**Робота №5. Процеси в ОС UNIX і керування ними**

### Мета

**Оволодіння практичними навичками роботи з процесами — створення і знищення, керування процесами та їх аналіз**

### Завдання для самостійної підготовки

1. Вивчити (теорія — [1, гл. 2], [5, розд. 3, 4]; моніторинг і керування — [2, розд. 9.3]):

 поняття процесу та його характеристики;

 виведення на екран списку процесів і його аналіз;

 фонові й активні процеси;

 пріоритет процесів і його зміна;

 відправлення сигналів процесам, організація перехоплення сигналів;

 виконання завдань у системі в заданий час і з заданою періодичністю.

1. Ознайомитись з такими командами UNIX:

ps, ptree, pgrep, kill, pkill, fg, bg, jobs, crontab,

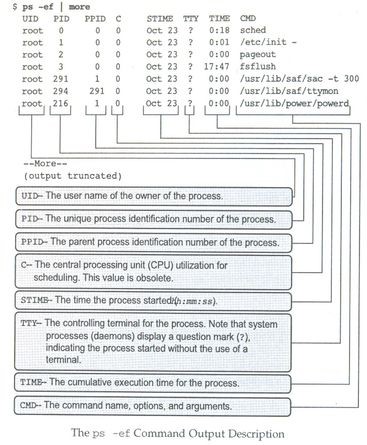
at

1. Зверніть увагу на використання параметру командного рядка "&"
2. Скласти послідовність команд для виконання необхідного варіанту завдання

## Довідковий матеріал

UNIX і UNIX-подібні системи – це багатозадачні операційні системи з розділенням часу. Це означає, що в системі одночасно виконується багато процесів. Кожний процес асоційований з певним користувачем, від імені якого цей процес діє. Для того, щоби переглянути список процесів, існує команда ps. Ця команда має багато ключів-модифікаторів, які визначають, яку саме інформацію про процеси повинна виводити команда. Слід зазначити, що в різних UNIX-подібних системах значення цих ключів може суттєво відрізнятись. Типові ключі: -a виводить інформацію про всі процеси, а не лише про процеси даного користувача, -l та -x виводять розширену інформацію про процес.

Кожний процес в системі має свій унікальний ідентифікатор – PID. За цим ідентифікатором можна звертатись до процесу. Крім того, кожний процес виникає не сам по собі – він має так званий батьківський процес, що характеризується ідентифікатором PPID(parent process ID). Таким чином утворюється ієрархія процесів, що бере початок від початкового процесу init.



Створення нового процесу у системі UNIX виконується у два етапи. Спочатку системний виклик fork() викликає “розщеплення” поточного процесу на два тотожні (різниця буде лише у тому, що процес-нащадок має інші PID і PPID). Далі виконується системний виклик exec(), який замінює контекст поточного процесу іншим контекстом. Розглянемо типовий приклад, коли один процес має запустити на виконання інший – командна оболонка sh виконує команду ls, для чого утворює новий процес, в якому виконується програма /usr/bin/ls. Для цього програміст, який пише код sh, повинен передбачити такі кроки:

1. Якщо під час синтаксичного аналізу командного рядка виявлено, що необхідно запустити на виконання деяку команду, то виконується виклик fork().
2. Наступна за fork() інструкція перевіряє значення, що повернув виклик. Якщо виклик fork() був успішний, то в результаті його виконання утворюється новий процес, тотожній батьківському, і операційна система буде виконувати їх обидва (в режимі розділення часу). Для процесу-нащадка значення, що повертає fork(), дорівнює 0, для батьківського процесу повертається значення PID нащадка.
3. Якщо це процес-нащадок, то виконується виклик exec(), якому в якості параметра передається ім’я файлу програми, яку необхідно запустити на виконання (в нашому прикладі – /usr/bin/ls). Якщо виклик був успішний, то на цьому виконання коду sh припиняється, і на його місце завантажується код ls.
4. Якщо це батьківський процес, то його дії залежать від того, які параметри були у командному рядку. Зокрема, якщо ls було запущено у пріоритетному режимі[[1]](#footnote-1), то виконання sh призупиняється до завершення процесунащадка, якщо ж ls було запущено у фоновому режимі, то sh видає необхідні повідомлення про запуск фонового процесу і продовжує свою роботу, тобто знову приймає і редагує командний рядок.

