

Цель работы: Практическое знакомство с командами, используемыми для контроля использования ресурсов и виртуальной файловой системой.

Ход работы:

1. Ознакомиться с теоретическим материалом.
2. Выполнить задания.
3. Ответить на контрольные вопросы.

Задание:

1. Вывести список всех процессов в системе.
2. Вывести дерево процессов.
3. С помощью команды `top` получить список процессов, потребляющих наибольшее количество процессорного времени.
4. Найти 2 процесса, имеющих более ДВУХ потоков.
5. Используя команду `top`, изменить приоритеты 2 процессов.
6. Получить список открытых файлов текущего пользователя.
7. Получить текущее состояние системной памяти.
8. Получить справку об использовании дискового пространства.
9. Вывести информацию о каком-либо процессе, используя содержимое каталога `/proc`.
10. Вывести информацию о процессоре ПК, используя содержимое каталога `/proc`.
11. Вывести список модулей, используемых в настоящий момент ядром ОС.

Описание выполнения работы

1. Вывести список всех процессов в системе.

Для вывода списка всех выполняющихся на компьютере в текущий момент процессах используется команда: `ps aux`

Значения используемых опций: `a` - `all` – процессы всех пользователей; `u` – ориентированная на пользователей (отображение информации о владельце); `x` – процессы, не контролируемые `ttys`.

Результат выполнения данной команды, представлен на рисунке 5.1.

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

student@student-VirtualBox:~$ ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.5  0.5 167480 11644 ?        Ss   15:02   0:03 /sbin/init s
root           2  0.0  0.0      0      0 ?        S    15:02   0:00 [kthreadd]
root           3  0.0  0.0      0      0 ?        I<   15:02   0:00 [rcu_gp]
root           4  0.0  0.0      0      0 ?        I<   15:02   0:00 [rcu_par_gp]
root           6  0.0  0.0      0      0 ?        I<   15:02   0:00 [kworker/0:0
root           8  0.0  0.0      0      0 ?        I<   15:02   0:00 [mm_percpu_w
root           9  0.0  0.0      0      0 ?        S    15:02   0:00 [ksoftirqd/0
root          10  0.0  0.0      0      0 ?        I    15:02   0:00 [rcu_sched]
root          11  0.0  0.0      0      0 ?        S    15:02   0:00 [migration/0
root          12  0.0  0.0      0      0 ?        S    15:02   0:00 [idle_inject
root          14  0.0  0.0      0      0 ?        S    15:02   0:00 [cpuhp/0]
root          15  0.0  0.0      0      0 ?        S    15:02   0:00 [kdevtmpfs]
root          16  0.0  0.0      0      0 ?        I<   15:02   0:00 [netns]
root          17  0.0  0.0      0      0 ?        S    15:02   0:00 [rcu_tasks_k
root          18  0.0  0.0      0      0 ?        S    15:02   0:00 [kauditd]
root          19  0.0  0.0      0      0 ?        S    15:02   0:00 [khungtaskd]
root          20  0.0  0.0      0      0 ?        S    15:02   0:00 [oom_reaper]
root          21  0.0  0.0      0      0 ?        I<   15:02   0:00 [writeback]
root          22  0.0  0.0      0      0 ?        S    15:02   0:00 [kcompactd0]
root          23  0.0  0.0      0      0 ?        SN   15:02   0:00 [ksmd]
root          24  0.0  0.0      0      0 ?        SN   15:02   0:00 [khugepaged]
root         116  0.0  0.0      0      0 ?        I<   15:02   0:00 [kintegrityd
root         117  0.0  0.0      0      0 ?        I<   15:02   0:00 [kblockd]
root         118  0.0  0.0      0      0 ?        I<   15:02   0:00 [blkcg_punt_
```

Рисунок 5.1 – Фрагмент вывода списка процессов системы

2. Вывести дерево процессов.

Для построения дерева процессов используются команды (результат работы каждой из них идентичен):

`ps tree`, `ps eJH`, `ps axjt`

Результат выполнения первой из приведенных выше команд, представлен на рисунке 5.2.



Рисунок 5.2 – Фрагмент вывода списка процессов системы в виде дерева

3. С помощью команды `top` получить список 5 процессов, потребляющих наибольшее количество процессорного времени.

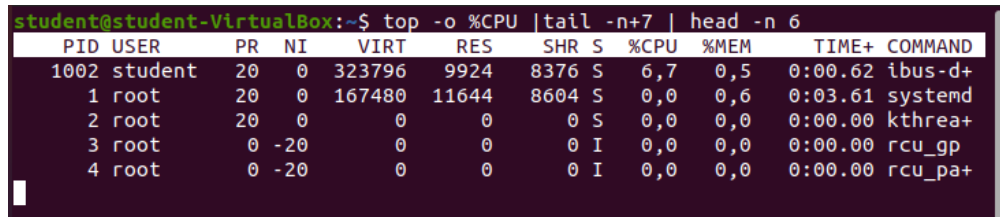
Для того чтобы вывести список процессов, отсортированных по потреблению наибольшего количество процессорного времени, необходимо воспользоваться командой: `top -o %CPU`.

Однако, данная команда выведет полный список всех процессов. Для того чтобы выбрать только первые 5 процессов, необходимо воспользоваться командами `tail` и `head`, которые отсекают от выходного результата команды строки с конца и с начала соответственно. В качестве параметра, определяющего количество отсекаемых строк, используется параметр `n`.

Таким образом, получаем команду:

