Міністерство освіти і науки України  
Національний технічний університет України

«Київський політехнічний інститут ім. І. Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

Лабораторна робота №6

з дисципліни «Операційна система UNIX»

«Робота з процесами ОС Linux»

Виконав:

студент 2-го курсу, НН ІАТЕ

групи ТР-23

Ровний Григорій Олександрович

Перевірила:

проф. Левченко Л.О.

КИЇВ 2023

**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 - одна з найбільш фундаментальних абстракцій в UNIX післяфайлів. Від оптимального налаштування підсистеми управління процесами та числаодночасно виконуваних процесів залежить завантаження ресурсів процесора, що впливає на продуктивність системи в цілому. Задача ядра – управлінняпроцесами.

Процес - це середовище виконання завдання, яке містить виконуваний код, системні дані, дані користувача і, а також набір додаткових ресурсів, отриманих під час виконання (ресурси пам'яті, доступ до пристроїв введення/виведення 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 та 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, містить драйвера, обслуговує файлову систему. Процес складається з інструкцій, які виконуються процесором.

**Типи процесів.** Системні процеси в Unix завжди розташовані в оперативній пам'яті. Їх виконуваніінструкції та дані знаходяться в ядрі, тому такі процеси є складовою ядра. Системніпроцеси можуть викликати функції, а також звертатися до даних, які не мають доступудо інших процесів.

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. Демони 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 НТТР та FТР, сервер системного журналу (syslogd). Зазвичай демони в кінці назви містять літеру «d».

Усі інші процеси, які виконуються в системі, вважаються прикладними. 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. 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 (IPC):

- канали (pipe, конвейєри та іменовані канали FIFO: First In First Out),

- сигнали (це асинхронне повідомлення процесу про 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.

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 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 операційно системи та мережевого інтерфейсу. Цьому типу сокета присвоюється INET-адреса, яка має адресу інтерфейсу і порту.

Процеси можуть виконуватися на передньому плані (режим за замовчуванням) і у фоновому режимі. 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. Фоновий процес після свого запуску завдяки використанню спеціальної команди командної оболонки відключається від клавіатури і екрану. 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.

**Моніторинг процесів.** Життєвий цикл процесу. Для управління процесами в Linux використовується двіоперації:

- 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. 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.

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 результатів. Параметр status використовується для позначення статусу процесу завершення. Інші програми - як і користувач оболонки - можуть перевіряти цю величину.

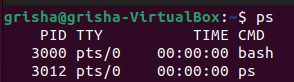
Є інша можливість отримати інформацію про процес, а саме про його дані - ідентифікатор процесу (PID); ідентифікатор батьківського процесу (PPID); ідентифікатор користувача (UID). Отримати PID, PPID і UID поточного процесу можна за допомогою системних викликів, оголошених в заголовки unistd.h: pid\_t getpid (void); pid\_t getppid (void); uid\_t getuid (void); 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.

**Поставлене завдання:**

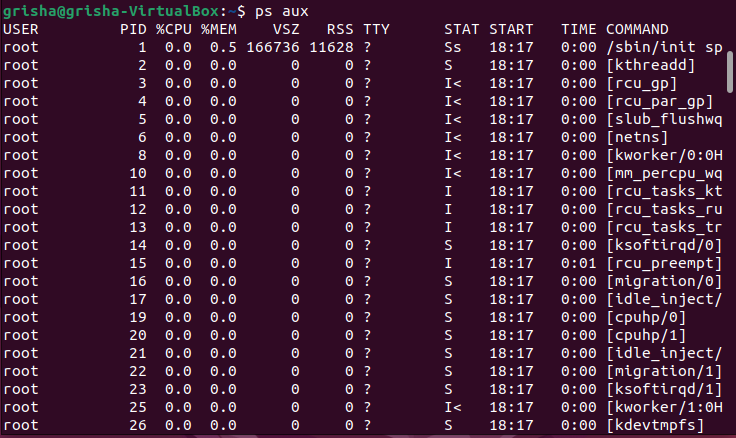
1. Ознайомтесь з роботою команд по управлінню процесами. Вивести на екран лістинг характеристик (у довгому і короткому форматах) процесів, ініціалізованих з вашого терміналу. Записати їх у файл. Проаналізувати і пояснити вміст кожного поля повідомлення.
2. Вивести всю ієрархію процесів поточної оболонки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

**Результат виконання роботи**

1. Для виведення списку процесів, ініційованого з терміналу, необхідно в терміналі виконати команду ps, в результаті чого отримуємо список процесів у короткому форматі, де PID (Process ID) – ідентифікатор процесу (у моєму випадку є лише два процеси – для оболонки bash та для команди ps), TTY – ідентифікатор терміналу (вказує з яким терміналом пов’язаний процес), TIME – час використання процесорного часу ЦП , CMD – команда, яка ініціювала процес.