Процеси можуть взаємодіяти між собою за допомогою так званих сигналів. Існує обмежена кількість сигналів, які мають свої числові ідентифікатори і мнемонічні позначення. Сигнали діють як переривання, тобто вони призупиняють процес, до якого вони направлені, і викликають відповідний обробник сигналу. Деякі із сигналів мають чітко визначене значення і обробляються системним обробником. Інші можуть перехоплюватись процесом, тобто процес може встановлювати для цих сигналів свій обробник. Звичайно, існують певні узгодження щодо призначення певних сигналів, і програмістам слід дотримуватись їх при розробці своїх обробників.

Користувач також може відправити процесу сигнал, для цього існує команда kill. Формат команди:

### kill [-<signal>] <PID>

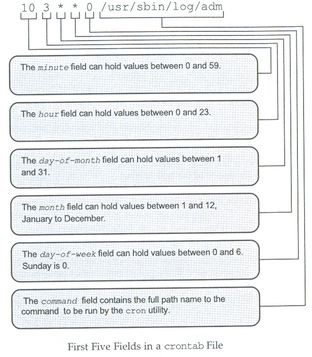
<signal>– це мнемонічне або числове позначення сигналу (наприклад, STOP, TERM, CONT, 9), а <PID> – ідентифікатор процесу, якому посилають сигнал. Якщо не вказати параметр <signal>, то буде відправлено сигнал завершення процесу TERM (15). Цей сигнал може перехоплюватись процесом, але існує сигнал KILL(9), який не перехоплюється і безумовно знищує процес (якщо у користувача є достатньо для цього прав). Таким чином можна зупинити будь-який свій процес, якщо над ним втрачене керування (або процес “завис”, що у UNIX-подібних системах трапляється нечасто, або користувач не знає, яку команду процес може сприйняти). Для цього слід зайти з іншої консолі (віртуального або фізичного терміналу) і дати команду kill -9 <PID>, де <PID> можна дізнатись за допомогою попередньої команди ps.

Команда ps без аргументів дає список лише процесів даного користувача і (зазвичай) лише пов’язаних з конкретним терміналом. При роботі в графічній оболонці, а також при роботі на персональному комп’ютері, де підтримується певна кількість так званих віртуальних терміналів, користувач може працювати в системі, використовуючи одночасно кілька терміналів (наприклад, на персональних комп’ютерах між ними можна переключатись комбінаціями клавіш Alt+F#, де F# – одна з функціональних клавіш). Однак в часи створення системи UNIX такі можливості не передбачались. Передбачалось, що користувач має доступ лише до одного терміналу. Тому було розроблено систему завдань і фонового виконання процесів, щоби користувач міг одночасно виконувати різні задачі.

Коли користувач дає команду з консолі, в системі запускається процес, або кілька процесів. Якщо в командному рядку утворюється конвеєр (наприклад, ls –l | wc –l), то всі процеси (у нашому прикладі ls і wc) запускаються одночасно. Разом вони складають так зване завдання. Завдання пов’язано з терміналом. Поки воно не завершиться, користувач не має можливості вводити наступну команду. Це так званий пріоритетний (foreground) режим виконання завдання. Щоби під час виконання завдання мати можливість запускати інше завдання, перше з них слід запустити в так званому фоновому (background) режимі. Для того, щоби запустити завдання в фоновому режимі, слід завершити рядок команди символом “&” (після пробілу). При цьому стандартний потік введення за умовчанням назначається порожньому файлу /dev/null. Слід врахувати, що завдання у фоновому режимі може намагатись здійснювати виведення на екран, заважаючи при цьому виводу пріоритетного процесу (спробуйте працювати, запустивши у фоновому режимі команду ping). Тому слід подбати, щоби фонові завдання здійснювали виведення у файли (див. Роботу №3). Завдання, що було запущено у пріоритетному режимі, можна перевести у фоновий. Для цього необхідно спочатку призупинити виконання завдання (комбінація клавіш Ctrl+Z). Далі можна поновити виконання завдання у пріоритетному режимі (команда fg) або у фоновому режимі (команда bg). Завдання мають свої номери. Переглянути їх можна за допомогою команди jobs.