```
top -o %CPU | tail -n +7 | head -n 6
```

Результат выполнения данной команды, представлен на рисунке 5.3.



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1002	student	20	0	323796	9924	8376	S	6,7	0,5	0:00.62	ibus-d+
1	root	20	0	167480	11644	8604	S	0,0	0,6	0:03.61	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.00	kthrea+
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_pa+

Рисунок 5.3 – Список 5 процессов, потребляющих наибольшее количество процессорного времени

4. Найти 2 процесса, имеющих более ДВУХ потоков.

Для того чтобы выбрать все процессы воспользуемся ключом «-e», который выводим список всех процессов. Получится команда:

```
ps -e
```

Для того чтобы получить необходимые нам колонки воспользуемся ключом «-o», параметры которого – необходимые для вывода колонки: pid – id процесса, cmd – команда запуска процессора, nlwp (Number Light-Weight Process) – число легковесных процессов в операционной системе, так как этот параметр позволяет пользователю использовать собственный формат ввода:

```
ps -e -o pid,cmd,nlwp
```

Отсортируем полученный результат по убыванию количества потоков

```
ps -e -o pid,cmd,nlwp --sort=-nlwp
```

Для того, чтобы получить только первые 2 процесса воспользуемся командой head -n 3. В конечном итоге получим команду:

```
ps -e -o pid,cmd,nlwp --sort=-nlwp | head -n 3
```

Результат выполнения данной команды, представлен на рисунке 5.4.

```
student@student-VirtualBox:~$ ps -e -o pid,cmd,nlwp --sort=-nlwp | head -n 3
  PID CMD                                NLWP
  2003 /usr/lib/firefox/firefox -n      51
  2081 /usr/lib/firefox/firefox -c      19
student@student-VirtualBox:~$
```

Рисунок 5.4 – Два процесса, имеющих более ДВУХ потоков

5. Используя команду `top`, изменить приоритеты 2 процессов. Изменим приоритеты двух процессов с PID 22452 и PID 2324.

Для просмотра информации об этих двух процессах воспользуемся командой `top: top -p22452,2324`

Результат выполнения данной команды, представлен на рисунке 5.5.

Как видим, значение приоритета обоих процессов равно 20. Повысим приоритет его у обоих процессов на 1. Это можно сделать интерактивно из утилиты `top`. Для этого запускаем утилиту `top` и нажимаем клавишу `<R>`. Нам будет предложено ввести уровень приоритета и PID процесса, аналогично вводу в режиме команд (рисунки 5.6 и 5.7).

```
top - 09:26:07 up 4:15, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 2 total, 0 running, 2 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.0 us, 0.8 sy, 0.0 ni, 98.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2018084 total, 74500 free, 941756 used, 1001828 buff/cache
KiB Swap: 2094076 total, 2089712 free, 4364 used. 836532 avail Mem
Renice PID 22452 to value -1
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2245	student	20	0	690184	48536	34560	S	1.3	2.4	0:28.11	gnome-sys+
2324	student	20	0	1629452	217140	66868	S	0.3	10.8	1:41.10	compiz

