

Лабораторна робота № 6

Робота з процесами ОС Linux

Мета роботи:

- набуття навичок управління процесами в оболонці Bash;
- опанування команд *ps*, *top*, *pstree*, *bg*, *fg*, *nice*, *renice*, *kill*, *killall*.

Теоретичні відомості

Управління процесами в Linux

Процеси - це одна з найбільш фундаментальних абстракцій в системах UNIX після файлів. Від оптимального налаштування підсистеми управління процесами та числа одночасно виконуваних процесів залежить завантаження ресурсів процесора, що безпосередньо впливає на продуктивність системи в цілому. Задача ядра – управління процесами. Необхідно чітко розуміти відмінності між процесом і програмою.

Процес - це середовище виконання завдання (оточення), яке містить виконуваний код, системні дані, дані користувача і, а також набір додаткових ресурсів, отриманих під час виконання (ресурси пам'яті, можливість доступу до пристроїв введення/виведення та різних системних ресурсів, включаючи послуги ядра. *Програма* - це файл, який містить виконуваний код, дані для ініціалізації та дані користувача.

Процес можна розглядати як сукупність даних ядра системи (*Kernel*), необхідних для опису образу програми в пам'яті і управління її виконанням, або як програму в стадії її виконання, тому що усі програми Unix представлені у вигляді процесів. Ядро ОС обробляє переривання від пристроїв, виконує запити системних процесів та додатків користувача, розподіляє віртуальну пам'ять, створює і знищує процеси, забезпечує багатозадачність за допомогою перемикачів між ними, містить драйвери пристроїв, обслуговує файлову систему. Процес складається з інструкцій, які виконуються процесором, даних та інформації про виконуване завдання, а саме: виділена пам'ять, відкриті файли і статус процесу.

Типи процесів

Системні процеси в Unix завжди розташовані в оперативній пам'яті. 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, наприклад, *системний процес init*, який запускається ядром системи при завантаженні і є одним з ключових процесів для нормального функціонування системи. Приклади системних процесів системні - *vm Daemon*, *pagezero*, *bufdaemon*, *synchronizer*.

Демони – неінтерактивний процес, який працює у фоновому режимі і не прив'язаний 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, мережеслужб і т.п. Демони не пов'язані із жодним користувачем, тобто не мають ніякого відношення до користувацьких процесів. 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».

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.

Процеси взаємодіють між собою засобами міжпроцесної взаємодії (*Interprocess Communication - IPC*), а саме:

- *канали* (*pipe*, конвейери та іменовані канали *FIFO: First In First Out*),
- *сигнали* (це асинхронне повідомлення процесу про будь-яку подію. Коли сигнал посланий процесу, операційна система перериває його виконання. Якщо процес встановив власний обробник сигналу, операційна система запускає цей обробник, передавши йому інформацію про сигнал. Якщо процес не встановив обробник, то виконується оброблювач за замовчуванням),

- *сокети*.

Сигнали посилаються наступними засобами:

- *розділювана пам'ять* (це пам'ять, яка дозволяє здійснювати обмін інформацією не через ядро, а через певну частину віртуального адресного простору, в яку розміщують та зчитують дані).

- *черги повідомлень* (обмін повідомленнями здійснюється наступним чином: один процес поміщає повідомлення в чергу за допомогою деяких системних викликів, а будь-який інший процес може прочитати його звідти, за умови, що і процес-джерело повідомлення і процес-приймач повідомлення використовують один і той же ключ для отримання доступу до черги).

Сокети представляють собою *віртуальний об'єкт*, який існує, поки на нього посилається хоча б один з процесів. Сокети UNIX бувають 2х типів: локальні і мережеві. *Локальному сокету* присвоюється UNIX-адреса і буде створений спеціальний файл (файл сокета) по заданому шляху, через який зможуть повідомлятися будь-які локальні процеси шляхом простого читання/запису з нього. При використанні *мережевого сокета* створюється абстрактний об'єкт, прив'язаний до порту операційної системи, який слухає, та мережевого інтерфейсу. Цьому типу сокета присвоюється INET-адреса, яка має адресу інтерфейсу і порту, який слухає.

Процеси можуть виконуватися на передньому плані (*foreground*) - режим за замовчуванням і у фоновому режимі (*background*). На передньому плані в кожний момент для поточного термінала може виконуватися тільки один процес. Однак користувач може перейти в інший віртуальний термінал і запустити на виконання ще один процес, а на іншому терміналі ще один і т. д. Процес переднього плану - це процес, з яким ви взаємодієте, він отримує інформацію з клавіатури (стандартне введення) і посилає результати на ваш екран (стандартне виведення).

