Laboratory Work №4.
Memory Management in Linux.

In Linux, the following basic rule applies: an unused page of RAM is considered a lost memory. RAM is wasted not only for data used by applied applications. It also stores data for the kernel itself and, most importantly, the data stored on the hard disk can be displayed in this memory, which is used for super-fast access to them - the top command indicates this in the columns "buffers / cache", "disk cache" or "cached". Cached memory is essentially free, as it can be quickly released if the running (or just started) program requires memory.

Saving the cache means that if someone else needs the same data again, there is a good chance that they will still be cached in RAM.

Therefore, the first thing you can use on your system is the free command, which will provide you with initial information about how your RAM is used.

```
xubuntu-home:~# free
     total       used       free     shared    buffers     cached
Mem: 1506       1373        133          0         40        359
-/+ buffers/cache:        972        534
Swap:                     486         24        462
```

The line -/+ buffers / cache shows how much memory is used and how much memory is free in terms of its use in applications. In this example, applications already use 972 MB of memory and another 534 MB of memory can be used.

Generally speaking, if you use at least a little swap memory, then using memory will not affect the performance of the system at all.

But if you want to get more information about your memory, then you should check the file/proc/meminfo:

```
xubuntu-home:~# cat /proc/meminfo
MemTotal:        1543148 kB
MemFree:          152928 kB
Buffers:           41776 kB
Cached:           353612 kB
SwapCached:         8880 kB
Active:           629268 kB
Inactive:         665188 kB
Active(anon):     432424 kB
Inactive(anon):   474704 kB
Active(file):     196844 kB
Inactive(file):   190484 kB
Unevictable:         160 kB
Mlocked:             160 kB
```

```
HighTotal:        662920 kB
HighFree:          20476 kB
LowTotal:         880228 kB
LowFree:          132452 kB
SwapTotal:        498684 kB
SwapFree:         470020 kB
Dirty:                44 kB
Writeback:             0 kB
AnonPages:        891472 kB
Mapped:           122284 kB
Shmem:              8060 kB
Slab:              56416 kB
SReclaimable:      44068 kB
SUnreclaim:        12348 kB
KernelStack:        3208 kB
PageTables:        10380 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     1270256 kB
Committed_AS:    2903848 kB
VmallocTotal:     122880 kB
VmallocUsed:        8116 kB
VmallocChunk:     113344 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       4096 kB
DirectMap4k:       98296 kB
DirectMap4M:      811008 kB
```

That means MemTotal (Total memory) and MemFree (Free memory) is understandable for everyone, the remaining values are explained further:

CACHED

The page cache in the Linux system ("Cached:" in meminfo) is in most systems the largest memory consumer. Every time you perform read () operation from a file located on a disk, the data is read into memory and placed in a page cache. After the read () operation is completed, the kernel can simply throw out the page of memory, since it is not being used. However, if you perform a second read operation on the same part of the file, the data will be read directly from the memory and no access to the disk will be made. This improves performance tremendously and, therefore, in Linux, caching of pages is intensively used: the bet is that if you access a page of disk memory, you will soon contact it again.

dentry/inode caches

Every time you execute an "ls" operation on the file system (or any other operation: open (), stat (), etc.), the kernel needs the data that is on the disk.

The kernel analyzes this data on the disk and places it in some data structures that are independent of the file system so that they can be processed in the same file system in the same way. In the same way as page caching in the above examples, the kernel can erase these structures after the "ls" command is completed. Nevertheless, the same assumption is made as before: if you once considered these data, you will necessarily read them again. The kernel stores this information in several places in the "cache", which are called the cache memory dentry and inode. Cache memory dentries are common to all file systems, but each file system has its own inodes cache.

This RAM is in meminfo part of "Slab:"

You can view a different cache memory and find its size using the following command:

```
head -2 /proc/slabinfo; cat /proc/slabinfo  | egrep dentry\|inode
```

Buffer Cache

The buffer cache ("Buffers: " in meminfo) is a close cousin to the dentry/inode cache. Data dentries and inodes, placed in memory, are a description of the structures on the disk, but they are arranged in different ways. This may be due to the fact that we have a structure in the copy located in the memory, like a pointer, but it does not exist on the disk. It may also happen that the bytes on the disk will be located in the wrong order, as the processor needs.

