Linux
System and process analysis with atop

JC van Winkel

Many thanks to Gerlof Langeveld, main author of atop.
`www.atoptool.nl`

$Timing(1m): start slide around 0s$

Welcome to this talk about Linux performance analysis and tuning with atop.

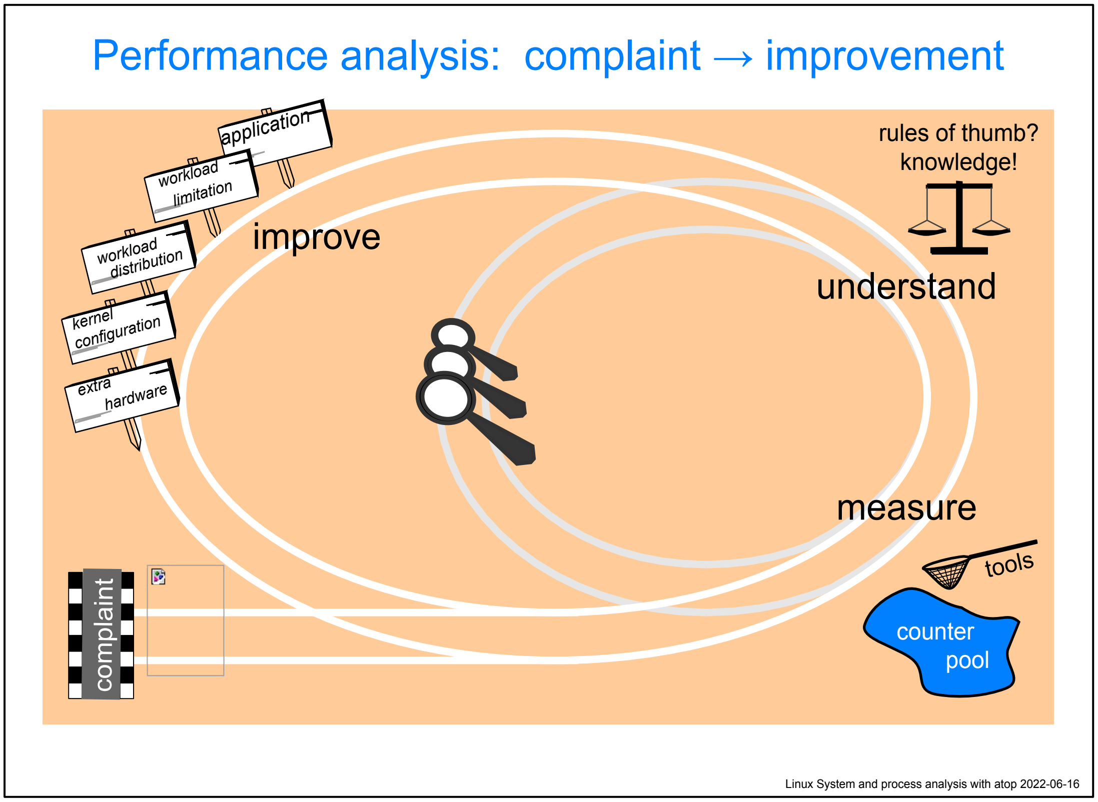
My name is JC van Winkel.

In my previous job, I was developing courses and teaching with AT Computing, a small spin off of the University of Nijmegen, the Netherlands.

We taught a lot of Linux classes and classes about languages stemming from the UNIX world, like C, C++, Python, perl.

There was also a class about Linux Performance analysis and Tuning class, which used atop a lot. And yes, the AT in atop stems from the name of the company.
It is open source, mostly written by Gerlof Langeveld and some contributions by me.
As linux gets more and more capabilities, atop is developed further to also be able to monitor those.

You can read all about atop at www.atoptool.nl



$Timing(3m): start slide around 1m$

Often performance analysis comes after complaints. Then one should know that, as with most troubleshooting, this is a iterative process

We start at the complaint.

- Make use of the many metrics that are being kept by the kernel
- Having some understanding of how the system works, we can zoom in: find other information and more detailed information from the metrics
- After several rounds of this, we can start improving things
- Your reflex might be: Buy a bigger (or more) machine(s)

However:

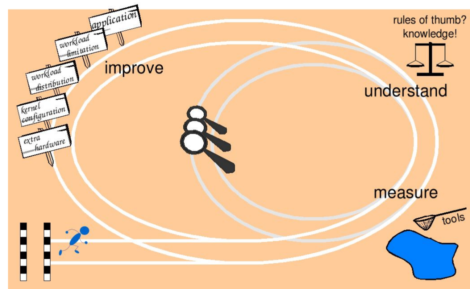
- most probable candidate: the application itself.
    - Consulting example: for (i=0; i<strlen(s); i++) { s is not changed here…. } where s was very long.
- Next thing could be limiting the workload or spreading the workload more over time
- Kernel configuration **can** help but requires quite some knowledge about the workings the kernel, including cpu/IO schedulters, the paging

- mechanism etc.
- Only after all the options mentioned above should you consider upgrading hardware.

Complaint

Reason for analysis

- interactive use:
  bad response times (seconds)
- batch processing:
  throughput (transactions/second)



Nature of complaint

- structural
  - system *always* slow
  - analysis easy (at any moment)
- incidental
  - system *sometimes* slow
  - analysis of historical data (logging necessary)

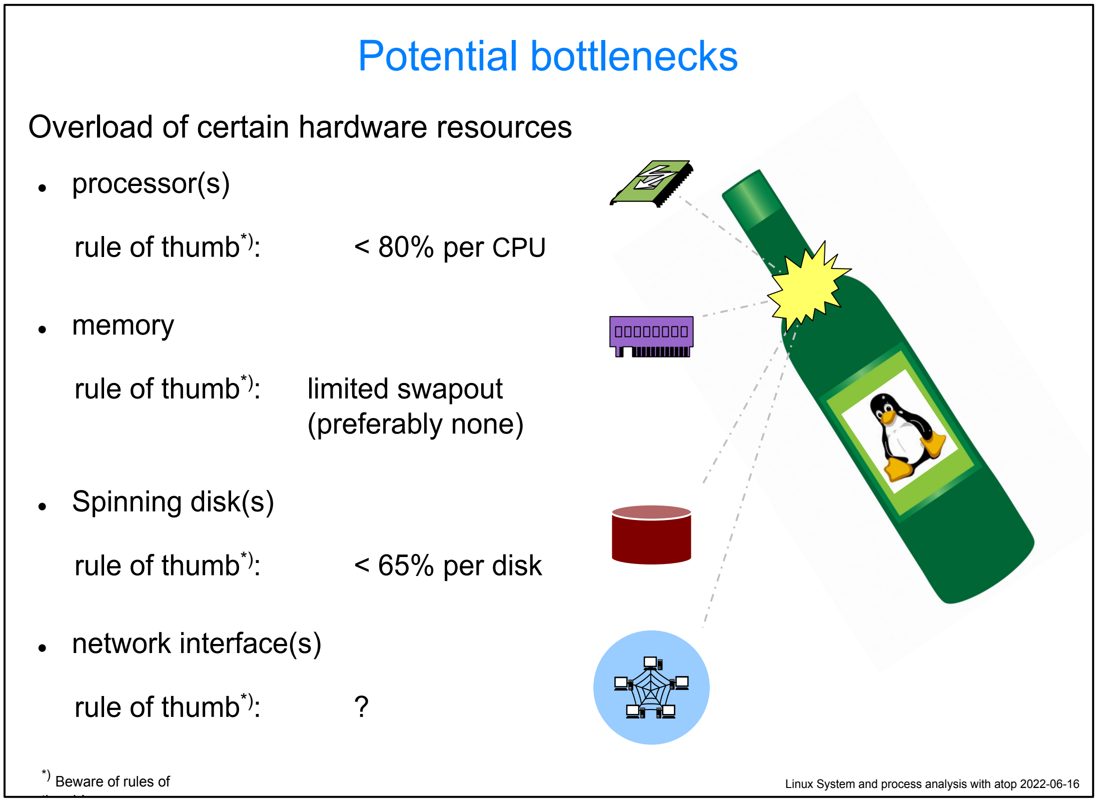
$Timing(2m): start slide around 4m$

The most common reason for analysis:

- Bad response times when using the system interactively. This means usually that the system feels slow **now**. i.e. you could look at measuring tools **now**
- Bad throughput. Here we're talking about slowness some time over the total processing of the job. i.e. when you look at the measuring tools, there may be nothing to see right now.

The problems can be

- happening all the time. This is easy to analyze (time frame wise)
- occurring irregularly. The system is not always slow. Harder to analyze, you need historic data for that.

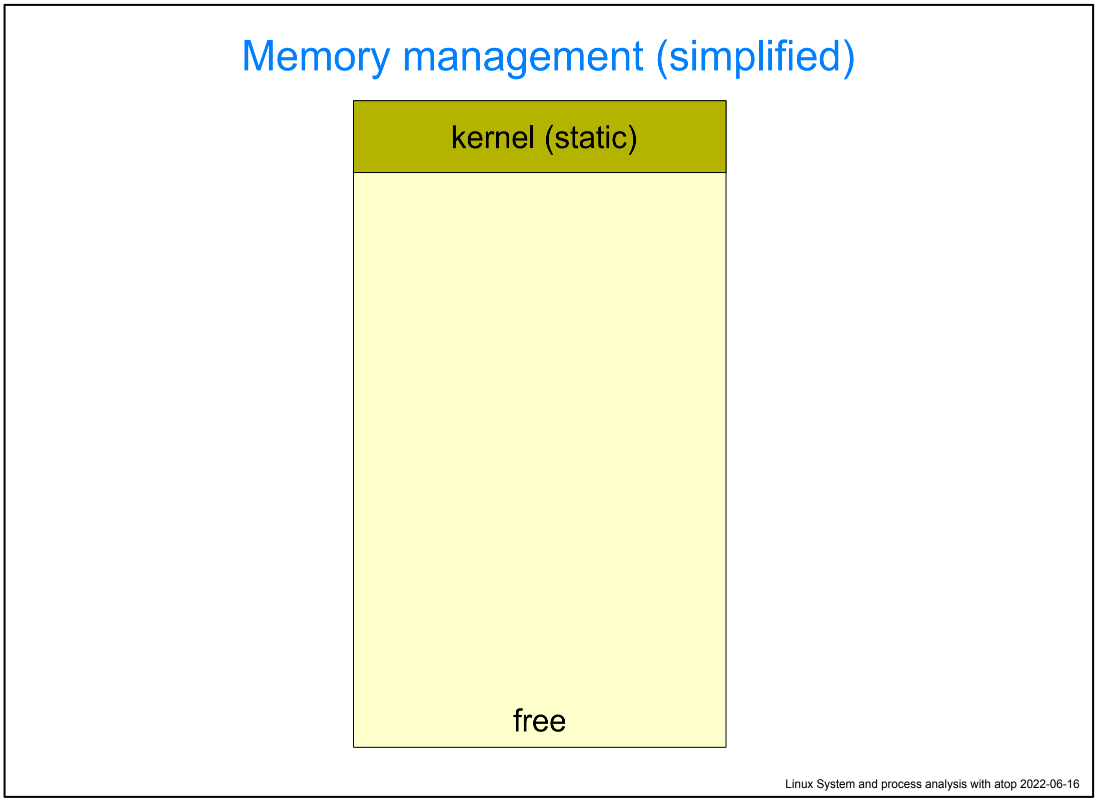


$Timing(3m): start slide around 6m$

In this talk we will look at four potential bottlenecks

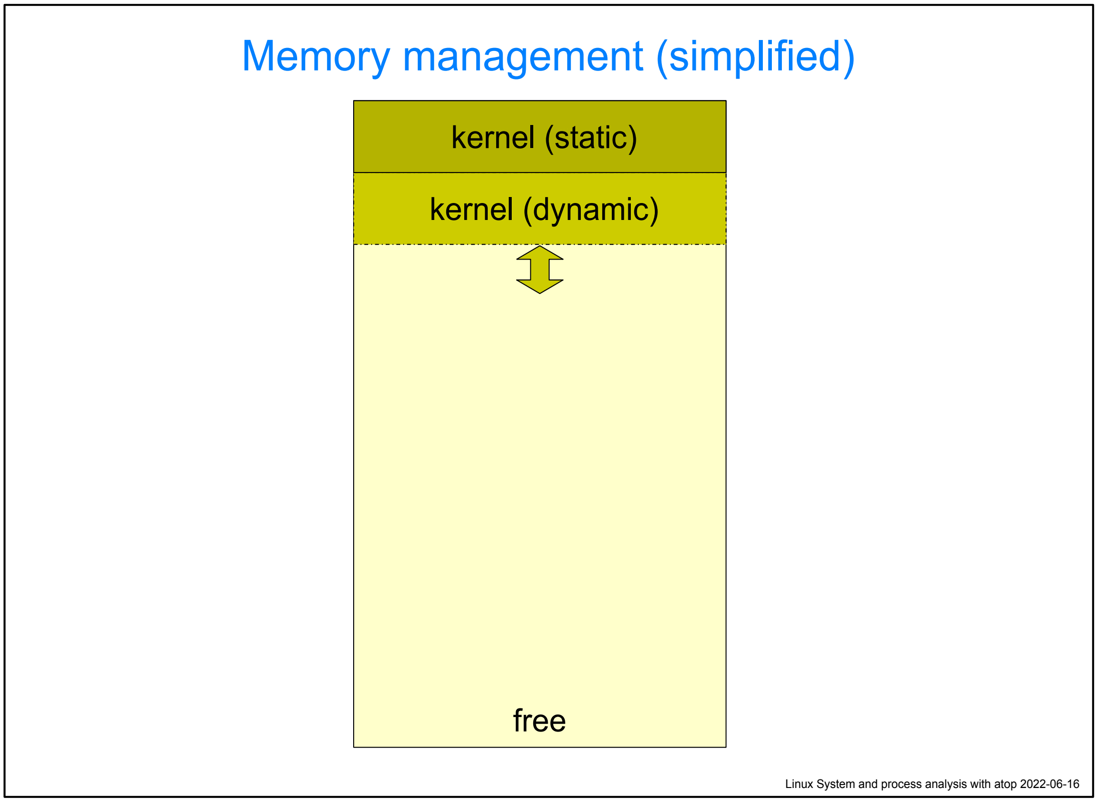
- CPU. We will want it to always have some "spare room" Note that this can be difficult to measure, especially when frequency scaling, turbo frequencies, thermal throttling and hyperthreading take place
- Memory. Having swapping is a bad sign. But even a system without any swapping that is currently using a lot of memory may feel slower as less memory is available for the page cache
- Spinning disks. These have a limited number of IOPS they can deliver. Getting close to 100% of that will lead to very long queues and hence waiting time. When swapping to disk, this may be even worse.
- Network interfaces. Here it is hard to give clear rules of thumb.