Фоновий процес після свого запуску завдяки використанню спеціальної команди командної оболонки відключається від клавіатури і екрану, тобто не очікує введення даних зі стандартного введення і не виводить інформацію на стандартне виведення, а

командна оболонка не очікує закінчення запущеного процесу, що дозволяє користувачеві негайно запустити ще один процес.

Зазвичай фонові процеси вимагають дуже великого часу для свого завершення і не потребують втручання користувача під час існування процесу. Наприклад, компіляцію програм або архівування великого обсягу інформації можна перевести у фоновий режим.

Для запуску програми в якості фонового процесу досить набрати в командному рядку ім'я програми і в кінці додати знак амперсанта (&), відокремлений пропуском від імені програми та її параметрів командного рядка, якщо такі є.

Наприклад, команда для запуску програми *yes* у фоновому режимі з подавленням виведення має вигляд:

```
qwe@vb:~$ yes "This is an example"
```

Команда *yes* використовується для відображення рядка декілька разів, поки вона не буде завершена за допомогою [Ctrl + C].

```
/home/user $ yes> /dev/null &
```

```
qwe@vb:~$ yes >/dev/null &  
[1] 3022
```

Після команди виводиться повідомлення, що складається з двох чисел. Перше число в дужках означає номер запущеного фонового процесу для користувача в поточному сеансі, з його допомогою можна проводити маніпуляції з цим фоновим процесом. Друге число показує ідентифікаційний номер (PID) процесу. Відмінності цих двох чисел достатньо суттєві. Номер фонового процесу унікальний тільки для користувача, що запускає цей фоновий процес. Тобто, якщо в системі три користувача вирішили запустити фоновий процес (перший для поточного сеансу), то в результаті у кожного користувача з'явиться фоновий процес з номером 1. Навпаки, ідентифікаційний номер процесу (PID) є унікальним для всієї операційної системи і однозначно ідентифікує в ній кожний процес.

Для перевірки стану фонових процесів можна скористатися командою командної оболонки - *jobs*.

```
/home/user$ jobs
```

```
[1] + Running          yes> / dev / null &
```

```
qwe@vb:~$ jobs
```

```
[1]+  Запущен          yes > /dev/null &
```

```
qwe@vb:~$
```

З вищенаведеного прикладу видно, що у користувача *user* в даний момент запущений один фоновий процес, і він виконується. Подавлення виведення команди здійснюється перенаправленням вихідного потоку на псевдопристрій */dev/null*. Все, що записується в цей файл, «зникає» назавжди.

Дві команди дуже корисні для перегляду працюючих в системі процесів, це *ps* (*process status*, показує моментальний знімок процесів в системі) і *top* (*table of processes*, дозволяє переглядати інформацію про процеси в реальному часі). Команда *ps* використовується для отримання списку запущених процесів і може показати їх PID, скільки пам'яті вони використовують, команду, якій вони були запущені і т.д. Команда *top* показує запущені процеси і оновлює екран кожні кілька секунд, що дозволяє спостерігати за роботою комп'ютера в реальному часі. Детальну інформацію про ці програми можна отримати на відповідних сторінках довідки (*man ps* і *man top*).

З точки зору ядра процес являє собою запис в **таблиці процесів**. Цей запис містить відомості про стан процесу і дані, що існують протягом усього часу його життя. Розмір таблиці процесів дозволяє запускати кілька сотень процесів одночасно. Процес також

використовує таблицю усіх відкритих процесом файлів, які зберігається в його адресному просторі. *Запис в таблиці процесів і простір процесу разом складають контекст, або оточення, процесу.*

Контекст кожного процесу містить:

- ***pid*** (*process ID* - ідентифікатор процесу), примусово призначається планувальником при запуску процесу;

- ***ppid*** (*parent process identifier*)) - ідентифікатор батьківського процесу, **UID** і **GID** - ідентифікатори прав процесу (**UID** - *user identifier*, ідентифікатор користувача і **GID** – *group identifier*, ідентифікатор групи процесів);

- ***tty*** - ім'я керуючого терміналу (термінал, з якого запущений процес);

- ***nice*** - пріоритет процесу або показник ввічливості;

- статус/стан процесу ***stat*** (***R=Running***, виконується; ***S=Sleeping*** – у стану очікування; ***D=Direct*** процес очікує певного сигналу виключно від апаратної частини; ***T=Tracing*** – процес знаходиться у режимі трювання/відладка програми; ***Z=Zombie***, у стані зомбі, тобто процес закінчився, але з деяких причин не звільнений з ядра; «<» підвищений пріоритет; «+» знаходиться в інтерактивному режимі);

- таблиця відкритих (використовуваних) файлів процесу;

- змінні оточення.

