



Тема: «Основи операційного керування процесами в *Unix*-подібних ОС»

Викладач: Олександр А. Блажко,
доцент кафедри ІС Одеської політехніки, blazhko@ieee.org

Мета роботи: отримання навичок в операційному керуванні процесами в *Unix*-подібних ОС засобами команд командної оболонки *Bash*.

План.

1 Теоретичні відомості

2 Завдання

1 Теоретичні відомості

1.1 Операційна трійка «Програма-Процесор-Процес»

1.1.1 Основні типи багатозадачності

Компонентна багатозадачність: декілька програм одночасно використовують різні апаратні компоненти (пристрої вводу/виводу, процесор), але лише одна програма використовує процесор

Інтерактивні програми активно використовують пристрої вводу/виводу при від час взаємодії з користувачем

Невитісняюча багатозадачність (*Non-preemptive multitasking*, *не упереджуюча, не превентивна багатозадачність*): кожна програма за бажанням користувача запускається на процесорі та звільняє його за власним бажанням або за бажанням користувача.

Витісняюча багатозадачність (*Preemptive multitasking*, *упереджуюча, превентивна багатозадачність*): кожна програма виконується на процесорі лише обмежений час, ОС слідкує за часом та періодично перериває роботу програми, надаючи процесор іншій програмі.

Термін "*процес*" вперше з'явився при розробці ОС *Multix* і має кілька визначень, які використовуються в залежності від контексту:

- це програма на стадії виконання;
- це "об'єкт", якому виділено процесорний час (рисунок 1).

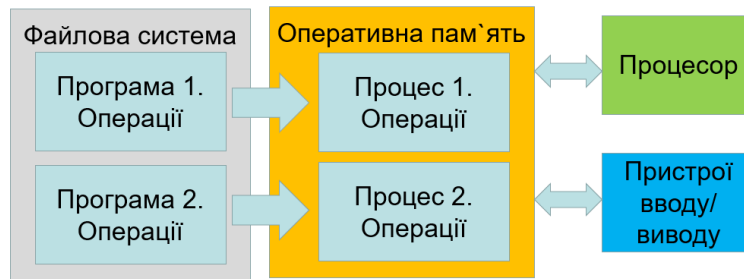


Рис. 1 – Файлова система з файлами-програмами, процесор та процес

1.1.2 Оцінка завантаженості процесора режиму витісняючої багатозадачності

Відомо, що режим витісняючої багатозадачності базується на розподілі часу (*Time Sharing*) між різними процесами, дозволяючи використовувати процесор більш раціонально. Наприклад, для середньостатистичного процесу обчислення займають лише 20% часу його перебування в пам'яті, то при п'яти процесах, які одночасно знаходяться в пам'яті, процесор буде завантажений постійно. Але в цей аналіз закладено абсолютно нереальний оптимізм, оскільки у ньому свідомо передбачається, що всі п'ять процесів ніколи не будуть одночасно перебувати в очікуванні закінчення якогось процесу вводу/виводу. Краще вибудовувати модель на основі імовірнісного погляду на використання процесору. Припустимо, що процес проводить частину свого часу p в очікуванні завершення операцій вводу/виводу. При одночасній присутності в пам'яті n процесів ймовірність того, що всі n процесів очікують завершення вводу/виводу (в разі чого процесор знаходиться у стані простоювання), дорівнює p^n .

Тоді час використання процесору обчислюється за формулою:

$$1 - p^n$$

На рисунку 2 показано час використання процесору у вигляді функції від аргументу n , який називається ступенем багатозадачності. Судячи з рисунку, якщо процес витрачає 80% свого часу на очікування завершення операцій вводу/виводу, то для зниження простоювання процесора до рівня не більше 10% в пам'яті можуть одночасно перебувати, принаймні, 10 процесів.

Коли ви зрозумієте, що до очікування операцій вводу/виводу відноситься і очікування інтерактивного процесу призначеного для користувача вводу із терміналу (або клацання кнопкою миші на значках екрану), стане зрозуміло, що час очікування завершення операцій вводу/виводу, що становить 80% і більше, не така вже й рідкість. Але навіть на серверах процеси, які здійснюють багато операцій вводу/виводу, часто мають такий же або навіть більший відсоток простоювання.

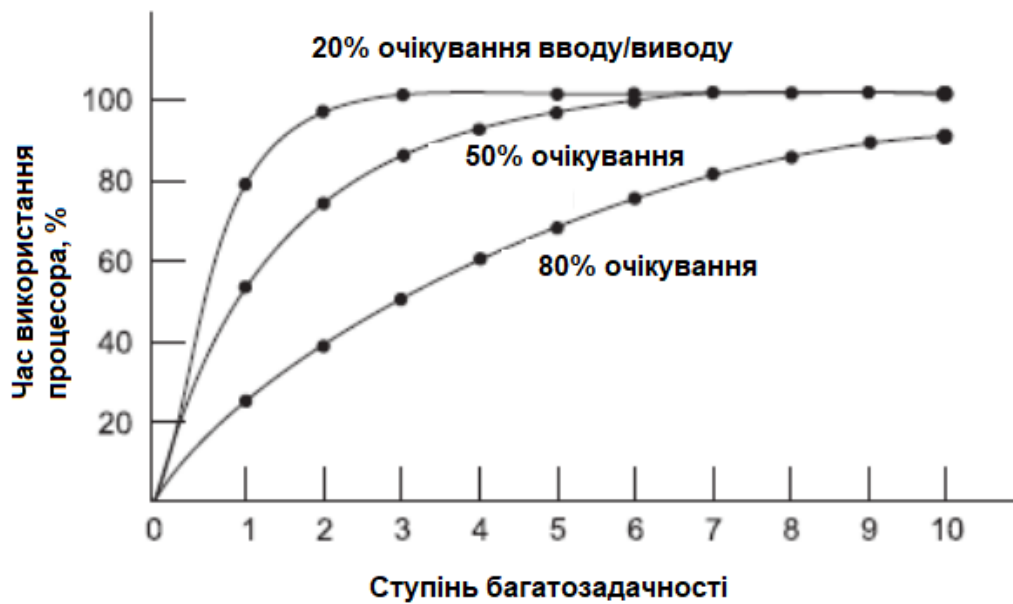


Рис. 2 - Час задіяння процесора у вигляді функції від кількості процесів, присутніх в пам'яті

Але слід зауважити, що розглянута імовірнісна модель носить дуже приблизний характер. У ній, безумовно, передбачається, що n процесів є незалежними один від одного, а значить, в системі з 5-ю процесами в пам'яті цілком припустимо мати 3 виконуваних процеси і 2 процеси на очікуванні. Але маючи один процесор, неможливо мати відразу 3 виконуваних процеси, тому процес, який стає готовим до роботи при зайнятому процесорі, змушений чекати своєї черги. Через це процеси не є незалежними. Більш точна модель може бути побудована з використанням теорії черг. Але, незважаючи на спрощеність представленої моделі, вона може бути використана для специфічних, хоча і досить приблизних прогнозів, що стосуються продуктивності центрального процесора.

Припустимо, наприклад, що пам'ять комп'ютера становить 8 Гбайт, ОС та її таблиці процесів займають до 2 Гбайт, а кожна програма також займає до 2 Гбайт. Цей обсяг дозволяє одночасно розмістити в пам'яті 3 програми. При середньому очікуванні вводу/виводу, який становить 80% часу, ми маємо завантаженість процесора (якщо ігнорувати витрати на роботу самої ОС), $1 - 0,8^3$, або близько 49%.

Розглянемо ефект від збільшення обсягу пам'яті ще на 8 Гбайт:

- ОС перейде від трикратної багатозадачності до семикратної;
- завантаженість процесора зросте з 49% до 79% - продуктивність процесора збільшиться на 30%.

Але збільшення пам'яті ще на 8 Гбайт підніме рівень продуктивності всього лише з 79 до 91%, тобто додатковий приріст продуктивності складе лише 12%. Використовуючи цю модель, власники комп'ютерів можуть прийти до висновку, що перше нарощування

обсягу пам'яті, на відміну від другого, стане непоганим внеском в підвищення продуктивності процесора.