Displaying memory in a command top: VIRT, RES и SHR

If you run the top command, the three lines will describe the use of memory. You must understand their meaning in order to understand how much memory is required for your server.

VIRT is an abbreviation of the virtual size of a process and represents the total amount of memory used: memory that is mapped to itself (for example, the memory of the video card for server X), files on the disk that are mapped to memory (especially Shared libraries) and memory shared with other processes. The value of VIRT indicates how much memory is currently available to the program.

RES is an abbreviation of resident size (the size of the resident part) and is an accurate indication of how much is actually consumed by the process of real physical memory. (Which also corresponds to the value found directly in the %MEM column). This value is almost always smaller than the size of the VIRT, because Most programs depend on the C library.

SHR shows how much of the value of VIRT is actually shared (from memory or through the use of libraries). In the case of libraries, this does not

necessarily mean that the entire library is in resident memory. For example, if the program uses only several library functions, then the entire library will be used to display it, which will be taken into account in the values of VIRT and SHR, but in fact, the library part containing the functions used will be loaded and this will be taken into account in the value RES.

Swap memory - swap

First of all, the kernel tries not to let the value of free RAM approach 0 bytes. This is due to the fact that when you need to free up the RAM, you usually need to allocate a little more memory. This is due to the fact that our kernel requires a kind of "workspace" to perform its actions, and therefore, if the amount of free RAM becomes zero, the kernel can do nothing more.

Based on the total amount of RAM and the ratio of its various types (high/low memory), the kernel heuristically determines the amount of memory as a working space, under which it feels comfortable. When this value is reached, the kernel begins to return memory for the other various tasks described above. The kernel can reclaim memory from any of these tasks.

However, there is another memory consumer, which we may have already forgotten: **user application data**.

Once the kernel decides that it does not need to receive memory from any other sources that we described earlier, it starts the swap memory. During this process, it receives the data of the user applications and writes them to a special place (or places) on the disk. Note that this happens not only when the RAM is close to populating, the kernel may decide to transfer the data in RAM as well, if they have not been used for some time (see "Swap memory").

For this reason, even a system with a huge amount of RAM (even if properly configured) can use swap memory. There are many memory pages that contain data for user applications, but these pages are rarely used. All this is the reason to transfer them to the swap partition and use the RAM for other purposes.

You can use the ***free*** command to check whether the swap memory is used; For example, which I have already used above, the last line of output data shows information about the size of the swap memory:

```
xubuntu-home:~# free
             total       used       free     shared    buffers     cached
Mem:          1506       1373        133          0         40        359
-/+ buffers/cache:        972        534
Swap:          486         24        462
```

We see that this computer already uses 24 megabytes of swap memory and 462 MB is available for use.

Thus, the very fact of using swap memory is not proof that there is too little RAM in the system with its current workload. The best way to determine this is using the ***vmstat*** command - if you see that many swap pages are moved to and from the disk, this means that the swap memory is being used actively, that the system is "stalling" or that it needs a new RAM because it will speed up the swap Application data.

On my laptop, when it's idle, it looks like this:

```
~ # vmstat 5 5
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 1  0      0 2802448  25856 731076    0    0    99    14  365  478  7  3 88  3
 0  0      0 2820556  25868 713388    0    0     0     9  675  906  2  2 96  0
 0  0      0 2820736  25868 713388    0    0     0     0  675  925  3  1 96  0
 2  0      0 2820388  25868 713548    0    0     0     2  671  901  3  1 96  0
 0  0      0 2820668  25868 713320    0    0     0     0  681  920  2  1 96  0
```

Note that in the output of the free command, you have only 2 values related to the swap: free memory and used memory, but for swap memory there is also another important value: **Swap cache** (The cache of the paging memory).