Перший процес, який ядро виконує під час запуску системи, називається *процесом ініціалізації (systemd)*. Його *pid* завжди 1, *ppid* – 0. Зазвичай *systemd process* в Linux є програмою ініціалізації.

Ядро Linux перебирає чотири виконуваних модуля в наступному порядку:

1. */sbin/init* - найбільш ймовірне розміщення процесу ініціалізації.

2. */etc/init* - наступне найбільш ймовірне розміщення процесу ініціалізації.

3. */bin/init* - резервне розміщення процесу ініціалізації.

4. */bin/sh* - місцезнаходження оболонки *Bourne* – оболонка Стіва Борна, яку ядро намагається запустити у випадку коли знайти процес ініціалізації не вдалося.

Після виконання ініціалізації системи запускаються різні сервіси та програми авторизації. Оскільки кожний процес належить певному користувачеві і групі, то ядро ОС Linux кожному процесу призначає числовий ідентифікатор (особистий номер) в діапазоні від 1 до 65535, тобто подвійне слово 2^{16} , *pid* та ідентифікатор батьківського процесу *ppid*. Усі процеси ОС Linux можна представити у вигляді дерева, в якому кореневим буде процес ініціалізації системи. Процеси, імена яких відображаються в квадратних дужках, наприклад, [keventd] - це процеси ядра. Такі процеси управляють складовими системи (менеджером пам'яті, планувальником часу процесора, менеджерами зовнішніх пристроїв тощо). Інші процеси це процеси користувача, які запущені або з командної оболонки (терміналу), або з графічної оболонки, або під час ініціалізації системи. Для отримання числового **UID** поточного користувача використовується наступна команда:

\$ id -u .

Можна також дізнатися **UID** будь-якого користувача системи:

\$ id -u USERNAMEUID.

Суперкористувача (з ім'ям *root* в переважній більшості випадків) завжди дорівнює 0:

\$ id -u root.

Для отримання імені користувача з числового **UID** застосовується бібліотечна функція *getpwuid()*, яка оголошена в заголовочному файлі ***pwd.h*** наступним чином:

struct passwd *getpwuid (uid_t UID);

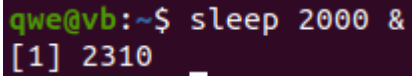
В структурі *passwd* нас цікавить тільки одне поле, яке містить ім'я користувача:

```
struct passwd
{
    /* ... */
    char *pw_name;
};
```

Якщо UID не відповідає жодному користувачеві системи, то *getpwuid()* повертає NULL.

Приклад виконання процесу у фоновому режимі (знак «&» вказує про відсутність зв'язку з терміналом, це означає, що непотрібно виконувати операції введення-

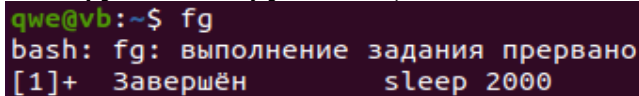
виведення):



Утиліта *sleep* виконує затримку на зазначений час, по закінченні цього часу завершується.

При створенні фоновому процесу *d* оболонці *bash* в квадратних дужках вказується номер завдання в системі ([1]) та його його *pid* (2310).

Для виведення процесу із фоновому режиму використовується команда *fg* і процесу надається функція керування (володіє потоками введення-виведення терміналу):



Моніторинг процесів

Життєвий цикл процесу. Для управління процесами в Linux використовується дві операції:

- створення нового процесу - виклик *fork()*,
- завершення поточного процесу, виклик *exit()* виконує основні кроки перед завершенням, а потім відправляє ядру команду припинити процес.

Системний виклик *fork()* - це операція, при якій процес копіює себе, і тим самим створює новий процес з унікальним ID, тобто запускає той же системний образ, що і поточний:

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

У разі успішного звернення до *fork()* створюється новий процес, в усіх відношеннях ідентичний викликаючому. Обидва процеси виконуються від точки звернення до *fork()*, як ніби нічого не відбувалося.

Новий процес є дочірнім по відношенню до викликаючого, який, в свою чергу, називається батьківським. У дочірньому процесі успішний запуск *fork()* повертає 0. В батьківському процесі *fork()* повертає *pid* дочірнього. Батьківський і дочірній процеси практично ідентичні, за винятком деяких особливостей:

- *pid* дочірнього процесу, призначається заново і відрізняється від батьківського;
- батьківський *pid* дочірнього процесу встановлений рівним *pid* батьківського процесу;
- ресурсна статистика дочірнього процесу обнуляється;
- будь-які очікуючі сигнали перериваються і не успадковуються дочірнім процесом.

