

Міністерство освіти і науки України  
НТУУ «КПІ ім. Ігоря Сікорського»  
Навчально-науковий інститут атомної та теплової енергетики  
Кафедра цифрових технологій в енергетиці

Лабораторна робота №6  
з дисципліни «Операційна система UNIX»  
Тема «Робота з процесами ОС Linux»  
Варіант №22

Студента 2-го курсу НН ІАТЕ гр. ТР-12

Ковальова Олександра

Перевірила: д.т.н., проф. Левченко Л. О.

**Мета роботи.** Набути навичок управління процесами в оболонці Bash, опанувати команди ps, top, pstree, bg, fg, nice, renice, kill, kill-all.

**Теоретична частина.** Програма – це файл, який містить виконуваний код, сегменти з даними для ініціалізації та з даними користувача. Процес представляє собою програму у стані виконання. Це абстракція, за допомогою якої можна керувати пам'яттю, часом роботи процесора та ресурсами введення-виведення.

Це аксіома філософії UNIX, яка дозволяє як можна більше працювати в контексті процесів, а не ядра. Системні і користувацькі процеси відповідають одним і тим же правилам, тобто мають спільне API (Applied Program Interface), тому можна використовувати один і той самий набір інструментів для керування ними.

Процес складається з адресного простору і набору структур даних всередині ядра. Адресний простір представляє собою набір сторінок пам'яті, які ядро виділило для використання процесу. Сторінки – це базові блоки пам'яті, розмір яких – від 1 до 8 Кбайт. Ці сторінки зберігають код та бібліотеки, які використовуються програмою, змінні процесу, його стеки і додаткову інформацію, потрібну ядру під час виконання. Віртуальний адресний простір виділяється в фізичній пам'яті випадковим чином і відслідковується таблицями сторінок ядра.

В структурах даних ядра зберігається багато інформації про кожен процес. До найбільш важливих відносять таку інформацію:

- Таблиця розподілу пам'яті, виділеної процесу;
- Поточний стан процесу (активний, в стані сну, виконується, зомбі);
- Пріоритет процесу;
- Інформація про ресурси, які використовує процесор (ЦП, пам'ять і т.д);
- Інформація про файли і мережеві порти, відкриті процесом;
- Маска сигналів (запис про те, які сигнали блокуються)
- Ім'я власника процесу.

Поток – це контекст виконання процесу. Кожний процес має як мінімум один потік, але деякі можуть мати й декілька. Кожний потік діє в адресному просторі зовнішнього процесу, та має свій стек і доступ до регістрів центрального процесора.

В сучасних комп'ютерних системах використовується декілька центральних процесорів та декілька ядер всередині кожного центрального процесора. Такі багатопотокові програми як Apache отримують максимальну користь з мультиядерних систем завдяки тому, що ці додатки можуть працювати з багатьма запитами одночасно.

Процеси поділяються на системні, демони (неінтерактивні, служби) і прикладні (інтерактивні).

1. Системні процеси є частиною ядра і завжди розташовані в оперативній пам'яті. Вони не мають відповідних їм програм у вигляді виконуваних файлів і запускаються особливим чином при ініціалізації ядра системи. Тобто їх виконувани інструкції та дані знаходяться в ядрі, тому такі процеси є складовою ядра. Системні процеси можуть викликати функції, а також звертатися до даних, які не мають доступу до інших процесів. Зазвичай, ці процеси вказані у квадратних дужках – []. Наприклад: [ksmd]. Окрім цього, процес init() також вважається системним, хоча не є частиною ядра. Всі процеси без батьків автоматично починають наслідуватись від цього процесу. Також, системні процеси не прив'язані до певного TTY або PTS.

2. Демони (сервіси, служби) – неінтерактивний процес, це програми, які працюють у фоновому режимі. Вони не мають графічного інтерфейсу і не прив'язана ні до якого керуючого терміналу. Зазвичай демони запускаються при ініціалізації системи, однак після ініціалізації ядра забезпечують роботу різних підсистем Unix: системи термінального доступу, системи друку, системи мережевого доступу, мережеских послуг і т.п. Демони не пов'язані із жодним користувачем, тобто не мають ніякого відношення до користувацьких процесів. Отримавши команду, вони виконують дію, для якої були створені. Наприклад, демон друку `cupsd` ставить в чергу документи, відправлені на друк, а потім посилає їх на принтер. Як правило, демони знаходяться у стадії очікування, поки для певного процесу не виникне потреба виконати певну послугу (звернення до архіву файлу, друк документу). Прикладами процесів-демонів слугують сервери протоколів HTTP (`httpd`) та FTP (`ftpd`), сервер системного журналу (`syslogd`), інші приклади - `usbd`, `sshd`. Зазвичай демони в кінці назви містять літеру «d».
3. Усі інші процеси, які виконуються в системі, вважаються прикладними або інтерактивними. Практично - це процеси, які запускаються під час роботи користувача. Наприклад, під час реєстрації користувача в системі запускається командний інтерпретатор (`shell`), який надає можливість працювати користувачу в Unix. Інші приклади інтерактивних процесів - `ls`, `sh`, `fsck`. Користувацькі процеси можуть виконуватися як в інтерактивному, так і у фоновому режимі, але виключно в рамках сеансу користувача. При виході з системи усі користувацькі процеси знищуються.

Для створення нового процесу існуючий процес, як правило, клонує сам себе за допомогою системного виклику `fork()`. Технічно, в системах Linux використовується системний виклик `clone`, (розширення системного виклику `fork`), який обробляє потоки і вмикає додаткові функції. Системний виклик `fork` залишається в ядрі для забезпечення зворотної сумісності, але насправді він виконує системний виклик `clone`. В результаті формується копія вихідного процесу, що має лише деякі відмінності. Зокрема, новому процесу надається власний ідентифікатор PID, і облік ресурсів ведеться для нього незалежно від предка.

Системний виклик `fork` має унікальну властивість: він повертає два різні значення. У дочірньому процесі це число 0, а батьківському – ідентифікатор PID процесу нащадка. Оскільки в іншому процесі ідентичні, вони мають перевіряти результат виклику, щоб визначити, що робити надалі.

Після завершення системного виклику `fork` дочірній процес зазвичай запускає нову програму за допомогою одного із системних викликів сімейства `exec`. Усі виклики цього сімейства замінюють програму, яку виконує процес, і встановлюють сегменти пам'яті у вихідний стан. Форми викликів `exec` розрізняються лише способами вказівки аргументів командного рядка та змінних середовища, що передаються новим програмі.

При завантаженні з системи ядро самостійно запускає кілька процесів.

Найбільш важливий з них - демон `init` або `systemd` з номером процесу, завжди рівний 1.

Ці процеси відповідають за виконання сценаріїв запуску системи, хоча характер їх дій у UNIX і Linux відрізняється. Усі процеси, крім тих, що створюються ядром є нащадками цих процесів.

Сигнали – це запити на переривання, що реалізуються на рівні процесів.

Існують понад тридцять різних сигналів, і вони знаходять різне застосування.

- Сигнали можуть надсилатися між процесами як засіб комунікації.
- Сигнали можуть надсилатися драйвером терміналу для знищення або призупинення процесів, коли користувач натискає спеціальні комбінації клавіш, такі як <Ctrl+C> або <Ctrl+Z>.
- Сигнали можуть надсилатися в різних цілях користувачем або адміністратором за допомогою команди kill.
- Сигнали можуть надсилатися ядром, коли процес виконує неприпустиму інструкцію, наприклад ділення на нуль

Коли надходить сигнал, можливий один із двох варіантів розвитку подій. Якщо процес призначив сигналу підпрограму обробки, то після виклику її надається інформація про контекст, у якому було згенеровано сигнал. Інакше ядро виконує від імені процесу дії, задані за замовчуванням. Ці дії залежать від сигналу. Багато сигналів призводять до завершення процесу, а в деяких випадках ще й створюються дампи пам'яті.

Функції, пов'язані з комбінаціями клавіш <Ctrl+Z> або <Ctrl+C>, можуть призначатися іншим клавіш за допомогою команди stty, але на практиці таке зустрічається дуже рідко.

