Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили

Г. В. Горбань

ОПЕРАЦІЙНІ СИСТЕМИ:
підготовка до виконання лабораторних робіт

Методичні вказівки

Випуск 368

Миколаїв – 2021

УДК 004.451.9
Г 67

Рекомендовано до друку вченою радою Чорноморського національного університету імені Петра Могили (протокол № 5 від 10 червня 2021 р.).

Рецензент:
Любченко В. В., доктор технічних наук, професор, професор кафедри системного програмного забезпечення Державного університету «Одеська політехніка».

Г 67 **Горбань Г. В.** Операційні системи: підготовка до виконання лабораторних робіт. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2021. – 148 с. (Методична серія ; вип. 368).

Методичні вказівки містять опис 21 лабораторної роботи. Всі лабораторні роботи присвячені вивченню операційної системи Linux, кожна з яких відповідає окремій темі. Перші 8 робіт розглядають інтерактивний режим роботи в командній оболонці. Лабораторні роботи 9–12 присвячено програмуванню сценаріїв командного інтерпретатора. У роботах 13–16 розглядаються більш складні аспекти роботи в командній оболонці операційної системи Linux. Останні роботи присвячено системному програмуванню мовою C у Linux. Методичні вказівки призначено для студентів спеціальностей 121 «Інженерія програмного забезпечення» та 122 «Комп’ютерні науки», а також можуть бути корисними для студентів інших спеціальностей галузі знань 12 «Інформаційні технології».

УДК 004.451.9

© Горбань Г. В., 2021

ISSN 1811-492X ©
© ЧНУ ім. Петра Могили, 2021

Зміст

Лабораторна робота № 1. Знайомство з операційною системою Linux ... 5

Лабораторна робота № 2. Робота з файловою системою Linux ... 12

Лабораторна робота № 3. Перенаправлення стандартних потоків даних. Використання конвеєрів для виконання команд ... 17

Лабораторна робота № 4. Знайомство з текстовими редакторами vi та nano ... 23

Лабораторна робота № 5. Розмежування прав доступу до файлів та каталогів ... 29

Лабораторна робота № 6. Використання регулярних виразів у Linux ... 36

Лабораторна робота № 7. Обробка текстових даних. Користувацьке оточення ... 43

Лабораторна робота № 8. Пошук та архівація файлів ... 51

Лабораторна робота № 9. Знайомство з процесами. Контроль ресурсів та планування задач ... 56

Лабораторна робота № 10. Розробка сценаріїв мовою оболонки bash (частина 1). Змінні, командні файли, файли ініціалізації ... 62

Лабораторна робота № 11. Розробка сценаріїв мовою оболонки bash (частина 2). Аргументи командного рядка, управляючі структури, цикли. Перевірка умов командою test ... 70

Лабораторна робота № 12. Розробка сценаріїв мовою оболонки bash (частина 3). Використання функцій у сценаріях ... 78

Лабораторна робота № 13. Розробка сценаріїв bash для графічних робочих столів. Використання пакета zenity ... 87

Лабораторна робота № 14. Мережеві засоби Linux ... 99

Лабораторна робота № 15. Робота з потоковим редактором sed ... 105

Лабораторна робота № 16. Мова обробки вхідного потоку та построкового розбору gawk у Linux ... 113

Лабораторна робота № 17. Розробка програм
мовою C/C++ в ОС Linux 121

Лабораторна робота № 18. Робота з файлами
та каталогами Linux мовою C 127

Лабораторна робота № 19. Робота з процесами Linux
програмним шляхом мовою C. Системний виклик fork() 131

Лабораторна робота № 20. Використання сигналів
у ОС Linux програмним шляхом мовою C 136

Лабораторна робота № 21. Програмування потоків у Linux 142

Лабораторна робота № 1.
Знайомство з операційною системою Linux

Операційна система Linux розрахована на багато користувачів. Для гарантування безпечної роботи користувачів і цілісності системи доступ до неї повинен бути санкціонований.

Для кожного користувача, якому дозволено вхід в систему, заводиться спеціальне реєстраційне ім'я – *username* чи *login* і зберігається спеціальний пароль – *password*, що відповідає цьому імені.

Вхід у систему

1. З локального комп'ютера

Якщо ОС Linux встановлена на локальному комп'ютері або встановлена віртуальна машина з такою ОС, а також у системі встановлена графічна оболонка поряд зі звичайними алфавітно-цифровими терміналами, найкраще це зробити з алфавітно-цифрового термінала або його емулятора. Щоб перейти з графічної оболонки в алфавітно-цифровий термінал, потрібно натиснуть ***<ctrl>*** + ***<alt>*** + ***<Fx>***, де ***Fx*** – одна з функціональних клавіш ***F1, F2 ... F6.***

Таким чином користувач може потрапити у буквено-цифрову консоль, на номер якої вказує покажчик ***tty***.

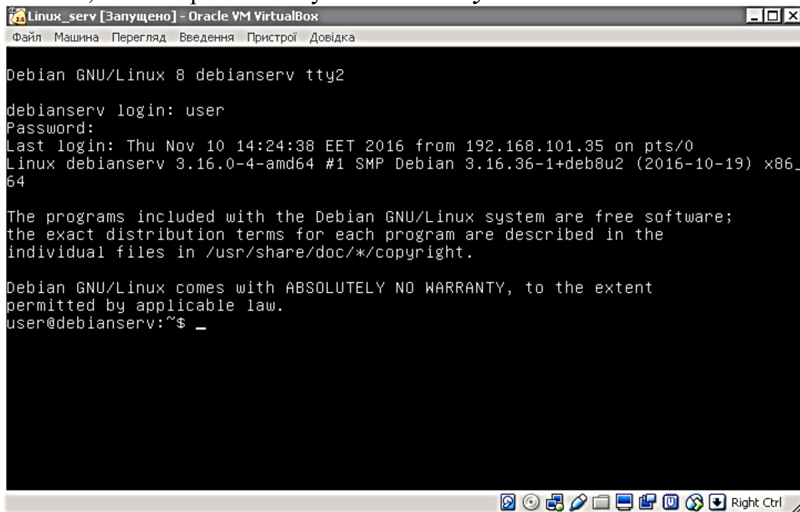


Рис. 1 – Режим командного рядка (друга консоль)

Наприклад, ***tty2*** означає, що користувач перебуває у другій консолі. За замовчуванням таких консолей 7. Причому сьома консоль є графічною. Переходити між консолями можна натискаючи клавіші ***<alt>*** + ***<Fx>***. Таким чином, повернутись до графічного режиму можна за допомогою клавіш ***<alt>*** + ***<F7>***.

