Laboratory Work №3.
Process Management.

The first thing we will learn is to determine which processes are running in the system. To do this, Linux (like all UNIX-based systems) has the ***ps*** command.

If you run it without any parameters, it displays a list of processes running in the current session. If you want to see a list of all the processes running on the system, you must set the same command with the parameter ***-ax.***

3.1 Command *ps*

When I looked in the man page devoted to the ***ps*** command, I was amazed at how many options it had. As it turned out, the ***GNU***-version of this program, which is part of Linux, supports options in the style of three different types of UNIX. Options in Unix98 style consist of one or more characters, before which (and) must be a hyphen. Options in the style of BSD have a similar appearance, only used without a hyphen. Options that are specific only to the GNU version are a word that must be preceded by two dashes. They can not be combined as single-letter options of the two preceding types. Thus, there are three peer-to-peer formats for specifying this command:

ps [-options]
ps [options]
ps [-- long_name_options [-- long_name_options] ...]

In this case, the options of different types can not be used in one command. Let's briefly describe the most important options.

The first group of options controls the output of the command. Regardless of the availability of options for this group, the ***ps*** command produces a separate line for each process, but the contents of this string can be different. Depending on the options that are:

- USER — the name of the owner of the process;
- PID — process identifier in the system;
- PPID — the identifier of the parent process;
- %CPU — percentage of CPU time (in percent) allocated
- %MEM — the percentage of real memory used by the process;
- VSZ — virtual process size (in kilobytes);
- RSS — the size of the resident set (the number of 1K pages in memory);

- STIME — time of process start;
- TTY — a reference to the terminal from which the process is started;
- S or STAT — process status;
- PRI — priority of planning;
- NI — the nice value (see the nice description below);
- TIME — how much CPU time the process took;
- CMD or COMMAND — command line for starting a program, executed by this process.

And also other fields, the full list of which is shown on the main page devoted to the ***ps*** command.

The values displayed in most of these fields you will understand without further explanation. In the Process status field, as mentioned above, the following values can be used:

- R — an executable process that expects only the moment when the task scheduler allocates to it the next time quantum;
- S — the process "sleeps";
- D — the process is paging on the disk;
- T — the stopped process;
- Z — the zombie process.

Next to the status indicator may be additional characters from the following set:

- W — the process does not have resident pages;
- < — high-priority process;
- N — low-priority process;
- L — the process has pages locked in memory.

The second group of options regulates which processes are included in the output of the command. To get a list of all processes, you must use the ***ps*** command with the options ***ax*** or ***-A***. The output in these two cases differs only in the CMD field: in the first case, a full command line for starting the program is issued, and in the second case only the name of the running program.

A description of all the ***ps*** program options can not be given here. Therefore, we give only a few examples of its application, which will show how to use this command in a standard situation.

In order to see all the processes in the system using the standard output form:

[user]$ ps –e

You can add the ***-o*** option to the same command, followed by a comma,

[user]$ ps –eo pid,user,cmd

In order to see all the processes in the system using the output form of BSD-systems:

[user]$ ps ax

In order to see all the processes in the system, using the graphical display of the relationship "ancestor-descendant":

[user]$ ps –ef

However, in order to see the "forest" of trees "ancestor-descendant", it is better to use a very interesting analogue of the ***ps -ef*** command - the ***pstree*** command.

In order to see how much % of the CPU and memory are occupied by the processes that you started:

[user]$ ps –u

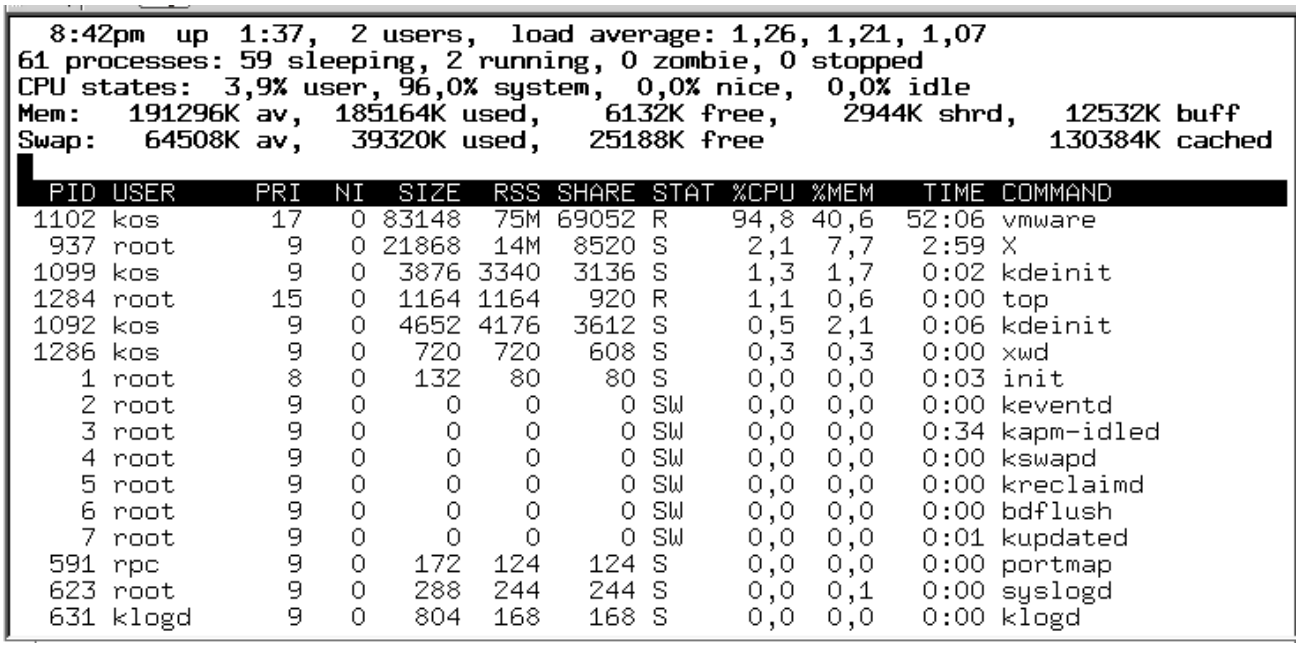
To find out the process priority and the nice value, use the ***-l*** option:

[user]$ ps –l

3.2 Command *top*

The **ps** command allows you to make a "snapshot" of the processes running on the system. Unlike ***ps***, the top command displays the state of the processes and their activity "in real time". Figure 1 shows the terminal window in which the top program is running.

As you can see, the top part of the window shows the astronomical time, the time elapsed since the system was started, the number of users in the system, the number of running processes and the number of processes in different states, data on CPU usage, memory and swap. And then there is a table that describes the individual processes. The number of rows displayed in this table is determined by the size of the window: how many rows are placed, and how many lines are displayed.



```
  8:42pm  up  1:37,  2 users,  load average: 1,26, 1,21, 1,07
61 processes: 59 sleeping, 2 running, 0 zombie, 0 stopped
CPU states:  3,9% user, 96,0% system,  0,0% nice,  0,0% idle
Mem:   191296K av,  185164K used,    6132K free,    2944K shrd,   12532K buff
Swap:   64508K av,   39320K used,   25188K free                  130384K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 1102 kos       17   0 83148  75M 69052 R    94,8 40,6  52:06 vmware
  937 root       9   0 21868  14M  8520 S     2,1  7,7   2:59 X
 1099 kos        9   0  3876 3340  3136 S     1,3  1,7   0:02 kdeinit
 1284 root      15   0  1164 1164   920 R     1,1  0,6   0:00 top
 1092 kos        9   0  4652 4176  3612 S     0,5  2,1   0:06 kdeinit
 1286 kos        9   0   720  720   608 S     0,3  0,3   0:00 xwd
    1 root       8   0   132   80    80 S     0,0  0,0   0:03 init
    2 root       9   0     0    0     0 SW    0,0  0,0   0:00 keventd
    3 root       9   0     0    0     0 SW    0,0  0,0   0:34 kapm-idled
    4 root       9   0     0    0     0 SW    0,0  0,0   0:00 kswapd
    5 root       9   0     0    0     0 SW    0,0  0,0   0:00 kreclaimd
    6 root       9   0     0    0     0 SW    0,0  0,0   0:00 bdflush
    7 root       9   0     0    0     0 SW    0,0  0,0   0:01 kupdated
  591 rpc        9   0   172  124   124 S     0,0  0,0   0:00 portmap
  623 root       9   0   288  244   244 S     0,0  0,1   0:00 syslogd
  631 klogd      9   0   804  168   168 S     0,0  0,0   0:00 klogd
```

Figure 1 – Output command top

The columns of the table are indicated in the same way as the output fields of the ***ps*** command (see Section 3.1), so no further explanation is required here.