Рисунок 5.5– Информация о процессах

```
top - 09:26:07 up 4:15, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 2 total, 0 running, 2 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.0 us, 0.8 sy, 0.0 ni, 98.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2018084 total, 74500 free, 941756 used, 1001828 buff/cache
KiB Swap: 2094076 total, 2089712 free, 4364 used. 836532 avail Mem
Renice PID 22452 to value -1
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
22452	student	20	0	690184	48536	34560	S	1.3	2.4	0:28.11	gnome-sys+
2324	student	20	0	1629452	217140	66868	S	0.3	10.8	1:41.10	compiz

Рисунок 5.6 – Изменение приоритета процесса с PID 22452

```
top - 09:26:37 up 4:16, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 2 total, 0 running, 2 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.8 us, 0.7 sy, 0.0 ni, 98.5 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2018084 total, 74360 free, 941872 used, 1001852 buff/cache
KiB Swap: 2094076 total, 2089712 free, 4364 used. 836408 avail Mem
Renice PID 2324 to value -1
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
22452	student	19	-1	690184	48536	34560	S	1.7	2.4	0:28.62	gnome-sys+
2324	student	20	0	1629444	217140	66868	S	0.3	10.8	1:41.30	compiz

Рисунок 5.7– Изменение приоритета процесса с PID 2324

Теперь мы можем видеть, что приоритет обоих процессов изменен и равен значению 19 (рисунок 5.8.).

```
top - 09:26:59 up 4:16, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 2 total, 0 running, 2 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.3 us, 0.7 sy, 0.0 ni, 98.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2018084 total, 73740 free, 942492 used, 1001852 buff/cache
KiB Swap: 2094076 total, 2089712 free, 4364 used. 835788 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
22452	student	19	-1	690184	48536	34560	S	1.7	2.4	0:29.10	gnome-sys+
2324	student	19	-1	1630016	217712	66868	S	1.0	10.8	1:41.64	compiz

Рисунок 5.8 – Информация о процессах

6. Получить список открытых файлов текущего пользователя

Команда `lsuf` (ListOpenFiles) без параметров выводит полный список открытых файлов. При этом пользователь-администратор получит несколько тысяч строк текста.

Для получения списка файлов, открытых конкретным пользователем, служит команда: `lsuf -u имя_пользователя`

Таким образом, в нашем случае (имя пользователя: `student`) необходимо выполнить команду:

`lsuf -u student`

Результат выполнения данной команды, представлен на рисунке 5.9.


```

ipe
lsof      2809 student    6r    FIFO          0,13      0t0      48635 p
ipe
lsof      2810 student   cwd     DIR           8,5        4096      295337 /
home/student
lsof      2810 student   rtd     DIR           8,5        4096        2 /
lsof      2810 student   txt     REG           8,5      175744      787140 /
usr/bin/lsof
lsof      2810 student   mem     REG           8,5       51832      797987 /
usr/lib/x86_64-linux-gnu/libnss_files-2.31.so
lsof      2810 student   mem     REG           8,5    14537584      794276 /
usr/lib/locale/locale-archive
lsof      2810 student   mem     REG           8,5     157224      798128 /
usr/lib/x86_64-linux-gnu/libpthread-2.31.so
lsof      2810 student   mem     REG           8,5       18816      797360 /
usr/lib/x86_64-linux-gnu/libdl-2.31.so
lsof      2810 student   mem     REG           8,5     584392      798071 /
usr/lib/x86_64-linux-gnu/libpcres2-8.so.0.9.0
lsof      2810 student   mem     REG           8,5     2029224      797225 /
usr/lib/x86_64-linux-gnu/libc-2.31.so
lsof      2810 student   mem     REG           8,5     163200      798225 /
usr/lib/x86_64-linux-gnu/libselinux.so.1
lsof      2810 student   mem     REG           8,5     191472      797012 /
usr/lib/x86_64-linux-gnu/ld-2.31.so
lsof      2810 student    4r    FIFO          0,13      0t0      48634 p
ipe
lsof      2810 student    7w    FIFO          0,13      0t0      48635 p
ipe
student@student-VirtualBox:~$

```

Рисунок 5.9– Фрагмент списка открытых файлов пользователя

7. Получить текущее состояние системной памяти

Текущее состояние системной памяти позволяет получить команда `free`.

По умолчанию все выводимые значения представлены в килобайтах. Значения в мегабайтах позволяет получить опция `-m`.

Таким образом, получаем команду: `free -m`

Результат выполнения данной команды, представлен на рисунке 5.10.