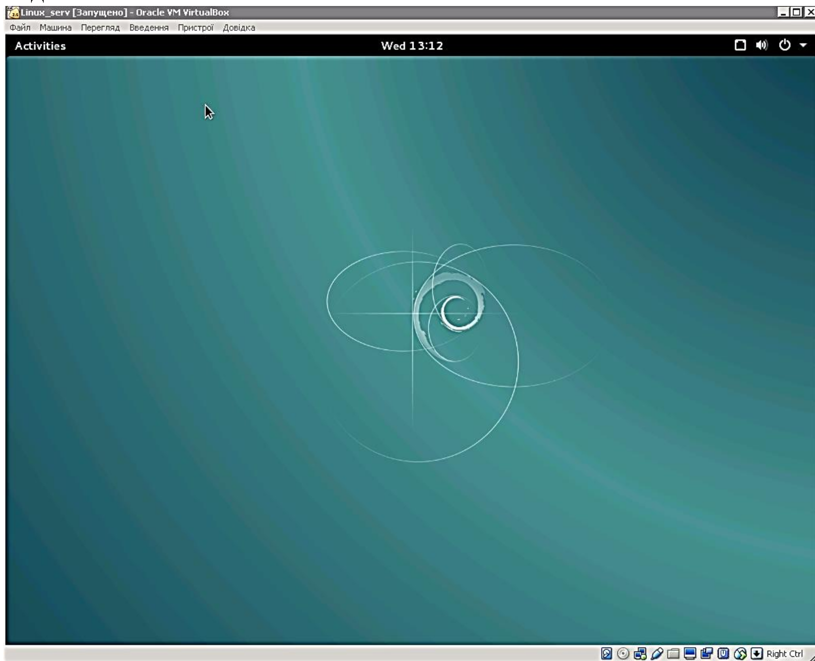


Рис. 2 – Графічний режим (сьома консоль)

2. З віддаленого комп'ютера

Вхід до ОС Linux з віддаленого комп'ютера можна здійснити за допомогою невеликої программи PuTTY, яка здійснює підключення до віддаленого комп'ютера з ОС Linux по локальній мережі за допомогою протоколу ssh. Недоліком PuTTY є те, що вона підтримує тільки консольний режим, до входу у графічний режим Linux з віддаленого комп'ютера використовуються зовсім інші механізми. Втім, для виконання більшої частини лабораторних робіт консольного режиму буде більш ніж достатньо.

Для того, щоб увійти до системи за допомогою PuTTY перший раз, для початку потрібно налаштувати з'єднання.

У локальній мережі університету встановлений сервер Linux (а саме – Debian 8.6), який має назву ***debianserv*** та IP-адресу ***192.168.96.164.*** Саме ці дані і потрібно ввести під час підключення через PuTTY.

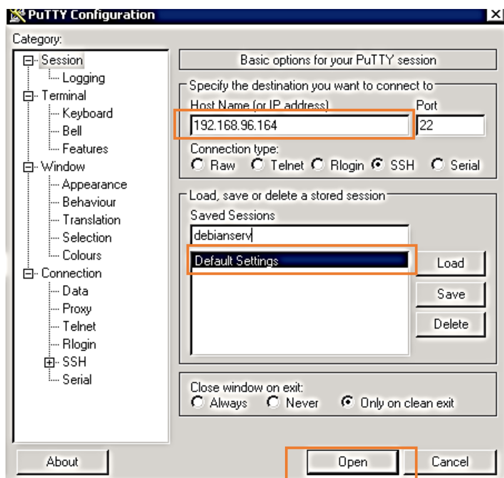


Рис. 3 – Налаштування підключення до сервера Linux у PuTTY

Зробивши один раз налаштування, його можна зберегти для того, щоб кожний раз не роботи нове налаштування для підключення до серверу.

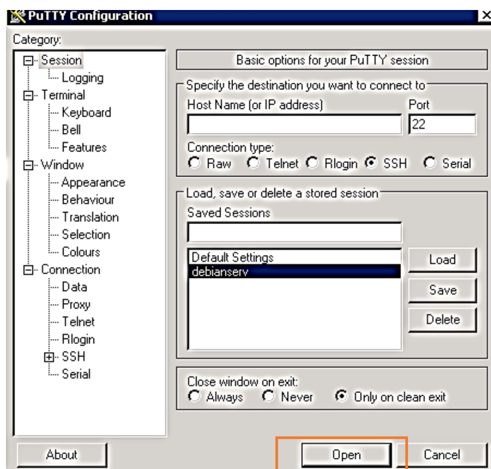


Рис. 4 – Підключення за збереженим налаштуванням у PuTTY

Після підключення PuTTY надає доступ до консольного режиму віддаленого сервера Linux.

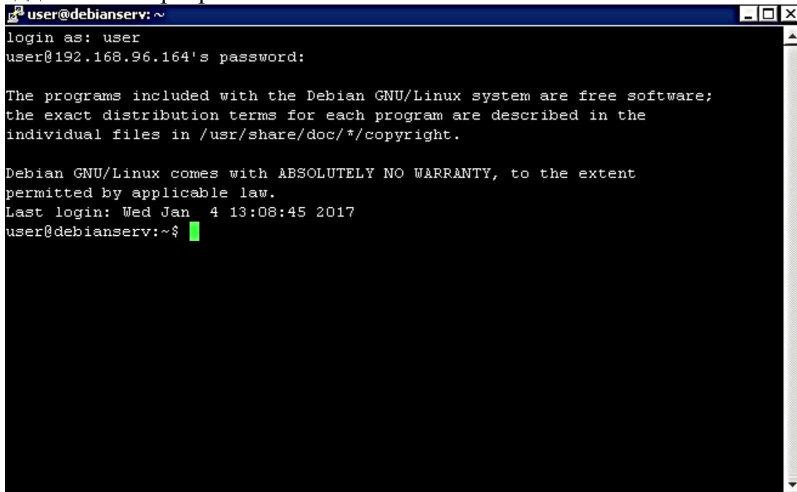


Рис. 5 – Сеанс роботи з віддаленим сервером Linux у PuTTY

Під час реєстрації в консолі на екрані з'являється напис, що пропонує ввести реєстраційне ім'я, як правило, це «login:». Набравши своє реєстраційне ім'я (у нашому випадку ***user***), натискаємо клавішу <Enter>. Після цього система запитує пароль, який відповідає введеному імені, видавши спеціальне запрошення – зазвичай «Password:».