1.1.3 Стани процесів

Для опису станів процесів використовується кілька моделей.

Найпростіша модель - це модель трьох станів (рисунок 3).

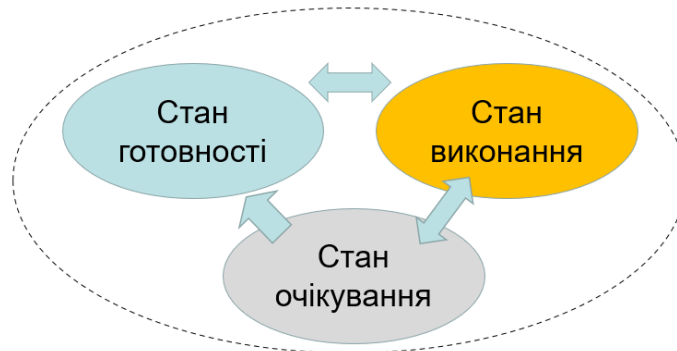


Рис. 3 – Модель 3х станів процесу

Модель складається зі станів:

- стан виконання;
- стан очікування;
- стан готовності.

Але до цих трьох станів необхідно додати стан народження і завершення.

Виконання - це активний стан, під час якого процес володіє всіма необхідними йому ресурсами. У цьому стані процес безпосередньо виконується процесором.

Очікування - це пасивний стан, під час якого процес заблокований, він не може бути виконаний, бо чекає якоїсь події, наприклад, введення даних або звільнення потрібного йому пристрою.

Готовність - це теж пасивний стан, процес теж заблокований, але на відміну від стану очікування, він заблокований не через внутрішні причини (адже очікування введення даних - це внутрішня, "особиста" проблема процесу - він може ж і не очікувати введення даних і вільно виконуватися - ніхто йому не заважає), а за зовнішніми причинами, що не залежать від процесу. Коли процес може перейти в стан готовності? Припустимо, що процес виконувався до введення даних. До цього моменту він був у стані виконання, потім перейшов у стан очікування - йому потрібно почекати, поки йому передадуть потрібні для роботи процесу дані. Коли потрібні дані були отримані, процес готовий перейти у стан виконання, але так як він не єдиний процес в системі, поки він був у стані очікування, процесор став виконувати інший процес. Тоді зазначений процес переходить у стан

готовності у будь-який момент почати виконуватися на процесорі, коли процесор звільниться.

Зі стану готовності процес може перейти тільки у стан виконання. У стані виконання може перебувати тільки один процес на один процесор. Якщо використовується n -процесорна машина, то одночасно в стані виконання можуть бути n процесів.

Зі стану виконання процес може перейти або в стан очікування або в стан готовності. У стан готовності процес може перейти, якщо під час його виконання, квант часу виконання завершився. Іншими словами, в ОС є спеціальна програма - планувальник, яка стежить за тим, щоб всі процеси виконувалися відведений їм час.

Більш складна модель - це модель, що складається з п'яти станів (рисунок 4).

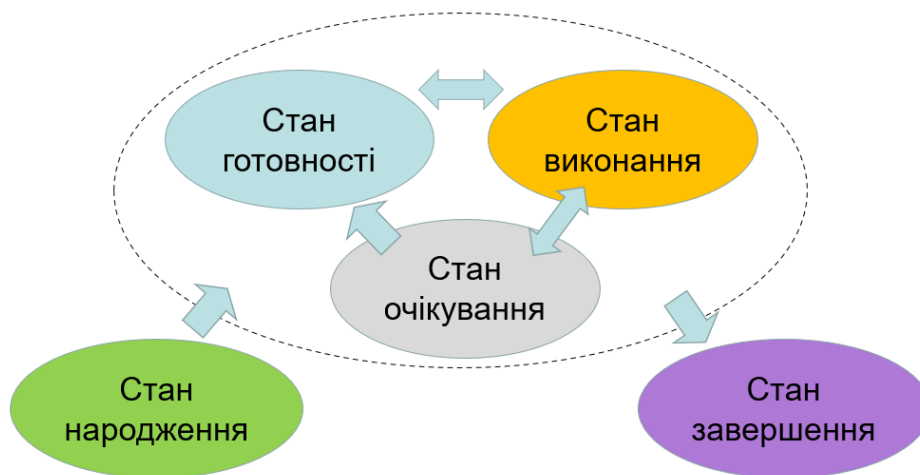


Рис. 4 – Модель 5-ти станів процесу

У цій моделі з'явилося два додаткових стани: народження процесу і завершення процесу.

Народження процесу - це пасивний стан, коли самого процесу ще немає, але вже готова структура для появи процесу.

Завершення процесу - самого процесу вже немає в оперативній пам'яті, але іноді його структура ще залишиється в списку процесів, коли процес перетворюється у так званий «зомбі»-процес.

1.2 Керування процесами *Unix*-подібних ОС

Відомо, що будь-яке керування будь-яким об'єктом включає в себе чотири основні операції (*CRUD*-операції):

- *Create* – створення об'єкта,
- *Read* – перегляд значень атрибутів об'єкта;

- *Update* – зміна значень атрибутів об'єкта;
- *Delete* – видалення об'єкта.

Для процесів ОС вказані операції створення, зміни і видалення представляються у вигляді семи додаткових операцій, які забезпечують перехід процесів між 5-ма станами (рисунок 5):

- 1) *створення процесу* – перехід зі стану народження у стан готовності;
- 2) *запуск процесу (або його вибір)* – перехід зі стану готовності у стан виконання;
- 3) *зміна пріоритету процесу* – перехід зі стану виконання у стан готовності у зв'язку із закінченням кванту часу виконання на процесорі;
- 4) *блокування процесу* – перехід зі стану виконання у стан очікування;
- 5) *пробудження процесу* – перехід зі стану очікування у стан готовності;
- 6) *відновлення процесу* – перехід зі стану очікування у стан виконання;
- 7) *знищення процесу* – перехід зі стану виконання у стан завершення;

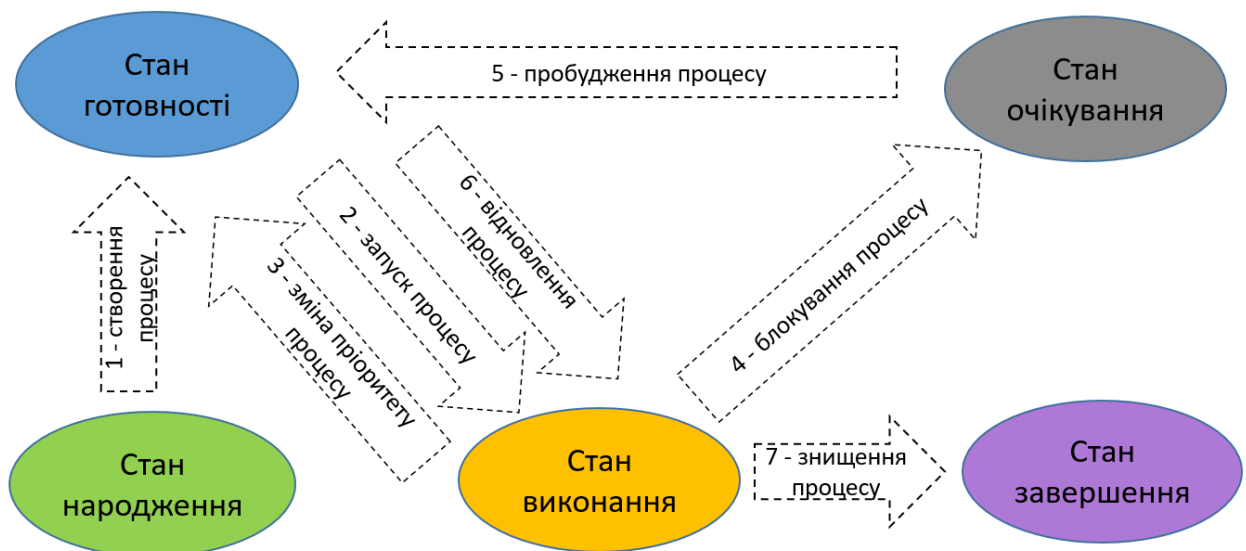


Рис. 5 – Операції керування процесами

1.2.1 Створення процесу

Для створення процесу ОС потрібно:

- присвоїти процесу ім'я;
- додати інформацію про процес у список процесів;
- визначити пріоритет процесу;
- сформувати блок управління процесом;
- надати процесу потрібні йому ресурси.