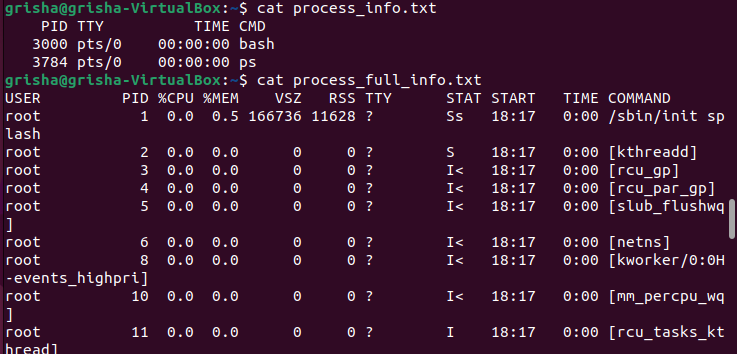


Виконаємо команду *ps aux* (де a – ключ для виведення процесів всіх користувачів, u – формат виведення, х – процеси не прив’язані до терміналу) для отримання розширеного виведення параметрів (BSD). У результаті чого отримуємо інформацію з такими полями: USER - ім'я користувача, якому належить процес, %CPU - відсоток використаного процесорного часу, %MEM - відсоток використаного об'єму оперативної пам'яті, VSZ - віртуальний розмір(в кілобайтах), RSS - доступний розмір (в кілобайтах), STAT - статус процесу : S – sleeping (очікування), R – running (виконується), START - час запуску процесу.

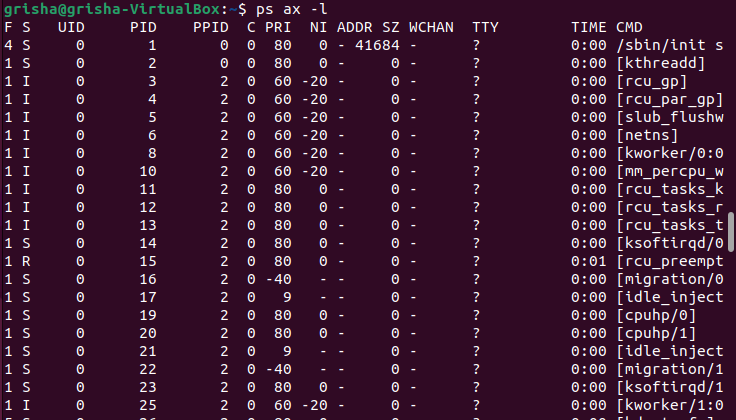


Запишемо результати виконання команд *ps* і *ps aux* у файли *process\_info.txt* та *process\_full\_info.txt* за допомогою оператора *>*, та перевіримо результат запису, зчитавши файл за допомогою команди *cat filename.txt.*

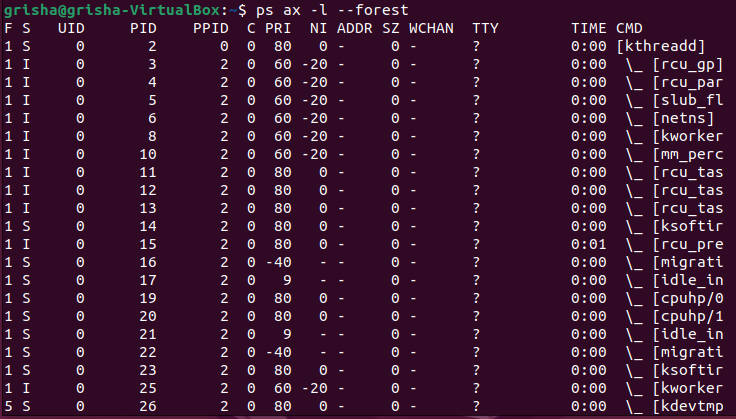


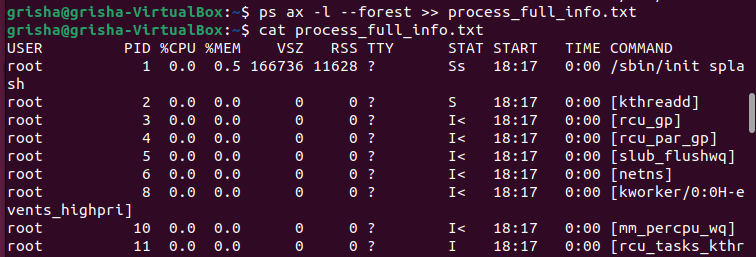


1. Щоб отримати всю ієрархію процесів, включаючи поля PID та PPID, введемо команду *ps ax –l.* У результаті виконання отримуємо розширений лістинг, з деякими додатковими полями: F – прапорець, що вказує на тип процесу (0 – стандартний процес), S – стан процесу: S – sleeping (очікування), R – running (виконується), UID(UserID) – айді користувача якому належить процес, PPID – ідентифікатор батьківського процесу, С – к-сть ядер, які використовує процесор, PRI і NI –пріоритети процесу, ADDR – адреса в пам’яті процесу, SIZE – к-сть використаної пам’яті, WCHAN – канал очікування.