Swap Cach

Caching the swap memory is essentially like page caching. The user application data page written to disk is very similar to the file data page on the disk. Each time a page is read from a swap file ("**si**" in **vmstat**), it is placed in the paging cache. Just like page caching, it's all done by the kernel. The kernel decides whether to return a specific page back to disk. If this is necessary, you can check if there is a copy of this page on the disk and you can simply throw the page out of memory. This will save us the cost of rewriting the page to disk.

Caching swap memory is really useful only when we read data from swap memory and never write to it. If we write to a page, the copy on the disc will not match the copy that is in memory. If this happens, then we should record the page on the disk just as we did the first time. Despite the fact that the cost of saving the entire page is greater than the cost of writing a small modified piece, the system will perform better.

Therefore, to find out that the swap memory is actually used, we need to subtract the SwapCached value from the SwapUsed value, you can find this information in / proc / meminfo.

Swap memory

When an application needs memory, and all RAM is completely occupied, the kernel has two ways to free up memory: it can either reduce the size of the disk cache in RAM by removing obsolete data, or it can flush a partition with a few rarely used ones Portions (pages) of the program. It is difficult to predict which method will be more effective. The kernel, based on the recent history of actions in the system, makes an attempt to approximately guess at the moment the effectiveness of each of these two methods.

Prior to kernels 2.6, the user was not able to influence these estimates, so that there could be situations where the kernel often made the wrong choice, which led to stalling and poor performance. In version 2.6, the memory swap situation was changed.

The memory paging is assigned a value from 0 to 100, which changes the balance between paging the application memory and freeing the memory cache. With a value of 100, the kernel will always prefer to find inactive pages and drop them to disk in the swap partition; In other cases this reset will be performed depending on how much memory the application occupies and how difficult it is to cache it when searching and deleting inactive items.

By default, this value is set to 60. A value of 0 gives something close to the old behavior when applications that need memory forced to slightly reduce the size of the RAM cache. For laptops, for which it is preferable to have discs with a slower rotation speed, it is recommended to use a value of 20 or less.

Task for laboratory work

1. Processes and memory used.

1.1 Issue 10 processes consuming the largest amount of memory

 # ps -auxf | sort -nr -k 4 | head -10

1.2 Issue 10 processes consuming the most CPU resource

 # ps -auxf | sort -nr -k 3 | head -10

2. **free** – memory usage

The free command shows the total amount of free memory and used by the system physical memory and swap memory, as well as the buffer sizes used by the kernel.

 # *free*

Example output:

```
total       used       free     shared    buffers     cached
Mem:      12302896    9739664    2563232          0     523124    5154740
-/+ buffers/cache:    4061800    8241096
Swap:      1052248          0    1052248
```

3***. iostat*** – average CPU usage, disk activity

The iostat command displays CPU usage statistics, as well as I/O statistics for devices, partitions, and network file systems (NFS).

 # *iostat*

Example output:

```
Linux 2.6.18-128.1.14.el5 (www03.nixcraft.in)      06/26/2009

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           3.50    0.09    0.51    0.03    0.00   95.86

Device:            tps   Blk read/s   Blk wrtn/s   Blk read   Blk wrtn
sda              22.04        31.88       512.03   16193351  260102868
sda1              0.00         0.00         0.00       2166        180
sda2             22.04        31.87       512.03   16189010  260102688
sda3              0.00         0.00         0.00       1615          0
```

4. sar – collecting and reporting system activity data

The **sar** command is used to collect information about system activity and to output it as a report or to save it. To see the value of the network activity scanner, type:

 # *sar -n DEV | more*

In order to see the values of network activity counters, starting from the 24th:

 # *sar -n DEV -f /var/log/sa/sa24 | more*

With the sar command, you can also output data in real time:

 # *sar 4 5*

Example output:

```
Linux 2.6.18-128.1.14.el5 (www03.nixcraft.in)              06/26/2009

06:45:12 PM       CPU     %user     %nice   %system   %iowait    %steal     %idle
06:45:16 PM       all      2.00      0.00      0.22      0.00      0.00     97.78
06:45:20 PM       all      2.07      0.00      0.38      0.03      0.00     97.52
06:45:24 PM       all      0.94      0.00      0.28      0.00      0.00     98.78
06:45:28 PM       all      1.56      0.00      0.22      0.00      0.00     98.22
06:45:32 PM       all      3.53      0.00      0.25      0.03      0.00     96.19
Average:          all      2.02      0.00      0.27      0.01      0.00     97.70
```