Процес не може взятися з порожнини, бо його обов'язково повинен запустити інший процес.

Процес, запущений іншим процесом, називається *процесом-нащадком* або *child-процесом*.

Процес, який запустив процес, називається *батьківським процесом* або *parent-процесом*.

У кожного процесу є два атрибути:

- *PID (Process ID)* - ідентифікатор процесу
- *PPID (Parent Process ID)* - ідентифікатор батьківського процесу.

Процеси створюють ієрархію у вигляді дерева. Найпершим *parent-процесом*, що стоїть на вершині цього дерева, є процес *init* або *system* (*PID* = 1).

1.2.2 Перегляд інформації про процеси *Unix*-подібних ОС

1.2.2.1 Команда *pstree*

Для швидкого перегляду ієрархії процесів у вигляді дерева використовується команда:

```
pstree [опції] [аргумент]
```

Аргументи команди:

- якщо в якості аргументу вказати *PID*, то команда виведе не все дерево, а тільки гілку нащадків процесу з цим *PID*, *pstree 124*
- ім'я користувача в якості аргументу вимагає вивести всі гілки процесів, запущених цим користувачем, наприклад, *pstree user1*.

Приклади корисних опцій:

- *p* – виводить поряд з ім'ям процесу у круглих дужках його *PID*;
- *u* – виводить поряд з ім'ям процесу у круглих дужках ім'я користувача, який запустив процес.

Для зручного перегляду великого дерева можна скористатися таким конвеєром:

```
pstree -up | less
```

Приклад результату виконання команди *pstree* представлено на рисунку 6, де першим процесом, процесом-пращуrom є процес *systemd*, *PID*=1

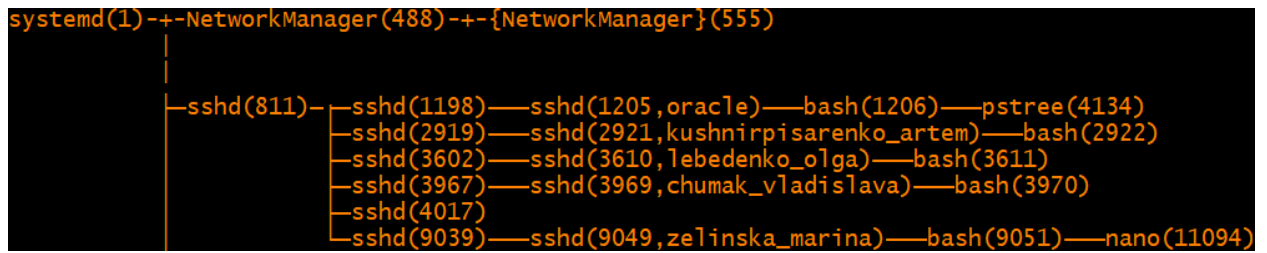


Рис. 6 – Приклад результату виконання команди *pstree*

На рисунку 6 можна помітити ланцюжок процесів *sshd-sshd-bash-pstree*, який пов'язано з користувачем *oracle* та відповідає наступному ланцюжку запуску *child*-процесів:

- процес *sshd(811)* – серверний процес (остання літера *d* – *daemon*), який запущений ще на етапі старту ОС та відповідає за встановлення з'єднання зовнішніх *ssh*-команд з ОС;
- процес *sshd(1198)* – *child*-процес для процесу *sshd(811)*, як копія *sshd*-процесу, яка автоматично створюється в ОС коли з'являється мережевий запит на з'єднання від зовнішньої *ssh*-команди;
- процес *sshd(1205, oracle)* – *child*-процес для процесу *sshd(1198)*, як копія *sshd*-процесу, яка автоматично створюється в ОС коли створюється псевдотермінал *pty* для користувача *oracle*;
- процес *bash(1206)* – *child*-процес для процесу *sshd(1205, oracle)*, який був запущений для початку роботи з оболонкою командного рядку *bash* від імені користувача *oracle*;
- *pstree(4134)* – *child*-процес для процесу *bash(1206)*, який був запущений під час виконання команди *pstree*.

1.2.2.2 Аналіз таблиці процесів командою *PS*

Інформація про процеси розташовується в спеціальній таблиці процесів.

В *Unix*-подібних ОС для отримання вмісту таблиці процесів використовується наступна команда:

```
ps [опції]
```

При відсутності опцій команда видає список процесів, запущених поточним користувачем з поточного псевдотерміналу, як показано на рисунку 7. Команда показала два процеси:

- перший процес має *PID*=21133, запущений із псевдотерміналу *pts/0* через команду *bash*;
- другий процес має *PID*=21161 запущено з того ж самого псевдотерміналу *pts/0* через команду *PS*.


```
[oracle@vpsj3IeQ ~]$ ps
  PID TTY          TIME CMD
 21133 pts/0    00:00:00 bash
 21161 pts/0    00:00:00 ps
[oracle@vpsj3IeQ ~]$ ps
```

Рис. 7 – Приклад результату виконання команди *ps*

Команда *PS* дозволяє зробити як би "миттєвий знімок", фотографію таблиці процесів, запущених в ОС у поточний момент часу.

Якщо треба проаналізувати окремий рядок відповіді, можна використати традиційний конвеєр з утилітою *AWK*, для якої в описі умов фільтрації необхідно вказати номер рядку, наприклад, *NR==2*, як показано нижче:

```
ps -u | awk 'NR==2 { print $0}'
```

Приклади опцій команди *PS* представлено в таблиці 1.

Таблиця 1 – Приклади опцій команди *ps*

Опція	Призначення
<i>A</i> або <i>e</i>	Отримати таблицю процесів зі спрощеним набором стовпчиків таблиці для всіх процесів ОС, наприклад: <i>ps -A</i>
<i>p</i> <i>PID</i>	Отримати таблицю процесів зі спрощеним набором стовпчиків таблиці для процесів із вказаними <i>PID</i> , наприклад: <i>ps -p 21133,21161</i>
<i>u</i>	Отримати таблицю процесів із розширеним набором стовпчиків для всіх процесів поточного користувача, наприклад: <i>ps -u</i>
<i>o</i> стовпчик	Отримати тільки вказані назви стовпчиків таблиці процесів, наприклад: <i>ps -o pid,ppid,cmd</i>

На рисунку 8 наведено приклад результату виконання команди *PS*, для якої вказано декілька назв колонок таблиці процесів. Аналіз результату виконання команди свідчить, що:

- команда *bash* запустила перший процес *PID*=21133;
- команда *PS* запустила другий процес *PID*=23698;
- другий процес *PID*=23698 є *child*-процесом для першого процесу, інакше кажучи, перший процес *PID*=21133 є *parent*-процесом для другого процесу.

```
[oracle@vpsj3IeQ ~]$ ps -o pid,ppid,cmd
  PID  PPID  CMD
 21133  21132  -bash
 23698  21133  ps -o pid,ppid,cmd
```

Рис. 8 – Приклад результату виконання команди

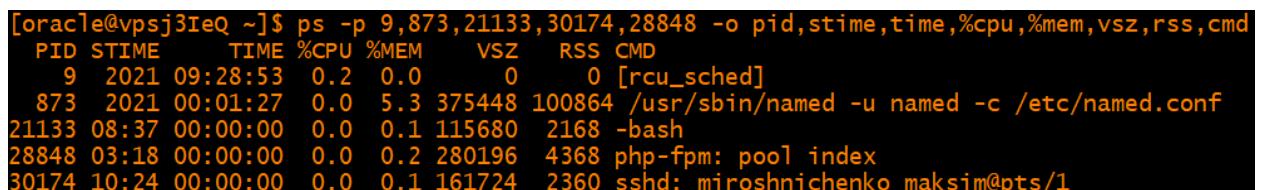
```
ps -o pid,ppid,cmd
```

Для опції *-o* можуть бути присутніми наступні колонки таблиці (значення вказувати в нижньому регістрі символів):