Найголовніші сигнали та їх коди: [1]

Таблица 4.1. Сигналы, которые должен знать каждый администратор\*

№ <sup>в</sup>	Имя	Описание	Реакция по умолчанию	Перехватывается?	Блокируется?	Дамп памяти?
1	HUP	Отбой	Завершение	Да	Да	Нет
2	INT	Прерывание	Завершение	Да	Да	Нет
3	QUIT	Выход	Завершение	Да	Да	Да
9	KILL	Уничтожение	Завершение	Нет	Нет	Нет
10	BUS	Ошибка на шине	Завершение	Да	Да	Да
11	SEGV	Ошибка сегментации	Завершение	Да	Да	Да
15	TERM	Запрос на завершение	Завершение	Да	Да	Нет
17	STOP	Остановка	Остановка	Нет	Нет	Нет
18	TSTP	Сигнал остановки, посылаемый с клавиатуры	Остановка	Да	Да	Нет
19	CONT	Продолжение после остановки	Игнорируется	Да	Нет	Нет
28	WINCH	Изменение окна	Игнорируется	Да	Да	Нет
30	USR1	Определяется пользователем	Завершение	Да	Да	Нет
31	USR2	Определяется пользователем	Завершение	Да	Да	Нет

\*Список названий сигналов и номеров также можно получить с помощью встроенной в оболочку **bash** команды **kill -l**.

\*Зависит от используемой системы. Подробнее см. файл `/usr/include/signal.h` или map-страницы интерактивного руководства для системного вызова `signal()`.

Може виникнути питання, а навіщо потрібні аж як мінімум 5 сигналів зупинки процесу? Відповідь проста – HUP за замовчуванням у багатьох процесах не закінчує програму, а перечитує конфігураційні файли. INT перериває лише одну задачу, якщо їх декілька, а не повністю виходить з програми. QUIT – вихід з утворенням дампу пам'яті. KILL – перехватити не можна, миттєве вимкнення процесу. TERM – «ввічливе

завершення», зазвичай використовується саме цей сигнал. STOP – не закінчує, а призупиняє процес. Потім, його можна буде повернути за допомогою сигналу CONT (continue, продовжити).

Команда ps – основний інструмент, яким системний адміністратор користується для поточного контролю процесів. Версії цієї команди відрізняються аргументами і вихідним форматом, але, по суті, видають ту саму інформацію. В основному, відмінність у версіях - це наслідок різних шляхів розвитку систем UNIX. Крім того, постачальники систем можуть налаштовувати цю команду з урахуванням конфігурації системи, так як вона тісно пов'язана з особливостями обробки процесів в ядрі і тому часто відображає зміни в ядрі.

За допомогою команди ps можна отримати інформацію про ідентифікатори PID, UID, пріоритеті та керуючому терміналі того чи іншого процесу. Вона також дозволяє з'ясувати, який обсяг пам'яті використовує процес, скільки часу центрального процесора зайняло його виконання, яким є його поточний стан (виконується, зупинений, простояє і т.д.). Процеси-зомбі в лістингу команди позначаються як <exiting> або <defunct>.

Приклад:

```
alex@oracle:~  
[alex@oracle ~]$ ps  
  PID TTY          TIME CMD  
 1310 pts/0        00:00:00 bash  
 1687 pts/0        00:00:00 ps  
[alex@oracle ~]$
```

Команда top відображає стан процесів, які здійснюються у режимі реальному часу. Тобто ця команда виконує функцію системного монітору і дозволяє виявити причини нестабільної роботи операційної системи та виявити процеси, які споживають більшість системних ресурсів.

```
alex@oracle:~  
top - 09:06:41 up 2:30, 2 users, load average: 0.00, 0.01, 0.00  
Tasks: 107 total, 1 running, 106 sleeping, 0 stopped, 0 zombie  
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.3 hi,  0.0 si,  0.0 st  
MiB Mem :  463.3 total,  142.9 free,  121.0 used,  199.4 buff/cache  
MiB Swap: 2048.0 total, 2034.5 free,   13.5 used.  329.2 avail Mem  


| PID  | USER | PR | NI  | VIRT   | RES  | SHR  | S | %CPU | %MEM | TIME+   | COMMAND                   |
|------|------|----|-----|--------|------|------|---|------|------|---------|---------------------------|
| 1689 | alex | 20 | 0   | 65440  | 4920 | 4152 | R | 0.3  | 1.0  | 0:00.02 | top                       |
| 1    | root | 20 | 0   | 172436 | 5128 | 3504 | S | 0.0  | 1.1  | 0:03.48 | systemd                   |
| 2    | root | 20 | 0   | 0      | 0    | 0    | S | 0.0  | 0.0  | 0:00.00 | kthreadd                  |
| 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_par_gp                |
| 6    | root | 0  | -20 | 0      | 0    | 0    | I | 0.0  | 0.0  | 0:00.00 | kworker/0:0H-events_high+ |
| 8    | root | 0  | -20 | 0      | 0    | 0    | I | 0.0  | 0.0  | 0:00.16 | kworker/0:1H-events_high+ |
| 9    | root | 0  | -20 | 0      | 0    | 0    | I | 0.0  | 0.0  | 0:00.00 | mm_percpu_wq              |
| 10   | root | 20 | 0   | 0      | 0    | 0    | S | 0.0  | 0.0  | 0:00.29 | ksoftirqd/0               |
| 11   | root | 20 | 0   | 0      | 0    | 0    | I | 0.0  | 0.0  | 0:00.21 | rcu_sched                 |
| 12   | root | rt | 0   | 0      | 0    | 0    | S | 0.0  | 0.0  | 0:00.01 | migration/0               |
| 14   | root | 20 | 0   | 0      | 0    | 0    | S | 0.0  | 0.0  | 0:00.00 | cpuhp/0                   |
| 16   | root | 20 | 0   | 0      | 0    | 0    | S | 0.0  | 0.0  | 0:00.00 | kdevtmpfs                 |
| 17   | root | 0  | -20 | 0      | 0    | 0    | I | 0.0  | 0.0  | 0:00.00 | netns                     |
| 18   | root | 20 | 0   | 0      | 0    | 0    | S | 0.0  | 0.0  | 0:00.00 | kauditd                   |
| 19   | root | 20 | 0   | 0      | 0    | 0    | S | 0.0  | 0.0  | 0:00.00 | khungd                    |

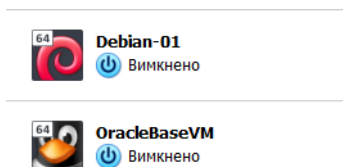

```

Кожному процесу планувальник ОС Linux призначає пріоритет, який впливає на те, як довго буде працювати процес. Це пояснюється тим, що частка ресурсів процесора, яка призначається процесу, зважується відповідно до його значення «ввічливості» (niceness). Пріоритети називаються в Linux значеннями ввічливості (NI), оскільки їх основна ідея - «бути ввічливим» по відношенню до інших процесів шляхом проходження процесної пріоритетності, дозволяючи іншим процесам споживати більше системного процесорного часу. Ядро Linux виділяє завданням з більш високим пріоритетом більший квант часу, а завданням з меншим пріоритетом - менший квант часу, який в середньому становить 200 і 10 мс відповідно.

Діапазон значень ввічливості - від «-20» (найвищий пріоритет) до «19» (нижчий пріоритет) включно, а значення за замовчуванням – 0 (процеси, запущені звичайними користувачами, зазвичай мають пріоритет 0). Таким чином, чим нижче значення ввічливості процесу, тим вище його пріоритет і більше квант часу.

### Хід роботи

При виконанні роботи використовувались два дистрибутива: Debian та Oracle Linux (Server). Другий більш оптимізований, так як там немає графічного інтерфейсу, та вже налаштовані порти, тобто можна підключатись за допомогою ssh. Це потрібно для пояснення деяких моментів.