5. *mpstat* – using the multiprocessor

The ***mpstat*** command displays the activity data of each available processor, processor 0 will be the first. The ***mpstat -P ALL*** command displays the average usage of resources for each processor:

 # mpstat -P ALL

Example output:

```
Linux 2.6.18-128.1.14.el5 (www03.nixcraft.in)              06/26/2009

06:48:11 PM  CPU   %user   %nice    %sys %iowait    %irq   %soft  %steal   %idle    intr/s
06:48:11 PM  all    3.50    0.09    0.34    0.03    0.01    0.17    0.00   95.86   1218.04
06:48:11 PM    0    3.44    0.08    0.31    0.02    0.00    0.12    0.00   96.04   1000.31
06:48:11 PM    1    3.10    0.08    0.32    0.09    0.02    0.11    0.00   96.28     34.93
06:48:11 PM    2    4.16    0.11    0.36    0.02    0.00    0.11    0.00   95.25      0.00
06:48:11 PM    3    3.77    0.11    0.38    0.03    0.01    0.24    0.00   95.46     44.80
06:48:11 PM    4    2.96    0.07    0.29    0.04    0.02    0.10    0.00   96.52     25.91
06:48:11 PM    5    3.26    0.08    0.28    0.03    0.01    0.10    0.00   96.23     14.98
06:48:11 PM    6    4.00    0.10    0.34    0.01    0.00    0.13    0.00   95.42      3.75
06:48:11 PM    7    3.30    0.11    0.39    0.03    0.01    0.46    0.00   95.69     76.89
```

6. *pmap* – the use of RAM processes

The ***pmap*** command provides information about the allocation of memory between processes. Using this command will find the cause of the bottlenecks associated with the use of memory.

 # pmap -d PID

In order to get information about using the process memory with ***pid # 47394***, type:

 # pmap -d 47394

Example output:

```
47394:   /usr/bin/php-cgi
Address           Kbytes Mode  Offset           Device    Mapping
0000000000400000    2584 r-x-- 0000000000000000 008:00002 php-cgi
0000000000886000     140 rw--- 0000000000286000 008:00002 php-cgi
00000000008a9000      52 rw--- 00000000008a9000 000:00000   [ anon ]
0000000000aa8000      76 rw--- 00000000002a8000 008:00002 php-cgi
000000000f678000    1980 rw--- 000000000f678000 000:00000   [ anon ]
000000314a600000     112 r-x-- 0000000000000000 008:00002 ld-2.5.so
000000314a81b000       4 r---- 000000000001b000 008:00002 ld-2.5.so
000000314a81c000       4 rw--- 000000000001c000 008:00002 ld-2.5.so
000000314aa00000    1328 r-x-- 0000000000000000 008:00002 libc-2.5.so
000000314ab4c000    2048 ----- 000000000014c000 008:00002 libc-2.5.so
.....
......
..
00002af8d48fd000       4 rw--- 0000000000006000 008:00002 xsl.so
00002af8d490c000      40 r-x-- 0000000000000000 008:00002 libnss_files-2.5.so
00002af8d4916000    2044 ----- 000000000000a000 008:00002 libnss_files-2.5.so
00002af8d4b15000       4 r---- 0000000000009000 008:00002 libnss_files-2.5.so
00002af8d4b16000       4 rw--- 000000000000a000 008:00002 libnss_files-2.5.so
00002af8d4b17000  768000 rw-s- 0000000000000000 000:00009 zero (deleted)
00007fffc95fe000      84 rw--- 00007ffffffea000 000:00000   [ stack ]
ffffffffff600000    8192 ----- 0000000000000000 000:00000   [ anon ]
mapped: 933712K    writeable/private: 4304K    shared: 768000K
```

The last line is very important:

- **mapped: 933712K -** total amount of memory allocated for files;
- **writeable/private: 4304K -** total private address space;
- **shared: 768000K -** the total amount of address space that this process uses in conjunction with other processes.