


←Reviewed 20130113 by hazel ->

Multi CPU performance tuning

The Linux SMP kernel (*Symmetric Multi Processing* - enabled in all 64-bit Slackware kernels as well as the 32-bit “-smp” kernels) does a good job of scheduling processes so that they take maximum advantage of your multiple CPU's or CPU cores.

In some cases however, you will want to be able to override the default behaviour of the kernel's scheduler - and limit one of your processes to run on a particular CPU or sub-set of CPU cores. This is called “forcing  [processor affinity](#)” or *CPU pinning*.

Processor affinity

For the sake of article clarity, let's talk about multi-core CPU's, multi-CPU systems, hyperthreading CPU's and such as “*a multi CPU system*”.

In a multi-CPU system, if a process (or a series of process threads) is started multiple times, the kernel process scheduler has a tendency to restart those processes or threads on the same CPU that ran the process thread before. This is potentially more efficient than selecting a random CPU core, because some of the process's state may have been preserved in that CPU's cache. CPU affinity is a natural behaviour of the kernel process scheduler.

Because the scheduler will do its best to keep processes running on the same CPU (as long as this is beneficial to overall performance) you usually do not have to interfere. However, some cases warrant pinning a process to a single CPU or a predefined subset. Let's look at two examples; the second will be discussed in more detail.

1. Commercial database vendors like [Oracle DB](#) or [IBM DB2](#) want you to pay per CPU. Typically, database servers have a massive amount of CPU cores and you want to be able to decide how much to pay for your database (more performance would mean more cost)
2. One process on a busy server is eating so many of your multi-CPU processor cycles that other programs' performance starts to suffer. Limiting your runaway process to a single CPU will free up resources for your other programs.

An actual use case for the second example, is running a Java based application like a [Minecraft](#) server. This is a resource-hungry beast, and it will claim much more than a full CPU if it runs a world with quite a number of players. A typical real-life load of a quad-core *AMD Athlon™ II X4 640 Processor* is higher than 12 with the Java process using nearly 200% CPU (i.e. it fully occupies half the available CPU cores). Other applications will be grateful if you manage to pin the Java process to a single core - and the game server's performance hit is not as large as you might fear.

How to pin this process to one of your CPU's?

Taskset

Slackware has a tool for setting the processor affinity for a certain task or process. It is called “taskset” and is part of the `util-linux` package. Run “`man 1 taskset`” if you want to know more about its purpose and command line syntax.



The `taskset` command is used to set or retrieve the CPU affinity of a running process given its PID, or to launch a new COMMAND with a given CPU affinity.

The CPU affinity is represented as a bitmask, with the lowest order bit corresponding to the first logical CPU and the highest order bit corresponding to the last logical CPU. For example:

```
0x00000001 is processor #0 (i.e. the 1st processor)
0x00000003 is processors #0 and #1 (1st and 2nd processor)
0x00000004 is processor #2 (the 3rd processor)
0xffffffff means all processors (#0 through #31)
```

Luckily you can use human-readable numbers instead of a bitmask if you add the “-c” command line parameter.

Examples:

- To pin an already running process 1337 to processor #0 (the 1st processor - remember, UNIX starts counting at zero) you would enter the following command:

```
$ taskset -p 0x00000001 1337
pid 1337's current affinity list: 0-3
pid 1337's new affinity list: 1
```

Confused by hexadecimal numbers? Then use the human-readable version:

```
$ taskset -cp 1 1337
```

You may notice the strange order of command line options and arguments; first come all the options and then follow the argument values in pre-defined order... `man taskset(1)` is your friend.

- To start a new process with a fixed CPU affinity, you omit the “-p” switch and append a command line, like this real-world example of starting a Minecraft server:

```
$ taskset -c 3 java -Xmx1024M -jar minecraft_server.jar nogui
```

which ties the Java process to the 4th CPU in your computer. The effect of this will be *immediately* obvious.

While you can pin a running process task to a sub-set of your available CPU's, that does not mean that the process and its threads will suddenly be limited to those CPU's you dictated. Rather, the kernel scheduler will now start preferring your appointed CPU instead of the CPU cores your processes may be currently running on. It can take quite a while (process threads have to stop and be started again) to see an obvious effect of using the `taskset` command on a running task. That is why it may be more effective to use `taskset` to start your program with the desired processor affinity.

Try assigning one of your CPU intensive programs to one CPU, and use `top` or `htop` to observe the non-uniform CPU load. For the `top` program, you have to press `1` in order to see processor statistics per CPU core. The `htop` command allows you to set processor affinity and show you a process list sorted by CPU.

If you want to know exactly what CPU your process task is running on, you can use this command (use “`pidof`” to find the Process ID of your program):

```
$ MYPID=$(/sbin/pidof java)
$ ps -eo psr,pid | grep $MYPID
3 4819
```

This shows that the Java process with PID 4819 is running on CPU #3 (the fourth CPU core of a quad-core system for instance). Indeed, this is the PID of that Minecraft server which we pinned to CPU #3 in the above example.

Sources

- Originally written by [Eric Hameleers](#)

[howtos](#), [hardware](#), [author alienbob](#)

From:

<https://docs.slackware.com/> - **SlackDocs**

Permanent link:

https://docs.slackware.com/howtos:hardware:processor_affinity

Last update: **2013/01/13 18:12 (UTC)**