```

student@student-VirtualBox:~$ free -m
              всего          занято          свободно          общая   буф./врем.   дост
упно
Память:      1987          1085           119           27           782           732
Подкачка:      687             0           686
student@student-VirtualBox:~$

```

Рисунок 5.10– Текущее состояние системной памяти

8. Получить справку об использовании дискового пространства.

Команда `df` выводит данные об объеме доступного дискового пространства (в килобайтах). Опция `-h` улучшает восприятие результатов (данные выводятся в табличной форме).

Команда `du` дает возможность узнать объем дисковой памяти, занимаемой каталогами и файлами.

Таким образом, получаем команду: `df -h`

Результат выполнения данной команды, представлен на рисунке 5.11.

```
student@student-VirtualBox:~$ df -h
Файл.система  Размер  Использовано  Дост  Использовано%  Смонтировано в
udev          967M    0             967M    0%             /dev
tmpfs         199M    1,3M         198M    1%             /run
/dev/sda5     15G     6,3G         7,3G    47%            /
tmpfs         994M    11M          984M    2%             /dev/shm
tmpfs         5,0M    4,0K         5,0M    1%             /run/lock
tmpfs         994M    0            994M    0%             /sys/fs/cgroup
/dev/loop0    55M     55M          0       100%            /snap/core18/1705
/dev/loop2    241M    241M         0       100%            /snap/gnome-3-34-1804/24
/dev/loop4    28M     28M          0       100%            /snap/snapd/7264
/dev/loop3    50M     50M          0       100%            /snap/snap-store/433
/dev/loop1    63M     63M          0       100%            /snap/gtk-common-themes/
1506
/dev/sda1     511M    4,0K         511M    1%             /boot/efi
tmpfs         199M    36K          199M    1%             /run/user/1000
student@student-VirtualBox:~$
```

Рисунок 5.11– Справка об использовании дискового пространства

9. Вывести информацию о каком-либо процессе, используя содержимое каталога `/proc`

В каталоге `/proc` содержится информация о каждом процессе, приведем информацию об основных файлах в этом каталоге.

В качестве процесса возьмем процесс с PID 1.

`cmdline`: этот (псевдо-) файл содержит полную командную строку, использованную для вызова процесса.

Результат выполнения данной команды `cat /proc/1/cmdline`, представлен на рисунке 5.12.

```
student@student-VirtualBox:~$ cat /proc/1/cmdline
/sbin/initstudent@student-VirtualBox:~$
```

Рисунок 5.12– Информация о процессе с PID 1, используя содержимое каталога `/proc`

`cwd`: эта символическая ссылка указывает на текущий рабочий каталог процесса (как следует из её имени).

Получим адрес ссылки при помощи команды: `readlink -f /proc/1/cwd`
Результат выполнения данной команды, представлен на рисунке 5.13.

```
student@student-VirtualBox:~$ readlink -f /proc/1/cwd
student@student-VirtualBox:~$
```

Рисунок 5.13– Текущий рабочий каталог процесса с PID 1

`environ`: этот файл содержит все переменные окружения, определенные для этого процесса, в виде ПЕРЕМЕННАЯ = значение. Как и в `cmdline` вывод вообще не отформатирован: нет разделителей строк для отделения различных переменных, и в конце нет разделителя строки.
`cat /proc/1/environ`

Результат выполнения данной команды, представлен на рисунке 5.14.

```
student@student-VirtualBox:~$ cat /proc/1/environ
cat: /proc/1/environ: Отказано в доступе
student@student-VirtualBox:~$ sudo cat /proc/1/environ
[sudo] пароль для student:
HOME=/init=/sbin/initNETWORK_SKIP_ENSLAVED=TERM=linuxBOOT_IMAGE=/boot/vmlinuz-5
.4.0-33-genericdrop_caps=PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
:/sbin:/binPWD=/rootmnt=/rootstudent@student-VirtualBox:~$
```

Рисунок 5.14 – Все переменные окружения, определенные для процесса с PID 1

`exe`: эта символическая ссылка указывает на исполняемый файл, соответствующий запущенному процессу.

Получим адрес ссылки при помощи команды: `readlink -f /proc/1/exe`

Результат выполнения данной команды, представлен на рисунке 5.15.

```
student@student-VirtualBox:~$ readlink -f /proc/1/exe
student@student-VirtualBox:~$
```

Рисунок 5.15 – Исполняемый файл, соответствующий запущенному процессу с PID 1

`fd`: этот подкаталог содержит список файловых дескрипторов, открытых в данный момент процессом.

Результат выполнения команды `ls /proc/1/fd`, представлен на рисунке 5.16.