Bottle:
https://www.publicdomainpictures.net/en/view-image.php?image=39397&picture=wine-bottle-blank-label



$Timing(1m): start slide around 9m$

Before we continue, I will give a brief overview of how memory is managed in Linux systems

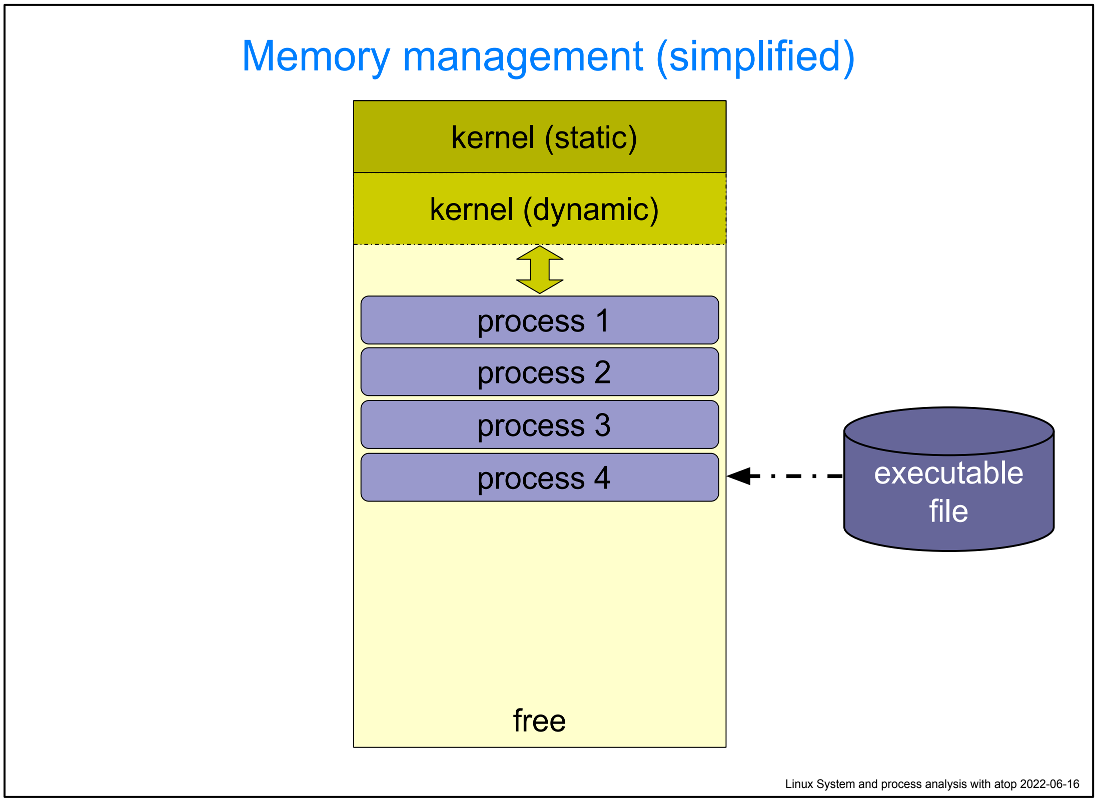


$Timing(1m): start slide around 10m$

Before we continue, I will give a brief overview of how memory is managed in Linux systems

In a just-booted system, there will basically only be memory being used by the kernel:

- The base kernel itself as loaded while booting
- some dynamically growing and shrinking memory for use by the kernel. This could be dynamically loaded modules, but also memory for data structures (the process table, open files table, network connections etc etc)



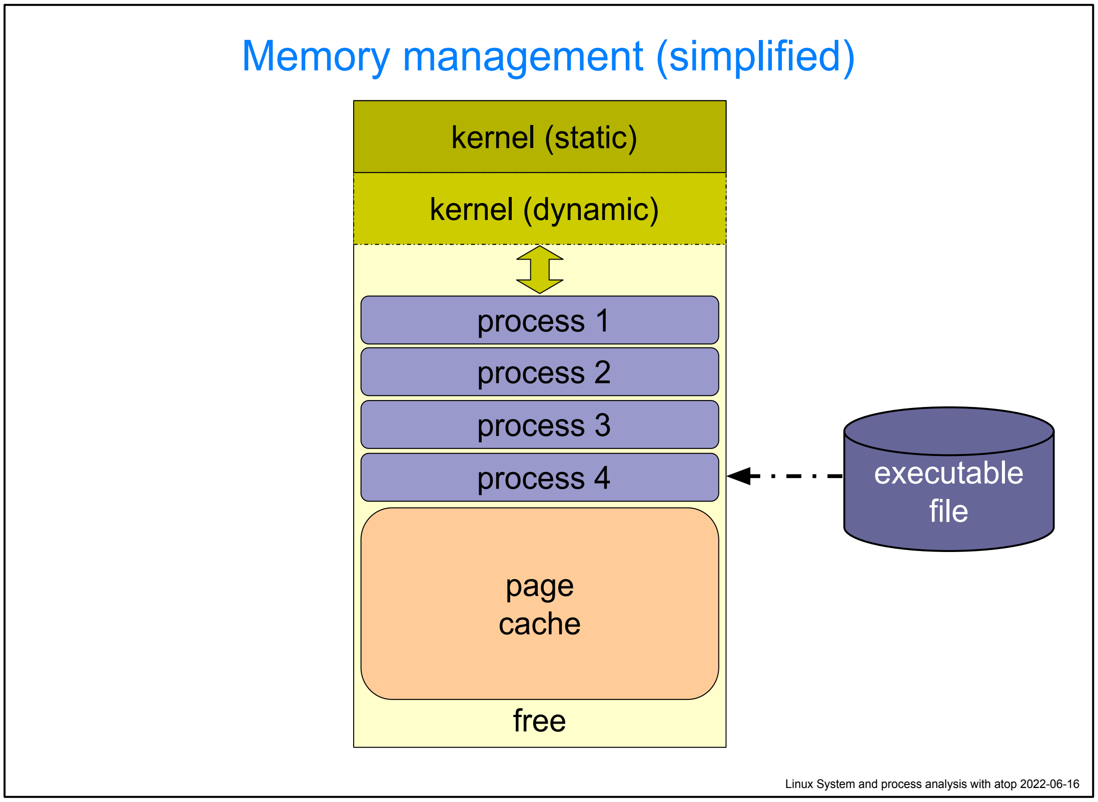
$Timing(1m): start slide around 11m$

Before we continue, I will give a brief overview of how memory is managed in Linux systems

In a just-booted system, there will basically only be memory being used by the kernel:

- The base kernel itself as loaded while booting
- some dynamically growing and shrinking memory for use by the kernel. This could be dynamically loaded modules, but also memory for data structures (the process table, open files table, network connections etc etc)

Next, when processes are started, their binary code is loaded from disk into memory. Processes can also grow by requesting dynamic memory (think: malloc).