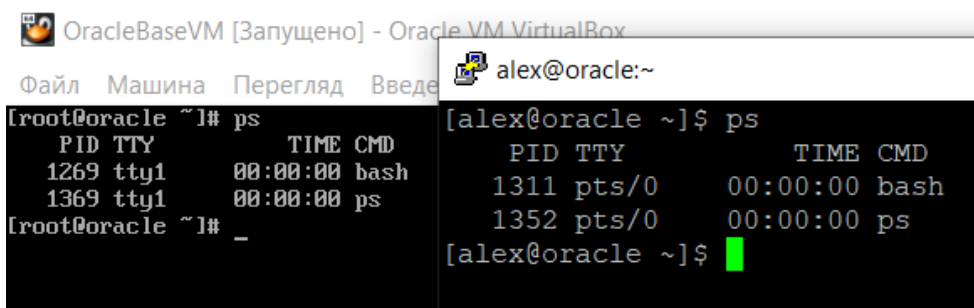


Для початку, виведемо список процесів у короткому форматі за допомогою команди ps.

```
alex@debian:~$ ps
  PID TTY          TIME CMD
 1300 pts/0        00:00:00 bash
 1993 pts/0        00:00:00 ps
alex@debian:~$
```

PID – унікальний номер процесу, TTY (TeleTypeWriter) – ім'я терміналу, TIME – час роботи процесу, CMD – команда, за допомогою якої запускався процес.

TTY може набувати значень «?» (термінал невідомий, або процес до нього не прив'язаний), TTY (безпосередньо основний термінал) та PTS (термінал прив'язаний через певну утиліту до основного)



На основній машині є лише CLI – тобто, Command Line Interface. Взаємодія йде безпосередньо від користувача root з терміналом. Тому стоїть значення tty. Після нього йде номер терміналу. Наприклад, можна переключитись на інший за допомогою комбінації клавіш <Ctrl+Alt+F(номер-терміналу)>

```
[root@oracle ~]# ps
  PID TTY          TIME CMD
 1378 tty2        00:00:00 bash
 1407 tty2        00:00:00 ps
[root@oracle ~]#
```

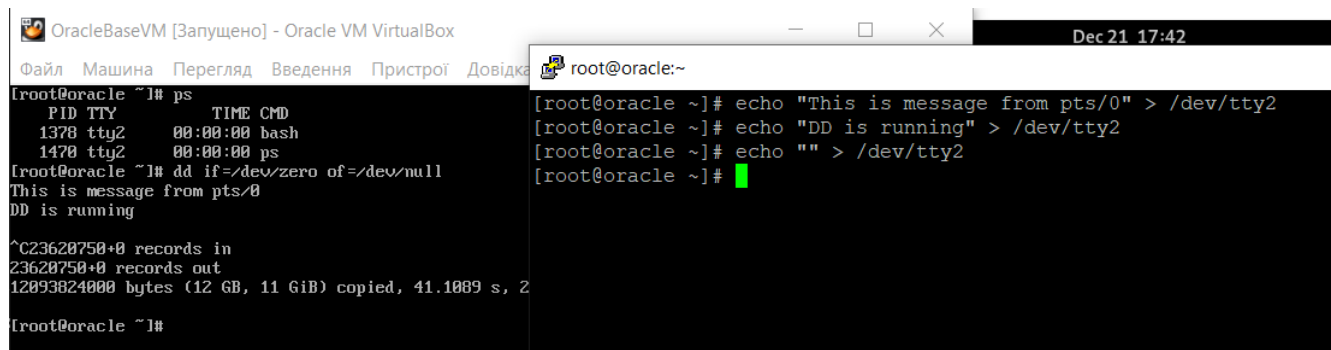
Активні термінали можна переглянути за допомогою команди “w”

```
[alex@oracle ~]$ w
10:47:04 up 50 min,  3 users,  load average: 0.16, 0.23, 0.11
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
root      tty1     -                09:58    5:43   0.06s  0.06s -bash
alex      pts/0    10.0.2.2        09:59    0.00s  0.04s  0.00s w
root      tty2     -                10:29   16.00s  0.06s  0.06s -bash
[alex@oracle ~]$
```

На скріншоті видно, що наразі активні 3 термінали: tty1, tty2, та pts/0 (підключення через ssh за допомогою утиліти Putty).

В лінуksі все є файлом – тому, виведення з терміналу також є файлом. Тобто, туди можна перенаправити якийсь текст як і у звичайний файл за допомогою каналу (звичайно, потрібно мати root права). Перевіримо це:

1. Перейдемо на tty2 (основна машина, Ctrl + Alt + F2).
2. Запустимо якийсь процес, який блокує введення для наглядності. Наприклад, будемо копіювати файли «з пустого в порожнє» - з файлу зі сміттям /dev/zero в «пустоту» - /dev/null. Для цього використаємо команду dd if=/dev/zero of=/dev/null.
3. На цьому етапі можна використовувати оператори каналів в іншому терміналі. В pts/0 використовуємо команду echo, і виведення переадресуємо за допомогою оператору каналу “>” в файл терміналу tty2 /dev/tty2.
4. Спостерігаємо за результатом! Бачимо, що повідомлення з'явилися незважаючи на запущений процес. Зупиняємо його, бачимо фінальне повідомлення, яке показує що процес весь цей час був запущений, а стандартний потік введення заблокований.

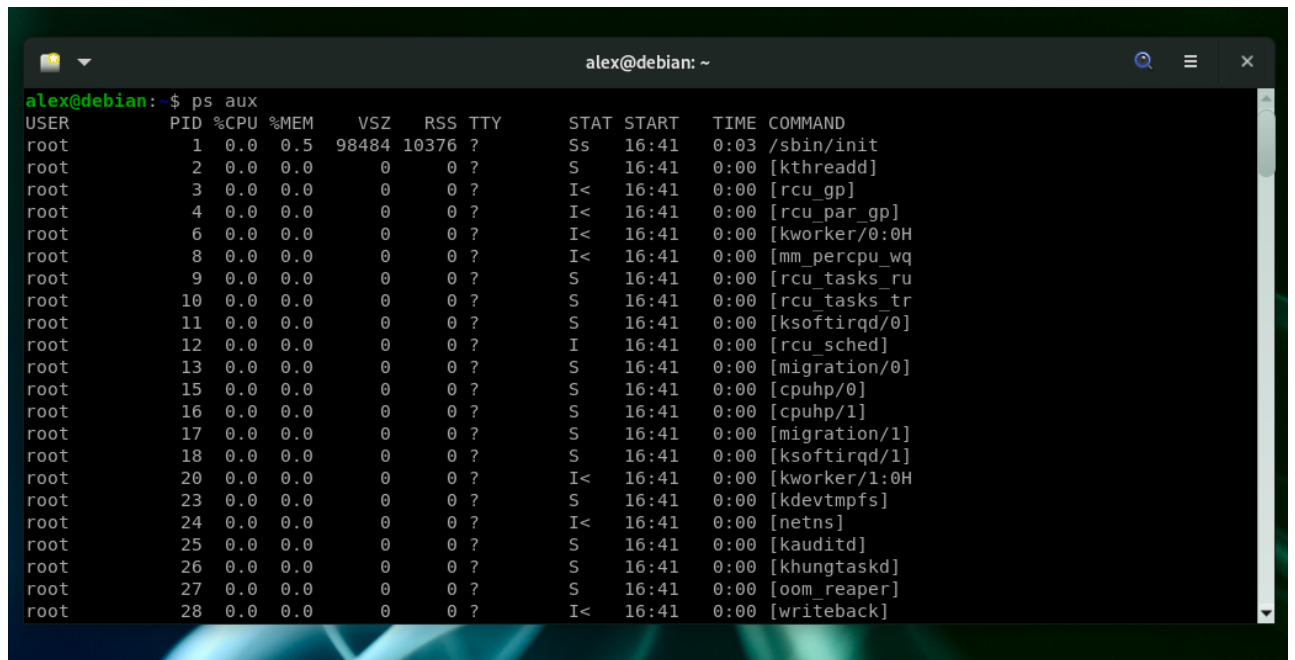


```
OracleBaseVM [Запущено] - Oracle VM VirtualBox
Файл  Машина  Перегляд  Введення  Пристрої  Довідка
root@oracle:~
[root@oracle ~]# ps
  PID TTY          TIME CMD
 1378 tty2        00:00:00 bash
 1470 tty2        00:00:00 ps
[root@oracle ~]# dd if=/dev/zero of=/dev/null
This is message from pts/0
DD is running
^C23620750+0 records in
23620750+0 records out
12093824000 bytes (12 GB, 11 GiB) copied, 41.1089 s, 291 MB/s
[root@oracle ~]#

Dec 21 17:42
root@oracle:~
[root@oracle ~]# echo "This is message from pts/0" > /dev/tty2
[root@oracle ~]# echo "DD is running" > /dev/tty2
[root@oracle ~]# echo "" > /dev/tty2
[root@oracle ~]#
```