```
student@student-VirtualBox:~$ ls /proc/1/fd
ls: невозможно открыть каталог '/proc/1/fd': Отказано в доступе
student@student-VirtualBox:~$ sudo ls /proc/1/fd
0    107  117  126  135  144  159  21  30  4   49  58  67  77  86  95
1    108  118  127  136  145  16   22  31  40  5   59  68  78  87  96
10   109  119  128  137  146  160  23  32  41  50  6   69  79  88  97
100  11   12   129  138  147  161  24  33  42  51  60  7   8   89  98
101  110  120  13   139  15   162  25  34  43  52  61  70  80  9   99
102  111  121  130  14   152  17   26  35  44  53  62  71  81  90
103  112  122  131  140  153  18   27  36  45  54  63  72  82  91
104  113  123  132  141  154  19   28  37  46  55  64  73  83  92
105  114  124  133  142  155  2    29  38  47  56  65  74  84  93
106  115  125  134  143  156  20   3   39  48  57  66  75  85  94
student@student-VirtualBox:~$
```

Рисунок 5.16– Список файловых дескрипторов, открытых в данный момент процессом с PID 1

maps: когда вы выводите содержимое этого именованного канала (при помощи команды `cat`, например), вы можете увидеть части адресного пространства процесса, которые в текущий момент распределены для файла. Вот эти поля (слева направо): адресное пространство, связанное с этим распределением; разрешения, связанные с этим распределением; смещение от начала файла, где начинается распределение; старший и младший номера (в шестнадцатеричном виде) устройства, на котором находится распределенный файл; номер inode файла; и, наконец, имя самого файла. Результат выполнения данной команды `cat /proc/1/maps`, представлен на рисунке 5.17.

```

student@student-VirtualBox:~$ cat /proc/1/maps
cat: /proc/1/maps: Отказано в доступе
student@student-VirtualBox:~$ sudo cat /proc/1/maps
56130efc9000-56130effb000 r--p 00000000 08:05 796641      /usr/l
ib/systemd/systemd
56130effb000-56130f0b4000 r-xp 00032000 08:05 796641      /usr/l
ib/systemd/systemd
56130f0b4000-56130f10a000 r--p 000eb000 08:05 796641      /usr/l
ib/systemd/systemd
56130f10a000-56130f150000 r--p 00140000 08:05 796641      /usr/l
ib/systemd/systemd
56130f150000-56130f151000 rw-p 00186000 08:05 796641      /usr/l
ib/systemd/systemd
5613100dd000-5613102f0000 rw-p 00000000 00:00 0          [heap]
7f0128000000-7f0128021000 rw-p 00000000 00:00 0
7f0128021000-7f012c000000 ---p 00000000 00:00 0
7f0130000000-7f0130021000 rw-p 00000000 00:00 0
7f0130021000-7f0134000000 ---p 00000000 00:00 0
7f013444e000-7f013444f000 ---p 00000000 00:00 0
7f013444f000-7f0134c4f000 rw-p 00000000 00:00 0
7f0134c4f000-7f0134c50000 ---p 00000000 00:00 0
7f0134c50000-7f0135450000 rw-p 00000000 00:00 0
7f0135450000-7f013545f000 r--p 00000000 08:05 797863      /usr/l
ib/x86_64-linux-gnu/libm-2.31.so
7f013545f000-7f0135506000 r-xp 0000f000 08:05 797863      /usr/l
ib/x86_64-linux-gnu/libm-2.31.so
7f0135506000-7f013559d000 r--p 000b6000 08:05 797863      /usr/l
ib/x86_64-linux-gnu/libm-2.31.so
7f013559d000-7f013559e000 r--p 0014c000 08:05 797863      /usr/l

```

Рисунок 5.17– Фрагмент адресного пространства процесса с PID 1

root: эта символическая ссылка указывает на корневой каталог, используемый процессом. Обычно это будет `readlink -f /proc/1/root`

Результат выполнения данной команды, представлен на рисунке 5.18.

```

student@student-VirtualBox:~$ readlink -f /proc/1/root
student@student-VirtualBox:~$ █

```

Рисунок 5.18– Корневой каталог, используемый процессом с PID 1

status: этот файл содержит разнообразную информацию о процессе: имя исполняемого файла, его текущее состояние, его PID и PPID, его реальные и эффективные UID и GID, его использование памяти и другие данные.

Результат выполнения команды `cat /proc/1/status`, представлен на рисунке 5.19.