- *user* – ім'я власника процесу;
- *pid* – ідентифікатор процесу;
- *ppid* – ідентифікатор *parent*-процесу;
- *tty* – назва псевдотерміналу, з якого запущено процес;
- *stime* – час старту процесу;
- *pri* – пріоритет планування процесу;
- *ni* – значення *nice* як протилежність пріоритету виконання процесу;
- *time* – заальний час перебування процесу на процесорі;
- *cmd* або *command* – командний рядок запуску програми, яким було запущено процес;
- *%cpu* – відсоток часу центрального процесора, виділеного даному процесу;
- *%mem* – відсоток реальної пам'яті, яка використовується даним процесом;
- *vsz* (*virtual size*) – обсяг віртуальної пам'яті (сторінок пам'яті), який виділено процесу в поточний момент (вимірюється у кілобайтах або у сторінках розміром 1Кб);
- *rss* (*resident set size*) – обсяг фізичної пам'яті, який реально використовує процес на поточний момент (вимірюється у кілобайтах або у сторінках розміром 1Кб);
- *s* або *stat* – статус процесу.

На рисунку 9 показано приклад результату виконання команди *PS*, для якої вказано декілько назв колонок таблиці процесів, що мають *PID* = 9, 873, 21133, 30174, 28848:

```
ps -p 9,873,21133,30174,28848  
-o pid,stime,time,%cpu,%mem,vsz,rss,cmd
```



PID	STIME	TIME	%CPU	%MEM	VSZ	RSS	CMD
9	2021 09:28:53	0.2	0.0	0	0	0	[rcu_sched]
873	2021 00:01:27	0.0	5.3	375448	100864	/usr/sbin/named -u named -c /etc/named.conf	
21133	08:37 00:00:00	0.0	0.1	115680	2168	-bash	
28848	03:18 00:00:00	0.0	0.2	280196	4368	php-fpm: pool index	
30174	10:24 00:00:00	0.0	0.1	161724	2360	sshd: miroshnichenko_maksim@pts/1	

Рис. 9 – Приклад результату виконання команди *PS*

Аналіз результату виконання попередньої команди *PS* свідчить, що під час виконання команди:

- процеси *PID*=9, *PID*=873 почали працювати ще у минулому 2021 році, тому для них значення *STIME* визначається лише роком;

- процеси $PID=21133$, $PID=28848$, $PID=30174$ почали працювати поточної доби, тому для них значення $STIME$ визначається лише часом та хвилинами;
- процес $PID=9$ за увесь час свого перебування в оперативній пам'яті реально працював на центральному процесорі 9 годин, 28 хвилин та 53 секунди, а процес $PID=873$ за увесь час свого перебування в оперативній пам'яті реально працював на центральному процесорі 1 хвилину та 27 секунд;
- процес $PID=9$ працював на центральному процесорі = 0.2% від загального часу роботи процесору, але інші процеси працювали на процесорі < 0.1%;
- процес $PID=9$ використовував пам'яті < 0.1%, процес $PID=873$ використовував найбільше пам'яті = 5.3%;
- процес $PID=873$ може використати віртуальної пам'яті у розмірі до 375448 Кбайт, але на поточний момент цей процес використав лише 100864 Кбайт.

Прокоментуємо особливості значень *nice*, яке пов'язано із пріоритетом виконання процесу. Пріоритет для кожного процесу встановлюється в той момент, коли процес породжується і при звичайному запуску команд або програм приймається рівним пріоритету *parent*-процесу. Пріоритет процесу вказує процесору групу пріоритету, в якій буде виконуватися процес з виділенням заданого для цієї групи кванту часу виконання на процесорі, враховуючи витісняючу багатозадачність ОС. Якщо у процесу пріоритет вище серед інших процесів, що знаходяться у стані виконання або готовності, тоді і квант часу для його виконання на процесорі більше, що забезпечить більшу швидкість роботи відповідної команди або програми та зменшення часу на її виконання.

В *Unix*-подібних ОС пріоритет процесу визначається значенням «*nice*», який в перекладі з англійської означає «милий», «ввічливий». «Ввічливий» процес завжди «поступається дорогою» іншим процесам. «Грубий» процес намагається сам обійти всі процеси, що знаходяться у стані виконання або готовності. З урахуванням цього поняття значення *nice* лежать в межах, зворотних терміну важливість (пріоритет):

- від *nice* = +20 – найменший пріоритет процесу, що виконується рідше за інших і тільки тоді, коли ніщо інше не займає процесор;
- до *nice* = -19 – найвищий пріоритет процесу, що виконується частіше за інших.

На рисунку 10 показано приклад результату виконання команди *PS*, для якої вказано декілька назв колонок таблиці процесів, що мають PID = 38, 26, 417, 21133, 30174, 28848:

```
ps -p 38,26,417,21133,30174,28848 -o user,pid,nice,pri,cmd
```

```
[oracle@vpsj3IeQ ~]$ ps -p 38,26,417,21133,30174,28848 -o user,pid,nice,pri,cmd
USER      PID    NI PRI  CMD
root       26   -20  39 [kblockd]
root       38   -20  39 [crypto]
root       417   -4  23 /sbin/auditd
oracle    21133    0  19 -bash
root      28848   10   9 php-fpm: pool index
miroshn+  30174    0  19 sshd: miroshnichenko_maksim@pts/1
```

Рис. 10 – Приклад результату виконання команди

```
ps -p 38,26,417,21133,30174,28848 -o user,pid,nice,pri,cmd
```

Аналіз результату виконання команди *PS* в колонках *NI* та *PRI* свідчить, що:

- процес, пов'язаний з системним програмним модулем *crypt* (API-функції для шифрування даних), та процес-демон *kblockd* (керування операціями вводу/виводу) знаходяться у групі процесів з найнижчим значенням *NI* = -20, тобто з найвищим пріоритетом *PRI* = 39;
- процес, пов'язаний з процесом-демоном *auditd* (керування аудитом та моніторингом в ОС) знаходиться к окремій групі процесів зі значенням *NI* = -4, тобто з пріоритетом *PRI* = 23;
- процеси звичайних користувачів *oracle* та *miroshnichenko_maksim* знаходиться к окремій групі процесів зі значенням *NI* = 10, тобто з пріоритетом *PRI* = 19.

Також прокоментуємо особливості значень статусу процесу *stat*. В колонці *STAT* процесу можуть стояти наступні значення статусу, який мав процес на момент виконання команди *PS*:

- *R* (*Running* або *Runnable*) – процес виконується на процесорі або чекає тільки моменту, коли планувальник завдань виділить йому черговий квант часу для початку роботи на процесорі з урахування алгоритму роботи витісняючої багатозадачності;
- *S* (*Interruptible Sleeping*) – процес "спить" або очікує завершення якоїсь події, не пов'язаної з операціями вводу/виводу;
- *D* (*Uninterruptible Sleeping*) – процес очікує завершення операцій вводу/виводу;
- *T* (*Stopped*) – процес призупинено;
- *Z* (*Zombie*) – процес-зомбі – процес, який вже не працює та як *child*-процес втратив контроль з боку свого *parent*-процесу у зв'язку з якоюсь помилкою, але від цього процесу ще не очищена таблиця процесів.

До вказаних значень статусу можуть бути додані додаткові два символи-коментарі:

- *s* – процес є лідером псевдотерміналу, тобто головним *parent*-процесом у ланцюжку запуску *child*-процесів власником псевдотерміналу;
- *+* (плюс) – процес знаходиться в інтерактивному режимі свого псевдотерміналу, наприклад, запущений користувачем з командного рядку не у фоновому режимі.