Стандартний виклик завершення поточного процесу *exit()* наступний:

```
#include <stdlib.h>
```

```
void exit(int status);
```

Виклик *exit()* виконує основні кроки перед завершенням, а потім відправляє ядру команду припинити процес. Ця функція не повертає ніяких результатів.

Параметр *status* використовується для позначення статусу процесу завершення. Інші програми - як і користувач оболонки - можуть перевіряти цю величину. Зокрема, статус *status & 0377* повертається батьківському процесу.

Є інша можливість отримати інформацію про процес, а саме про його дані - ідентифікатор процесу (PID); ідентифікатор батьківського процесу (PPID); ідентифікатор користувача (UID).

Отримати PID, PPID і UID поточного процесу можна за допомогою наступних системних викликів, оголошених в заголовки *unistd.h*: *pid_t getpid (void); pid_t getppid (void); uid_t getuid (void);* Типи даних *pid_t* та *uid_t* є цілими числами, розмірність яких залежить від реалізації. Щоб використовувати ці типи, потрібно включити в програму заголовочний файл *sys/types.h*. Системний виклик *getpid()* повертає ідентифікатор поточного процесу, *getppid()* - батьківського, а *getuid()* - ідентифікатор користувача, від імені якого виконується процес.

Для виведення списку усіх процесів в в поточній оболонці використовується команда **ps (Get-Process):** *ps [PID] options.*

Команда *ps* має три типи стилів представлення опцій:

- стиль Unix98: *ps [-опції]* (використовується дефіс);
- в стилі BSD (*Berkeley Software Distribution*): *ps [опції]* (без дефіса);
- в стилі GNU-версії: *ps [-- довге ім'я опції [--довге ім'я опції] ...]* (використовується два дефіса).

Команда *ps* без будь-яких аргументів відображає процеси для поточної оболонки.

```
qwe@vb:~$ ps
  PID TTY          TIME CMD
 2251 pts/1        00:00:00 bash
 2829 pts/1        00:00:00 ps
```

Виводиться *PID* (ідентифікатор процесу), *TTY* (ідентифікатор терміналу), *TIME* (час ЦП), *CMD* (ім'я команди).

Виведення усіх процесів в різних форматах

Кожний активний процес в системі Linux в загальному форматі (Unix/Linux) відображається з опціями «-A» або «-e»: *ps -A* або *ps -e*:

```
qwe@vb:~$ ps -A
  PID TTY          TIME CMD
    1 ?            00:00:08 systemd
    2 ?            00:00:00 kthreadd
    3 ?            00:00:00 rcu_gp
    4 ?            00:00:00 rcu_par_gp
    6 ?            00:00:00 kworker/0:0H-kblockd
    8 ?            00:00:00 mm_percpu_wq
    9 ?            00:00:00 ksoftirqd/0
   10 ?            00:00:03 rcu_sched
   11 ?            00:00:00 migration/0
   12 ?            00:00:00 idle_inject/0
   13 ?            00:00:05 kworker/0:1-events
   14 ?            00:00:00 cpuhp/0
   15 ?            00:00:00 cpuhp/1
```

Довідка про опції команди виводиться як *man ps*.

Відображення усіх процесів у форматі BSD (розширене виведення параметрів): **ps au** або **ps aux**

```
qwe@vb:~$ ps au
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
qwe       1047  0.0  0.3 174068  6332 tty2    Ssl+ 17:40   0:00 /usr/lib/gdm3/g
qwe       1049  0.0  2.6 373164 54680 tty2    Sl+  17:40   0:10 /usr/lib/xorg/X
qwe       1081  0.0  0.7 584408 16248 tty2    Sl+  17:40   0:00 /usr/lib/gnome-
qwe       1373  0.8 15.1 2837208 308508 tty2    Sl+  17:41   1:32 /usr/bin/gnome-
qwe       1396  0.0  0.3 319104  8008 tty2    Sl  17:41   0:00 ibus-daemon --x
qwe       1401  0.0  0.3 244736  6860 tty2    Sl  17:41   0:00 /usr/lib/ibus/i
qwe       1405  0.0  1.3 296584 26720 tty2    Sl  17:41   0:00 /usr/lib/ibus/i
qwe       1409  0.0  1.1 219644 24032 tty2    Sl  17:41   0:00 /usr/lib/ibus/i
qwe       1521  0.0  0.4 323228  9848 tty2    Sl+  17:41   0:00 /usr/lib/gnome-
```