$Timing(1m): start slide around 12m$

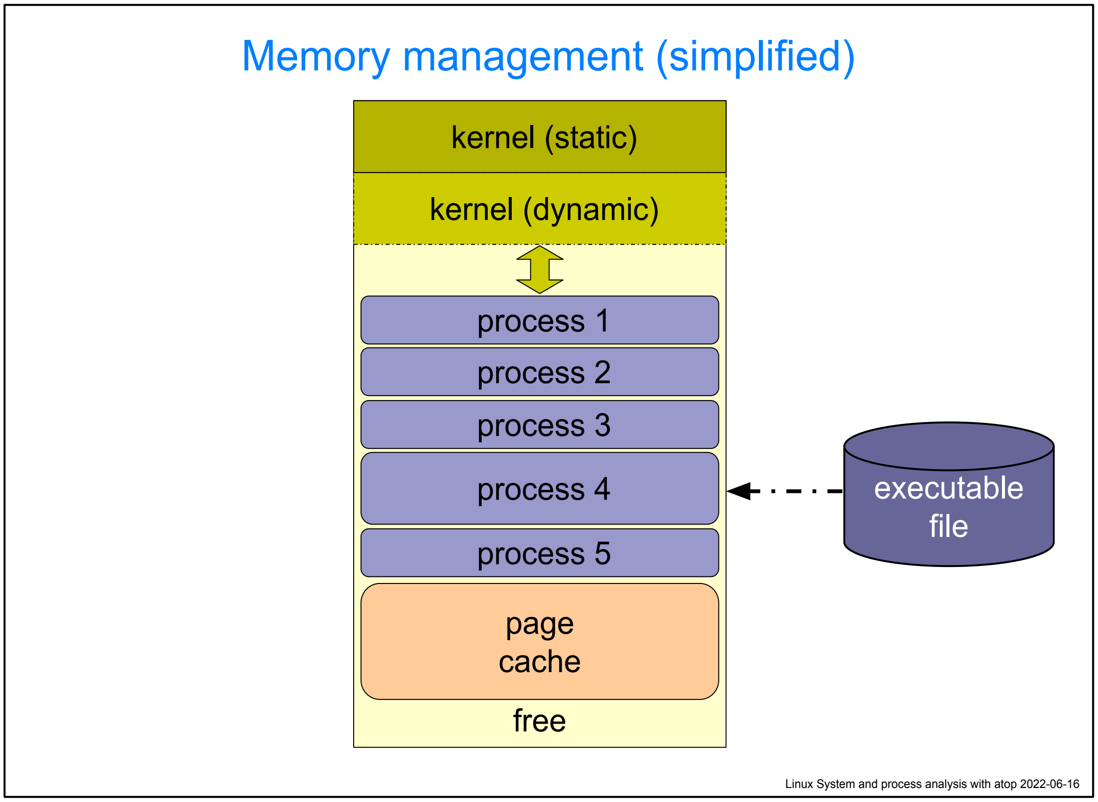
Before we continue, I will give a brief overview of how memory is managed in Linux systems

In a just-booted system, there will basically only be memory being used by the kernel:

- The base kernel itself as loaded while booting
- some dynamically growing and shrinking memory for use by the kernel. This could be dynamically loaded modules, but also memory for data structures (the process table, open files table, network connections etc etc)

Next, when processes are started, their binary code is loaded from disk into memory. Processes can also grow by requesting dynamic memory (think: malloc).

Memory that is not in use by the processes, will be used to by the kernel for the page cache. As files are read and written, the kernel keeps the blocks in memory in case the same is to be read again. This can save a lot of disk IO. The kernel wil always make sure there is a little but of memory free so when starting a new process (or when a process decides to grow its memory footprint) we don't have to shrink the page cache right away. Like keeping a rainy day fund.



$Timing(1m): start slide around 13m$

Before we continue, I will give a brief overview of how memory is managed in Linux systems

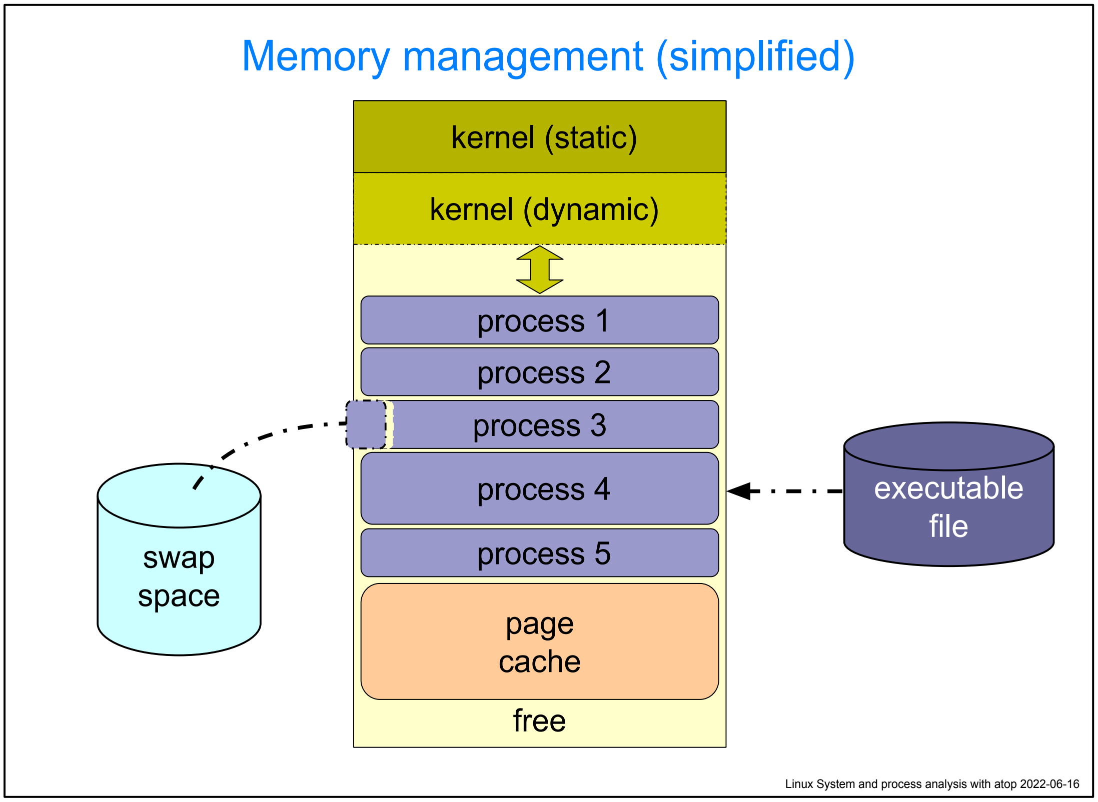
In a just-booted system, there will basically only be memory being used by the kernel:

- The base kernel itself as loaded while booting
- some dynamically growing and shrinking memory for use by the kernel. This could be dynamically loaded modules, but also memory for data structures (the process table, open files table, network connections etc etc)

Next, when processes are started, their binary code is loaded from disk into memory. Processes can also grow by requesting dynamic memory (think: malloc).

Memory that is not in use by the processes, will be used to by the kernel for the page cache. As files are read and written, the kernel keeps the blocks in memory in case the same is to be read again. This can save a lot of (slow) disk IO. The kernel wil always make sure there is a little but of memory free so when starting a new process (or when a process decides to grow its memory footprint) we don't have to shrink the page cache right away. A bit like keeping a rainy day fund.