На рисунку 11 показано приклад результату виконання команди *PS* у конвеєрному ланцюжку з командою *GREP*:

```
ps -o pid,ppid,stat,cmd
```

Аналіз результату виконання конвеєру команд в колонці *stat* (5-та колонка) з рисунку 11 свідчить, що:

- перший процес, пов'язаний з командою *bash* є лідером псевдотерміналу *pts/0* та в момент перегляду таблиці процесів перебував у стані *Interruptible Sleeping*, тому що, дійсно, на той момент команда *bash* вже завершила всі свої дії із запуску *child*-процесів, пов'язаних з виконанням конвеєру команд, та чекає на нові команди;
- другий процес, пов'язаний з командою *ps*, в момент перегляду таблиці процесів перебував у стані *Running* або *Runnable*, знаходячись в інтерактивному режимі псевдотерміналу *pts/0*;
- третій процес, пов'язаний з командою *GREP*, в момент перегляду таблиці процесів перебував у стані *Interruptible Sleeping*, знаходячись в інтерактивному режимі псевдотерміналу *pts/0*.

```
[oracle@vpsj3IeQ ~]$ ps -o user,TTY,pid,ppid,stat,cmd | grep oracle
oracle pts/0 21133 21132 ss -bash
oracle pts/0 29644 21133 R+ ps -o user,TTY,pid,ppid,stat,cmd
oracle pts/0 29645 21133 S+ grep --color=auto oracle
```

Рис. 11 – Приклад результату виконання команди *ps -o pid,ppid,stat,cmd*

1.2.2.3 Аналіз таблиці процесів командою *TOP*

Вже визначалося, що команда *PS* дозволяє зробити як би "миттєвий знімок", фотографію таблиці процесів, запущених в ОС у поточний момент часу. Але, якщо необхідно відображати стан процесів і їх активність "в реальному режимі часу", використовується команда *TOP*, яка оновлює на екрані дані про таблицю процесів кожні кілька секунд, а також надає додаткову статистичну інформацію про загальний стан ОС.

На рисунку 12 представлено результат виконання команди *top*.

```
top - 20:42:43 up 162 days, 5:30, 4 users, load average: 0.50, 1.17, 0.94
Tasks: 128 total, 1 running, 127 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.3 us, 0.5 sy, 0.0 ni, 98.5 id, 0.0 wa, 0.0 hi, 0.5 si, 0.2 st
KiB Mem : 1881836 total, 883128 free, 145088 used, 853620 buff/cache
KiB Swap: 4194300 total, 3609888 free, 584412 used. 1687644 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
  488 root        20   0  476436   1140   452  S   1.7   0.1   2935:47 NetworkManager
  274 root        20   0      0      0      0  S   0.3   0.0    7:14.28 jbd2/vda1-8
 2021 mysql      20   0 1272260   1076      0  S   0.3   0.1  170:33.48 mysqld
27079 oracle     20   0  162104   2280  1592  R   0.3   0.1    0:00.11 top
```

Рис. 12 – Приклад результату виконання команди *top*

У верхній частині вікна відображається:

- поточний час на сервері;
- час, що минув з моменту останнього запуску ОС;
- поточна кількість активних користувачів в ОС;
- середнє завантаження серверу (*load average*);
- кількість запущених процесів;
- кількість процесів, які знаходяться у станах *running*, *sleeping*, *stopped*, *zombie*;
- дані про використання процесору, оперативної пам'яті і *swap*-пам'яті.

Після опису статистичних значень команда *TOP* надає таблицю, яка характеризує окремі процеси у вигляді наступних колонок: *PID*, *USER*, *PR* – *Priority*, *NI*, *VIRT* (*Virtual memory* – загальний обсяг пам'яті, який можна використовувати процесом в даний момент), *RES* (*Resident memory size* – обсяг фізичної пам'яті, вже зайнятої процесом), *SHR* (*Shared Memory size* – обсяг *VIRT*-пам'яті, який використовується процесом спільно з іншими процесами, наприклад, для файлів динамічних бібліотек), *S* – *Process Status*, *%CPU*, *%MEM*, *TIME+* – час виконання процесу, *COMMAND* – команда, яка запустила процес.

Наприклад, як видно на рисунку 11, процес *PID*=488 на поточний момент:

- може використати 476436 Кб віртуальної пам'яті;
- вже використав 1140 Кб віртуальної пам'яті;
- спільно з іншими процесами використовує 452 Кб віртуальної пам'яті.

Список процесів може бути відсортований через використання сполучень клавіш:

<Shift> + <N> – сортувати процеси по *PID*;

<Shift> + <P> – сортувати процеси по % використання процесора;

<Shift> + <M> – сортувати процеси по % використання пам'яті;

<Shift> + <T> – сортувати процеси за часом їх виконання.

1.2.2.4 Особливості розрахунку *load average* – середньої завантаженості серверу

Середнє завантаження серверу (*load average*) обчислюється як довжина черги виконання в ОС і, якщо значення = 0, тоді сервер нічого не виконує, якщо значення >= 1, тоді є процеси, які очікують своєї черги на виконання на процесорі. В команді *TOP* завантаження відображається у вигляді трьох значень, які є усередненими величинами за останню 1 хвилину, останні 5 хвилин та останні 15 хвилин з метою аналізу зміни завантаженості серверу.

Наведемо приклад розрахунку значення *load average*.

Початкові значення завдання.

Одного разу команда *TOP* показала наступне значення *load average*: 0.50, 1.21, 0.95

Припустимо, що за наступні 4 хвилини в ОС на однопроцесорному сервері статистика присутності процесів була така: 1-ша хвилина – $P1$; 3-тя хвилина – $P1, P2$; 4-та хвилина – $P3, P4, P5, P6$. Припустимо також, що в інші невказані хвилини статистика присутності визначалася у відповідності з присутністю процесів у хвилини між ними. Треба визначити значення *load average*, яке буде за останні 5 хвилин?

Рішення завдання. У 1-шу хвилину присутній лише один процес, тому він не чекав у стані готовності, а постійно працював у стані виконання на процесорі, і довжина черги за першу хвилину $L1 = 0$. Процес $P1$ продовжив працювати у 3-тю хвилину, тому наприклад, на 2-й хвилині він також був присутній з довжиною черги $L2 = 0$. На 3-й хвилині присутні 2 процеси: один - у стані готовності, інший - у стані виконання. Тому довжина черги за 3-тю хвилину $L3 = 1$. На 4-тій хвилині з чотирма процесами довжина черги $L4 = 3$.

$$\text{load average}(4 \text{ хвилини}) = (L1 + L2 + L3 + L4) / 4 = (0 + 0 + 1 + 3) / 4 = 1.0$$

Для визначення *load average*(5 хвилин) необхідно врахувати середнє значення, яке показувала команда *TOP* за останню хвилину:

$$\begin{aligned} \text{load average}(5 \text{ хвилин}) &= (\text{load average}(1 \text{ хвилина}) + \text{load average}(4 \text{ хвилини})) / 2 \\ &= (0.5 + 1.0) / 2 = 1.5 / 2 = 0.75 \end{aligned}$$

1.2.2.5 Відмінності в алгоритмах обчислення завантаженості процесора командами *TOP* та *PS*

Якщо подивитися коментарі визначення частки часу використання процесора для команд *TOP* та *PS*, представлені в документації (команди *man top* і *man ps*), то бачимо, що:

- команда *TOP* показує завантаження процесора, яке проявилось з моменту останнього оновлення екрану, виражену у відсотках від загального часу роботи процесора в межах поточного відрізка часу роботи процесу;

- команда *PS* показує завантаження процесора, представлене у відсотках часу, витраченого на виконання процесу на процесорі протягом усього часу життя процесу.

Отже, припустимо, у вас є процес, який був запущений тиждень тому і за цей час він використовував в середньому 5% процесорного часу. Якщо раптом він стане інтенсивно завантажувати процесор, постійно споживаючи 90%, то команда *TOP* зразу покаже 90%, а команда *PS* продовжить показувати ті ж 5%, але з поступовою зміною до 90% протягом тривалого періоду часу.