The contents of the window are updated every 5 seconds. The list of processes can be sorted by the CPU time used (by default), by memory usage, by PID, by run-time. You can switch display modes with the commands that the program top perceives. These are the following commands

- <Shift>+<N> — sort by PID;
- <Shift>+<A> — sort processes by age;
- <Shift>+<P> — sort the processes using the CPU;
- <Shift>+<M> — sorting processes for using memory;
- <Shift>+<T> — sorting by execution time.

In addition to the commands that define the sorting mode, the ***top*** command accepts a number of commands that allow you to control the processes interactively. Using the ***<K>*** command, you can terminate some process (its PID will be requested), and with the ***<R>*** command you can override the nice value for some process. Thus, these two commands are similar to the commands ***kill*** and ***renice*****.**

3.3 Priorities, *nice* and *renice*

We already spoke briefly about what priority is, at the beginning of this chapter, but some facts need to be stated further. Priority for each process is established at the moment when the process is generated. The priority of the

process is determined by the so-called "***nice***" value, which lies in the range from +20 (the lowest priority, the process is performed only when nothing else takes up the processor), to -20 (highest priority).

The nice value is set for each process at the time this process is spawned and, when the commands or programs are normally started, is set to the priority of the parent process. But there is a special ***nice*** command that allows you to change the value of ***nice*** when you start the program. The format of using this program:

nice [- adnice] command [args]

where adnice is the value (-20 to +19) added to the nice value of the parent process.

The received amount will be a nice value for the process being started. A negative value can only be set by the superuser. If the option ***- adnice*** is not specified, the default for the child process is set to ***nice***, increased by 10 compared to the value of the ***nice*** parent process. Obviously, if you are not a superuser, then this command only makes sense if you want to start a process with a low priority.

Another command, ***renice*****,** is used to change the ***nice*** value for already running processes. Its format is as follows:

renice priority [[-p] PID] [[-g] grp] [[-u] user]

For example, the command

[root]# renice -1 987 –u daemon –p 32

increases by 1 the priority of processes with PID 987 and 32, as well as all processes of the user daemon.

The superuser can change the priority of any process in the system. Other users can change the priority value only for those processes for which the user is the owner. In this case, the normal user can only reduce the priority value, but can not increase the priority, even for returning the value of the good to the default value. Therefore, processes with a low priority can not produce "high-priority children".

3.4 Signals and command *kill*

Signals are a means by which processes can send messages about certain events in the system. The processes themselves can also generate signals by which they transmit certain messages to the kernel and other processes. Signals can be used to carry out such actions of process control, such as pausing a process, starting a suspended process, shutting down the process. In total there are 63 different signals in Linux, their list can be viewed by command

[user]$ kill –l

Signals are usually denoted by numbers or symbolic names. All names start on ***SIG***, but this prefix is sometimes omitted: for example, the signal with the number 1 is designated as either ***SIGHUP***, or simply as ***HUP***.

When the process receives a signal, then one of two possible developments is possible. If a processing subprogram is defined for a given signal, this subroutine is called. Otherwise, the kernel executes on behalf of the process the action defined by default for this signal. Calling a processing subroutine is called intercepting a signal. When the execution of the processing subroutine is completed, the process resumes from the point where the signal was received. You can force the process to ignore or block certain signals. The ignored signal is simply discarded by the process and has no effect on it. The blocked signal is queued for delivery, but the kernel does not require any action from the process before the signal is unblocked. After the signal is released, the program of its processing is called only once, even if during the period of blocking this signal was received several times.

In table. 1. some of the frequently occurring signals are listed.