When more memory is needed for a new process, the page cache will shrink



$Timing(2m): start slide around 14m$

Before we continue, I will give a brief overview of how memory is managed in Linux systems

In a just-booted system, there will basically only be memory being used by the kernel:

- The base kernel itself as loaded while booting
- some dynamically growing and shrinking memory for use by the kernel. This could be dynamically loaded modules, but also memory for data structures (the process table, open files table, network connections etc etc)

Next, when processes are started, their binary code is loaded from disk into memory. Processes can also grow by requesting dynamic memory (think: malloc).

Memory that is not in use by the processes, will be used to by the kernel for the page cache. As files are read and written, the kernel keeps the blocks in memory in case the same is to be read again. This can save a lot of (slow) disk IO. The kernel wil always make sure there is a little but of memory free so when starting a new process (or when a process decides to grow its memory footprint) we don't have to shrink the page cache right away. A bit like keeping a rainy day fund.

When more memory is needed for a new process, the page cache will shrink

If even more memory is needed (e.g. by a process starting to use more dynamically allocated memory (malloc), something has to give. The kernel will start searching for memory that can be made available. Parts of processes that have not been used for longer time may be discarded (e.g. for code pages) or they can be written to the swap space. Should a program then reference that (no longer available) memory, some empty space is searched for and the relevant page is loaded back from the swap space.

This is a normal process (imagine a process that had a lot of start-up code but once the program has been initialized, keeping that startup code in memory is a waste.) But if we start moving memory to swap even though it is being used actively, we may get into a state where a lot of memory accesses need a disk access. We call this state "thrashing". You will notice the system can do nothing.

Overview generic measurement tools

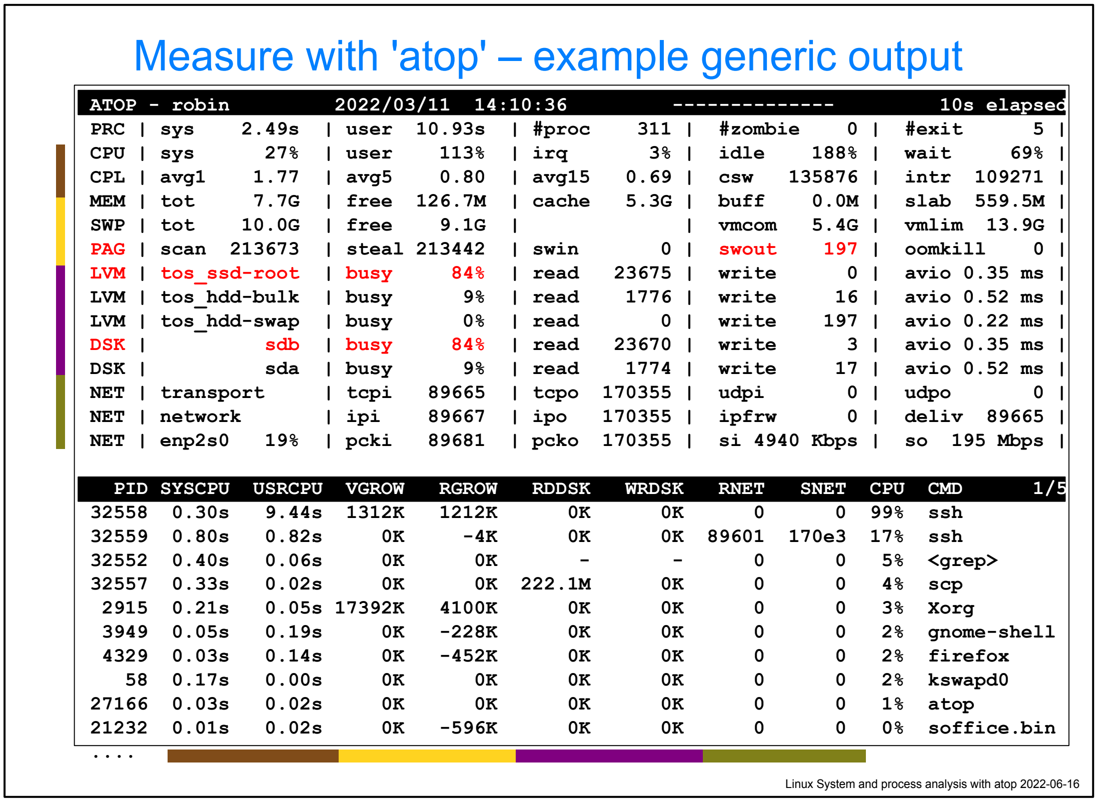
Generic tools	sar	vmstat	iostat	top	atop
Live measurements	✔	✔	✔	✔	✔
Historical data (logging)	✔	✘	✘	✘	✔
System-level data					
Processor	✔	✔	✔	✔	✔
Memory	✔	✔	✘	✔	✔
Disk	✔	✘	✔	✘	✔
Network	✔	✘	✘	✘	✔
Process-level data					
Processor	✘	✘	✘	✔	✔
Memory	✘	✘	✘	✔	✔
Disk	✘	✘	✘	✘	✔
Network	✘	✘	✘	✘	✔ with *netatop* kernel module

Linux System and process analysis with atop 2022-06-16

$Timing(3m): start slide around 16m$

There are many tools available on Linux for measuring the state of your system

- sar allows you to do measurements of CPU, Memory, Disk and network activity. Because it can be told take periodic snapshots and log thise, you can also see what happened "last night around 21:15". It cannot tell you **what process** was using this, just a total on the system
- vmstat and iostat can give you information about processor, memory (vmstat) and IO (iostat), but only "in the moment". These two also only look at the total system state
- top is a very popular tool that does allow you to look at both the system state and the per-process state, but only for CPU and memory usage, not for IO and network. Also, it does not do logging so you cannot inspect previous "episodes" when the system was misbehaving
- atop (www.atoptool.nl) can do all of these. And with the help of a kernel module, also downloadable (source) from www.atoptool.nl, you can even get per process network bandwidth consumption, right now and in the past.



$Timing(4m): start slide around 19m

Here we see the "generic" output page of atop.

The top part of the screen is overall system state, with indicators in red (and blinking) for resources that are (close to) overloading (according to some rules of thumb)
We can see lines for:

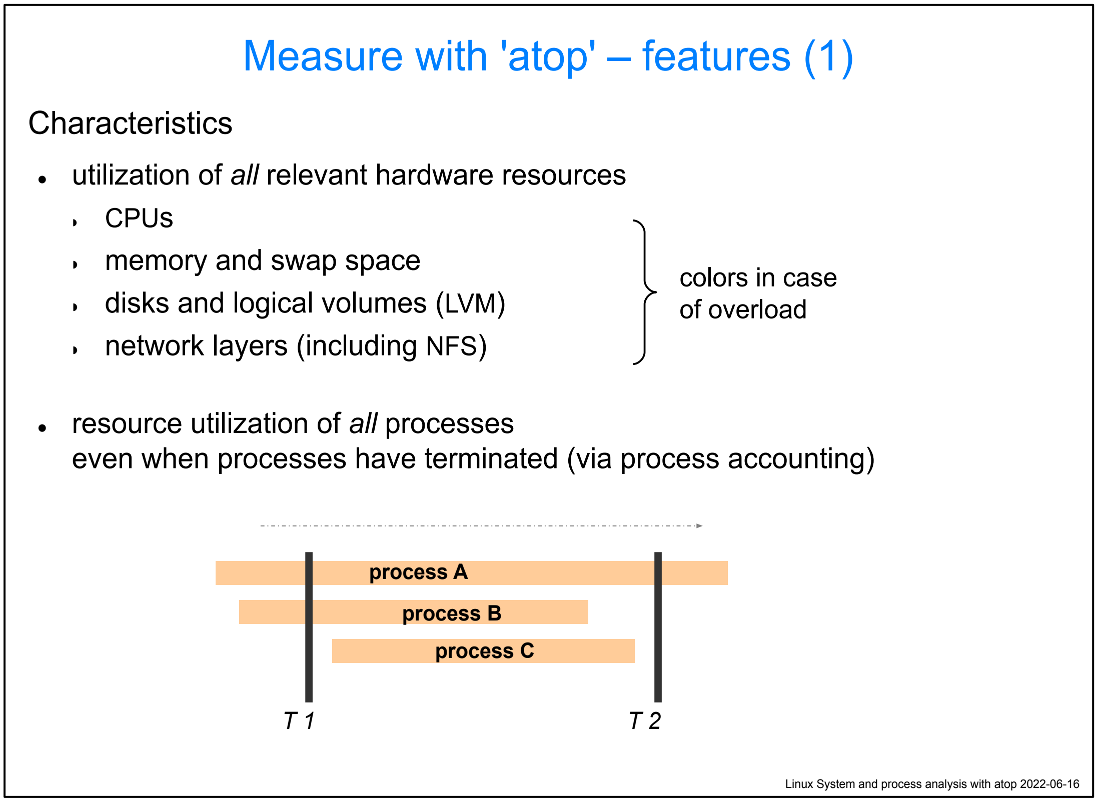
- CPU and Processes (PRC/CPU/CPL). This includes the number of processes, the number of exited processes (5 here), interrupts and the load averages.
- Memory, swapping and page activity. Here we see that 197 pages have been swapped out and that 213673 pages were scanned to see if we can swap them
- Disk activity (LVM/MDD/DSK). This lists the %busy (sdb is busy here 84% of the time), reads and writes and more.
- Network (tranport, IP, interface). Here we see how much the network layers were doing.

For Processes, we see the same categories:

- SYSCPY, USRCPU: CPU usage **since the last snapshot**

- VGROW, RGROW: virtual/resident memory **growth since the last snapshot**
- RDDSK, WRDSK: the volume of disk reads/writes **since the last snapshot**
- RNET, SNET: the number of TCP/UDP packets received/sent **since the last snapshot**

Note that if you make your screen wider, there will be more information. For example, the CPU line will show the frequency scaling that is going on, the memory line will show how many pages in the cache are dirty (waiting to be written out) and for processes, you can see the number of theads and the amount of schedule waiting time it has encountered.



$Timing(2m): start slide around 23m$

An important difference between top and atop, is that atop turns on process accounting, making it possible to keep track of processes that have exited since the last snapshot.

The graph shows why this is important.

- For process A, both top and atop will show the same: the measured resource consumption up to T1 and up to T2l
- For process B, top will only report about the consumption at T1, but it does not show how much it used at T2. atop will show all o this
- For process C, top won't even know that it existed. Atop will report on the resources it consumed during its existence, eventhough it did not exist at either snapshot.

Note that this means that for top, the "processes" part of the screen will not sum up to what is shown in the "System" part of the screen, but atop will.

Measure with 'atop' – features (2)

Data recording

- permanent recording (default interval: 10 minutes)

`$ atop -r`	today's data since midnight
`$ atop -r 20220223`	data of specific date
`$ atop -r yy`	data of day before yesterday

  - branch to interval (**b**)
  - next interval (**t**)
  - previous interval (**T**)
  - rewind (**r**)

  - daily logfiles stored in directory **/var/log/atop**

- incidental recording

`$ atop -w /tmp/trial.atop 60 10`	record 10 intervals of 60 seconds
`$ atop -r /tmp/trial.atop`	data of specific date

- file format might change: convert with command **atopconvert**

$Timing(2m): start slide around 25m$

atop can be instructed to make regular snapshots of the entire system. You can read these recorded files and see what happened.

You can then walk through these logs with the regular atop interface, where 't' will skip to the next snapshot and 'T' will get you back to the previous one.

Measure with 'atop' – features (3)

Characteristics

- dynamic scaling for additional columns (wider screens)

- per interval only show
  - *utilized* system resources (*all* system resources: `f`)
  - *active* processes/threads (*all* processes/threads: `a`) scroll with arrow keys

- show individual threads for multi-threaded processes (`y`)

- select process/thread information

generic – default	(`g`)	various	(`v`)	disk	(`d`)
scheduling	(`s`)	memory	(`m`)	network	(`n`)

- **sort** criterion with same metrics

cpu	(`C`)	disk	(`D`)	autofocus	(`A`)
memory	(`M`)	network	(`N`)		

$Timing(2m): start slide around 27m$

atop has several ways to slide and dice your data:

- You can zoom in on individual threads (y)
- You can select to per process get **more** data about a specific resource (eg more memory info, more disk IO details, memory etc)
- You can sort not just on CPU consumption, but also over the other resources

Measure with 'atop' – features (4)

Other characteristics

- filter processes: view only
  - per user (`U`) – regular expression
  - program name (`P`) – regular expression
  - command line info (`/`) – regular expression
  - PIDs (`I`) – comma-separated list
  - per container (`J`) – container ID

- accumulate utilization of processes
  - per user (`u`)
  - per program (`p`)
  - per container (`j`)

- miscellaneous
  - pause measurement (`z`) interval (`i`) full command line (`c`)
  - utilization per second (`1`) kill (`k`)

$Timing(2m): start slide around 29m$

You can tell atop to filter in the process view

You can tell atop to add up all the resource usage per user, program-name or container

Measure with 'atop' – features (5)

Parsing data

- generate *parseable* output with ***-Plabel[,label]...***
  - guaranteed format and order of metrics
  - for live and recorded measurements
  - generic layout

```
$ atop -r  -PMEM,PRM
...
SEP
MEM systema 1520859311 2022/03/12 13:55:11 600 4096 1980942 .....
PRM systema 1520859311 2022/03/12 13:55:11 600 1 (systemd) S ....
PRM systema 1520859311 2022/03/12 13:55:11 600 2 (kthreadd) S ...
...
SEP
...
```

metrics for one interval

label host epoch date time interval label-specific data

  - labels
    - system level metrics: **CPU**, **MEM**, **SWP**, **PAG**, **DSK**, **NET**, ...
    - process level metrics: **PRC**, **PRM**, **PRD**, **PRN**, **PRG**, ...

$Timing(1m): start slide around 31m$

Since atop logs data, it can be interesting to also use atop in a pipeline to create reports.

Using the -P flag allows you to tell which system or process oriented values you want to see.