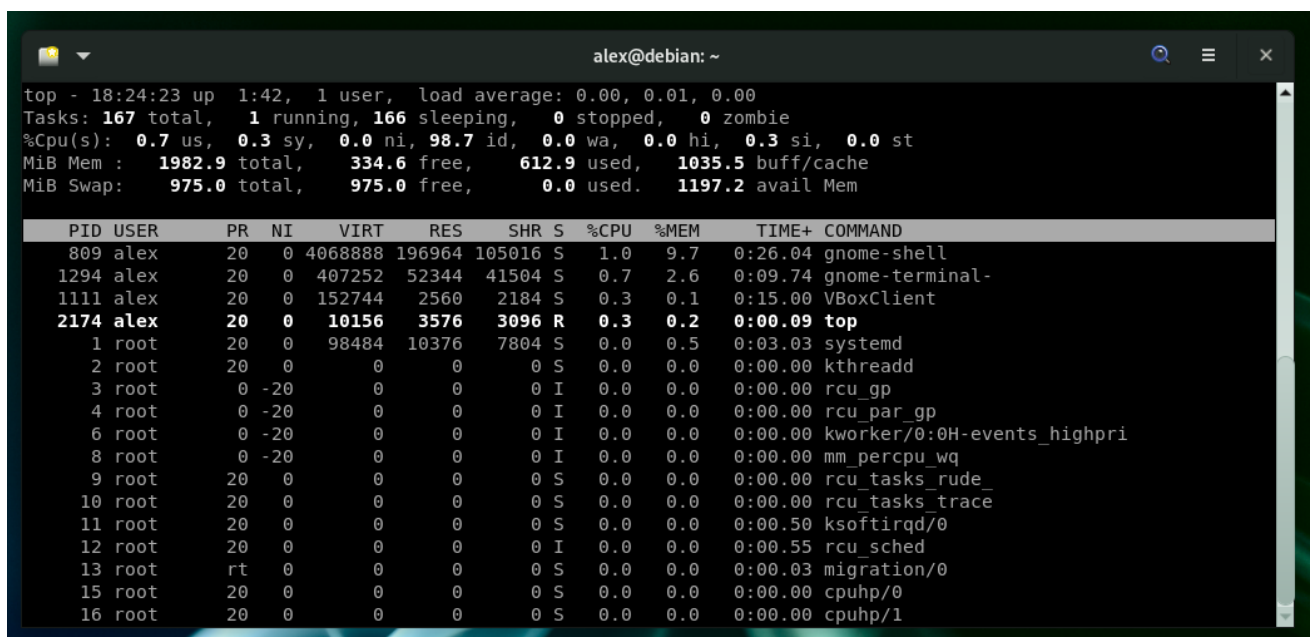
Повертаємось назад до Debian. Виведемо повний лістинг команди `ps` за допомогою ключів `aux` (`a` – ключ для виведення процесів всіх користувачів, `u` – виведення в user-friendly форматі, `x` – виведення процесів не прив’язаних до терміналу). Ці ключі працюють так в BSD форматі – в UNIX або GNU форматах можуть працювати зовсім по іншому.



```
alex@debian: ~  
alex@debian:~$ ps aux  
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND  
root         1  0.0  0.5 98484 10376 ?        Ss   16:41   0:03 /sbin/init  
root         2  0.0  0.0      0      0 ?        S    16:41   0:00 [kthreadd]  
root         3  0.0  0.0      0      0 ?        I<   16:41   0:00 [rcu_gp]  
root         4  0.0  0.0      0      0 ?        I<   16:41   0:00 [rcu_par_gp]  
root         6  0.0  0.0      0      0 ?        I<   16:41   0:00 [kworker/0:0H  
root         8  0.0  0.0      0      0 ?        I<   16:41   0:00 [mm_percpu_wq  
root         9  0.0  0.0      0      0 ?        S    16:41   0:00 [rcu_tasks_ru  
root        10  0.0  0.0      0      0 ?        S    16:41   0:00 [rcu_tasks_tr  
root        11  0.0  0.0      0      0 ?        S    16:41   0:00 [ksoftirqd/0]  
root        12  0.0  0.0      0      0 ?        I    16:41   0:00 [rcu_sched]  
root        13  0.0  0.0      0      0 ?        S    16:41   0:00 [migration/0]  
root        15  0.0  0.0      0      0 ?        S    16:41   0:00 [cpuhp/0]  
root        16  0.0  0.0      0      0 ?        S    16:41   0:00 [cpuhp/1]  
root        17  0.0  0.0      0      0 ?        S    16:41   0:00 [migration/1]  
root        18  0.0  0.0      0      0 ?        S    16:41   0:00 [ksoftirqd/1]  
root        20  0.0  0.0      0      0 ?        I<   16:41   0:00 [kworker/1:0H  
root        23  0.0  0.0      0      0 ?        S    16:41   0:00 [kdevtmpfs]  
root        24  0.0  0.0      0      0 ?        I<   16:41   0:00 [netns]  
root        25  0.0  0.0      0      0 ?        S    16:41   0:00 [kauditd]  
root        26  0.0  0.0      0      0 ?        S    16:41   0:00 [khungtaskd]  
root        27  0.0  0.0      0      0 ?        S    16:41   0:00 [oom_reaper]  
root        28  0.0  0.0      0      0 ?        I<   16:41   0:00 [writeback]
```

Також, можна використовувати ключ `w` (до 4 разів). Він використовується для того щоб виведення працювало повністю – на попередньому скріншоті виведення `command` обрізається.

У цієї утиліти є мінус – вона статична. На екран виводиться одномоментний зліпок процесів системи. Цю проблему вирішує утиліта `top`, яка виводить дані в динамічному форматі:



```
alex@debian: ~  
top - 18:24:23 up 1:42, 1 user, load average: 0.00, 0.01, 0.00  
Tasks: 167 total, 1 running, 166 sleeping, 0 stopped, 0 zombie  
%Cpu(s): 0.7 us, 0.3 sy, 0.0 ni, 98.7 id, 0.0 wa, 0.0 hi, 0.3 si, 0.0 st  
MiB Mem : 1982.9 total, 334.6 free, 612.9 used, 1035.5 buff/cache  
MiB Swap: 975.0 total, 975.0 free, 0.0 used, 1197.2 avail Mem  
  
  PID USER      PR  NI  VIRT  RES  SHR S  %CPU  %MEM    TIME+  COMMAND  
  809 alex       20   0 4068888 196964 105016 S   1.0   9.7   0:26.04 gnome-shell  
 1294 alex       20   0 407252 52344 41504 S   0.7   2.6   0:09.74 gnome-terminal-  
 1111 alex       20   0 152744 2560 2184 S   0.3   0.1   0:15.00 VBoxClient  
 2174 alex       20   0 10156 3576 3096 R   0.3   0.2   0:00.09 top  
    1 root       20   0 98484 10376 7804 S   0.0   0.5   0:03.03 systemd  
    2 root       20   0      0      0      0 S   0.0   0.0   0:00.00 kthreadd  
    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_par_gp  
    6 root        0 -20      0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri  
    8 root        0 -20      0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq  
    9 root       20   0      0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_rude_  
   10 root       20   0      0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_trace  
   11 root       20   0      0      0      0 S   0.0   0.0   0:00.50 ksoftirqd/0  
   12 root       20   0      0      0      0 I   0.0   0.0   0:00.55 rcu_sched  
   13 root       rt    0      0      0      0 S   0.0   0.0   0:00.03 migration/0  
   15 root       20   0      0      0      0 S   0.0   0.0   0:00.00 cpuhp/0  
   16 root       20   0      0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
```



Тут можна побачити параметр uptime, скільки користувачів зараз працюють у системі, навантаження процесів за останні 5, 10, 15 хвилин. Другий рядок: скільки процесів у цілому, скільки працюють, скільки «сплять», скільки зупинено, скільки зомбі. Третій рядок відноситься до процесору, четвертий до ОЗУ, п'ятий до файлу підкачки.