Table 1 – Signals

№	Name	Description	Can intercept	Can block	Hot keys
1	HUP	Hangup. Clear	Yes	Yes	
2	INT	Interrupt. In the case of executing simple commands causes termination of the execution, in interactive programs -termination of the active process	Yes	Yes	<Ctrl>+<C> or <Del>
3	QUIT	As a rule, the signal is stronger than Interrupt	Yes	Yes	<Ctrl>+<\>

4	ILL	Illegal Instruction. The CPU encountered an unfamiliar command (in most cases this means that a program error has been committed). A signal is sent to the program in which the problem occurred.	Yes	Yes	
8	FPE	Floating Point Exception. Computational error For example, division by zero	Yes	Yes	
9	KILL	Always stops the process	No	No	
11	SEGV	Segmentation Violation. Access to an unauthorized area of memory	Yes	Yes	
13	PIPE	An attempt was made to transfer data using a pipeline or a FIFO queue, but there is no process that can take this data	Yes	Yes	
15	TERM	Software Termination.The requirement to finish the process	Yes	Yes	
17	CHLD	Change the status of the child process	Yes	Yes	
18	CON T	Continue the paused process	Yes	Yes	
19	STOP	Suspending the process	No	No	
20	TSTR	Stop signal generated by the keyboard. Translate the Process to the background	Yes	Yes	<Ctrl>+<Z>

As you can see, some signals can be generated using certain key combinations. But such combinations do not exist for all signals. But there is a ***kill*** command, which allows you to send any signal to the specified process. As already mentioned, with this command you can get a list of all possible signals, if you specify the ***-l*** option. If you specify the signal number after this option, its symbolic name will be given, and if you specify a name, we will get the corresponding number.

To send a signal to a process (or a group of processes), you can use the ***kill*** command in the following format:

[user]$ kill [-сигн] PID [PID..]

Where signal is the signal number, and if the signal indication is omitted, a signal 15 (***TERM***) is sent. The most common signal is 9 (***KILL***), with which the super user can complete any process. But this signal is very "rude", so to speak, so its use can lead to a breakdown in the system. Therefore, in most cases it is recommended to use ***TERM*** or ***QUIT*** signals, which terminate the process more "softly".

Naturally, most often the kill command is forced to use the superuser. It must use it to kill zombie processes, hanging processes (they are shown in the *ps* listing as <exiting>), processes that take too much CPU time or too much memory, etc. A special case is the processes launched by the attacker . But the discussion of this special case is beyond the scope of this book.

3.5 Converting the process to the background

If you start a process by running the program from the command line, then the process usually starts, they say, "in the foreground." This means that the process is "tied" to the terminal from which it is running, perceiving input from this terminal and executing its output. But you can start the process in the background when it is not connected to the terminal. To start the process in the background, add the ***&*** symbol at the end of the program's command line.

The ***bash*** shell has two built-in commands, which are used to move processes to the foreground or return them to the background. But before talking about these commands, you need to talk about the team jobs. It is always called with no arguments and shows the tasks that are run from the current shell instance. At the beginning of each line of output of this command, the ordinal number of the task is indicated in the form of a number in square brackets. After the number, the status of the process is indicated: *stopped, running, or suspended*. At the end of the line, you specify the command that is executed by this process. One of the numbers of the running tasks is marked with a **+** sign, and one with the **-** sign. A process marked with a **+** sign will be considered by default the argument of the ***fg*** or ***bg*** commands if they are called without parameters. A process marked with a **-** sign will receive a **+** sign, if for any reason the process that was marked with a **+** sign is completed.

And now you can tell about the ***fg*** and ***bg*** commands, which serve for transferring processes to the foreground or returning them to the background. As an argument to both these commands are transferred the numbers of those jobs that are present in the output of the jobs command. If there are no arguments, then a job marked with a **+** sign is implied. The ***fg*** command takes the process specified in the argument to the foreground, and the ***bg*** command

switches the process to the background. With one ***bg*** command, you can put several processes into the background at once, but you only need to return them to the foreground one at a time.

4.6. Command nohup

Suppose you run several processes from the **b**ash shell, some of them in the background. And for some reason completed the current session of the shell. At the end of the session, the shell sends all the processes it generates a "hang-back" signal, according to which some of the processes generated by it can end, which is not always desirable. If you want to run in the background a program that must be run after you exit the shell, you need to run it using the nohup utility. This is done like this:

nohup command &

The process launched in this way will ignore the signals sent to it (if possible, see Table 1). The standard output stream and the standard error stream for this command are redirected to the file *nohup.out* or ***$HOME/nohup.out.***

The nohup command has the side effect that the ***nice*** value for the process being started is increased by 5, that is, the process is performed with a lower priority.

