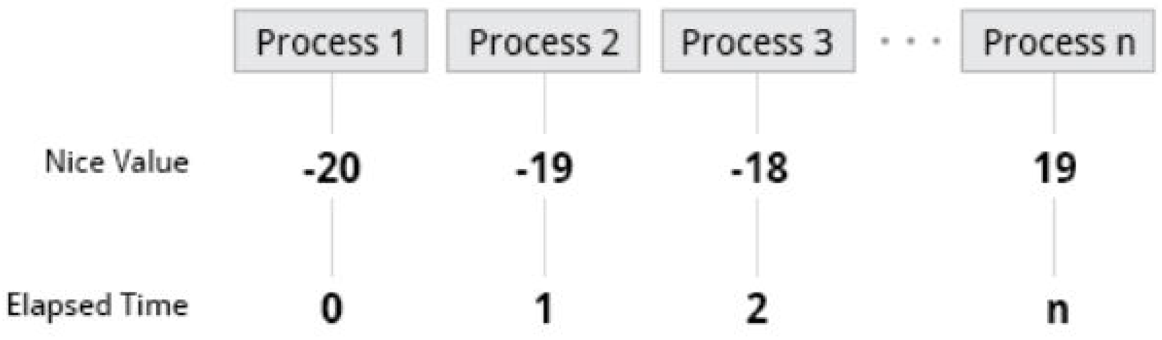
More About Priorities:

1. At any given time, many processes are running (i.e. in the run queue) on the system.
2. However, a CPU can actually accommodate only one task at a time, just like a car can have only one driver at a time.
3. Some processes are more important than others, so Linux allows you to set and manipulate process priority.
4. Higher priority processes grep preferential access to the CPU.
5. The priority for a process can be set by specifying a **nice value**, or **niceness**, for the process.
6. The lower the **nice value**, the higher the priority. Low values are assigned to important processes, while high values are assigned to processes that can wait longer.
7. A process with a high **nice value** simply allows other processes to be executed first.
8. In Linux, a **nice value** of **-20** represents the highest priority and **+19** represents the lowest.
9. We can change the priority of the process using **renice** command
   Command: renice priority PPID
   Example: renice +5 3077



```
c8:/tmp>cat nice.out
nice=-20 time=   1.46209 secs pid= 51568  t_cpu=   1.43334  t_sleep= 0.0261177  nsched=   51  avg timeslice =  0.0281047
nice=-15 time=    1.7626 secs pid= 51562  t_cpu=   1.43729  t_sleep=  0.319698  nsched=  210  avg timeslice = 0.00684424
nice=-10 time=   2.31245 secs pid= 51560  t_cpu=    1.4379  t_sleep=  0.863796  nsched=  303  avg timeslice = 0.00474553
nice= -5 time=   2.73582 secs pid= 51548  t_cpu=   1.44821  t_sleep=   1.27848  nsched=  476  avg timeslice = 0.00304246
nice=  0 time=   3.07847 secs pid= 51544  t_cpu=    1.4445  t_sleep=   1.62375  nsched=  422  avg timeslice =   0.003423
nice=  5 time=   3.69831 secs pid= 51542  t_cpu=   1.43476  t_sleep=   2.25625  nsched=  359  avg timeslice = 0.00399655
nice= 10 time=   4.25169 secs pid= 51540  t_cpu=   1.43955  t_sleep=   2.80296  nsched=  340  avg timeslice = 0.00423397
nice= 15 time=   4.60337 secs pid= 51538  t_cpu=   1.44216  t_sleep=   3.15102  nsched=  520  avg timeslice = 0.00277339
nice= 19 time=   3.91182 secs pid= 51536  t_cpu=   1.43978  t_sleep=   2.46218  nsched=  596  avg timeslice = 0.00241573
c8:/tmp>
```

nice Output



Nice Values