Увага! Пароль потрібно набирати уважно, оскільки він на екрані не відображається.

Якщо все було зроблено правильно, на екрані з'являється запрошення до введення команд операційної системи, яке має вигляд ***<імя_користувача>@<назва_комп'ютера>:~$***.

Оскільки у поточному сеансі ім'ям користувача є *user*, а назвою комп'ютера – *debianserv*, то запрошення має вигляд ***user@debianserv:~$*** (далі під час приведення прикладів як запрошення в описах лабораторних робіт для скорочення буде використовуватись тільки знак ***$***).

Команда man – універсальний довідник

У процесі вивчення операційної системи Linux досить часто може знадобитись інформація про те, що робить та чи інша команда або системний виклик, які у них параметри і опції, для чого призначені деякі системні файли, яким є їхній формат та ін.

Для отримання довідки служить утиліта ***man***.

Користуватися утилітою *man* достатньо просто – набрати команду:

$ man < ім'я>,

де ***<ім'я>*** – це імя команди, утиліти, системного виклику, бібліотечної функції або файлу, інформацію про які потрібно дізнатись.

Приклад. *$ man passwd* – отримати інформацію про команду *passwd*, що встановлює пароль.

Щоб перегорнути сторінку отриманого опису, якщо він не помістився на екрані повністю, слід натиснути клавішу <*проміжок*>. Для прокрутки одного рядка потрібно скористатись клавішею <*Enter*>. Повернутися на сторінку назад дозволить одночасне натискання клавіш <*Ctrl*> і <*b*>. Припинити перегляд інформації можна з допомогою клавіші <*q*>.

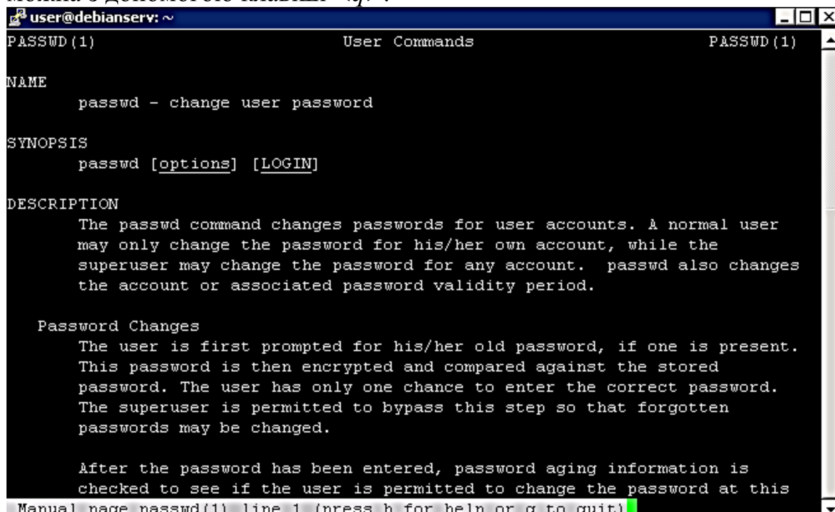


Рис. 6 – Приклад роботи команди утиліти *man* для виведення інформації про команду *passwd*

Іноді імена команд інтерпретатора і системних викликів або які-небудь ще імена збігаються. Тоді щоб знайти цікаву для вас інформацію, необхідно поставити утиліті man категорію, до якої належить ця інформація (номер розділу). У Linux прийнятий такий розподіл:

1. Виконувані файли або команди інтерпретатора.
2. Системні виклики.

3. Бібліотечні функції.
4. Спеціальні файли (зазвичай файли пристроїв)
5. Формат системних файлів і прийняті угоди.
6. Ігри (зазвичай відсутні).
7. Макропакет і утиліти – такі, як сам man.
8. Команди системного адміністратора.
9. Підпрограми ядра (нестандартний розділ).

Якщо ви знаєте розділ, до якого відноситься інформація, то утиліту *man* можна викликати в Linux з додатковим параметром:

$ man <номер_розділу> < ім'я>

В інших операційних системах цей виклик може виглядати інакше.

Для отримання точної інформації про розбиття на розділи, формі вказівки номера розділу і додаткові можливості утиліти *man* можна набрати команду:

$ man man

Завершення роботи системи Linux

Якщо ви працюєте з ОС Linux, не можна виключати комп'ютер простим відключенням живлення, як це було під MS-DOS. Вимикання ПК відбувається командою ***shutdown***.

Команда *shutdown* має такий синтаксис:

$ shutdown <options> <time> < warning-massage>

З опцій програми *shutdown* найбільш часто використовуються такі:

–h – повна зупинка системи (комп'ютер буде вимкнений);

–r – перезавантажити систему.

Параметр time вказує час, коли повинна бути виконана команда (не обов'язково виконувати її негайно). Час можна вказати у формі затримки. Наприклад, якщо ви хочете, щоб система зупинилася через 5 хвилин, введіть команду:

$ shutdown –r +5

що буде означати «зупинити систему через 5 хвилин і перезавантажитися після того, як робота буде коректно завершена».

$ shutdown -h 0

коли ви захочете просто вимкнути комп'ютер. Еквівалентом команди є команда ***halt***.

Слід зазначити, що команда shutdown може бути виконана тільки суперкористувачем (у Linux цей користувач завжди має ім'я ***root***).

Звичайні користувачі можуть тільки завершити поточний сеанс. Для цього потрібно виконати команду: ***$ exit*** або ***$ logout***

Завдання

1. За допомогою програми PuTTY під'єднатись до сервера debianserv у мережі.
2. Увійти до системи за допомогою свого логіна та пароля, попередньо зареєструвавшись у системі за допомогою викладача.
3. Використовуючи утиліту ***man***, знайти інформацію про команди ***pwd, who, whoami, last***
4. Використовуючи команди who, whoami, last отримаєте відомості про користувачів, що працюють у системі.

Контрольні питання

1. Як здійснити вхід у систему Linux з локального комп'ютера?
2. Як здійснити вхід у систему Linux з віддаленого комп'ютера?
3. Для чого призначена программа PuTTY?
4. Як знайти довідкову інформацію про команди Linux?
5. За допомогою яких команд можна отримати інформацію про користувачів, що працюють у системі?
6. Які дії виконує команда pwd?
7. Як вийти з системи Linux?

Лабораторна робота № 2.
Робота з файловою системою Linux

Файлова система Linux

Файлова система – це структура, за допомогою якої ядро операційної системи організовує і представляє користувачам ресурси пам'яті системи. Цього стосується і пам'ять на різного роду носіях інформації. Ємність і кількість носіїв є різною в різних системах. Ядро об'єднує ці ресурси в єдину ієрархічну структуру, яка починається в каталозі / і розгалужується, охоплюючи довільне число підкаталогів.