Task for laboratory work

Scenario 1: System information logging

In this scenario, the commands that provide information about the system are studied.

Initial conditions: Command line after logging in.

1. Determine the name of the current UNIX system using the ***uname -a*** command.
2. Display the contents of the proc directory with the ***ls*** /proc command:
3. Display the current user sessions using the ***who*** command:
4. List all mounted devices using the mount command:
5. Display the load of mounted disks using the ***df -h*** command:
6. Display information about all running processes using the ***ps aux*** command.
7. Leave in the output of the previous command only system processes using the command ***ps aux | grep -v user***:
8. Display the process hierarchy with the command ***pstree***
9. Consider the behavior of processes interactively with the help of the command ***top***.

Scenario 2: Managing processes using signals

In this scenario, we study signals, control keys for transmitting signals to processes, commands for processes control.

Initial conditions: Command line after logging in.

1. Run the ***yes*** command, which produces an infinite output of the ***y*** character on the screen. Interrupt it by pressing Ctrl-C.
2. In this case, the ***SIGTERM*** signal was sent to the running process - the program was terminated.
3. Run the collection of information about all files of the system using the command ***find /> files.txt.***
4. Find the ***id*** of the process that was just started with the command ***ps aux |grep find*** launched in another terminal.
5. Send the end signal to this process with the command ***kill 8178***, specifying the process ID as the parameter.
6. If you try to terminate the system process, for example with the ***kill 1*** command, an error message will appear:
7. Only superuser can send signals to system processes.
8. An alternative way to send signals to processes - by the name of the process, rather than by ***PID*** - is the ***killall*** command. After executing the

command ***killall bash***, we will terminate all the command shells, and thus the user sessions.

11. Prior to that, we only sent a signal to complete the process. It can be intercepted and ignored by programs. A non-interceptable signal is the ***SIGKILL*** signal, which can be sent, for example, with the following command:

killall -SIGKILL find

Scenario 3: Running tasks in the background

In this scenario, work is being done with the tasks of the command shell, running tasks in the background.

Initial conditions: Command line after logging in.

1. Run a long command, for example ***find /> files.txt*** and suspend its execution by pressing ***Ctrl-Z***. In this process, the signal ***SIGSTOP*** is sent.

2. The command is suspended and stored as task 1 (number in square brackets).

3. The current list of running tasks in the command shell can be viewed with the ***jobs*** command.

4. Resume the execution of the task with the command ***fg 1***, the argument of which is the job number.

5. If you pause the process again, you can start the task in the background: ***bg 1***.

6. Then you can continue to work on the command line.

7. Commands can be started immediately in the background. It is necessary to add the symbol "***&***" (ampersand) to the end of the command line:

find / -name "* .xml"> xml-list &

10. In this case, the command shell displays the job number and PID of the created process.

Scenario 4: Running Daemons

In this scenario, daemons are considered as processes that are not associated with any terminal.

Initial conditions: Command line after logging in.

1. If you run the command in the background and exit the command shell: for example, ***find / -name "*.html" -exec grep -Hn "linux loader" \{} \; &*** and ***exit***, then the running command will end with the ***SIGHUP*** signal.

2. To ensure that programs do not receive a ***SIGHUP*** signal, a special command ***nohup*** is used:

nohup find / -name "*.txt" -exec grep -Hn "linux loader" \{} \; &

3.This program will exit correctly after the search is completed.

Scenario 5: Changing the priorities of running programs

This scenario examines the UNIX priority mechanism and commands for changing the priorities of the processes being started.

Initial conditions: Command line after logging in.

1. Each process in the system has its own priority level (in UNIX it is called "**nice**"), which can be seen with the ***ps -l*** command in the "**NI**" column.

2. By default, the priority of simple user processes is zero.

3. To start the process with a given priority, use the ***nice*** command. For example, let's start creating an archive with a lower priority:
nice -n 10 tar -cjf libraries.tar.bz2 /usr/lib/.

4. To change the priority of an already running program, use the command ***renice***. For example, to lower the priority of a previously running ***backup process tar -cjf libraries.tar.bz2/usr/lib &*** use the command
renice +10 -p 3442:

5. Common users can not increase the priority of processes, only to lower.