Відображення запущених користувацьких процесів: **ps -x**

```
qwe@vb:~$ ps -x
  PID TTY          STAT TIME  COMMAND
   950 ?            Ss   0:00 /lib/systemd/systemd --user
   951 ?            S    0:00 (sd-pam)
  1042 ?           SLL   0:00 /usr/bin/gnome-keyring-daemon --daemonize --login
  1047 tty2        Ssl+  0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SH
  1049 tty2        Sl+   0:11 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1
  1077 ?           Ss   0:00 /usr/bin/dbus-daemon --session --address=systemd: --n
  1081 tty2        Sl+   0:00 /usr/lib/gnome-session/gnome-session-binary --session
  1340 ?           Ss   0:00 /usr/bin/ssh-agent /usr/bin/im-launch env GNOME_SHELL
  1344 ?           Ssl   0:00 /usr/lib/at-spi2-core/at-spi-bus-launcher
  1349 ?           S    0:00 /usr/bin/dbus-daemon --config-file=/usr/share/default
  1351 ?           Sl    0:00 /usr/lib/at-spi2-core/at-spi2-registryd --use-gnome-s
  1373 tty2        Sl+   1:39 /usr/bin/gnome-shell
```

Команда **ps -l** у форматі BSD виводить розширений лістинг.

```
qwe@vb:~$ ps -l
 F S  UID      PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
 0 S  1000    2777    2764  0  80   0 - 4775 do_wai pts/0    00:00:00 bash
 0 R  1000    2833    2777  0  80   0 - 5001 -      pts/0    00:00:00 ps
```

Для виведення усіх процесів, які виконуються з правами користувача root (реальний і ефективний ідентифікатор, Real і Effective ID), використовується команда **ps -U root -u root**

```
qwe@vb:~$ ps -U root -u root
  PID TTY          STAT TIME  CMD
    1 ?            00:00:10 systemd
    2 ?            00:00:00 kthreadd
    3 ?            00:00:00 rcu_gp
    4 ?            00:00:00 rcu_par_gp
    6 ?            00:00:00 kworker/0:0H-kblockd
    8 ?            00:00:00 mm_percpu_wq
    9 ?            00:00:00 ksoftirqd/0
   10 ?           00:00:05 rcu_sched
   11 ?           00:00:00 migration/0
   12 ?           00:00:00 idle_inject/0
   13 ?           00:00:07 kworker/0:1-events
   14 ?           00:00:00 cpuhp/0
   15 ?           00:00:00 cpuhp/1
```

Відобразити процес за його **pid**: **ps -fp 1351**

```
qwe@vb:~$ ps -fp 1351
  UID      PID  PPID  C STIME TTY          TIME CMD
qwe       1351   950  0 17:41 ?          00:00:00 /usr/lib/at-spi2-core/at-spi2-r
```

Відобразити процес за його **ppid**: **ps -f --ppid 950**

```
qwe@vb:~$ ps -f --ppid 950
  UID      PID  PPID  C STIME TTY          TIME CMD
qwe       951   950  0 17:40 ?          00:00:00 (sd-pam)
qwe      1077   950  0 17:40 ?          00:00:00 /usr/bin/dbus-daemon --session
qwe      1344   950  0 17:41 ?          00:00:00 /usr/lib/at-spi2-core/at-spi-bu
qwe      1351   950  0 17:41 ?          00:00:00 /usr/lib/at-spi2-core/at-spi2-r
qwe      1382   950  0 17:41 ?          00:00:00 /usr/lib/gvfs/gvfsd
qwe      1387   950  0 17:41 ?          00:00:00 /usr/lib/gvfs/gvfsd-fuse /run/u
qwe      1397   950  0 17:41 ?          00:00:00 /usr/libexec/xdg-permission-sto
qwe      1412   950  0 17:41 ?          00:00:00 /usr/lib/ibus/ibus-portal
qwe      1423   950  0 17:41 ?          00:00:00 /usr/lib/gnome-shell/gnome-shel
qwe      1429   950  0 17:41 ?          00:00:00 /usr/libexec/evolution-source-r
```

Вивести інформацію про усі процеси **ps -e**

```
qwe@vb:~$ ps -e
  PID TTY          TIME CMD
    1 ?        00:00:12 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 rcu_gp
    4 ?        00:00:00 rcu_par_gp
    6 ?        00:00:00 kworker/0:0H-kblockd
    8 ?        00:00:00 mm_percpu_wq
    9 ?        00:00:00 ksoftirqd/0
   10 ?        00:00:06 rcu_sched
   11 ?        00:00:00 migration/0
   12 ?        00:00:00 idle_inject/0
   13 ?        00:00:08 kworker/0:1-cgroup_destroy
   14 ?        00:00:00 cpuhp/0
   15 ?        00:00:00 cpuhp/1
   16 ?        00:00:00 idle_inject/1
```