Ланцюжок імен каталогів, через які необхідно пройти для доступу до заданого файлу, разом з ім'ям цього файлу називається колійним ім'ям файлу (Pathname). Колійні імена можуть бути повними або відносними. В будь-який момент кожен процес прив'язаний до певного поточного каталогу. Відносні імена інтерпретуються з поточного каталогу.

Файлове дерево може бути довільного розміру. Однак існують певні обмеження, що залежать від конкретної операційної системи. Як правило, ім'я каталогу не повинно містити більше 256 символів, а у визначенні одного шляху не повинно бути більше 1023 символів.

В ОС Linux існує вісім типів файлів:

1. **Звичайний файл** – це просто послідовність байтів. Звичайний файл може містити виконувану програму, главу книги, графічне зображення та ін.
2. **Каталоги** – можуть містити файли будь-яких типів в будь-яких поєднаннях. Спеціальні імена **.** і **..** позначають відповідно сам каталог і його батьківський каталог.
3. **Файли пристроїв** – дозволяють програмам взаємодіяти з апаратними засобами і периферійними пристроями системи. У конфігуруванні ядра до нього додаються ті модулі, які знають, як взаємодіяти з кожним з пристроїв системи. За всю роботу з керування конкретним пристроєм відповідає спеціальна програма, яка називається драйвером пристрою.
4. **Доменні гнізда (sokets) Linux** – це з'єднання між процесами, які дозволяють їм взаємодіяти, не підпадаючи під вплив інших процесів. Доменні гнізда Linux локальні для конкретного хост-комп'ютера. Звернення до них здійснюється через об'єкт файлової системи, а не через мережевий порт.
5. **Іменовані канали** – також, як і доменні гнізда, забезпечують взаємодію двох незв'язаних процесів, які виконуються на одній машині.

6. **Жорсткі посилання** – це скоріше не тип файлу, а його додаткове ім'я. У кожного файлу є як мінімум одне посилання. Як правило, це ім'я, під яким він був створений. Додаванням посилання створюється псевдонім файлу. Посилання неможливо відрізнити від імені файлу, до якого вона приєднана: в ОС Linux вони ідентичні. Linux підраховує кількість посилань, що вказують на кожен файл, і не звільняє блоки даних файлу до тих пір, поки не видалить його останнє посилання.

7. **Символічні посилання** – забезпечують можливість вказувати замість імені файлу ім'я посилання. Символічне посилання містить ім'я файлу, на який воно посилається.

Імена файлів можуть складатися з будь-яких символів, за винятком слеша і символу з кодом нуль. Максимальна довжина імені файлу визначається конкретною системою. Для кожного файлу визначено власника цього файлу і групу власника цього файлу. Для кожного файлу визначаються права доступу власника файлу, групи, всіх інших. Є три типи прав доступу: читання, запис, виконання / пошук.

Змінити права доступу до файлу може тільки власник і привілейований користувач (root).

Особливості формування файлового простору

Файловий простір Unix-систем є ієрархією файлів, яка має єдиний спільний корінь – так званий кореневий каталог, що позначається знаком «/». Щоб однозначно ідентифікувати будь-який файл, можна вказати шлях до цього файлу від кореневого або поточного каталогу. Всі елементи шляху відокремлюються один від одного символом «/». Якщо перший символ рядка також «/», то шлях бере початок в кореневому каталозі, в іншому випадку – в поточному. Шлях з єдиним ім'ям позначає файл в поточному каталозі.

Приклади:

– ***`docs.ps`*** – файл з ім'ям docs.ps в поточному каталозі;

– ***`/usr/doc/FAQ/README`*** – файл з ім'ям README в каталозі ***`/usr/doc/FAQ`***;

– ***`work/thesis.tex`*** – файл ***`thesis.tex`*** в підкаталозі ***`work`*** поточного каталогу.

Поняття поточного каталогу дещо відрізняється від такого в системі MS-DOS або Windows. В Unix у кожного процесу власний поточний каталог. Кореневий каталог файлового дерева Unix зазвичай містить такі підкаталоги (в різних системах ця структура може відрізнятися):

– ***/bin*** – мінімальний набір виконуваних файлів, необхідний для працездатності системи;

– ***/etc*** – файли конфігурації системи;

– ***/dev*** – файли пристроїв;

– ***/home*** – домашні каталоги користувачів;

– ***/lib*** – основні системні бібліотеки та модулі;

– ***/root*** – каталог адміністратора системи ***root***;

– ***/proc*** – файли-образи процесів, що виконуються;

– ***/sbin*** – мінімальний набір утиліт адміністратора;

– ***/tmp*** – каталог для тимчасових файлів;

– ***/usr*** – основний обсяг файлів системи: встановлені програми, бібліотеки, вихідні коди ядра, файли даних та інше;

– ***/var*** – каталог для інформації, що змінюється (облікових даних, поштових скриньок, черг принтера, відформатованих сторінок документації, логів та ін.).

Слід зазначити, що символ «/» не є частиною імен каталогів, а лише вказує, що такі елементи знаходяться в кореневому каталозі. У кожному каталозі також існує два особливих «підкаталоги» з іменами «***..***» і «***.***». Перший з них служить вказівником на однозначно визначений батьківський каталог (що знаходиться на рівень вище), а другий – на поточний каталог. Наприклад, шлях «***../readme***» Вказує на файл «***readme***», який знаходиться в батьківському каталозі (на щабель вище), а шлях «***./readme.now***» вкаже на файл «***readme.now***», який знаходиться в поточному каталозі.

Основні команди для роботи з файлами та каталогами

1. ***$ pwd*** – визначити поточний каталог.

2. ***$ cd <каталог>*** – змінює поточний каталог на зазначений. Якщо параметр опущений, то поточним стає домашній каталог.

3. ***$ ls [-alFR] <файл ...>*** – виводить список файлів в зазначеному (або поточному) каталозі.

Ключі:

– ***-a*** – змушує виводити всі файли;

– ***-l*** – служить для виведення докладної інформації про файли;

– ***-F*** – призводить до того, що до імен каталогів додається символ «/», до імен посилань – «@», до імен виконуваних файлів – «*»;

– ***-R*** – виводиться список файлів не тільки зазначеного каталогу, але і його підкаталогів.