```
student@student-VirtualBox:~$ cat /proc/1/status
Name:      systemd
Umask:     0000
State:     S (sleeping)
Tgid:      1
Ngid:      0
Pid:       1
PPid:      0
TracerPid: 0
Uid:       0      0      0      0
Gid:       0      0      0      0
FDSize:    256
Groups:
NSTgid:    1
NSpid:     1
NSpgid:    1
NSSid:     1
VmPeak:    233016 kB
VmSize:    167480 kB
VmLck:      0 kB
VmPin:      0 kB
VmHWM:     11644 kB
VmRSS:     11640 kB
RssAnon:           3036 kB
RssFile:           8604 kB
RssShmem:           0 kB
VmData:    19168 kB
VmStk:      132 kB
VmExe:      940 kB
```

Рисунок 5.19—Фрагмент информации о процессе с PID 1

10. Вывести информацию о процессоре ПК, используя содержимое каталога /proc

/proc/cpuinfo: этот файл содержит, как видно из его имени, информацию о процессорах машины: `cat /proc/cpuinfo`

Результат выполнения данной команды, представлен на рисунке 5.20.

```

student@student-VirtualBox:~$ cat /proc/cpuinfo
processor       : 0
vendor_id      : AuthenticAMD
cpu family     : 21
model          : 2
model name     : AMD FX-4330 Quad-Core Processor
stepping       : 0
microcode      : 0x60000626
cpu MHz        : 4018.036
cache size     : 2048 KB
physical id    : 0
siblings       : 1
core id        : 0
cpu cores      : 1
apicid         : 0
initial apicid : 0
fpu            : yes
fpu_exception  : yes
cpuid level    : 13
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
                 pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt rdtscp lm co
                 nstant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid tsc_known_freq pni pclmu
                 lqdq monitor ssse3 cx16 sse4_1 sse4_2 x2apic popcnt aes xsave avx hypervisor la
                 hf_lm cr8_legacy abm sse4a misalignsse 3dnowprefetch ssbd vmmcall arat
bugs           : fxsavleak sysret_ss_attrs null_seg spectre_v1 spectre_v2 sp
                 ec_store_bypass
bogomips       : 8036.07
TLB size       : 1536 4K pages

```

Рисунок 5.20– Фрагмент информации о процессоре ПК

Вывести список модулей, используемых в настоящий момент ядром /proc/modules: этот файл содержит список модулей, используемых ядром в настоящий момент, вместе со счетчиком использования каждого из модулей. Эта информация используется командой `lsmod`, которая отображает её в более удобной для чтения форме. Для вывода воспользуемся командой `cat`.

Результат выполнения команды `cat/proc/modules`, представлен на рисунке 5.21.