Перша колонка – ProcessID, друга – користувач, який запустив процес. Третя – пріоритет, четверта – параметр Nice. Далі йдуть колонки пов'язані з пам'яттю, статус процесу, стан завантаженості процесору, пам'яті, та інше.

Але – ця утиліта, як і ps, є досить старою. Можна використати один з найвідоміших аналогів – htop. Але перед цим, її потрібно встановити:

```
alex@debian: ~  
alex@debian:~$ sudo apt install htop  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
The following package was automatically installed and is no longer required:  
  linux-image-5.10.0-18-amd64  
Use 'sudo apt autoremove' to remove it.  
Suggested packages:  
  strace  
The following NEW packages will be installed:  
  htop  
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.  
Need to get 127 kB of archives.  
After this operation, 328 kB of additional disk space will be used.  
Get:1 http://deb.debian.org/debian bullseye/main amd64 htop amd64 3.0.5-7 [127 kB]  
Fetched 127 kB in 0s (428 kB/s)  
Selecting previously unselected package htop.
```

Запускаємо:

```
alex@debian: ~  
0[|] 1.3% Tasks: 102, 232 thr; 1 running  
1[|] 1.3% Load average: 0.42 0.97 0.59  
Mem[|||||] 666M/1.94G Uptime: 02:00:12  
Swp[|] 36.0M/975M  


| PID   | USER      | PRI | NI | VIRT  | RES   | SHR   | S | CPU% | MEM% | TIME+   | Command                                        |
|-------|-----------|-----|----|-------|-------|-------|---|------|------|---------|------------------------------------------------|
| 13627 | alex      | 20  | 0  | 8664  | 4596  | 3544  | R | 1.3  | 0.2  | 0:00.20 | htop                                           |
| 1163  | alex      | 20  | 0  | 149M  | 1340  | 1108  | S | 0.7  | 0.1  | 0:17.73 | /usr/bin/VBoxClient --draganddrop              |
| 1     | root      | 20  | 0  | 99644 | 9764  | 6884  | S | 0.0  | 0.5  | 0:04.39 | /sbin/init                                     |
| 213   | root      | 20  | 0  | 48580 | 17472 | 14320 | S | 0.0  | 0.9  | 0:00.94 | /lib/systemd/systemd-journald                  |
| 244   | root      | 20  | 0  | 23860 | 6348  | 3480  | S | 0.0  | 0.3  | 0:00.86 | /lib/systemd/systemd-udev                      |
| 430   | root      | 20  | 0  | 230M  | 7300  | 6336  | S | 0.0  | 0.4  | 0:00.19 | /usr/libexec/accounts-daemon                   |
| 432   | avahi     | 20  | 0  | 7332  | 3236  | 2888  | S | 0.0  | 0.2  | 0:00.07 | avahi-daemon: running [debian.local]           |
| 434   | root      | 20  | 0  | 6748  | 2640  | 2432  | S | 0.0  | 0.1  | 0:00.01 | /usr/sbin/cron -f                              |
| 435   | messagebu | 20  | 0  | 9836  | 5468  | 3572  | S | 0.0  | 0.3  | 0:03.35 | /usr/bin/dbus-daemon --system --address=system |
| 436   | root      | 20  | 0  | 248M  | 16292 | 13920 | S | 0.0  | 0.8  | 0:00.49 | /usr/sbin/NetworkManager --no-daemon           |
| 439   | root      | 20  | 0  | 230M  | 8840  | 5992  | S | 0.0  | 0.4  | 0:00.39 | /usr/libexec/polkitd --no-debug                |
| 440   | root      | 20  | 0  | 215M  | 4088  | 3148  | S | 0.0  | 0.2  | 0:00.19 | /usr/sbin/rsyslogd -n -iNONE                   |
| 441   | root      | 20  | 0  | 227M  | 7016  | 4568  | S | 0.0  | 0.3  | 0:00.05 | /usr/libexec/switcheroo-control                |
| 442   | root      | 20  | 0  | 22176 | 5992  | 5148  | S | 0.0  | 0.3  | 0:00.32 | /lib/systemd/systemd-logind                    |
| 443   | root      | 20  | 0  | 384M  | 13140 | 9132  | S | 0.0  | 0.6  | 0:00.15 | /usr/libexec/udisks2/udisksd                   |
| 446   | root      | 20  | 0  | 14620 | 4172  | 3556  | S | 0.0  | 0.2  | 0:00.08 | /sbin/wpa_supplicant -u -s -O /run/wpa supplic |

  
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice + F9Kill F10Quit
```

Бачимо більш зручний інтерфейс, який нагадує файловий менеджер mc. Видно всі ті ж самі колонки, також зверху з'явився «ползунок» завантаженості. Цифри – ядра процесору.

Запишемо у файл результат команди ps у звичайному варіанті та у подовженому за допомогою канальних операторів.

Короткий варіант:

```
alex@debian:~$ ps > ./Documents/Lab6/ps.txt
alex@debian:~$ cat ./Documents/Lab6/ps.txt
  PID TTY          TIME CMD
 1300 pts/0        00:00:00 bash
 13701 pts/0        00:00:00 ps
alex@debian:~$
```

Повний варіант:

```
alex@debian: ~
alex@debian:~$ ps auxwww > ./Documents/Lab6/ps-full.txt
alex@debian:~$ cat ./Documents/Lab6/ps-full.txt
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.4 99644  9764 ?        Ss   17:19   0:04 /sbin/init
root         2  0.0  0.0      0     0 ?        S    17:19   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   17:19   0:00 [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   17:19   0:00 [rcu_par_gp]
root         6  0.0  0.0      0     0 ?        I<   17:19   0:00 [kworker/0:0H-events_highpri]
root         8  0.0  0.0      0     0 ?        I<   17:19   0:00 [mm_percpu_wq]
root         9  0.0  0.0      0     0 ?        S    17:19   0:00 [rcu_tasks_rude_]
root        10  0.0  0.0      0     0 ?        S    17:19   0:00 [rcu_tasks_trace]
root        11  0.0  0.0      0     0 ?        S    17:19   0:01 [ksoftirqd/0]
root        12  0.0  0.0      0     0 ?        I    17:19   0:01 [rcu_sched]
root        13  0.0  0.0      0     0 ?        S    17:19   0:00 [migration/0]
root        15  0.0  0.0      0     0 ?        S    17:19   0:00 [cpuhp/0]
root        16  0.0  0.0      0     0 ?        S    17:19   0:00 [cpuhp/1]
root        17  0.0  0.0      0     0 ?        S    17:19   0:00 [migration/1]
root        18  0.0  0.0      0     0 ?        S    17:19   0:00 [ksoftirqd/1]
root        20  0.0  0.0      0     0 ?        I<   17:19   0:00 [kworker/1:0H-events_highpri]
root        23  0.0  0.0      0     0 ?        S    17:19   0:00 [kdevtmpfs]
root        24  0.0  0.0      0     0 ?        I<   17:19   0:00 [netns]
root        25  0.0  0.0      0     0 ?        S    17:19   0:00 [kauditd]
root        26  0.0  0.0      0     0 ?        S    17:19   0:00 [khungtaskd]
root        27  0.0  0.0      0     0 ?        S    17:19   0:00 [oom_reaper]
```

USER – ім'я користувача, який є власником процесу (root),

PID – ідентифікатор процесу,

%CPU – відсоток використання центрального процесора даним процесом,

%MEM – відсоток використання оперативної пам'яті даним процесом.

VSZ – обсяг віртуальної пам'яті, яка використовується процесом,

RSS – розмір сторінок пам'яті, виділених процесу,

TTY – номер терміналу,

STAT – вказує на статус процесу: S=sleep (очікує подій), R=running (виконується), Z=zombie (очікує батьківський процес) (S),

START – час, коли був запущений процес,

TIME+ – загальний час активності процесу,

COMMAND – ім'я процесу.