4. ***$ mkdir <каталог>*** – створює каталог.
5. ***$ rmdir <каталог>*** – видаляє каталог.
6. ***$ cp [-rp] <файл1> <файл2>, $ cp [-rp] <файл ...> <каталог>*** – копіює один файл в інший або копіює файли в зазначений каталог.

Ключі:

– ***-r*** – призначений для копіювання каталогів;

– ***-p*** – дозволяє зберігати власників файлів, режим доступу та час доступу і зміни.

7. ***$ rm [-r] <файл ...>*** – видаляє файли. Ключ ***-r*** дозволяє видаляти каталоги.
8. ***$ mv <файл1> <файл2>, $ mv <файл ...> <каталог>*** – переміщує один файл в інший або переміщує файли в заданий каталог.
9. ***$ ln [-s] <файл> <посилання>*** – створює посилання на файл. Ключ ***–s*** вказує на те, що буде створене символічне посилання, інакше буде створене жорстке посилання.
10. ***$ touch <файл>*** – створення порожнього файлу;
11. ***$ less <файл>*** – переглянути вміст файлу.

Завдання

Порада! Перед виконанням певної команди спочатку бажано ознайомитись з її додатковими можливостями за допомогою довідника ***man***.

Після кожного виконаного завдання не забувайте робити скріншоти для їх вставки у звіт!

1. Увійдіть у систему Linux та визначте поточний каталог, в якому ви перебуваєте.
2. Перейдіть до кореневого каталогу.
3. Перегляньте вміст кореневого каталогу, використовуючи різні режими.
4. Зробіть копію екрана для використання у звіті з лабораторної роботи.
5. Поверніться в домашній каталог.
6. Створіть у домашньому каталозі каталог «***test***» та перейдіть у нього.
7. Створіть каталог «***test2***».
8. Створіть файл «***text***» в каталозі «***test2***».
9. Перейменуйте файл «***text***» в «***textSIT***».

10. Скопіюйте у файл «***textSIT***» дані з файла ***/etc/passwd*** та перегляньте його вміст.
11. Скопіюйте файл «***textSIT***» в каталог «***test2***» під ім'ям «***copy.txt***».
12. Створіть жорстке посилання «***link***» на файл «***copy.txt***».
13. Створіть символічне посилання «***simlink***» на файл «***copy.txt***».
14. Перегляньте результати в поточному каталозі. Зверніть увагу на те, як відрізняється відображення жорстких та символічних посилань та на їх розмір.
15. Видаліть файл «***copy.txt***» та подивіться вміст поточного каталогу. Зверніть увагу на те, як змінилось відображення символічних посилань.
16. Видаліть всі створені вами у цій лабораторній роботі файли, посилання і каталоги.

Контрольні питання

1. Чим відрізняються результати виконання команд ***ls -F*** і ***ls -la***?
2. За допомогою якої команди і як можна перемістити файл в інший каталог?
3. Куди здійснюється перехід під час виконання команд ***cd*** без параметрів та з параметром «**-**»?
4. Як перейти до домашнього каталогу іншого користувача?
5. Що таке жорстке та символічне посилання та у чому різниця між ними?
6. Як здійснити перегляд підкаталогів і їх вмісту?
7. Як здійснити перегляд прихованих файлів у домашньому каталозі?
8. Як здійснити створення нового каталогу і необхідних підкаталогів рекурсивно?
9. Як здійснити рекурсивне копіювання всіх файлів з одного каталогу в інший?
10. Як здійснити рекурсивне копіювання всіх файлів і підкаталогів з одного каталогу в інший?
11. Як рекурсивно видалити всі файли і підкаталоги в певному каталозі?

Лабораторна робота № 3. Перенаправлення стандартних потоків даних. Використання конвеєрів для виконання команд

Стандартні потоки введення-виведення

З кожною програмою, що запускається з командного рядка Unix, пов’язані три стандартних потоки даних:

– стандартний потік введення (stdin);
– стандартний потік виведення (stdout);
– стандартний потік помилок (stderr).

1. Стандартний потік введення

Команди, які потребують вхідних даних, зазвичай читають інформацію зі стандартного потоку введення. Наприклад, команда ***wc*** підраховує кількість рядків, слів та символів у вхідних даних. Якщо запустити цю команду без аргументів, то ***wc*** чекатиме вхідні дані з термінала (щоб закінчити введення даних, потрібно натиснути комбінацію клавіш ***Ctrl-D***):

```
$ wc
two words
<Ctrl-D>
1 2 10
```

У цьому прикладі команда ***wc*** прочитала введений користувачем текст зі стандартного потоку введення (куди користувач ввів текст «two words»). За замовчуванням цей потік з’єднаний з терміналом (з клавіатурою) користувача, але допускається його перенаправлення. Щоб зв’язати дані стандартного вхідного потоку з довільним файлом, можна використовувати операцію перенаправлення «**<**», наприклад:

```
$ wc < /etc/passwd
28 37 1052
```

У такому випадку команда wc вже не вимагає введення з клавіатури, оскільки вона вже отримала вхідні дані з файлу ***/etc/passwd***. Зауважимо, що ця команда може мати практичне застосування – перша цифра означає кількість рядків у файлі ***/etc/passwd***, що відповідає кількості користувачів, зареєстрованих в системі.

2. Стандартний потік виведення

Стандартний потік виведення – це потік, куди програми записують вихідні дані. У попередньому прикладі команда ***wc*** виводила результат (три числа) саме в цей потік. Так само працюють і більшість інших неінтерактивних команд (включаючи ***echo***, ***pwd*** і ***ls***).

Подібно стандартному потоку введення вихідний потік спочатку пов'язаний з терміналом і також допускає перенаправлення. Для зв'язування стандартного потоку виведення з файлом використовується операція «***>***», наприклад:

```
$ ls> filelist.txt
```

У цьому прикладі команда ***ls***, замість того, щоб вивести список файлів на екран, записала його у файл з ім'ям «***filelist.txt***». При цьому, якщо файл з таким ім'ям не існував, він буде створений, а інакше його старий вміст буде втрачений.

Існує й інша можливість перенаправлення виведення, коли нові вихідні дані будуть дописані в кінець існуючого файлу. Для цього використовується операція «***>>***». У наступному прикладі поточні дата і час будуть дописані в кінець файлу з ім'ям «***dates.txt***»:

```
$ date >> dates.txt
```

3. Стандартний потік помилок

Повідомлення про помилки виводяться в стандартний потік помилок. Наприклад, нехай виконується спроба отримати список файлів в каталозі без відповідних прав доступу:

```
$ ls -l /home/ftp/bin/
ls: /home/ftp/bin/: Access denied
```

У цьому випадку команда ***ls*** вивела повідомлення у стандартний потік помилок.