```

student@student-VirtualBox:~$ cat /proc/modules
nls_iso8859_1 16384 1 - Live 0x0000000000000000
snd_intel8x0 45056 2 - Live 0x0000000000000000
snd_ac97_codec 131072 1 snd_intel8x0, Live 0x0000000000000000
edac_mce_amd 32768 0 - Live 0x0000000000000000
ac97_bus 16384 1 snd_ac97_codec, Live 0x0000000000000000
snd_pcm 106496 2 snd_intel8x0,snd_ac97_codec, Live 0x0000000000000000
crct10dif_pclmul 16384 1 - Live 0x0000000000000000
snd_seq_midi 20480 0 - Live 0x0000000000000000
ghash_clmulni_intel 16384 0 - Live 0x0000000000000000
snd_seq_midi_event 16384 1 snd_seq_midi, Live 0x0000000000000000
snd_rawmidi 36864 1 snd_seq_midi, Live 0x0000000000000000
joydev 24576 0 - Live 0x0000000000000000
aesni_intel 372736 0 - Live 0x0000000000000000
snd_seq 69632 2 snd_seq_midi,snd_seq_midi_event, Live 0x0000000000000000
crypto_simd 16384 1 aesni_intel, Live 0x0000000000000000
cryptd 24576 2 ghash_clmulni_intel,crypto_simd, Live 0x0000000000000000
glue_helper 16384 1 aesni_intel, Live 0x0000000000000000
snd_seq_device 16384 3 snd_seq_midi,snd_rawmidi,snd_seq, Live 0x0000000000000000
0
snd_timer 36864 2 snd_pcm,snd_seq, Live 0x0000000000000000
snd 90112 11 snd_intel8x0,snd_ac97_codec,snd_pcm,snd_rawmidi,snd_seq,snd_seq_de
vice,snd_timer, Live 0x0000000000000000
soundcore 16384 1 snd, Live 0x0000000000000000
input_leds 16384 0 - Live 0x0000000000000000
vboxguest 348160 0 - Live 0x0000000000000000 (0)
mac_hid 16384 0 - Live 0x0000000000000000
serio_raw 20480 0 - Live 0x0000000000000000
sch_fq_codel 20480 2 - Live 0x0000000000000000

```

Рисунок 5.21 – Фрагмент списка модулей, используемых в настоящий момент ядром ОС

Контрольные вопросы

1. Команды вывода списка процессов.

Для вывода списка всех выполняющихся на компьютере в текущий момент процессах используется команда: `ps aux`

Значения используемых опций: `a` - `all` – процессы всех пользователей; `u`

Ориентированная на пользователей (отображение информации о владельце); `x` – процессы, не контролируемые `ttys`.

Полезные ключи: `-e` – вывод сведений обо всех процессах и `o` – пользовательский вывод.

2. Команда получения списка потоков.

Для получения информации о потоках заданного процесса используется опция `-L`, на пример `ps -fLC swriter.bin` выводит список потоков приложения `writer Open Office`.

3. Команда для завершения приложений.

Завершение процесса выполняется командой `kill` сигнал `PID`.

Сначала процессу посылается сигнал `-15`. Если это не помогает, `129`, используется крайняя мера - посылается сигнал `-9`.

4. Состояния процесса Linux.

Каждый из каталогов содержит одинаковые пункты, краткое описание некоторых из них:

`cmdline`: этот (псевдо-) файл содержит полную командную строку, использованную для вызова процесса `134` программой и ее аргументами нет пробелов, а в конце строки нет разделителя строки. Чтобы просмотреть его, вы можете использовать: `perl -ple 's,\00, ,g' cmdline`.

`cwd`: эта символическая ссылка указывает на текущий рабочий каталог процесса (следует из имени). `environ`: этот файл содержит все переменные окружения, определенные для этого процесса, в виде ПЕРЕМЕННАЯ=значение. Как и в `cmdline` вывод вообще не отформатирован: нет разделителей строк для отделения различных

переменных, и в конце нет разделителя строки. Единственным решением для его просмотра будет: `perl -pl -e 's,\00,\n,g' environ exe`: эта символическая ссылка указывает на исполняемый файл, соответствующий запущенному процессу.

`fd`: этот подкаталог содержит список файловых дескрипторов, открытых в данный момент процессом.

`maps`: когда вы выводите содержимое этого именованного канала (при помощи команды `cat`, например), вы можете увидеть части адресного пространства процесса, которые в текущий момент распределены для файла. Вот эти поля (слева направо): адресное пространство, связанное с этим распределением; разрешения, связанные с этим распределением; смещение от начала файла, где начинается распределение; старший и младший номера (в шестнадцатиричном виде) устройства, на котором находится распределенный файл; номер `inode` файла; и, наконец, имя самого файла.

`root`: эта символическая ссылка указывает на корневой каталог, используемый процессом. Обычно это будет `/`.

`status`: этот файл содержит разнообразную информацию о процессе: имя исполняемого файла, его текущее состояние, его `PID` и `PPID`, его реальные и эффективные `UID` и `GID`, его использование памяти и другие данные.

5. Получение информации о потоках процесса.

Как известно, процесс может иметь параллельно выполняющиеся потоки (`threads`) или облегченные процессы (`LWP`, `Light Weight Process`). Для получения информации о потоках заданного процесса используется опция `-L`, например `ps -fLC swriter.bin` выводит список потоков приложения `writer Open Office`. Процессы, использующие более одного потока – редактор звуковых файлов `audacity` и `soffice.bin`, а также демоны (службы в терминологии `Windows`). Как указано выше, многопоточные процессы помечено символом `l` в колонке состояния.

6. Примеры многопоточных процессов.

Программные потоки являются базовым элементом многозадачного программного окружения. Программный поток может быть описан как среда выполнения процесса; поэтому каждый процесс имеет как минимум один программный поток. Многопоточность предполагает наличие нескольких параллельно работающих (на многопроцессорных системах) и обычно синхронизируемых сред выполнения процесса.