1.2.3 Керування процесами через сигнали ОС

Для керування процесами в *Unix*-подібних ОС використовуються сигнали як засіб, за допомогою якого процесам можна передати повідомлення про деякі події в ОС.

Самі процеси теж можуть генерувати сигнали, за допомогою яких вони передають певні повідомлення ядру ОС і іншим процесам.

В *Unix*-подібних ОС для передачі сигналів процесу використовується команда:

```
kill [-сигнал] PID [PID ...],
```

де – сигнал - номер або символічне ім'я сигналу, список яких можна дізнатися по команді *kill -l*

Всі назви сигналів починаються на *SIG*, але цю приставку іноді опускають: наприклад, сигнал з номером 1 позначають або як *SIGHUP*, або просто як *HUP*.

За замовчуванням команда *kill* здійснює відправку сигналу *SIGTERM* або *TERM*.

У таблиці 2 наведені деякі з найпоширеніших сигналів.

Таблиця 2 – Приклади сигналів

№	Назва	Опис	Комбінація клавіш
1	<i>HUP</i>	<i>Hangup</i> . Відміна роботи процесу	
2	<i>INT</i>	<i>Interrupt</i> . В інтерактивних програмах - припинення активного процесу	<Ctrl>+<C> або <Ctrl>+
3	<i>QUIT</i>	Як правило, сильніше сигналу <i>Interrupt</i>	<Ctrl>+< >
9	<i>KILL</i>	Завершення роботи процесу	
15	<i>TERM</i>	<i>Software Termination</i> . Вимагання завершити процес (програмне завершення)	
18	<i>CONT</i>	Продовжити виконання призупиненого процесу	
19	<i>STOP</i>	Призупинення виконання процесу	
20	<i>TSTR</i>	Сигнал призупинення процесу, генеруємий клавіатурою. Переводить процес у фоновий режим	<Ctrl>+<Z>

Значення ідентифікаторів < 0 використовуються для вказівки груп процесів, яким потрібно відправити сигнали. Ідентифікатор -1 є спеціалізованим ідентифікатором і використовується для відправки сигналів всім запущеним процесам за винятком *kill* та *init*.

На рисунку 13 представлено фрагменти роботи двох псевдотерміналів. В лівому псевдотерміналі користувачем *oracle* запущено команду *ping localhost*, яка періодично передає мережеві пакети на локальну адресу сервера. В правому псевдотерміналі послідовно виконуються наступні команди:

1) *ps -u oracle -o pid,ppid,cmd,stat* - отримати перелік процесів, запущених користувачем *oracle*;

- 2) `kill -19 6766` - призупинити виконання процесу команди *ping* (*STOP*) з *PID*=6766;
- 3) `kill -18 6766` - повторно запустити процес команди *ping* (*CONT*) з *PID*=6766;
- 4) `kill -9 6766` - завершити роботу процесу команди *ping* (*TERM*) з *PID*=6766.

The image shows two terminal windows. The left window shows a continuous ping to localhost, which is then stopped and restarted. The right window shows the output of the 'ps' command, identifying the ping process (PID 6766) and then using 'kill' to stop and restart it.

```

[oracle@vpsj3IeQ ~]$ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.023 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.031 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.035 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.035 ms
64 bytes from localhost (127.0.0.1): icmp_seq=5 ttl=64 time=0.034 ms
64 bytes from localhost (127.0.0.1): icmp_seq=6 ttl=64 time=0.032 ms
64 bytes from localhost (127.0.0.1): icmp_seq=7 ttl=64 time=0.031 ms
64 bytes from localhost (127.0.0.1): icmp_seq=8 ttl=64 time=0.040 ms
64 bytes from localhost (127.0.0.1): icmp_seq=9 ttl=64 time=0.033 ms
64 bytes from localhost (127.0.0.1): icmp_seq=10 ttl=64 time=0.040 ms
64 bytes from localhost (127.0.0.1): icmp_seq=11 ttl=64 time=0.043 ms
[2]+  Stopped                  ping localhost
[oracle@vpsj3IeQ ~]$ 64 bytes from localhost (127.0.0.1):
icmp_seq=12 ttl=64 time=0.030 ms
64 bytes from localhost (127.0.0.1): icmp_seq=13 ttl=64 time=0.038 ms

[oracle@vpsj3IeQ ~]$ ps -u oracle -o pid,ppid,cmd,stat
  PID  PPID  CMD                                STAT
   864    847  sshd: oracle@pts/0                  S
   865    864  -bash                               Ss
  1205   1198  sshd: oracle@pts/1                  S
  1206   1205  -bash                               Ss
  6641   865  ping localhost                       T
  6766   865  ping localhost                       S+
  6772   1206  ps -u oracle -o pid,ppid,cm R+

[oracle@vpsj3IeQ ~]$ kill -19 6766
[oracle@vpsj3IeQ ~]$ ps -u oracle -o pid,ppid,cmd,stat
  PID  PPID  CMD                                STAT
   864    847  sshd: oracle@pts/0                  S
   865    864  -bash                               Ss+
  1205   1198  sshd: oracle@pts/1                  S
  1206   1205  -bash                               Ss
  6641   865  ping localhost                       T
  6766   865  ping localhost                       T
  6786   1206  ps -u oracle -o pid,ppid,cm R+

[oracle@vpsj3IeQ ~]$ kill -18 6766
[oracle@vpsj3IeQ ~]$ kill 6766
[oracle@vpsj3IeQ ~]$
  
```

Рис. 13 – Приклад результату виконання команди *ps*

1.2.4 Керування процесами у фоновому режимі роботи

1.2.4.1 Ступені свободи (незалежності) процесів від псевдотерміналів

Розглянемо ступені свободи (незалежності) процесів від псевдотерміналів.

Мінімальний ступінь свободи – режим *foreground* («передній план»):

- процес є результатом запуску програми з командного рядку;
- процес "прив'язується" до псевдотерміналу, з якого він запущений.

Середній ступінь свободи – режим *background* («задній план») – процес є результатом запуску програми з командного рядку із завершальним символом `&` (фоновий режим).

Середній ступінь свободи – режим «*background without hanging up*» – процес ігнорує сигнал *HUP* (команда *nohup*).

Максимальний ступінь свободи – *daemon*-процес («даймон», не демон):

- «не прив'язані» до псевдотерміналу;
- найчастіше для взаємодії з іншими процесами надають свої копії;
- найчастіше запускаються під час старту ОС;
- найчастіше у назві мають останню літеру *d*.

1.2.4.2 Керування процесами в режимах *foreground/background*

В оболонці *bash* є дві вбудовані команди, які керують переводом процесів на передній план або повернення їх у фоновий режим:

- команда *fg* – перевід процесу на передній план;
- команда *bg* – перевід процесу на задній план, у фоновий режим.

Команда *jobs* викликається без аргументів і показує завдання, запущені з поточного екземпляра *shell*. На початку кожного рядка виводу цієї команди вказується порядковий номер завдання у вигляді числа в квадратних дужках. Після номера вказується стан процесу: *stopped* (зупинений), *running* (виконується) або *suspended* (припинений). В кінці рядка вказується команда, яка виконується даним процесом. Один з номерів завдань, що виконуються, позначений знаком +, а ще один позначений знаком -. Процес, позначений знаком +, буде за замовчуванням вважатися аргументом команд *fg* або *bg*, якщо вони викликаються без параметрів. Процес, позначений знаком -, отримає знак +, якщо тільки завершиться з якої-небудь причини процес, який був позначений знаком +. Як аргумент обом цим командам передаються номери тих завдань, які присутні у результаті виконання команди *jobs*. Якщо аргументи відсутні, то мається на увазі завдання, позначене знаком +. Однею командою *bg* можна перевести у фоновий режим відразу кілька процесів, а ось повертати їх на передній план необхідно по одному.