Вивести дерево процесів **ps -f --forest** або **pstree** або **pstree <ім'я користувача>**

```
qwe@vb:~$ ps -f --forest
UID          PID  PPID  C  STIME TTY          TIME CMD
qwe           2251 2240  0 17:43 pts/1      00:00:00 bash
qwe           3787 2251  0 21:29 pts/1      00:00:00 \_ ps -f --forest

qwe@vb:~$ pstree
systemd───ModemManager───2*[{ModemManager}]
          └─NetworkManager───dhclient
                                └─2*[{NetworkManager}]
accounts-daemon───2*[{accounts-daemon}]
acpid
avahi-daemon───avahi-daemon
boltd───2*[{boltd}]
colord───2*[{colord}]
cron
cups-browsed───2*[{cups-browsed}]
cupsd
dbus-daemon
fprintd───{fprintd}
fwupd───4*[{fwupd}]
gdm3───gdm-session-wor───gdm-x-session───Xorg───3*[{Xorg}]
                                     └─gnome-session-b───evolution-+
                                                         gnome-shel+
                                                         gnome-soft+
                                                         gsd-a11y-s+
                                                         gsd-clipbo+
                                                         gsd-color+
```

Команда top

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.

```
qwe@vb:~$ top

top - 22:03:01 up 4:23, 1 user, load average: 0,33, 0,14, 0,09
Tasks: 198 total, 1 running, 197 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3,1 us, 2,2 sy, 0,0 ni, 94,7 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
MiB Mem : 1990,8 total, 275,7 free, 842,2 used, 872,8 buff/cache
MiB Swap: 947,2 total, 930,0 free, 17,3 used. 976,6 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
 1373 qwe        20   0 2837736 302844 101860 S   6,0  14,9  3:29.73 gnome-sh+
   415 root        20   0  29448  13392   9188 S   1,7   0,7  4:59.30 plymouthd
 1049 qwe        20   0 373520  46340  34664 S   1,3   2,3  0:34.12 Xorg
 1070 root        20   0 381564  42272  23780 S   0,7   2,1  0:47.73 containe+
   350 root        20   0  19252   5176   2924 S   0,3   0,3  0:05.91 systemd-+
   604 root        20   0 246352   7140   6340 S   0,3   0,4  0:00.60 accounts+
   751 root        20   0 504740  72864  40028 S   0,3   3,6  1:03.65 dockerd
 3158 root        20   0      0      0      0 I   0,3   0,0  0:01.20 kworker/+
 3375 root        20   0      0      0      0 I   0,3   0,0  0:02.65 kworker/+
 3865 qwe        20   0  20456   3708   3272 R   0,3   0,2  0:00.10 top
      1 root        20   0 165284  10392   7740 S   0,0   0,5  0:13.75 systemd
      2 root        20   0      0      0      0 S   0,0   0,0  0:00.00 kthreadd
```

Перший рядок (**top**) – загальна інформація:

- поточний час (22:03:01),
- час роботи системи (up 4 day, 4:23),
- кількість відкритих сесій користувачів (1 user)

- середнє завантаження системи (*load average*: 0,33, 0,14, 0,09), три значення відповідають завантаженню в останню хвилину, п'ять хвилин і п'ятнадцять хвилин відповідно.

Другий рядок (task) – статистика процесів:

- загальна кількість процесів в системі (198 *total*),
- кількість працюючих в даний момент процесів (1 *running*),
- кількість очікуючих подій процесів (197 *sleeping*),
- кількість зупинених процесів (0 *stopped*),
- кількість процесів, які очікують батьківський процес для передачі статусу завершення (0 *zombie*).

Третій рядок (%Cpu(s)) - статистика використання центрального процесора (*cpu*):

- відсоток використання центрального процесора для користувача процесів (3,1% *us*),
- відсоток використання центрального процесора системними процесами (2,2% *sy*),
- відсоток використання центрального процесора процесами з пріоритетом, підвищеним за допомогою виклику *nice* (0,0% *ni*),
- відсоток часу, коли центральний процесор не використовується (94,7% *id*),
- відсоток використання центрального процесора процесами, які очікували завершення операцій введення-виведення (0,0% *wa*),
- відсоток використання центрального процесора обробниками апаратних переривань (0,0% *hi* - Hardware IRQ (апаратні переривання)),
- відсоток використання центрального процесора обробниками програмних переривань (0,0% *si* - Software Interrupts (програмні переривання)),
- кількість ресурсів центрального процесора "запозичених" у віртуальній машині гіпервізором для інших завдань (таких, як запуск іншої віртуальної машини); це значення дорівнюватиме нулю на настільних комп'ютерах і серверах, які не використовують віртуальні машини (0,0% *st* - Steal Time (запозичений час)).

