

Using Top More Efficiently

Contributed by Mulyadi Santosa
03-28-06

For desktop users, monitoring resource usage is an important task. By doing this, we can locate system bottleneck, planning what to do to optimize our system, identifying memory leak and so on. The problem is, which software one should use and how to use it according to our need.

Among many monitoring tools that available, most people use "top" (a part of procs package). Top provide almost everything we need to monitor our system's resource usage within single shot. In this article, all the information are based on procs 3.2.5 running on top of Linux kernel 2.6.x

Here, we assume that procs package is already installed and run well in your Linux system. No previous experience with top is needed here, but if you had given it a try briefly, that would be an advantage.

Here are some challenges:

A. Interactive or batch mode?

By default, top is invoked using interactive mode. In this mode, top runs indefinitely and accepts keypress to redefine how top works. But, sometimes you need to post-process the top's output and this is hardly achieved using this mode. The solution? Use batch mode.

```
$ top -b
```

You will get output like below:

```
top - 15:22:45 up 4:19, 5 users, load average: 0.00, 0.03, 0.00
Tasks: 60 total, 1 running, 59 sleeping, 0 stopped, 0 zombie
Cpu(s): 3.8% us, 2.9% sy, 0.0% ni, 89.6% id, 3.3% wa, 0.4% hi, 0.0% si
Mem: 515896k total, 495572k used, 20324k free, 13936k buffers
Swap: 909676k total, 4k used, 909672k free, 377608k cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
 1 root  16  0 1544 476 404 S 0.0 0.1 0:01.35 init
 2 root  34 19  0  0  0 S 0.0 0.0 0:00.02 ksoftirqd/0
 3 root  10 -5  0  0  0 S 0.0 0.0 0:00.11 events/0
```

Uh, wait, it runs repeatedly, just like interactive mode does. Don't worry, limit its repetition with -n. So, if you just want single shot, type:

```
$ top -b -n 1
```

The real advantage of this mode is you can easily combine in with at or cron. Together, top can snapshot resource usage at certain time

unattendedly. For example, using `at`, we can schedule `top` to run 1 minute later.

```
$ cat ./test.at
TERM=linux top -b -n 1 >/tmp/top-report.txt
$ at -f ./test.at now+1minutes
```

Careful reader might ask "why do I need to set `TERM` environment before invoking `top` when creating new `at` job?". The answer is, `top` needs this variable set but unfortunately "`at`" isn't retained it from the time of invocation. Simply set it like above and `top` will work smoothly.

B. How to monitor certain processes only?

Sometimes, we are only interested on several processes only, maybe just 4 or 5 of the whole existing processes. For example, if you want monitor process identifier (PID) 4360 and 4358, you type:

```
$ top -p 4360,4358
OR
$ top -p 4360 -p 4358
```

Seems easy, just use `-p` and list all the PIDs you need, each separated with comma or simply use `-p` multiple times coupled with the target PID.

Another possibility is just monitoring process with certain user identifier (UID). For this need, you can use `-u` or `-U` option. Assuming user "johndoe" has UID 500, you can type:

```
$ top -u johndoe
OR
$ top -u 500
OR
$ top -U johndoe
```

The conclusion is, you can either use the plain user name or the numeric UID. "-u, -U? Those two are different?" Yes. Like almost any other GNU tools, options are case sensitive. `-U` means `top` will find matching effective, real, saved and filesystem UIDs, while `-u` just find matching effective user id. Just for reminder, every *nix process runs using effective UID and sometimes it isn't equal with real user ID. Most likely, one is interested in effective UID as filesystem permission and operating system capability are checked against it, not real UID.

While `-p` is just command-line option only, both `-U` and `-u` can be used inside interactive mode. Like you guess, press 'U' or 'u' to filter the processes based on their user name. Same rule is applied, 'u' for effective UID and 'U' for real/effective/saved/filesystem user name. You will be asked to enter the user name or the numeric UID.

{mospagebreak title=Fast or slow update?}
C. Fast or slow update?

Before we answer this question, let's take a short look on how top really works. Strace is your friend here:

```
$ strace -o /tmp/trace.txt top -b -n 1
```

Use you favourite text editor and load /tmp/trace.txt. What do you think? A lot of jobs for single invocation, that is what I think and maybe you'll agree. One of the jobs top must do in every iteration is opening many files and parsing their contents, as shown by the number:

```
$ grep open\(/tmp/hasil.txt | wc -l
```

Just for illustration, in my Linux system, it yields 304. Closer look reveals that top iterates inside /proc directory to gather processes information. /proc itself is pseudo filesystem, meaning it doesn't exist on real disk but is created on the fly by the Linux kernel and live on RAM. Within directory such as /proc/2097 (2097 is a PID), Linux kernel exports information about related process and this is where top gathers processes information along with resource usage.

Also try these:

```
$ time top -b -n 1
```

This will give you illustration how fast top works on single round. In my system, this yields around 0.5-0.6 seconds. Look at "real" field, not the "user" or "system" field because "real" reflects the total time top needs to work.