Виводимо всі процеси включно з PPID за допомогою команди `ps ax -l`:

```
alex@debian: ~  
alex@debian:~$ ps ax -l  
F S    UID      PID      PPID  C  PRI  NI ADDR  SZ  WCHAN  TTY          TIME CMD  
4 S    0        1        0  0  80   0 - 41295 -    ?           0:04 /sbin/init  
1 S    0        2        0  0  80   0 -    - -    ?           0:00 [kthreadd]  
1 I    0        3        2  0  60 -20 -    - -    ?           0:00 [rcu_gp]  
1 I    0        4        2  0  60 -20 -    - -    ?           0:00 [rcu_par_gp]  
1 I    0        6        2  0  60 -20 -    - -    ?           0:00 [kworker/0:0H-events_highpri]  
1 I    0        8        2  0  60 -20 -    - -    ?           0:00 [mm_percpu_wq]  
1 S    0        9        2  0  80   0 -    - -    ?           0:00 [rcu_tasks_rude_]  
1 S    0       10        2  0  80   0 -    - -    ?           0:00 [rcu_tasks_trace]  
1 S    0       11        2  0  80   0 -    - -    ?           0:01 [ksoftirqd/0]  
1 I    0       12        2  0  80   0 -    - -    ?           0:01 [rcu_sched]  
1 S    0       13        2  0 -40   - -    - -    ?           0:00 [migration/0]  
1 S    0       15        2  0  80   0 -    - -    ?           0:00 [cpuhp/0]  
1 S    0       16        2  0  80   0 -    - -    ?           0:00 [cpuhp/1]  
1 S    0       17        2  0 -40   - -    - -    ?           0:00 [migration/1]  
1 S    0       18        2  0  80   0 -    - -    ?           0:00 [ksoftirqd/1]  
1 I    0       20        2  0  60 -20 -    - -    ?           0:00 [kworker/1:0H-events_highpri]  
5 S    0       23        2  0  80   0 -    - -    ?           0:00 [kdevtmpfs]  
1 I    0       24        2  0  60 -20 -    - -    ?           0:00 [netns]  
1 S    0       25        2  0  80   0 -    - -    ?           0:00 [kauditd]  
1 S    0       26        2  0  80   0 -    - -    ?           0:00 [khungtaskd]  
1 S    0       27        2  0  80   0 -    - -    ?           0:00 [oom_reaper]  
1 I    0       28        2  0  60 -20 -    - -    ?           0:00 [writeback]
```

F – поле для прапорців, S – стан, PPID – PID батьківського процесу, C – пріоритет процесу для планувальника завдань, WCHAN – виводить функцію ядра, в якій спить процес, або виводить мінус. Інші стовбці були описані раніше.

Виведемо цей же список, але деревом (`ps ax -l -forest`):

```
alex@debian: ~  
alex@debian:~$ ps ax -l --forest  
F S    UID      PID      PPID  C  PRI  NI ADDR  SZ  WCHAN  TTY          TIME CMD  
1 S    0        2        0  0  80   0 -    - -    ?           0:00 [kthreadd]  
1 I    0        3        2  0  60 -20 -    - -    ?           0:00 \ [rcu_gp]  
1 I    0        4        2  0  60 -20 -    - -    ?           0:00 \ [rcu_par_gp]  
1 I    0        6        2  0  60 -20 -    - -    ?           0:00 \ [kworker/0:0H-events_highpri]  
1 I    0        8        2  0  60 -20 -    - -    ?           0:00 \ [mm_percpu_wq]  
1 S    0        9        2  0  80   0 -    - -    ?           0:00 \ [rcu_tasks_rude_]  
1 S    0       10        2  0  80   0 -    - -    ?           0:00 \ [rcu_tasks_trace]  
1 S    0       11        2  0  80   0 -    - -    ?           0:01 \ [ksoftirqd/0]  
1 I    0       12        2  0  80   0 -    - -    ?           0:01 \ [rcu_sched]  
1 S    0       13        2  0 -40   - -    - -    ?           0:00 \ [migration/0]  
1 S    0       15        2  0  80   0 -    - -    ?           0:00 \ [cpuhp/0]  
1 S    0       16        2  0  80   0 -    - -    ?           0:00 \ [cpuhp/1]  
1 S    0       17        2  0 -40   - -    - -    ?           0:00 \ [migration/1]  
1 S    0       18        2  0  80   0 -    - -    ?           0:00 \ [ksoftirqd/1]  
1 I    0       20        2  0  60 -20 -    - -    ?           0:00 \ [kworker/1:0H-events_highpri]  
5 S    0       23        2  0  80   0 -    - -    ?           0:00 \ [kdevtmpfs]  
1 I    0       24        2  0  60 -20 -    - -    ?           0:00 \ [netns]  
1 S    0       25        2  0  80   0 -    - -    ?           0:00 \ [kauditd]  
1 S    0       26        2  0  80   0 -    - -    ?           0:00 \ [khungtaskd]  
1 S    0       27        2  0  80   0 -    - -    ?           0:00 \ [oom_reaper]  
1 I    0       28        2  0  60 -20 -    - -    ?           0:00 \ [writeback]  
1 S    0       29        2  0  80   0 -    - -    ?           0:01 \ [kcompactd0]
```

Та допишемо у файл `ps-full.txt`:

```
alex@debian: ~  
alex@debian:~$ ps ax -l --forest >> ./Documents/Lab6/ps-full.txt  
alex@debian:~$ cat ./Documents/Lab6/ps-full.txt  
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND  
root         1  0.0  0.4 99644  9764 ?        Ss   17:19   0:04 /sbin/init  
root         2  0.0  0.0      0     0 ?        S    17:19   0:00 [kthreadd]
```

Переглянемо список процесів свого користувача за допомогою ключів -u та -U, які визначають Real ID (RUID) – ідентифікатор користувача, який запустив процес – та Effective UID, який визначає права доступу процесу до системних ресурсів. [2]

```
alex@debian:~$ ps -U alex -u alex
  PID TTY          TIME CMD
  673 ?            00:00:00 systemd
  674 ?            00:00:00 (sd-pam)
  688 ?            00:00:00 pipewire
  689 ?            00:00:01 pulseaudio
  691 ?            00:00:01 tracker-miner-f
  694 ?            00:00:00 gnome-keyring-d
  699 tty2        00:00:00 gdm-wayland-ses
  702 ?            00:00:01 dbus-daemon
  710 tty2        00:00:00 gnome-session-b
  719 ?            00:00:00 pipewire-media-
  723 ?            00:00:00 gvfsd
```

Виведемо дерево процесів для свого користувача.

```
alex@debian:~$ pstree alex
gdm-wayland-ses─gnome-session-b─3*[{gnome-session-b}]
                  └2*[{gdm-wayland-ses}]

gnome-keyring-d─3*[{gnome-keyring-d}]

systemd─(sd-pam)
        └2*[VBoxClient─VBoxClient─2*[{VBoxClient}]]
          └VBoxClient─VBoxClient─3*[{VBoxClient}]
            └VBoxClient─VBoxDRMClient
              └at-spi2-registr─2*[{at-spi2-registr}]
                └blueman-tray─3*[{blueman-tray}]
                  └dbus-daemon
                    └dconf-service─2*[{dconf-service}]
                      └evolution-addre─5*[{evolution-addre}]
                        └evolution-calen─8*[{evolution-calen}]
                          └evolution-sourc─3*[{evolution-sourc}]
                            └gjs─4*[{gjs}]
                              └gnome-session-b
                                └at-spi-bus-laun─dbus-daemon
                                  └3*[{at-spi-bus-laun}]
                                    └blueman-applet─3*[{blueman-applet}]
                                      └evolution-alarm─5*[{evolution-alarm}]
                                        └gnome-software─5*[{gnome-software}]
                                          └gsd-disk-utilit─2*[{gsd-disk-utilit}]
```

Виведемо список сигналів, доступних користувачу.

```
alex@debian:~$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
 6) SIGABRT     7) SIGBUS     8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2   13) SIGPIPE    14) SIGALRM    15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD   18) SIGCONT    19) SIGSTOP    20) SIGTSTP
21) SIGTTIN    22) SIGTTOU   23) SIGURG     24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF   28) SIGWINCH   29) SIGIO       30) SIGPWR
31) SIGSYS     34) SIGRTMIN  35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX

alex@debian:~$
```

Проведемо декілька експериментів. (Знову на Oracle Linux Server).

Потрібна утиліта, з якою можна зробити щось цікаве використовуючи сигнали.