Важливою можливістю є запуск певних завдань за розкладом. Це реалізується за допомогою програми-демона crond, який переглядає свої файли crontab (окремі для кожного користувача) і запускає завдання згідно розкладу від імені того користувача, в файлі якого це завдання задано. Для зміни розкладу завдань треба відредагувати файл crontab і після цього перезапустити crond, що може зробити лише root та ті користувачі, яким це право надано. Останнім рекомендується для цього користуватись командою crontab, яка після редагування автоматично перезапускає crond.

Користувачу дозволено виконувати команду crontab тільки за умови, що його ім’я зустрічається в файлі cron.allow і не зустрічається в файлі cron.deny (в різних UNIX-подібних системах ці файли можуть знаходитись в різних каталогах, наприклад /usr/lib/cron/ або /etc/cron.d/). Якщо обидва файли відсутні, то тільки root може користуватись командою crontab. Якщо cron.allow не існує, cron.deny існує, але не містить імен, то використовувати команду crontab дозволено усім. Файли cron.allow і cron.deny повинні містити по одному імені в рядку.



Існує спеціальна команда at, яка призначена для одноразового виконання заданої команди в будь-який заданий час. Правила дозволу і заборони користування командою at аналогічні crontab (файли at.allow і at.deny). Завдання, що задані командою at, виконуються тим же демоном crond.

## Завдання до виконання

1. Перегляньте список процесів користувача (вас).
2. Перегляньте повний список процесів, запущених у системі. При цьому гарантуйте збереження інформації від "утікання" з екрана (якщо процесів багато). Зверніть увагу на ієрархію процесів. Простежте через поля PID і PPID всю ієрархію процесів тільки-но запущеної вами команди, починаючи з початкового процесу init. Зверніть увагу на формування інших полів виводу.
3. Запустіть ще одну оболонку shell. Перегляньте повний список процесів, запущених вами, при цьому зверніть увагу на ієрархію процесів і на їхній зв'язок з терміналом. Використовуючи команду kill, завершіть роботу в цій оболонці.
4. Перегляньте список задач у системі і проаналізуйте їхній стан.

##### 5. Запустіть фоновий процес командою find / -name "\*c\*" -print > file 2> /dev/null & [[2]](#footnote-2)

1. Визначте його номер. Відправте сигнал призупинення процесу. Перегляньте список задач у системі і проаналізуйте їхній стан. Продовжить виконання процесу. Знову перегляньте список задач у системі і проаналізуйте його зміну. Переведіть процес в активний режим, а потім знову у фоновий. Запустіть цей процес із пріоритетом 5.
2. Виведіть на екран список усіх процесів, запущених не користувачем **root**.
3. Організуйте виведення на екран календаря <2010+ №варіанту> року через 1 хвилину після поточного моменту часу.

Організуйте періодичне (щоденне) видалення в домашньому каталозі усіх файлів з розширенням \*

1. Про пріоритетний і фоновий режими див. далі [↑](#footnote-ref-1)
2. Увага! Ця команда коректно працює в оболонках sh і bash, і не працює коректно у оболонках csh і tcsh (це пов’язано з правилами перенаправлення потоку помилок, див. Роботу №3). [↑](#footnote-ref-2)