So, realizing this fact, it will be wise to use moderate update interval. Browsing RAM based filesystem takes time too, so be wise. As rule of thumb, 1 to 3 seconds interval is enough for most users. Use -d in command line option or press "s" inside interactive mode to set it. You can use fractional number as interval, e.g: 2.5, 4.1 and so on

When we should faster than 1 seconds?

- You need more samples during a time. For this need, better use batch mode and redirect standart output to a file so you can analyze it better.

- You don't mind with extra CPU load carried by top. Yes, it is small but it is still a load. If your Linux system is relatively idle, feel free to use short interval, but if not, better preserve your CPU time for more important task.

One way to reduce top's work is by monitoring certain PIDs only.

This way, top won't need to traverse all the /proc sub-directory. How about user name filtering? It won't do any good. User name filtering brings extra work for top, thus combining it with very short interval will increase CPU load.

Of course, whenever you need to force the update, just press Space and top will refresh the statistic right away.

{mospagebreak title=Fields we need}

D. Fields that we need

By default, top starts by showing the following task's property:

FieldDescription

PID : Process ID

USER :Effective User ID

PR : Dynamic priority

NI :Nice value, also known as base priority

VIRT : Virtual Size of the task. This includes the size of process's executable binary, the data area and all the loaded shared libraries.

RES : The size of RAM currently consumed by the task. Swapped out portion of the task is not included.

SHR : Some memory areas could be shared between two or more task, this field reflects that shared areas. The example of shared area are shared library and SysV shared memory.

S : Task status

%CPU : The percentage of CPU time dedicated to run the task since

the last top's screen update.

%MEM : The percentage of RAM currently consumed by the task.

TIME+ : The total CPU time the task has been used since it started.
"+" sign means it is displayed with hundredth of a second granularity. By default, TIME/TIME+ doesn't account the CPU time used by the task's dead children.

COMMAND : Showing program names.

But, there are more. Here, I will just explain fields that might interest you:

FieldDescription

nFLT (key 'u')

Number of major page fault since the process is started.

Technically, page fault happens when the task access a non existant page in its address space. A page fault is said as "major" if kernel needs to access the disk to make the page available. On the contrary, soft minor page fault means the kernel only need to allocate pages in RAM without reading anything from disk.

For illustration, consider the size of program ABC is 8 kB and assume the page size is 4 kB. When the program is fully loaded to RAM, there will be 2 times major page fault ($2 * 4$ kB). The program itself allocates another 8 kB for temporary data storage in RAM. Thus, there will be 2 minor page fault.

A high number of nFLT could mean:

- The task is aggressively load some portions of its executable or library from the disk.

- The task is accessing a page that is swapped out

-

It is normal if you see a high number of major page fault when a program is run for first time. On the next invocations, buffer is utilized so likely you will see "0" or low number of nFLT. But, if a program is continuously triggering major page fault, big chance your program needs larger RAM size than currently installed.

nDRT (key 'v')

The number of dirty pages since they are written back to the disk.

Maybe you wonder, what is dirty page? First, a little background. As you know, Linux employ caching mechanism, so everything that is read from disk is also cached in RAM. The advantage of this action is, subsequent read to the same disk block can be served from RAM thus reading completes faster.

But it also costs something. If the buffer's content is modified, it needs to be synchronized. Thus, sooner or later this modified buffer (dirty page) must be written back. The failure on the synchronization might cause data inconsistency on related disk.

On mostly idle to fairly loaded system, nDRT is usually below 10 (this is just a raw prediction) or mostly zero. If it is constantly bigger than that:

- The task is aggressively write something to file(s).
It is so often that disk I/O can't keep up with it
- The disk suffers I/O congestion, thus even the task only modifies small portion of file(s), it must wait a bit longer to be synchronized. Congestion happens when many processes access the disk at a time but cache hit is low.

These days, (1) unlikely happens because I/O speed is getting faster and less CPU demanding (thanks to DMA). So (2) has bigger probability.

Note: On 2.6.x, this field is always zero without unknown reason.

P (key 'j')

Last used CPU. This field only has meaning in SMP environment. SMP here refers to Hyperthread, multi core or true multi processor. If you just have one processor (non multi core, not HT), this field will always show '0'.

In SMP system, don't be surprised if this field change sometimes. That means, the Linux kernel tries to move your task to the other CPU which is considered less loaded.

CODE (key 'r') and DATA (key 's')

CODE simply reflects the size of your application code, while DATA reflects the size of data segment (stack, heap, variables but not shared libraries). Both are measured in kilobyte.

DATA is useful to show how much your application allocates memory. Sometimes, it can also reveal memory leaks. Of course, you need better tool such as valgrind to differentiate between repetitive memory allocation or growing memory leaks if DATA continuously climbs up.

Note: DATA, CODE, SHR, SWAP, VIRT, RES are all measured in page size (4KB in Intel architecture). Read only data section is also calculated as CODE size, thus sometimes it is larger than the actual text (executable) segment.