Перенаправлення стандартного потоку помилок здійснюється не так просто, як стандартного виведення. Щоб перенаправити стандартний потік помилок, потрібно вказати його дескриптор файлу. Програма може здійснювати виведення в будь-який з кількох нумерованих файлових потоків. Перші три з них ми згадали як стандартні потоки введення, виведення і помилок. Командна оболонка посилається на них як на файлові дескриптори 0, 1 і 2 відповідно.

Командна оболонка підтримує синтаксис перенаправлення файлів з використанням номерів файлових дескрипторів. Оскільки стандартному потоку помилок відповідає файловий дескриптор 2, ми можемо перенаправити його, як показано нижче:

```
$ ls -l /home/ftp/bin/ 2> last-error.txt
```

Операції перенаправлення введення-виведення можна комбінувати, наприклад:

```
$ wc < /etc/passwd 2> errors.txt > result.txt
```

У цьому випадку стандартний потік помилок буде перенаправлений у файл ***errors.txt***, а стандартний потік виведення – у файл ***result.txt***.

Перенаправлення стандартних потоків виведення і помилок в один файл

Іноді необхідно зберегти все виведення команди в один файл. Для цього потрібно перенаправити відразу два потоки: виведення і помилок. Зробити це можна двома способами.

Перший (традиційний) працює в старих версіях командної оболонки:

```
ls -l /etc/passwd > output.txt 2>&1
```

Тут виконується два перенаправлення. Спочатку – перенаправлення стандартного потоку виведення у файл ***output.txt***, а потім, з використанням нотації ***2>&1***, – перенаправлення файлового дескриптора 2 (стандартний потік помилок) у файловий дескриптор 1 (стандартний потік виведення).

Сучасні версії bash підтримують другий, більш простий метод виконання перенаправлення цього виду:

```
$ ls -l / etc/passwd &> output.txt
```

У цьому прикладі використовується єдиний оператор ***&>***, що перенаправляє стандартний потік виведення і стандартний потік помилок у файл ***output.txt***.

Команда cat як універсальна команда для створення, копіювання та об'єднання файлів

Команда ***cat*** часто використовується для створення файлів (хоча можна скористатися й командою ***touch***). За командою cat на стандартний вивід (тобто на екран) виводиться вміст зазначеного файлу (або декількох файлів, якщо їхні імена послідовно задати як аргументи команди). Якщо вивід команди cat перенаправляти у файл, то можна одержати копію якогось файлу:

```
$ cat file1 > file2
```

Первісне призначення команди ***cat*** саме й припускало перенаправлення виведення, тому що ця команда створена для конкатенації, тобто об'єднання декількох файлів в один:

$ cat file1 file2 ... file > new-file

Саме можливості перенаправлення введення та виведення цієї команди й використовуються для створення нових файлів. Для цього на вхід команди ***cat*** направляють дані з потоку стандартного введення (тобто із клавіатури), а виведення команди – у новий файл:

$ cat > newfile

Після того, як ви надрукуєте все, що хочете, натисніть комбінацію клавіш ***<Ctrl>+<D>*** або ***<Ctrl>+<C>***, і все, що ви ввели, буде записано в newfile. Звичайно, у такий спосіб створюються, в основному, короткі текстові файли.

Конвеєри

Існує інший корисний спосіб перенаправлення вводу-виведення – конвеєри команд. Операція «**|**» дозволяє перенаправити стандартний потік виведення однієї команди на стандартний потік введення іншої команди. Наприклад:

$ ls -l /etc | less

У цьому прикладі команда ***ls*** виводить довгий список файлів у каталозі ***/etc***, ці дані потрапляють на вхід програми ***less***, яка дозволяє перегортати текст за допомогою клавіш управління курсором. Так здійснюється «об'єднання» двох незалежних команд в один «конвеєр».

Конвеєри часто використовуються для виконання складних операцій з даними. Вони дозволяють об'єднати разом кілька команд. Часто команди, використовувані таким способом, називають ***фільтрами***. Фільтри приймають введення, змінюють його певним чином і виводять результат.

Деякі команди-фільтри:

1. ***$ sort [<файл>]*** – сортування вхідних даних за алфавітом.
2. ***$ uniq [<файл>]*** – пошук або видалення рядків, що повторюються. Використання команди без ключа видалить всі дублікати та виведе рядки без них назалежно від того, чи є дублікати у певних рядків, чи ні. Використання команди з ключем ***–d*** навпаки виведе список рядків, що мають дублікати.
3. ***$ wc <файл >***.
4. ***$ grep <шаблон> [<файл...>]*** – пошук рядків, що відповідають шаблону.
5. ***$ head (tail)*** – виведення перших (останніх рядків файлу).

Використання фільтрів у конвеєрах

Розглянемо більш складний приклад використання конвеєрів. Припустимо, що нам потрібно знайти всі файли в списку програм, які мають у своєму імені послідовність символів «***zip***». Результати такого пошуку можуть підказати нам, які програми в системі мають відношення до стискання файлів. Такі програми можуть знаходитись у каталогах ***/bin*** та ***/usr/bin***.

Цю задачу можна виконати за допомогою такої послідовності дій:

```
$ ls /bin /usr/bin | sort | uniq | grep zip
bunzip2
bzip2
gunzip
gzip
unzip
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
```

У цьому випадку були використані конвеєри за певною послідовністю дій:

1. Виведення вмісту каталогів (команда ***ls***).
2. Сортування списку вмісту каталогів в алфавітному порядку, оскільки під час виведення спочатку знаходився вміст ***/bin***, а потім – ***/usr/bin*** (команда ***sort***).
3. Видалення у відсортованому списку записів-дублікатів, оскільки у каталогах ***/bin*** та ***/usr/bin*** можуть знаходитись каталоги або файли з однаковими назвами (команда ***uniq***).
4. Пошук у отриманому списку назв каталогів або файлів, які мають у собі послідовність симолів «***zip***» (команда ***grep***).

Завдання

1. Створити порожній файл ***first.txt***.
2. Додати рядок тексту в кінець файлу «***Hello, world***» шляхом перенаправлення виводу.
3. Переглянути вміст файлу ***first.txt***.
4. Скопіювати вміст файлу ***first.txt*** у файл ***1.txt***.
5. Створити новий файл ***second.txt*** та записати у нього декілька рядків на власний розсуд.

