

# LINUX PROCESSES

Jim Jackson <jj@franjem.org.uk>

March 14, 2006

---

## Contents

<b>1 In the Beginning.....</b>	<b>3</b>
<b>2 Unix Processes...</b>	<b>4</b>
<b>3 Starting A New Unix process...</b>	<b>5</b>
<b>4 Parental Responsibilities...</b>	<b>6</b>
<b>5 The GrandDaddy of them all...</b>	<b>7</b>
<b>6 Families of processes...</b>	<b>8</b>
<b>7 ... Complex Processes - Threads...</b>	<b>9</b>
<b>8 Some CLI commands for viewing Process data...</b>	<b>10</b>

References to use of **ps** refers to GNU **ps**

This document can be found at <http://www.comp.leeds.ac.uk/jj/linux/processes.html>

---

## 1 **In the Beginning.....**

- Computers ran one program at a time - they didn't have the resources to do much else
  - Eventually they got powerfull enough to waste processing on what to do next.....
  - They even got pretty smart at it :-)
  - They could even run one program for a franction of a second, save its context, and move on to running another program for a fraction of a second, save context, move on.... etc
  - This gives the appearance of several programs running at the same time.
-

## 2 Unix Processes...

- In Unix parlance, a process was the name given to a section of program that is separately scheduled to run
  - The OS keeps an internal list of processes that are in memory and are executing or waiting to execute.
  - Each Process is given a number to identify it, call the Process ID or **pid** .
  - The OS periodically interrupts the running process, and schedules the next process to run. This is the **scheduler**
  - The OS will also schedule when a process starts to wait for resources, e.g. for keyboard input, or the next block from disk
-

### 3 Starting A New Unix process...

- Traditionally the only way to start a new process was for an existing process to **fork**
- The process that called **fork** becomes the parent of the new **child** process. The **child** is a copy of the parent with a new **pid**
- Usually the **child** then mutates by using **exec** to load a new program in the current process's space
- An example of processes begetting processes .....

```
|-xterm(778)---bash(782)---vi(1212)
```

- **xterm** will have **forked** and **execed** **bash** which has in turn **forked** and **execed** **vi**

**xterm** is the parent of **bash** which in turn is the parent of **vi**

see <http://yolinux.com/TUTORIALS/ForkExecProcesses.html> a YoLinux Tutorial on Fork, Exec and Process control

---

## 4 Parental Responsibilities...

- A program that has **forked** a **child** is supposed to **wait** for the child to finish, and tidy afterwards. If the **parent** terminates before the **child** the **child** is said to be **Orphaned**
- Process number 1 inherits **orphans**
- If a **child** terminates but its parent does not catch the termination signal, the **child** process cannot fully terminate and becomes a **zombie** . They are called **zombies** because they can't be killed.
- **zombies** might not disappear until the next reboot, but as long as there only a few, are benign. killing the parent of a **zombie** may get rid of the zombie

see <http://www.linuxsa.org.au/tips/zombies.html>

---

## 5 The GrandDaddy of them all...

- If a **process** needs a **process** to beget it, then we have a problem, how does the first process get started?
  - As part of the Unix kernel boot process, it starts **process 1** , usually the program **init**
  - **init** then follows scripts/config\_data to do all that's necessary to ge the system started.
  - All **processes** are offspring of **process 1**
  - Don't try killing **process 1**
-

## 6 Families of processes...

- Each **process** is a member of a **process group** (**pgid**), which is a collection of processes with a common **process leader** . A new **process** is initially in the **process group** of its parent.
- A **process group** is a member of a **session**, and a **session** can have a controlling **tty**
- **pgid** of a **process** is the **pid** of the **process leader**
- A **process** can leave home and declare independence and make itself a **process leader** of a new **process group**

Why **process groups** & **sessions** ? It's mainly to provide job control facilities. For more info see <http://www.win.tue.nl/~aeb/linux/lk/lk-10.html>

---

## 7 ... Complex Processes - Threads...

Some OSes, including other **Unixes**, have the separate concept of a **process** optionally consisting of several separate **threads**, or **light weight processes**. Each **thread** is part of the same program but are scheduled individually.

Until comparatively recently Linux just had **processes**, but it provides a more versatile sort of **fork**, called **clone**, which allows more control over the relationship between **parent** and **child**, allowing them to share memory space, file handles, signals etc and so provide **thread** facilities using Linux **processes**

Since Linux 2.4.19 (or so) threads can share the **pid** of the **parent process** and have a separate **thread** id, **tid** . Most **processes** have just the one **thread** and so their **tid** is same as their **pid** For more info see <http://yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html> for a tutorial on POSIX threads usage under Linux.

---

## 8 Some CLI commands for viewing Process data...

**ps uax** List all processes given some basic usage on utility use

**pstree** Lists processes as a tree showing parental relationships

**pstree -p** ditto but showing process numbers (pid)

**ps -eo pid,ppid,user,pgid,args --sort pid** list all processes showing process id, parent pid, user name, process group id and command arguments, sorted by pid

**ps -eLf** list all processes showing pid, ppid, tid (LWP), and number of threads in group, NLWP

An example of a thread....

UID	PID	PPID	LWP	C	NLWP	STIME	TTY	TIME	CMD
...									
root	1753	1	1753	0	2	Feb27	?	00:00:01	auditd
root	1753	1	1755	0	2	Feb27	?	00:00:03	auditd

Notice the same PID but with different LWP (TID).

see the **man ps** more far more details than you can possible want.

---

HomePage <http://www.comp.leeds.ac.uk/jj>