Наведемо приклад. Нехай в терміналі запущено команду *top*. Якщо виконати комбінацію клавіш *<Ctrl>+<Z>* (сигнал призупинення процесу), тоді процес *top* буде призупинено, як показано на рисунку 14.

Команда *bg* покаже призупинений процес, як видно з рисунку 14: наявність символу *&* вказує на те, що вона представлена як команда, запущена у фоновому режимі.

```
18 root      20   0          0      0      0 S   0.0  0.0
19 root          0 -20          0      0      0 S   0.0  0.0
20 root      20   0          0      0      0 S   0.0  0.0
21 root          0 -20          0      0      0 S   0.0  0.0
22 root          0 -20          0      0      0 S   0.0  0.0
23 root          0 -20          0      0      0 S   0.0  0.0
[2]+  Stopped                  top
[oracle@vpsj3IeQ ~]$
[oracle@vpsj3IeQ ~]$ bg
[2]+  top &
[oracle@vpsj3IeQ ~]$

[2]+  Stopped                  top
[oracle@vpsj3IeQ ~]$
[oracle@vpsj3IeQ ~]$ fg
```

Рис. 14 – Приклад результату призупинення процесу *top* та виконання команди *bg*

Якщо виконати команду *fg*, вона поверне призупинений процес до стану виконання.

1.2.4.3 Керування процесами в режимі «*background without hanging up*»

При завершенні поточної сесії роботи з ОС (закриття псевдотерміналу) автоматично будуть завершені всі процеси, які були з нього запущені, тому що при завершенні сесії оболонка командного рядка посилає всім породженим нею процесам сигнал *HUP* - "відбій". Якщо необхідно, щоб деякі процеси продовжували працювати у фоновому режимі, їх потрібно запускати за допомогою утиліти *nohup* в форматі:

nohup команда &

Запускаємий таким чином процес буде ігнорувати сигнал *HUP* (якщо це можливо). Але команда *nohup* збільшує значення *nice* на 5 пунктів для працюючого процесу, тоді процес буде виконуватися з більш низьким пріоритетом.

1.2.5 Керування пріоритетами виконання процесів

Існує спеціальна команда *nice* для встановлення початкового значення пріоритету процесу під час запуску команди:

nice [-n value] command [args]

де *value* - значення (від -20 до +19), що додається до значення *nice* процесу-батька, тому отримана сума і буде значенням *nice* для нового процесу.

Якщо опція *-n* не задана, то за замовчуванням для *child*-процесу значення *nice* збільшується на +10 у порівнянні із *nice*-значенням *parent*-процесу.

Негативні значення *nice*, які забезпечать більший пріоритет може встановлювати тільки привілейований користувач. Процес *bash*-команди звичайних користувачів запускається із *nice*=0, тому за замовчуванням всі його *child*-процеси будуть мати *nice*=0.

Приклад запуску процесу файл-скрипту *file.sh* із поточного каталогу у фоновому режимі із пріоритетом за замовчуванням:

nice ./file.sh &

Команда *renice* змінює значення *nice* для процесів, які вже виконуються.

Синтаксис команди:

renice -n value -u user1 -p PID

Наприклад, для збільшення на 10 пунктів пріоритету процесу (зменшення його *nice*-значення) з PID 987, необхідно виконати:

renice -n 10 -p 987

Наприклад, для зменшення на 5 пунктів пріоритету процесу (збільшення його *nice*-значення), запущеного користувачем *user1*, необхідно виконати:

renice -n -5 -u user1

Адміністратор ОС може змінити пріоритет будь-якого процесу в системі. Інші користувачі можуть змінювати значення пріоритету тільки для тих процесів, для яких даний користувач є власником. При цьому звичайний користувач може тільки зменшити значення пріоритету (збільшити значення *nice*), але не може збільшити пріоритет, навіть для повернення значення *nice* до значення, що встановлюється за умовчанням. Тому процеси з низьким пріоритетом не можуть породити "високопріоритетних" *child*-процесів.

1.3 Керування поведінкою процесів на мові програмування *Bash*

На рисунку 15 показано файл-скрипт на мові програмування *bash* з ім'ям *e1.sh*, який виконує у безкінечному циклі процес складання чисел. Для надання прав на виконання файлу, необхідно виконати: `chmod +x e1.sh`

```
#!/bin/bash
counter=0
while [[ 1=1 ]]
do
    (( counter++ ))
done
```

Рис. 15 – Приклад файл-скрипту *bash*-програми для проведення експериментів

Після запуску файлу, наприклад, командою `./e1.sh &`, створюється новий фоновий процес.

Таблицю процесу запущеної команди можна отримати, запустивши псевдотермінал з командою:

```
ps -o pid,%cpu,cmd | grep 'e.\.sh$'
```

Для призупинки процесу с *PID* = 3445 можна скористатися командою:

```
kill -19 3445
```

Для зручної перевірки переводу процесу у сплячий режим, наприклад, на 10 хвилин, можна додати у файл-скрипт команду *sleep 10m*, як показано на рисунку 16.

```
#!/bin/bash
sleep 10m
x=0
while [ true ]
do
    (( x++ ))
    #echo $x
done
```

Рис. 16 – Приклад *bash*-програми для проведення експериментів в режимі сну

Після повторного запуску програми, її процес перейде у стан *S*, еквівалентний знаходженню у стані очікування або готовності.

На рисунку 17 показано приклад скрипт-файлу *start.sh*, який запускає із затримкою у 2 секунди інші два скрипт-файли у фоновому режимі, встановивши їх пріоритет на 5 позиції нижче ніж пріоритет процесу, пов'язаного із *bash*-оболонкою.

```
#!/bin/bash
sleep 1
nice -n 5 ./e2.sh &
sleep 1
nice -n 5 ./e3.sh &
```

Рис. 17 – Приклад скрипт-файлу *start.sh* із запуском інших двох скрипт-файлів зі змененим пріоритетом у фоновому режимі

На рисунку 18 наведено фрагмент таблиці процесів (стовпчики *PID*, *PPID*, *STAT*, *NI*, *%CPU*, *CMD*) 3-х запущених скрипт-файлів.

PID	PPID	TT	STAT	NI	%CPU	CMD
358856	358855	pts/4	Ss	0	0.0	-bash
358895	358856	pts/4	R	0	39.2	/bin/bash ./e1.sh
358898	1	pts/4	RN	5	12.7	/bin/bash ./e2.sh
358902	1	pts/4	RN	5	12.7	/bin/bash ./e3.sh

Рис. 18 – Фрагмент таблиці процесів (стовпчики *PID*, *PPID*, *STAT*, *NI*, *%CPU*, *CMD*) 3-х запущених скрипт-файлів

Аналіз значень стовпчиків *PPID* та *STAT* дає наступний висновок:

- після завершення виконання скрипт-файлу *start.sh* ОС надає запущеним процесам, пов'язаним зі скрипт-файлами *e2.sh* та *e3.sh*, новий *parent*-процес з *PID=1*;
- завантаженість процесору, пов'язаного зі скрипт-файлом *e1.sh*, = 39.2% – у три рази більше ніж завантаженість інших двох процесів, які мають значно нижчий пріоритет;
- два процеси, пов'язані зі скрипт-файлами *e2.sh* та *e3.sh*, маючи між собою однаковий пріоритет, однаково завантажують процесор.

2 Завдання

Виконати наступні дії з підготовки до виконання завдань роботи:

- 1) встановити з'єднання з віддаленим *Linux*-сервером з IP-адресою = 46.175.148.116, логіном, наданим вам викладачем, та паролем, зміненим вами у попередній роботі;
- 2) перейти до каталогу *Git*-репозиторія;
- 3) створити нову *Git*-гілку з назвою «*Laboratory-work-7*»;
- 4) перейти до роботи зі створеною гілкою;
- 5) створити каталог з назвою «*Laboratory-work-7*»;
- 6) перейти до каталогу «*Laboratory-work-7*»;
- 7) в каталозі «*Laboratory-work-7*» створити файл *README.md* та додати до файлу рядок тексту із темою лабораторної роботи «*Основи операційного керування процесами в Unix-подібних ОС*» як заголовок 2-го рівня *Markdown*-форматування;
- 8) виконати операції з оновлення *GitHub*-репозиторію змінами *Git*-репозиторія через послідовність *Git*-команд *add*, *commit* із коментарем «*Changed by Local Git*» та *push*;
- 9) на веб-сервісі *GitHub* перейти до створеної гілки «*Laboratory-work-7*»;
- 10) перейти до каталогу «*Laboratory-work-7*» та розпочати процес редагування файлу *README.md* ;
- 11) в подальшому за результатами рішень кожного наступного розділу завдань до файлу *README.md* додавати рядки як заголовки 3-го рівня *Markdown*-форматування з назвами розділу, а також знімки екрані та підписи до знімків екранів з описом пунктів завдань.