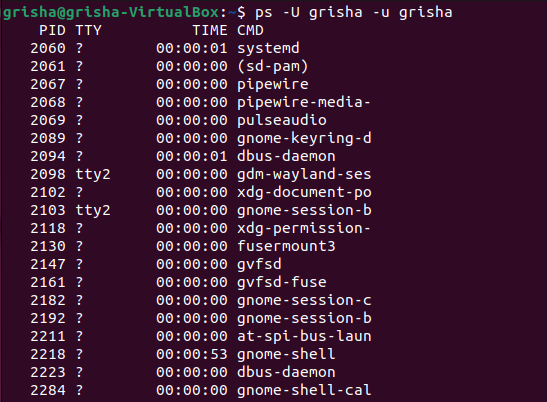


1. Спробуємо вивести даний список у вигляді дерева. Для цього існує команда *ps ax – l --forest.* Доповнимо файл process\_full\_info.txt отриманим результатом (*ps ax –l –forest >> process\_full\_info.txt*).

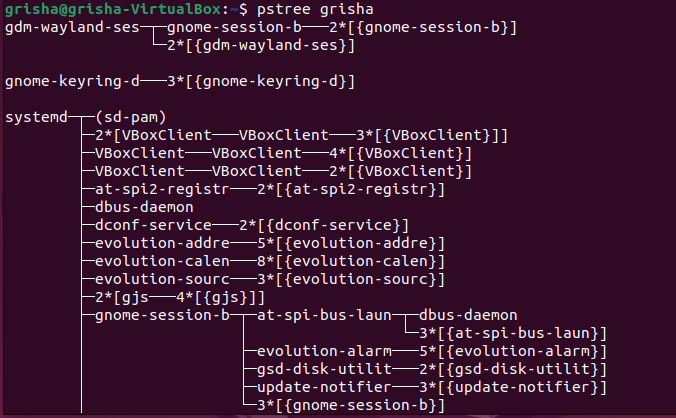




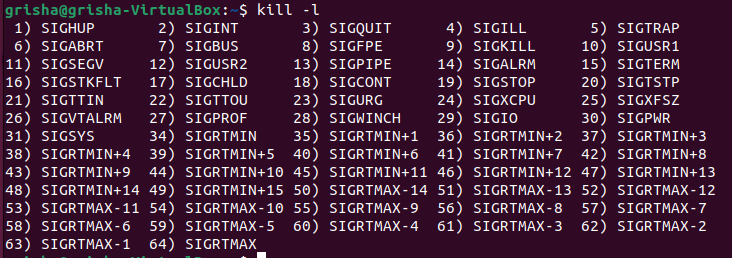
1. Для того, щоб переглянути список процесів користувача grisha, необхідно скористатися командою *ps –U grisha –u grisha*.

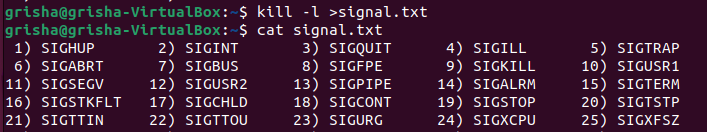


1. Виведемо список процесів користувача grisha у вигляді дерева. Спеціально для цього існує команда *pstree grisha*.

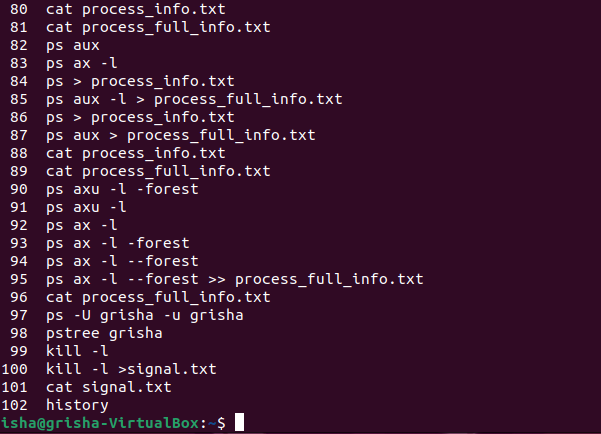


1. Переглянемо та запишемо у файл signal.txt список доступних сигналів користувача (kill -l). В результаті отримуємо інформацію про різні типи сигналів, які можна відправити процесам для керування їх роботою.





1. На завершення вводимо history для виведення списку використаних команд



**Висновок:**

У результаті виконання лабораторної роботи було набуто практичні навички з управління процесами оболонки Bash; розглянуто такі команди як: ps, pstree, kill з різними ключами, необхідні для керування процесами та моніторингу системи.