Підсумовування усіх значень повинно дорівнювати 100%.

Четвертий і п'ятий рядки - статистика використання пам'яті (*memory usage*):

У четвертому і п'ятому рядках виводиться інформація про використання фізичної оперативної пам'яті і розділу підкачки відповідно. Значення в порядку проходження: загальна кількість пам'яті (*total*), кількість використовуваної пам'яті (*used*), кількість вільної пам'яті (*free*), кількість пам'яті в кеші буферів (*buffers*).

Наступні рядки - список процесів:

PID - ідентифікатор процесу,

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

PR - пріоритет процесу,

NI - значення "NICE", яке впливає на пріоритет процесу,

VIRT - обсяг віртуальної пам'яті, яка використовується процесом,

RES - обсяг фізичної пам'яті, який використовується процесом,

SHR - обсяг розділюваної пам'яті процесора,

S - вказує на статус процесу: *S=sleep* (очікує подій), *R=running* (виконується),

Z=zombie (очікує батьківський процес) (*S*),

%CPU - відсоток використання центрального процесора даним процесом,

%MEM - відсоток використання оперативної пам'яті даним процесом,

TIME+ - загальний час активності процесу,
COMMAND - ім'я процесу.

Пріоритети процесів

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» (*nice*ness). 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.

Linux надає декілька викликів для отримання і установки значення ввічливості процесу. Найпростіший з них - *nice()*:

```
#include <unistd.h>
int nice(int inc);
```

Успішна робота *nice()* збільшує значення ввічливості процесу на значення *inc* і повертає оновлену величину. Тільки процес з характеристикою *CAP_SYS_NICE* (яким фактично володіє користувач *root*) може встановити негативну величину *inc*, зменшуючи його значення ввічливості і, таким чином, підвищуючи пріоритет процесу. Процеси без прав *root* можуть тільки знижувати свої пріоритети (збільшуючи значення ввічливості).

Кращим рішенням є застосування системних викликів *getpriority()* і *setpriority()*, які надають більше можливостей для управління і контролю, але складніші у використанні:

```
#include <sys/time.h>
#include <sys/resource.h>
int getpriority(int which, int who);
int setpriority(int which, int who, int prio);
```

Ці виклики впливають на процес, групу процесів або користувачів, що визначено за допомогою *which* і *who*. Значення *which* повинно бути *PRIO_PROCESS*, *PRIO_PGRP* або *PRIO_USER*, в той час як *who* вказує ідентифікатор процесу, групи процесів або ідентифікатор користувача відповідно. Якщо значення *who* дорівнює 0, то виклик працює з поточними ідентифікатором процесу, групи процесів або ідентифікатором користувача відповідно.

Виклик *getpriority()* повертає найбільший пріоритет (найменшу чисельну величину значення люб'язності) кожного із зазначених процесів. Виклик *setpriority()* встановлює значення пріоритету кожного з зазначених процесів, яке дорівнює *prio*. Як і *nice()*, тільки процес з властивістю *CAP_SYS_NICE* може збільшити пріоритет процесу (знизити чисельне значення люб'язності). Отже, тільки процес з цією властивістю може збільшити або зменшити пріоритет процесу, що не належить користувачу, який викликає.

Використання команди **nice**: **nice [-n <число>]<команда>**. Команда вказує пріоритет, з яким процес буде виконуватися після запуску. Від'ємне значення пріоритету має право вказувати виключно суперкористувач (root). Якщо ключ «-n» не був введений в команді, то використовується число 10.

```
sudo nice -10 sleep 1000
```

```
qwe@vb:~$ sudo nice -n 10 sleep 1000
```

Для зміни пріоритету процесу використовується команда **renice -n <число> <список ідентифікаторів процесів>**

Команда **renice** працює аналогічно **nice**, з тією різницею, що змінюється не пріоритет створюваного процесу, а пріоритети усіх запущених процесів, ідентифікатори яких входять у список, розділений пробілами та переданий команді в якості параметра. Число у складі ключа «-n» додається до поточного пріоритету процесу, що пришвидчує виконання процесу. Така зміна виконується виключно від імені адміністратора (root): **sudo renice -n -10 PID**.

Управління сигналами

Одним із способів взаємодії між процесами в Linux є сигнали; сигнали - це програмні переривання, які можуть бути послані процесу за допомогою системного виклику. Кожному виду сигналів в системі відповідає назва і номер. Багато сигнали мають спеціальний сенс. Зокрема, сигнал **SIGTERM** повідомляє процес про необхідність завершення, сигнал **SIGTERM** використовується для безумовного завершення процесу, сигнали **SIGSTOP** і **SIGCONT**, відповідно, зупиняють і відновлюють виконання процесу. Сигнали **SIGKILL** (примусове завершення процесу) і **SIGSTOP** неможливо ні перехопити, ні ігнорувати, тому що вони адресовані не процесу, а планувальнику ОС.