Measure with 'atop' – use case

Determine time/reason of process termination

- example: system time inaccurate because **ntpd/chronyd** disappeared

```
$ atop -r y | egrep '^ATOP|chronyd'
....

ATOP - systema     2022/03/12  15:39:01      --------------      10m0s elapsed
 1124  0.00s  0.00s      0K      0K      0K      0K      2      3    0% chronyd
ATOP - systema     2022/03/12  15:49:01      --------------      10m0s elapsed
 1124  0.00s  0.01s      0K      0K      0K      0K      2      3    0% chronyd
ATOP - systema     2022/03/12  15:59:01      --------------      10m0s elapsed
 1124  0.00s  0.00s      0K      0K       -       -      1      3    0% <chronyd>
ATOP - systema     2022/03/12  16:09:01      --------------      10m0s elapsed
ATOP - systema     2022/03/12  16:19:01      --------------      10m0s elapsed
```

```
$ atop -r y  -b 15:59 -v
/chronyd
ATOP - systema     2022/03/12  15:59:01      --------------      10m0s elapsed
....
  PID  PPID  RUID     RGID      STDATE     STTIME     ENDATE     ENTIME   ST  EXC S   CPU CMD
 1124     -  chrony   chrony  2022/03/12 08:55:12  2022/03/12  15:56:41 -S     9 E    0% <chronyd>
/kill
31622     -  root     root    2022/03/12 15:56:41  2022/03/12  15:56:41 NE     0 E    0% <kill>
```

interactive session

Linux System an

$Timing(2m): start slide around 32m$

Let's take a look at this case study:

On a system, we noticed that ntpd or chronyd was not running yesterday, leading to the clock to skew.

With atop -r y | egrep '^ATOP|chronyd' we get the ATOP lines (with the time stamp) and the chrony lines to show that chronyd was running

But we also see that it had exited in the snapshot of 15:59.

Lets start an interactive session and start at the snapshot of 15:59.

There we can see that cronyd exited with exit code 9 at 15:56:41, but also that the kill command also exited at 15:56:41, run by root…

Measure with 'atop' – processor analysis

```
ATOP - myhost                2022/02/23  10:40:33             10 seconds elapsed
PRC | sys    0.50s | user    3.05s | #proc     114 | #zombie    0 | #exit      4
CPU | sys       5% | user      31% | irq        0% | idle    163% | wait      1%
cpu | sys       3% | user      25% | irq        0% | idle     70% | cpu001 w  1%
cpu | sys       2% | user       5% | irq        0% | idle     92% | cpu000 w  0%

  PID  SYSCPU  USRCPU  VGROW  RGROW USERNAME THR  ST EXC  S  CPU CMD
30261   0.18s   2.88s 15236K 10528K gerlof     1  N-   -  R  31% grep
    ?   0.11s   0.02s     0K     0K gerlof     0  NE   2  E   1% <grep>
30204   0.06s   0.05s     0K     0K gerlof     1  --   -  S   1% sshd
30255   0.05s   0.02s  4120K   612K gerlof     1  N-   -  S   1% find
    ?   0.03s   0.03s     0K     0K gerlof     0  NE   2  E   1% <grep>
    ?   0.03s   0.02s     0K     0K gerlof     0  NE   2  E   0% <grep>
30198   0.02s   0.01s     0K     0K gerlof     1  --   -  R   0% atop
    ?   0.02s   0.01s     0K     0K gerlof     0  NE   2  E   0% <grep>
27511   0.00s   0.01s     0K     0K gerlof     4  --   -  S   0% soffice.bin
17668   0.00s   0.00s     0K     0K ntp        1  --   -  S   0% ntpd
....
```

Keystroke s shows:

```
  PID TRUN TSLPI TSLPU  POLI NICE PRI RTPR CURCPU ST EXC  S  CPU CMD
30261    1     0     0  norm    0 124    0      1 N-   -  R  31% grep
    ?    0     0     0     -    -   -    -      - NE   2  E   1% <grep>
30204    0     1     0  norm    0 115    0      1 --   -  S   1% sshd
30255    0     1     0  norm    0 116    0      0 N-   -  S   1% find
...
```


$Timing(2m): start slide around 34m$

Let's now look at how the different lines in atop's system output and the columns in the processes output relate.

We can see that can see that 3.05 user time was used, we also see which processes those were. That includes exited processes (between angle brackets and with an exit code listed). top would not take those into account.

By using the 's' command (scheduling) we get to see more scheduling information, such as the priority, the number of theads in state R (running), I (Interupptible sleep), U (uninterruptible sleep).

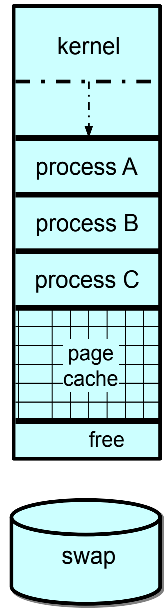
Measure with 'atop' – memory analysis

```
MEM | tot  993.8M | free   5.6M | cache 846M | buff   4.8M | slab  32.4M |
SWP | tot    2.0G | free   2.0G |             | vmcom 398.3M | vmlim  2.5G |
PAG | scan  22440 | stall    55 |             | swin       2 | swout     3 |

  PID  SYSCPU  USRCPU  VGROW  RGROW USERNAME THR  ST EXC  S  CPU CMD
30484   0.46s   7.02s  1548K  1548K gerlof     1  --   -  R  75% grep
27238   0.03s   0.27s     0K     4K wwwadm     5  --   -  R   3% firefox-
30555   0.01s   0.15s  8640K  5616K root       1  N-   -  D   2% mrtg
27187   0.08s   0.07s     0K     0K wwwadm     1  --   -  R   2% sshd
   43   0.04s   0.00s     0K     0K root       1  --   -  S   0% kswapd0
27511   0.00s   0.03s     0K     0K gerlof     4  --   -  R   0% soffice.
27470   0.01s   0.02s     0K     0K gerlof     1  --   -  S   0% sshd
30198   0.01s   0.01s     0K     0K gerlof     1  --   -  R   0% atop
17963   0.01s   0.00s     0K     0K apache     1  --   -  S   0% httpd
30557   0.00s   0.01s  4088K  1844K mailman    1  N-   -  D   0% python
27251   0.00s   0.00s     0K   -12K wwwadm     1  --   -  S   0% gvim
17964   0.00s   0.00s     0K     0K apache     1  --   -  S   0% httpd
17919   0.00s   0.00s     0K     0K root       1  --   -  S   0% httpd
.....
```

Keystroke **m** shows:

```
  PID MINFLT MAJFLT  VSTEXT  VSIZE  RSIZE  VGROW  RGROW  MEM CMD
27511      0      0    242K 246.3M 89788K     0K     0K   9% soffice.
27238      1      1   9516K  98.3M 32884K     0K     4K   3% firefox-
27251      0      0   2054K 19456K 10240K     0K   -12K   1% gvim
17963      2      0    249K 21092K  6312K     0K     0K   1% httpd
17964      3      0    249K 20964K  6236K     0K     0K   1% httpd
.....
```



$Timing(2m): start slide around 36m$

In the general overview, we will have information on a system level about memory consumption: the amount of free memory, in the page cache (cache + buff) and the amount of dynamically allocated memory by the kernel

Per process, we can see how memory usage by the processes has **grown** since the last snapshot, both the Virtuam footprint as the amount of that that is actually resident in memory. It is one thing to know a process has a certain footprint, but it is more intesting to see how much it is growing (just like we are not so much interested in how much CPU time a process has used up to know in its entire runtime, but it is interesting to see how much extra CPU is has used since the last snapshot).

Typing m will give you more memory related data: page faults, the code footprint. Making the terminal wider will also show you how much of the footprint is shared, is in the swap space and how much memory is locked.