Використаємо ту ж саму утиліту dd для низькорівневого копіювання, тому що їй можна посилати сигнал USR1 і таким чином отримувати статистику копіювання.

```
...
    treat 'seek=N' as a byte count (oflag only)

Sending a USR1 signal to a running 'dd' process makes it print I/O statistics to standard
error and then resume copying.

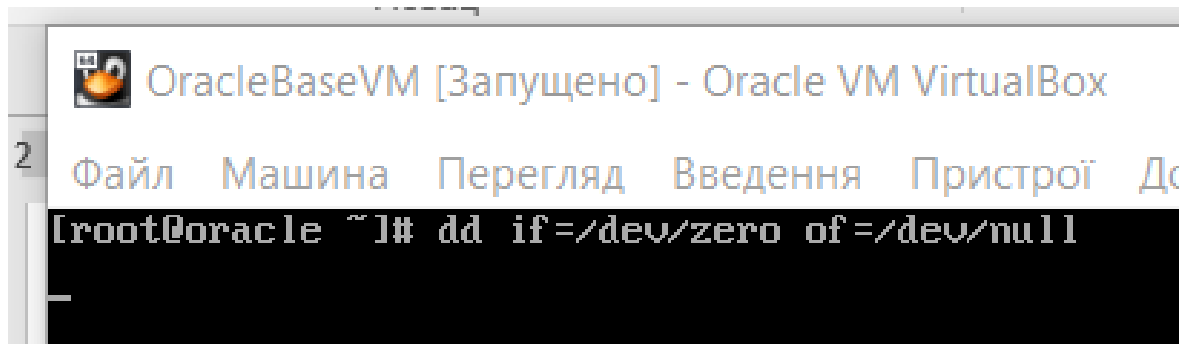
Options are:

--help display this help and exit
```

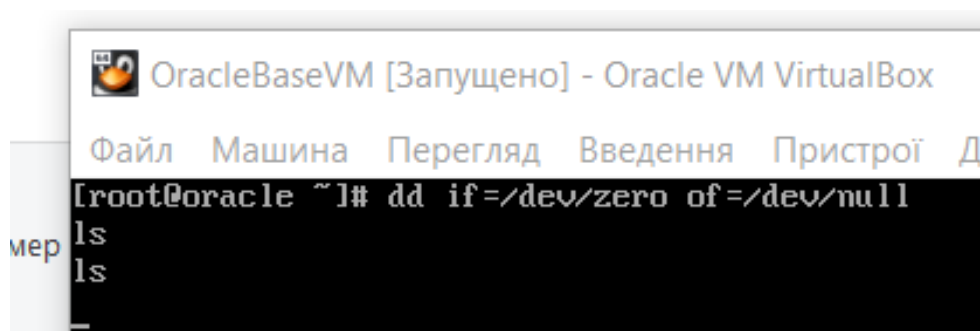
Запускаємо два термінали:

```
[alex@oracle ~]$ w
 15:54:56 up 2 min,  2 users,  load average: 0.34, 0.18, 0.07
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
root      tty1      -               15:53    16.00s  0.03s  0.03s -bash
alex      tty2      -               15:54     0.00s  0.02s  0.00s w
[alex@oracle ~]$
```

В першому запускаємо копіювання:



Процес копіювання прив'язаний до першого терміналу, тому забирає собі потік виведення.



Можна вводити безліч команд, але це не спрацює – потік виведення зайнятий.

Переключаємось на другий термінал. Перенаправляємо за допомогою конвеєра результат роботи команди ps в утиліту grep. Шукаємо процес dd. Знаходимо його, він є під номером 1357. Викликаємо команду kill з ключем -s, тобто вказуємо що хочемо передати сигнал, та передаємо ім'я сигналу USR1 як аргумент та номер процесу. Після цих дій натискаємо Enter. Але – нічого немає.

```
Файл  Машина  Перегляд  Введення  Пристрої  Довідка
[alex@oracle ~]$ ps aux | grep "dd"
root      2  0.0  0.0   0   0 ?        S   15:52   0:00 [kthreadd]
root     97  0.0  0.0   0   0 ?        I<  15:52   0:00 [ipv6_addrconf]
dbus     734  0.0  1.1 56368 5524 ?        Ss  15:52   0:00 /usr/bin/dbus-daemon --system --systemd-acti-
dress=systemd --nofork --nopidfile --systemd-activation --syslog-only
root    1357  0.0  0.2   7356 1008 tty1    R+  15:56   3:14 dd if=/dev/zero of=/dev/null
alex    1377  0.0  0.2 12176 1180 tty2    R+  15:59   0:00 grep --color=auto dd
[alex@oracle ~]$ sudo kill -s USR1 1357
[alex@oracle ~]$ _
```

Нічого немає тому, що сигнал передався процесу, який прикріплений до першого терміналу. Тому повідомлення з'явилося в цьому місці.

```
Файл  Машина  Перегляд  Введення  Пристрої  Довідка
[root@oracle ~]# dd if=/dev/zero of=/dev/null
ls
ls
117385625+0 records in
117385625+0 records out
6010144000 bytes (60 GB, 56 GiB) copied, 206.19 s, 291 MB/s
_
```

Використовуючи текстовий редактор ві напишемо маленький скрипт для виведення статистики по цьому процесу:

```
#!/bin/bash

while true; do
    kill -s USR1 1405
    sleep 2
done
```

Спробуємо запустити:

```
[alex@oracle Lab61]$ sudo chmod +x dd-stats.sh
[alex@oracle Lab61]$ ./dd-stats.sh
_
```

Результат:

```
105709340+0 records in
105709339+0 records out
54123181568 bytes (54 GB, 50 GiB) copied, 188.52 s, 287 MB/s
106805989+0 records in
106805988+0 records out
54684665856 bytes (55 GB, 51 GiB) copied, 190.662 s, 287 MB/s
107813230+0 records in
107813229+0 records out
55200373248 bytes (55 GB, 51 GiB) copied, 192.838 s, 286 MB/s
108222549+0 records in
```

Кожні 2 секунди бачимо виведення статистики. Експеримент виконано успішно!

Запустимо процес копіювання ще раз. Як і минулого разу бачимо, що виведення заблоковане. Подамо сигнал SIGTERM для зупинки (комбінація клавіш Ctrl+Z):

```
[root@oracle ~]# dd if=/dev/zero of=/dev/null  
^Z  
[1]+  Stopped                  dd if=/dev/zero of=/dev/null  
[root@oracle ~]#
```

Бачимо, що процесу призначився локальний номер 1. Також стоїть плюс, що означає, що це останній процес над яким виконувались маніпуляції.

Цей список можна вивести за допомогою команди jobs:

```
[root@oracle ~]# jobs  
[1]+  Stopped                  dd if=/dev/zero of=/dev/null  
[root@oracle ~]# _
```

Як бачимо, процес зупинений. Можна додати на введення ще декілька процесів:

```
[root@oracle ~]# dd if=/dev/zero of=/dev/null  
^Z  
[2]+  Stopped                  dd if=/dev/zero of=/dev/null  
[root@oracle ~]# dd if=/dev/zero of=/dev/null  
^Z  
[3]+  Stopped                  dd if=/dev/zero of=/dev/null  
[root@oracle ~]# jobs  
[1]  Stopped                  dd if=/dev/zero of=/dev/null  
[2]-  Stopped                  dd if=/dev/zero of=/dev/null  
[3]+  Stopped                  dd if=/dev/zero of=/dev/null  
[root@oracle ~]#
```

У такому випадку, щоб запустити на фоні процес використовується команда bg. Для того щоб запустити на передньому фоні, відповідно, fg:

```
[root@oracle ~]# jobs  
[1]  Stopped                  dd if=/dev/zero of=/dev/null  
[2]-  Stopped                  dd if=/dev/zero of=/dev/null  
[3]+  Stopped                  dd if=/dev/zero of=/dev/null  
[root@oracle ~]# bg 2  
[2]- dd if=/dev/zero of=/dev/null &  
[root@oracle ~]# jobs  
[1]-  Stopped                  dd if=/dev/zero of=/dev/null  
[2]   Running                  dd if=/dev/zero of=/dev/null &  
[3]+  Stopped                  dd if=/dev/zero of=/dev/null  
[root@oracle ~]#
```