Для того щоб послати сигнал процесу з програми-оболонки:

kill [-s <назва сигналу>] <список ідентифікаторів процесів>

Для виведення усіх сигналів, які підтримує ця система, використовується ключ «-l»: **kill -l**

```
qwe@vb:~$ 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
```

Для зупинення виконання певного процесу, наприклад, з ідентифікатором 123, треба скористатися командою **kill -s SIGSTOP 123**. Для його відновлення використовуємо команду **kill -s SIGCONT 123**.

Для завершення процесу з ідентифікатором 123 використовуємо команду **kill 123**.

Найбільш вживані сигнали в Linux:

SIGINT (№ 2) - користувач натиснув Ctrl+C,

SIGKILL (№ 9) – припинення виконання процесу,

SIGTERM (№ 15) – попередження, що процес незабаром буде знищений,

SIGCONT (№ 18) - продовження призупиненого процесу,
SIGSTOP (№ 19) – припинення виконання процесу,
SIGTSTP (№ 20) - натискання клавіш<CTRL>+<Z>, сигналу викликає зупинку виконання процесу.

Команда **killall -s SIGNAL процес** надсилає сигнал всім процесам з іменем **процес**. Якщо сигнал не вказаний, надсилається SIGTERM.

Команди оболонки для роботи у фоновому режимі: jobs, bg (background) та fg (foreground) Команда **jobs** виводить список процесів, які виконуються у фоновому режимі. Зі знаком «+» відображається процес, з якими працювали останнім, «-» - процес, з яким працювали передостаннім.

fg <номер_завдання> - переводить процес на передній план,

bg <номер_завдання> - переводить процес на задній план.

Номер завдання - це не PID, а число, яке команда **jobs** виводить в квадратних дужках. З ключем **-I** вона буде виводити, крім того, і PID процесу.

Оскільки перелічені команди - не самостійна утиліти, а підкоманди **bash**, довідку по ним потрібно запитувати так: **help <підкоманда>**.

Завдання:

1. Ознайомитися з теоретичними матеріалом по лабораторній роботі. Набути навичок по роботі з процесами.
2. Опанувати команди для управління процесами.
3. Підготувати звіт для викладача про виконання лабораторної роботи і представити його.

Хід виконання роботи

1. Ознайомтеся з роботою команд по управлінню процесами.
2. Вивести на екран лістинг характеристик (у довгому і короткому форматах) процесів, ініціалізованих з вашого терміналу. Записати їх у файл. Проаналізувати і пояснити вміст кожного поля повідомлення.
3. Вивести всю ієрархію процесів поточної оболонки разом полями *pid* та *ppid*.
4. Побудувати дерево процесів, які визначені у попередньому пункті. Результат виконання вивести на екран і дописати в файл.
5. Переглянути список процесів вашого користувача.
6. Вивести список процесів вашого користувача у вигляді дерева (команда *ps tree*).
7. Переглянути список сигналів вашого користувача. Записати у окремий файл
8. За допомогою команди *history* виведіть команди, які ви використовували.

Підготувати звіт

1. Описати хід виконання поставлених завдань, надаючи знімок екрану (screenshot).
2. Висновки по роботі.

Контрольні питання

1. Що таке процес?
2. Які відмінності процесу та програми?
3. Які вам відомі типи процесів?

4. Що таке контекст процесу?
5. Що таке *pid*, *ppid*?
6. Як управляти пріоритетами процесу?
7. Які вам відомі команди для роботи з процесами та їх призначення?
8. Що таке сигнали?
9. Як управляти роботою сигналів?
10. Які вам відомі команди для роботи у фоновому режимі?

Література

1. Уорд Б. Внутреннее устройство Linux. Санкт-Петербург : Питер, 2016. 384 с.
2. Негус К., Казн Ф. Ubuntu и Debian Linux для продвинутых: более 1000 незаменимых команд. Санкт-Петербург : Питер, 2014. 384 с.
3. Немет Эви, Гарт Снайдер, Трент Хейн, Бэн Уэйли. Unix и Linux: руководство системного администратора, 4-е изд. : Пер. с англ. Москва : ООО “И.Д. Вильямс”, 2012. 1312 с.
4. Колисниченко Д. Linux от новичка к профессионалу. Санкт-Петербург : БХВ-Питер, 2016. 672 с.