2.1 Перегляд таблиці процесів

Виконати наступні завдання.

2.1.1 Отримати ієрархію процесів у вигляді дерева процесів, враховуючи наступне:

- імена користувачів, які запустили процеси;
- *PID* процесів.

2.1.2 Отримати на екран назви запущених процесів ОС, враховуючи, що, ймовірно, вони можуть бути процесами-даймонами.

2.1.3 Отримати таблицю процесів, враховуючи наступне:

- процеси запущені від імені вашого користувача;
- таблиця процесів має розширений набір стовпчиків;
- для першого рядка опису процесу визначити значення *Virtual memory size* та *Resident memory size*.

2.1.4 Отримати таблицю процесів, враховуючи наступне:

- процес є запущеною *bash*-оболонкою;
- таблиця процесів містить лише стовпчики: *TTY, USER, PID, STIME, CMD*

2.1.5 Отримати таблицю процесів, враховуючи наступне:

- процеси знаходяться у стані сну;
- процеси не мають псевдотерміналу;
- таблиця процесів містить лише стовпчики: *PID, CMD*

2.1.6 Використовуючи команду *TOP*, отримати список процесів, відсортованих за % використання оперативної пам'яті.

2.2 Керування станами процесів

Виконати наступні завдання.

2.2.1 У поточному (1-му) псевдотерміналі виконати команду *ping localhost* у фоновому режимі.

2.2.2 Запустити 2-й псевдотермінал роботи з *Linux*-сервером.

У 2-му псевдотерміналі:

- для команди *ping* отримати таблицю її процесу (стовпчики *PID, STAT, CMD*);
- призупинити виконання процесу команди *ping*;
- отримати таблицю процесу команди *ping*;
- відновити виконання призупиненого процесу *ping*.
- остаточно завершити роботу процесу команди *ping*.

2.2.3 У 1-му псевдотерміналі запустити команду *ping* у режимі «*background without hanging up*».

У другому псевдотерміналі для команди *ping* отримати таблицю її процесу (стовпчики *PID, PPID, TTY, STAT, CMD*) для вашого поточного користувача.

2.2.4 Закрити перший псевдотермінал.

У другому псевдотерміналі для команди *ping* отримати таблицю її процесу (стовпчики *PID, PPID, TTY, STAT, CMD*) для вашого поточного користувача.

Порівняти зміст таблиці процесу з пунктом 2.2.3 та зробити висновок про стан процесу та його новий *parent*-процес.

2.2.11 Завершити роботу процесу команди *ping*.

2.3 Керування пріоритетами виконання процесів

Виконати наступні завдання.

2.3.1 Створити скрипт-файл на мові програмування *bash*, який виконує операцію

безкінечного циклічного складання за формулою: $x = x + n$, де початкове значення x = номер вашого варіанту, значення n - номер вашого варіанту. Назва файлу збігається з транслітерацією вашого прізвища із розширенням *.sh*, наприклад, *blazhko.sh*

Надати скрипт-файлу права на виконання та запустити його у фоновому режимі.

2.3.2 Переглянути таблицю процесів для процесу, пов'язаного із запущеним скрипт-файлом із урахуванням набору стовпчиків: *PID, PPID, TTY, STAT, NI, % CPU, CMD*.

Зробити висновки про стан процесу та завантаженість процесору.

2.3.3 Виконати команду призупинення процесу, запущеного у пункті 2.3.1.

Ще раз переглянути таблицю процесів для призупиненого процесу.

Зробити висновки про зміни стану процесу та завантаженості процесору.

2.3.4 Повторити попереднє завдання, запустивши команду *ps* із затримкою не менше 1 хвилини, наприклад, через командний ланцюжок *sleep 1m ; ps ...*

Порівняти значення завантаженості процесору із результатом попереднього завдання та зробити відповідний висновок.

2.3.5 Відновити роботу призупиненого процесу.

Ще раз переглянути таблицю процесів для процесу, пов'язаного із скрипт-файлом.

Зробити висновки про стан процесу.

2.3.6 Створити два файли як символічні зв'язки на створений скрипт-файл із іменами як імена поточного файлу з додаванням цифр 2 і 3, відповідно, наприклад: *blazhko2.sh, blazhko3.sh*

2.3.7 Створити скрипт-файл на мові програмування *bash*, який:

- а) запускає два створені скрипт-файли у фоновому режимі;
- б) перед запуском кожного скрипт-файлу вказується затримка n секунд, де значення n – номер вашого варіанту;
- в) для кожного скрипт-файлу встановлюється пріоритет процесу, який буде на 10 пунктів нижче ніж пріоритет поточного процесу *bash*-оболонки.

2.3.8 Запустити файл-скрипт, створений у попередньому пункті.

Переглянути таблицю процесів для трьох вами запущених процесів, пов'язаних із скрипт-файлами. Зробити висновки за поточними значеннями *PPID, TTY, NI* та *%CPU*.

2.3.9 Зменшити на 9 пунктів пріоритет процесу, пов'язаного з першим файлом.

Переглянути таблицю процесів для трьох вами запущених процесів, пов'язаних із скрипт-файлами.

Зробити висновки щодо змін значень *% CPU* для кожного процесу: як вони змінилися?

2.3.10 Переглянути таблицю процесів для трьох скрипт-файлів через команду *top*.

Порівняти отримані значення *%CPU* зі значеннями, які були отримані у попередньому пункті завдання та надати висновки за результатом порівняння.

2.3.11 Завершити роботу трьох запущених процесів.

2.4 Підготовка процесу *Code Review* для надання рішень завдань лабораторної роботи на перевірку викладачем

2.4.1 На веб-сервісі *GitHub* зафіксувати зміни у файлі *README.md*

2.4.2 Скопіювати файли, які було створено у попередніх розділах завдань, в каталог «*Laboratory-work-7*» *Git*-репозиторію.

2.4.3 Оновити *Git*-репозиторій змінами нової гілки «*Laboratory-work-7*» з *GitHub*-репозиторію.

2.4.4 Оновити *GitHub*-репозиторій змінами нової гілки «*Laboratory-work-7*» *Git*-репозиторію.

2.4.5 Виконати запит *Pull Request*.

Примітка: Увага! Не натискайте кнопку «Merge pull request»!

Це повинен зробити лише рецензент, який є вашим викладачем!

Коли рецензент-викладач перегляне ваше рішення він виконає злиття нової гілки та основної гілки, натиснувши кнопку «*Merge pull request*». Якщо рецензент знайде помилки, він повідомить про це у коментарях, які з'являться на сторінці *Pull request*.

2.5 Оцінка результатів виконання завдань лабораторної роботи

Оцінка	Умови
+3 бали	1) всі рішення відповідають завданням 2) <i>Pull Request</i> представлено не пізніше найближчої консультації після офіційного заняття із захисту лабораторної роботи
-0.5 балів за кожну помилку	в рішенні є помилка, про яку вказано в <i>Code Review</i>
-1 бал	<i>Pull Request</i> представлено пізніше дати найближчої консультації після офіційного заняття із захисту лабораторної роботи за кожний тиждень запізнення
+2 бали	1) ви увійшли до трійки перших з вашої групи, хто без помилок виконав усі завдання; 2) отримано правильну відповідь на два запитання, які стосуються призначення команд зі знімків екранів