Measure with 'atop' – disk analysis

```
LVM |   vg00-lvusr | busy     95% | read    6668 | write    2592 | avio 1.17 ms |
LVM |  vg00-lvhome | busy      2% | read       0 | write      28 | avio 9.21 ms |
LVM |  vg00-lvroot | busy      1% | read      12 | write      15 | avio 5.00 ms |
LVM |   vg00-lvvar | busy      1% | read       0 | write      10 | avio 10.9 ms |
DSK |          sda | busy     95% | read    5439 | write     172 | avio 1.94 ms |

  PID  SYSCPU  USRCPU  VGROW  RGROW  RDDSK   WRDSK ST EXC S  DSK  CMD
14318   0.61s   0.04s     0K     0K 77132K     0K --   - D  95%  grep
 1915   0.01s   0.00s     0K     0K     0K  3584K --   - D   4%  jbd2/dm-5-8
 1959   0.00s   0.00s     0K     0K     0K    48K --   - S   0%  flush-253:4
 1920   0.00s   0.00s     0K     0K     0K    28K --   - S   0%  jbd2/dm-2-8
 2283   0.00s   0.00s     0K     0K     0K     4K --   - S   0%  NetworkManager
 2749   0.00s   0.00s     0K     0K     0K     4K --   - S   0%  ksmtuned
  494   0.00s   0.00s     0K     0K     0K     4K --   - S   0%  jbd2/dm-0-8
 1905   0.00s   0.00s     0K     0K     0K     4K --   - S   0%  jbd2/dm-4-8
 3771   0.03s   0.07s     0K     0K     0K     0K --   - S   0%  plugin-contain
....
```

Keystroke `d` shows:

```
  PID       TID      RDDSK      WRDSK     WCANCL      DSK     CMD
14318         -     77132K         0K         0K      95%     grep
 1915         -         0K      3584K         0K       4%     jbd2/dm-5-8
 1959         -         0K        48K         0K       0%     flush-253:4
 1920         -         0K        28K         0K       0%     jbd2/dm-2-8
 2283         -         0K         4K         0K       0%     NetworkManager
.....
```


$Timing(2m): start slide around 38m$

For disk analysis, in the system area, atop shows # of reads writes and how busy the disk or LVM volume is. More column will also show the volume of the reads and writes.

Per process you can see the volume of data that was written by each process since the previous snapshot

pressing 'd', also shows the number of writes that were cancelled.

Measure with 'atop' – network analysis

```
NET | transport     | tcpi  133548 | tcpo   18457 | udpi        2 | udpo        0 |
NET | network       | ipi   133562 | ipo    18457 | ipfrw    0    | deliv 133559 |
NET | eth0      75% | pcki  130078 | pcko  629402 | si 8818 Kbps | so  752 Mbps |

  PID  SYSCPU  USRCPU VGROW  RGROW  RDDSK WRDSK  RNET  SNET S  NET CMD
14387   2.19s   0.12s    0K     0K     0K    0K  13e4  17e3 S 100% attract
14326   0.10s   0.15s    0K     4K     0K    0K  3426  1903 S   0% ssh
10316   0.00s   0.00s    0K     0K     0K    0K    20    10 S   0% ssh
 7135   0.10s   0.39s    0K   516K     0K    0K    12     8 S   0% thunderbird-bi
10310   0.00s   0.00s    0K     0K     0K    0K     1     2 S   0% ssh
 3655   0.45s   0.83s  232K   104K     0K    0K     0     0 S   0% gnome-terminal
.....
```

Keystroke **n** (with kernel module 'netatop') shows:

```
  PID      TCPRCV TCPSND  UDPRCV UDPSND     BANDWI   BANDWO   NET  CMD
14387      130084  16558       0      0  7284 Kbps 750 Mbps  100%  attract
14326        3426   1903       0      0  1520 Kbps 109 Kbps    0%  ssh
10316          20     10       0      0     1 Kbps   1 Kbps    0%  ssh
 7135          12      8       0      0     0 Kbps   0 Kbps    0%  thunderbird-bi
10310           1      2       0      0     0 Kbps   0 Kbps    0%  ssh
 3655           0      0       0      0     0 Kbps   0 Kbps    0%  gnome-terminal
.....
```


$Timing(2m): start slide around 40m$

Finally, we can see system level networking information, not only on transport (TCP/UDP), network (IP) and interface level, but if you use NFS, it will also show lines for that.

Per process, we will see the number of TCP and UDP sends and receives and the bandwidth in and out.

For per process information, the netatop module is required.

Live demo

A bold strategy :-)

Let's catch a memory leak…

$Timing(10m): start slide around 42m$

The atop demo file used can be found here:
for atop 2.7: https://www.atoptool.nl/download/leakerdemo-2.7
for atop 2.8: https://www.atoptool.nl/download/leakerdemo-2.8

Download the file, then use:
atop -r <filename>
use "t" to step from snapshot to snapshot (and "T" to go back one snapshot)

20:50:17 system in use but in steady state, A youtube video is playing in firefox, openoffice is used…
20:50:47 some memory users are started
20:51:17 no new usemems, memory pressure there, but still 1.6GB in cache
20:52:17 leaker is starting up, cache is shrinking
20:52:27 first swapouts happening, disk 14% busy
20:52:37 Kernel less happy, more swaps, disk 47% busy, firefox loses mem
20:52:57 swap disk 98% busy
20:53:37 leaker grows less aggressively, but is losing RSIZE
20:53:57 leaker exits, sdb now busy with reads
20:54:27 usemems leave, sdb relatively quiet

Measure with 'atopsar' – features

Characteristics

- system reports

live: **atopsar** [-**flags**...] **interval** [**samples**]
past: **atopsar** [-**flags**...] [-**r** **file**|**date**|**y**...]

- examples: live CPU utilization

```
$ atopsar -c 60 5
myhost       ....                                       x86_64  2022/02/23

14:22:11  cpu %usr %nice   %sys %irq %softirq   %steal    %wait %idle
14:23:11  all    2     0      8    1        1        0       88     0
14:24:11  all   32     0     38    1        1        0       28     0
14:25:11  all   43     0     45    2        1        0        0     9
14:26:11  all    2     0      2    1        1        0       15    79
14:27:11  all    2     0      1    0        0        0       11    86
```

```
$ atopsar -A -r 20220223
$ atopsar -A -r yyyyy
```


$Timing(1m): start slide around 52m$

UNIX systems used to have a "System Activity Reporter" that uses logs that were kept.

Since atop keeps logs as well, atop also offers a sar like interface with the atopsar command.

Installation

Installation of **`atop`**/**`atopsar`**

- distribution repository
  - Ubuntu, Debian, ...
  - RedHat, Fedora, … (from EPEL)
- website: [**`www.atoptool.nl`**](http://www.atoptool.nl) (RPM or source tarball)
- GitHub: **`git://github.com/atoptool/atop.git`**

Installation of **`netatop`** kernel module

- website [**`www.atoptool.nl`**](http://www.atoptool.nl) (source tarball)

$Timing(1m): start slide around 53m$

Tux logo: https://commons.wikimedia.org/wiki/File:Tux.svg, License: The copyright holder of this file allows anyone to use it **for any purpose, provided that** the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
Attribution: lewing@isc.tamu.edu Larry Ewing and The GIMP

Steoscope Image by https://vectorportal.com/vector/stethoscope-image/22292 License https://creativecommons.org/licenses/by/4.0/