SWAP (key 'p')

The size of swapped out portion of a task's virtual memory image. This field is sometimes confusing, here is why:

Logically, you would expect this field really shows whether

your program is partially swapped out and how much. But the reality shows otherwise. Even the "Swap used" field shows 0, you will be surprised that SWAP field of each tasks show greater than zero number. So, what's wrong?

This comes from the fact that top use this formula:

$$\begin{aligned} \text{VIRT} &= \text{SWAP} + \text{RES or equal} \\ \text{SWAP} &= \text{VIRT} - \text{RES} \end{aligned}$$

As explained previously, VIRT includes anything inside task's address space, no matter it is in RAM, swapped out or still not loaded from disk. While RES represents total RAM consumed by this task. So, SWAP here means it represents the total amount of data being swapped out OR still not loaded from disk. Don't be fooled by the name, it doesn't just represent the swapped out data.

To display the above fields, press 'f' inside the interactive mode. Then press the related key (mentioned above inside the parentheses). Those keys toggle the related fields, so press once to show it, press again to hide it. To find out whether the fields are displayed or not, simply watch the series of letters on the first line (at the right of "Current Fields"). Uppercase means the fields is shown, lower case means the opposite. Press Enter after you are satisfied with the selection.

Sorting use similar way. Press 'O' (upper case) followed by a key representing the field. Don't worry if you don't remember the key map, top will show it. The new sort key will be marked with asterisk and the letter will change to upper case, so you can notice it easily. Press Enter after you are finished

```
{mospagebreak title=Multi view are better than one?}
```

E. Multi view are better than one?

In different situations, sometimes we want to monitor different system property. For example, at one time you want to monitor %CPU and cpu time spent by all tasks. At another time, you want to monitor resident size and total page faults of all tasks. Rapidly press 'f' and change the visible fields? I don't think this is a smart choice.

Why don't you use Multiple Windows mode? Press 'A' (upper case) to switch to multi windows view. By default, you will see 4 different set of field groups. Each field groups has a default label/name:

- 1st field group: Def

- 2nd field group: Job

- 3rd field group: Mem

- 4th field group: Usr

1st field group is the usual group you see in single window view, while the rest are hidden. Inside multi window mode, press 'a' or 'w' to cycle through all the available windows. Pay attention, switching to another window also change the active window (also known as current window). If you are not sure which one is currently the active one, just look at the first line of top's display (at the left of current time field). Another way to change active window is by pressing 'G' followed by windows number (1 to 4).

Active window is the one which react to user input, so make sure to select your preferred window first before doing anything. After that, you can do anything exactly like you do in single window mode. Usually, what you want to do here is customizing field display, so just press 'f' and start customizing.

If you think 4 is too much, just switch to a field group and press '-' to hide it. Please note, even you hide current field group, that doesn't mean you also change the active group. Press '-' once again to make current group visible.

If you are done with multi window mode, press 'A' again. That also make active group as the new field group of single window mode.

F. "How come there is only so few free memory on my Linux PC?"

Come to same question? No matter how much you put RAM in your motherboard, you quickly notice the free RAM is reduced so fast. Free RAM miscalculation? No!

Before answering this, first check the memory summary located on the upper side of top's display (you may need to press 'm' to unhide it). There, you will find two fields: buffers and cached. "Buffers" represent how much portion of RAM is dedicated to cache disk block. "Cached" is similar like "Buffers", only this time it caches pages from file reading. For thorough understanding of those terms, refer to Linux kernel book like Linux Kernel Development by Robert M. Love.

It is enough to understand that both "buffers" and "Cached" represent the size of system cache. They dynamically grow or shrink as requested by internal Linux kernel mechanism.

Besides consumed by cache, the RAM itself is also occupied by application data and code. So, to conclude, free RAM size here means RAM area that isn't occupied by cache nor application data/code. Generally, you can consider cache area as another "free" RAM since it will be shrunk gradually if the application demands more memory.

On the task point of view, you might wonder which field truly represent memory consumption. VIRT field? certainly not! Recall that this field represent everything inside task address space, including the related shared libraries. After reading top source code and proc.txt (inside Documentation/filesystem folder of kernel source's tree), I conclude that RSS field is the best field describing task's memory consumption. I said "best" because you should consider it as approximation and isn't 100% accurate on all time.

G. Working with many saved configurations

Wanna keep several different configuration of top so you can easily switch between preconfigured display? Just create symbolic link to the top binary (name it anything you like):

```
# ln -s /usr/bin/top /usr/bin/top-a
```

Then run the new "top-a". Do the tweak and press 'W' to save the configuration. It will be saved under ~/.top-arc (the format is your top alias name+'rc').

Now run the original top to load your first display alternative, top-a for the second one and so on.

```
{mospagebreak title=Conclusion}
```

H. Conclusion

There are numerous tricks to use top more efficiently. The key is by knowing what you really need and possibly a little good understanding of Linux low level mechanism. The statistics isn't always correct, but at least it is helpful as a overall measurement. All these numbers are gathered from /proc, so make sure it is mounted first!

Reference:

- Understanding The Linux Kernel, 2nd edition.
- Documentation/filesystems/proc.txt inside kernel source tree.
- Linux kernel source.