6. Створити файл ***home.txt***, записавши у нього назви домашніх каталогів всіх користувачів.
7. Підрахувати кількість користувачів системи, що мають домашній каталог, як кількість рядків файлу ***home.txt***.
8. Об'єднати файли ***orig.txt***, ***second.txt*** та ***home.txt*** в один з ім'ям ***big.txt***.
9. Переглянути файл ***big.txt***, переконавшись, що він містить рядки з перерахованих файлів.
10. За допомогою конвеєрів знайдіть у файлі ***home.txt*** назви каталогів користувачів, що є студентами вашої групи, які мають логіни, що складаються з цифр, та виведіть їх у відсортованому порядку.

Контрольні питання

1. Як перенаправити стандартний потік введення?
2. Як перенаправити стандартний потік виведення у файл:
   - з його перезаписом;
   - з додаванням у кінець існуючого файлу.
3. Як перенаправити стандартний потік помилок у файл?
4. Як перенаправити стандартні потоки виведення та помилок в один файл?
5. Для чого призначена команда ***cat***? Наведіть приклади різних використань команди.
6. Для чого призначені конвеєри та як запустити їх на виконання?
7. Наведіть приклади команд-фільтрів. Для чого вони призначені?

Лабораторна робота № 4.
Знайомство з текстовими редакторами
vi та nano

Текстовий редактор vi

Текстовий редактор **vi** є ідеологічним нащадком стандартного для системи UNIX текстового редактора **ed**. Для роботи в редакторах такого типу необхідно знати, в якому з режимів – командному або текстовому – знаходиться користувач, який попросив ресурси редактора. У командному режимі об’єктом роботи редактора є файл в цілому, в текстовому – окремий рядок (рядки) файлу. Власне робота з редактором **vi** починається з виклику:

$ vi [<ім’я_файлу>],

де ***<ім’я_файлу>*** – ім’я текстового файлу, відносно якого виконуються модифікуючі цей текст процедури редактора **vi**. За відсутності в системі файлу з таким ім’ям він буде створений для початку на час сеансу з **vi**, а за наявності файлу з такою назвою до виклику **vi** відобразить перші рядки цього файлу на екрані відеотерміналу. У будь-якому випадку після виклику редактора він переходить в командний режим.

Більшість дистрибутивів Linux мають не справжній редактор **vi**, а його покращену заміну з іменем **vim** (скорочено від Vi Improved – Vi покращений).

```
                              VIM - Vi IMproved

                               version 7.4.576
                           by Bram Moolenaar et al.
          Modified by pkg-vim-maintainers@lists.alioth.debian.org
                Vim is open source and freely distributable

                       Become a registered Vim user!
              type  :help register<Enter>   for information

              type  :q<Enter>               to exit
              type  :help<Enter>  or  <F1>  for on-line help
              type  :help version7<Enter>   for version info

                       Running in Vi compatible mode
              type  :set nocp<Enter>        for Vim defaults
              type  :help cp-default<Enter> for info on this
```

Рис. 1 – Зовнішній вигляд текстового редактора **vim** для нового файлу

Для користувача, що знаходиться у командному режимі візуального редактора **vi**, можливо виконання таких дій, необхідних для редагування дій:

– переміщення у файлі, що відкритий для редагування (переміщення курсора або маркера);
– контекстний пошук і заміна у файлі;
– вихід з редактора зі збереженням або без збереження змін у файлі;
– введення команд редагування у позиційованій області файлу.

Для повернення з текстового режиму в командний користувачеві, що знаходиться в текстовому режимі, достатньо хоча б один раз натиснути клавішу ***<ESC>***. Для користувача, що знаходиться в текстовому режимі візуального редактора **vi**, доступні наступні, необхідні для редагування, дії:

– введення (вставка) тексту з позиції / після позиції, зазначеної маркером;
– введення (вставка) тексту під новостворений порожній рядок над рядком файлу, зазначеним маркером.

Для пересування по екрану використовуються такі клавіші:

– ***h*** – ліворуч;
– ***j*** – вниз;
– ***k*** – вгору;
– ***l*** – праворуч;
– ***^*** – перейти на початок рядка;
– ***$*** – перейти до кінця рядка;
– ***w*** – перейти до початку наступного слова;
– ***e*** – перейти до кінця слова;
– ***b*** – перейти на початок слова;
– ***cw*** – замінити слово;
– ***cl*** – замінити літеру;
– ***gg*** – перейти на початок документа;
– ***G*** – перейти в кінець документа;
– ***<номер>G*** – перейти на рядок з номером <номер>.

Можна використовувати зв’язок ***<номер><команда переміщення>***. Наприклад:

- ***3w*** – перейти до початку четвертого слова відносно поточної позиції;
- ***4e*** – перейти до кінця четвертого слова відносно поточної позиції.

Для переведення редактора **vi** в режим введення / вставки тексту відразу за позицією курсора в редагованому файлі служить команда ***a***, а для виконання введення / вставки тексту в позиції курсора – команда ***i***.

Після введення будь-якої з цих команд редактор переходить в текстовий режим до натискання клавіші ***<ESC>***, що повертає його знову в командний режим. У текстовому режимі користувач має можливість безпосереднього введення з клавіатури символів тексту, користуватися клавішею ***<Backspace>*** і клавішами табуляції.

Для переведення редактора **vi** з командного режиму в режим введення тексту у знову утворений порожній рядок над позиціонованим служить команда ***O***, а для тих же дій над позиціонованим рядком – команда ***o***. В обох випадках буде спочатку утворений порожній рядок, курсор буде переміщений в першу позицію цього нового рядка і **vi** перемкнеться в текстовий режим до натискання ***<ESC>***.

Виконання в зоні команд дій із запису змін і виходу з редактора кодується послідовністю:

: wq

а для відмови від запису змін, зроблених під час редагування, використовується послідовність:

: q!

Командний інтерфейс візуального редактора **vi** набагато багатше розглянутих тут випадків, однак уже цих прикладів достатньо, щоб усвідомити гнучкість цього програмного засобу.

Для більш детального ознайомлення рекомендується переглянути навчальну програму з редактора **vi**. Для цього необхідно в терміналі ввести команду:

$ vimtutor

Текстовий редактор nano

Nano – консольний текстовий редактор для UNIX і UNIX-подібних операційних систем, заснований на бібліотеці curses і поширюваний під ліцензією GNU GPL. На сьогодні включено в деякі дистрибутиви Linux (включаючи Debian та Ubuntu) за замовчуванням і не потребує встановлення.