Программные потоки имеют свои идентификаторы (thread ID) и могут выполняться независимо друг от друга. Они делят между собой одно адресное пространство процесса и используют эту особенность в качестве преимущества, позволяющего не использовать каналы IPC (систем меж процессного взаимодействия - разделяемой памяти, каналов и других систем) для обмена данными. Программные потоки процесса могут взаимодействовать-например, независимые потоки могут получать/изменять значение глобальной переменной. Эта модель взаимодействия исключает лишние затраты ресурсов на вызовы IPC на уровне ядра. Поскольку потоки работают в едином адресном пространстве, переключения контекста для потока быстры и не ресурсоемки.

Многопоточные процессы помечено символом l в колонке состояния ps.

Примером может служить браузер Firefox.

7. Необходимость использования потоков.

Выделим основные преимущества использования потоков:

Потоки удобно использовать при необходимости выполнения в процессе нескольких действий сразу (пример: одновременная обработка на сервере запросов нескольких пользователей)

Ускорение работы приложений, использующих ввод, обработку и вывод данных за счет возможности распределения этих операций по отдельным потокам. Это дает возможность не прекращать выполнение программы во время возможных простоев из-за ожидания при чтении/записи данных

Как правило, переключение между потоками происходит быстрее и требует

меньших затрат системных ресурсов, по сравнению с переключением между процессами

8. Процессы-зомби: как они появляются, как их найти и что с ними делать?

Потоки удобно использовать при необходимости выполнения в процессе нескольких действий сразу (пример: одновременная обработка на сервере запросов нескольких пользователей).

Как правило, переключение между потоками происходит быстрее и требует меньших затрат системных ресурсов, по сравнению с переключением между процессами.

9. Содержимое вывода команды `top`.

Команда `top` представляет динамически обновляемые сведения о процессах и о том, какой объем системных ресурсов использует каждый из них.

В первых пяти строках команда `top` отображает подробную информацию о системе, а затем описывает каждый выполняющийся процесс. Выходные данные сортируются по уровню загрузки ЦП данным процессом.

В первой строке программа сообщает текущее время, время работы системы (1 час 15 мин), количество зарегистрированных (`login`) пользователей (3 users), общая средняя загрузка системы (`load average`). Общей средней загрузкой системы называется среднее число процессов, находящихся в состоянии выполнения (R) или в состоянии ожидания (D). Общая средняя загрузка измеряется каждые 1, 5 и 15 минут.

Во второй строке вывода программы `top` сообщается, что в списке процессов находятся 132 процесса, из них 131 спит (состояние готовности или ожидания), 1 выполняется (на виртуальной машине только 1 процессор), 0 процессов зомби и 0 остановленных процессов.

В третьей-пятой строках приводится информация о загрузке процессора CPU в режиме пользователя и системном режиме, использования памяти и файла подкачки.

В таблице отображается различная информация о процессе. Рассмотрим

колонки PID (идентификатор процесса), USER (пользователь, запустивший процесс), S (состояние процесса) и COMMAND (команда, которая была введена для запуска процесса).

Колонка S может содержать следующие значения:

R – процесс выполняется или готов к выполнению (состояние готовности); помечено символом l в колонке состояния ps. Примером может служить браузер Firefox.

10. Как получить информацию о процессах системы, используя файловую систему /proc?

Каталог /proc содержит всю информацию о всех процессах в системе, для доступа к информации о процессе используется следующая команда:

`/proc/[pid]/stat`

Где на месте stat может быть любой другой тип возвращаемой информации.

11. Команды для получения информации об открытых файлах.

Команда lsof (List open files) без параметров выводит полный список открытых файлов. Пользователь-администратор получит несколько тысяч строк текста.

Для получения списка файлов, открытых конкретным пользователем, служит команда lsof -u имя_пользователя

12. Получение информации о состоянии системной памяти.

Текущее состояние системной памяти позволяет получить команда

По умолчанию все значения представлены в килобайтах. Значения в М позволяет получить опция -m.

13. Получение информации об использовании дискового пространства.

Команда df выводит данные об объеме доступного дискового пространства (в Кбайтах). Опция -h улучшает восприятие результатов.

Команда du дает возможность узнать объем дисковой памяти, занимаемой каталогами и файлами.

14. Назначение файловой системы /proc.

Файловая система /proc является механизмом для ядра и его модулей, позволяющим посылать информацию процессам (отсюда и название /proc). С помощью этой виртуальной файловой системы можно работать с внутренними структурами ядра, получать полезную информацию о процессах и изменять установки (меняя параметры ядра) на лету. Файловая система /proc располагается в памяти в отличие от других файловых систем, которые располагаются на диске.