Запускаємо другий процес на фоні. Як бачимо, в полі команди з'явився амперсанд – це означає, що процес виконується у фоновому режимі. Ці маніпуляції з посиланням сигналу SIGTERM та переведенням у фон можна не робити, а відразу запускати у фоновому режимі вказуючи той самий амперсанд:

```
[root@oracle ~]# jobs
[1]-  Stopped                  dd if=/dev/zero of=/dev/null
[3]+  Stopped                  dd if=/dev/zero of=/dev/null
[root@oracle ~]# dd if=/dev/zero of=/dev/null &
[4] 1630
[root@oracle ~]# jobs
[1]-  Stopped                  dd if=/dev/zero of=/dev/null
[3]+  Stopped                  dd if=/dev/zero of=/dev/null
[4]   Running                  dd if=/dev/zero of=/dev/null &
[root@oracle ~]# _
```

Перезапустимо усі 3 процеси у фоні:

```
[root@oracle ~]# jobs
[1]   Running                  dd if=/dev/zero of=/dev/null &
[2]-  Running                  dd if=/dev/zero of=/dev/null &
[3]+  Running                  dd if=/dev/zero of=/dev/null &
[root@oracle ~]# _
```

Та подивимось що відбувається з процесором за допомогою команди top:

```
top - 16:56:22 up 1:04, 2 users, load average: 0.67, 0.56, 0.95
Tasks: 108 total, 4 running, 104 sleeping, 0 stopped, 0 zombie
%Cpu(s): 32.5 us, 65.1 sy, 0.0 ni, 0.0 id, 0.0 wa, 2.4 hi, 0.0 si, 0.0 st
MiB Mem : 463.3 total, 166.4 free, 86.5 used, 210.4 buff/cache
MiB Swap: 2048.0 total, 1999.7 free, 48.2 used, 361.0 avail Mem
```

PID	USER	PR	NI	UIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
20525	root	20	0	7356	952	884	R	32.9	0.2	0:05.67	dd
20526	root	20	0	7356	1036	968	R	32.2	0.2	0:05.05	dd
20527	root	20	0	7356	944	876	R	32.2	0.2	0:04.73	dd
777	root	20	0	493200	4716	2616	S	0.3	1.0	0:12.66	tuned

Як бачимо, процесор був майже не зайнятий – тому на процеси dd можна виділити майже порівну по 33 відсотки. Як бачимо, у процесів пріоритет 20 – доволі низький. Якщо коефіцієнт піс буде від’ємний, то пріоритет підвищиться, якщо додатний – то навпаки. Перевіримо:

```
[root@oracle ~]# renice -n 10 20525
20525 (process ID) old priority -10, new priority 10
[root@oracle ~]# top_
```

Підвищуємо коефіцієнт до 10. Основний пріоритет повинен стати більшим на 10, тобто 30. Більший – менш важливий, 0 – найвищий.

Результат:

```
top - 17:01:50 up 1:09, 2 users, load average: 3.60, 2.54, 1.72
Tasks: 108 total, 5 running, 103 sleeping, 0 stopped, 0 zombie
%Cpu(s): 32.3 us, 63.3 sy, 1.3 ni, 0.0 id, 0.0 wa, 3.0 hi, 0.0 si, 0.0 st
MiB Mem : 463.3 total, 153.4 free, 87.6 used, 222.2 buff/cache
MiB Swap: 2048.0 total, 2000.0 free, 48.0 used. 359.9 avail Mem
```

PID	USER	PR	NI	UIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
20526	root	20	0	7356	1036	968	R	46.0	0.2	2:12.51	dd
20527	root	20	0	7356	944	876	R	45.7	0.2	2:12.18	dd
20525	root	30	10	7356	952	884	R	5.3	0.2	1:09.85	dd
20537	root	20	0	65452	4416	3760	R	0.7	0.9	0:00.32	top

Дійсно, процес з PID 20525 отримав пріоритет 30, тоюто нижче ніж 20. Тому замість розділення ресурсів 33-33-33 отримуємо 45-45-5.

Додамо ще один процес за допомогою команди пісе, коефіцієнт назначаємо 0, тобто основний пріоритет повинен бути 20:

```
[alex@oracle Lab6]$ nice -n 0 dd if="/dev/zero" of="/dev/null"
```

І дійсно, бачимо що створився ще один процес з пріоритетом 20. Ресурси розподілились рівномірно.

PID	USER	PR	NI	UIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
20550	alex	20	0	7356	1016	948	R	30.1	0.2	0:18.93	dd
20527	root	20	0	7356	944	876	R	29.8	0.2	3:48.62	dd
20526	root	20	0	7356	1036	968	R	29.5	0.2	3:48.95	dd
20525	root	30	10	7356	952	884	R	3.0	0.2	1:20.20	dd

Закриємо всі 4 процеси за допомогою команди killall – передамо їй коректний сигнал про завершення, тобто SIGTERM.

```
[alex@oracle Lab6]$ sudo killall -TERM dd
[alex@oracle Lab6]$ _
```

Перевіряємо:

```
top - 17:11:57 up 1:19, 1 user, load average: 1.80, 3.22, 2.63
Tasks: 105 total, 2 running, 103 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.3 us, 0.7 sy, 0.0 ni, 98.3 id, 0.0 wa, 0.7 hi, 0.0 si, 0.0 st
MiB Mem : 463.3 total, 124.4 free, 87.7 used, 251.2 buff/cache
MiB Swap: 2048.0 total, 2002.0 free, 46.0 used. 359.8 avail Mem
```

PID	USER	PR	NI	UIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
20537	root	20	0	65452	4416	3760	R	0.7	0.9	0:02.03	top
777	root	20	0	493200	4728	2616	S	0.3	1.0	0:14.61	tuned
1	root	20	0	90984	5956	3708	S	0.0	1.3	0:03.43	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp

Процесор не завантажений, процеси коректно вимкнулись (нема повідомлення “KILLED”). Експеримент завершено успішно! History команд не наведена, тому що історія зайняла б декілька листків.

## **Контрольні запитання:**

### **1) Що таке процес?**

Процес – це програма у стані виконання.

### **2) Які відмінності у процесу та програми?**

Програма повинна мати лише місце для зберігання коду, а процес повинен мати доступ до апаратних ресурсів [3]. Програма зберігається на жорсткому диску поки її не видалить користувач, а процес завершується після виконання програми. Окрім цього ще є дуже багато відмінностей.

### **3) Які вам відомі типи процесів?**

Системні (ядерні), демони (служби, не інтерактивні процеси) та користувацькі.

### **4) Що таке контекст процесу?**

Контекстом процесу є потік. Кожний процес має як мінімум один потік, але деякі можуть мати й декілька. Кожний потік діє в адресному просторі зовнішнього процесу, та має свій стек і доступ до реєстрів центрального процесора. Це потрібно для виконання багатьох дій одночасно, що економить час.

### **5) Що таке pid, ppid?**

PID – Process ID, унікальний номер процесу. PPID – унікальний номер батьківського процесу.

### **6) Як управляти пріоритетами процесу?**

За допомогою таких утиліт як nice та renice – тобто підвищувати або віднімати коефіцієнт «ввічливості», який напряму впливає на пріоритет процесу.

### **7) Які вам відомі команди для роботи з процесами та їх призначення?**

ps, top, htop – виведення інформації; bg, fg – переведення програми в фон або «на передній план», kill – обмін сигналами і так далі.

### **8) Що таке сигнали?**

Сигнали – це запити на переривання, що реалізуються на рівні процесів.

### **9) Як управляти роботою сигналів?**

За допомогою команд kill та killall можна робити запити сигналами.

### **10) Які вам відомі команди для роботи у фоновому режимі?**

fg (foreground), bg (background) є основними командами для переведення процесів у певний режим.

**Висновок:** за результатами виконання цієї лабораторної роботи було ознайомлено з такими поняттями як процес, сигнал, канал, та командами ps, top, pstree, bg, fg, nice, renice, kill, killall. Була виконана робота з опрацювання отриманих знань, були проведені певні експерименти результати яких виявились успішними і були описані у роботі.

## **Додаткові джерела:**

- 1) Немет, Эви, Снайдер – Unix и Linux: руководство системного администратора, 5-е изд.: Пер. с англ. - СПб. : ООО "Диалектика", 2020. - 1 168 с. : ил. - Парал. тит. англ.
- 2) PortaOne Education Centre – Processes lection
- 3) [Guru99.com – Program vs Process.](https://www.guru99.com/program-vs-process.html)