Щоб запустити **nano**, слід відкрити термінал і виконати:

$ nano [<ім'я_файлу>]

Редактор **nano** розроблено для емуляції функціональності та простоти використання оригінального редактора **UW Pico**. Редактор розбитий на 4 основні частини: верхній рядок містить версію програми, поточне ім'я файлу, що редагується, і чи були внесені зміни в

поточний файл. Друга частина – це головне вікно редагування, в якому відображено редагований файл. Рядок стану – 3 рядок знизу – показує різні важливі повідомлення. Два рядки внизу показують найбільш часто використовувані комбінації клавіш.

Рис. 2 – Зовнішній вигляд текстового редактора **nano** для порожнього файлу

Система позначень комбінацій клавіш така:

– комбінації з ***<Ctrl>*** позначені символом ***«^»*** і вводяться за допомогою натиснутої клавіші ***<Ctrl>*** або подвійного натискання ***<Esc>***;

– комбінації з ***<Esc>*** позначені символом «***Meta***» (***«m»***) і можуть бути введені за допомогою кнопок ***Esc***, ***Alt*** або ***Meta*** залежно від використовуваної клавіатури.

Також натискання ***<Esc>*** двічі і подальше введення тризначного числа від 000 до 255 введе відповідний символ.

Нижче перераховані деякі важливі комбінації клавіш, що використовуються у редакторі **nano**:

– ***<Ctrl>+<G>*** або ***<F1>*** – показати довідку;

– ***<Ctrl>+<X>*** або <*F2*> – закрити поточний буфер / вийти з **nano**;

– ***<Ctrl>+<O>*** або ***<F3>*** – записати поточний файл на диск;

– ***<Ctrl>+<J>*** або ***<F4>*** – вирівняти поточний абзац;

– ***<Ctrl>+<R>*** або ***<F5>*** – вставити інший файл в поточний;

– ***<Ctrl>+<W>*** або ***<F6>*** – шукати текст або регулярний вираз;

– ***<Ctrl>+<Y>*** або ***<F7>*** – перейти на попередній екран;

– ***<Ctrl>+<V>*** або ***<F8>*** – перейти на наступний екран;

– ***<Ctrl>+<K>*** або ***<F9>*** – вирізати поточний рядок і зберегти її в буфері обміну;

– ***<Ctrl>+<U>*** або ***<F10>*** – вставити вміст буфера обміну в поточний рядок;

– ***<Ctrl>+<C>*** або ***<F11>*** – показати положення курсора;

– ***<Ctrl>+<T>*** або ***<F12>*** – перевірити орфографію, якщо є;

– ***<ESC>+<\>*** або ***<ESC>+|*** – на перший рядок файлу;

– ***<ESC>+</>*** або ***<ESC>+?*** – на останній рядок файлу;

– ***<Ctrl>+<_>*** або ***<Esc>+<G>*** – перейти на вказаний номер рядка і ряд;

– ***<Ctrl>+<\>*** або ***<Esc>+<R>*** – замінити текст або регулярний вираз;

– ***<Ctrl>+<^>*** або ***<Esc>+<Alt>*** – відзначити текст в поточній позиції курсора;

– ***<Esc>+<W>*** – повторити останній пошук;

– ***<Esc>+<^>*** або ***<Esc>+<6>*** – копіювати поточний рядок і зберегти її в буфері обміну;

– ***<Esc>+<}>*** – збільшити відступ рядка;

– ***<Esc>+<{>*** – зменшити відступ рядка;

– ***<Ctrl>+<F>*** – вперед на один символ;

– ***<Ctrl>+<B>*** – назад на один символ;

– ***<Ctrl>+<Space>*** – вперед на одне слово;

– ***<Esc>+<Space>*** – назад на одне слово;

– ***<Ctrl>+<P>*** – на попередній рядок;

– ***<Ctrl>+<N>*** – на наступний рядок;

– ***<Ctrl>+<Alt>*** – на початок поточного рядка;

– ***<Ctrl>+<E>*** – в кінець поточного рядка;

– ***<Esc>+<(>*** або ***<Esc>+<9>*** – на початок поточного абзацу; потім наступного абзацу;

– ***<Esc>+<)>*** або ***<Esc>+<0>*** – кінець поточного абзацу; потім наступного абзацу;

– ***<Esc>+<]>*** – на відповідну дужку;

– ***<Esc>+<->*** або ***<Esc>+<_>*** – прокрутити один рядок вгору, не переміщаючи курсор;

– ***<Esc>+<+>*** або ***<Esc>+<=>*** – прокрутити один рядок вниз, не переміщаючи курсор;

– ***<Esc>+<<>*** або ***<Esc>+<,>*** – переключити на попередній буфер;

– ***<Esc>+<>>*** або ***<Esc>+<.>*** – переключити на наступний буфер.

Завдання

1. Створіть у вашій домашній директорії піддиректорію ***Text Files***.
2. Згідно з вашим порядковим номером у загальному списку групи та файлу «**Додаток до ЛР №4**», занесіть перший текст у файл ***mytext.1.txt*** за допомогою редактора **vi**, а другий текст – у файл ***mytext.2.txt*** за допомогою редактора **nano**.
3. На початку тексту в обох файлах додайте шифр своєї групи і свій номер за журналом групи. Результат зберегти у файлі.
4. У кінці тексту в обох файлах додайте у текст що-небудь на зразок «**Your most humble obedient servant, ..., student**», результат зберегти.
5. Виконайте завдання пунктів 3 і 4 в нових рядках, що створюються в середній частині текстів відповідних файлів, результат зберегти.

№	**Перший текст**	**Другий текст**	**№**	**Перший текст**	**Другий текст**
1	Текст 5	Текст 4	**13**	Текст 6	Текст 3
2	Текст 3	Текст 5	**14**	Текст 4	Текст 6
3	Текст 1	Текст 6	**15**	Текст 3	Текст 1
4	Текст 2	Текст 3	**16**	Текст 5	Текст 2
5	Текст 6	Текст 2	**17**	Текст 1	Текст 4
6	Текст 4	Текст 1	**18**	Текст 2	Текст 5
7	Текст 3	Текст 6	**19**	Текст 4	Текст 3
8	Текст 5	Текст 3	**20**	Текст 5	Текст 1
9	Текст 2	Текст 1	**21**	Текст 2	Текст 6
10	Текст 6	Текст 4	**22**	Текст 3	Текст 4
11	Текст 1	Текст 5	**23**	Текст 6	Текст 5
12	Текст 4	Текст 2	**24**	Текст 1	Текст 2

Контрольні питання

1. Командний режим редактора **vi** та основні його команди.
2. Перехід з командного режиму до текстового режиму редактора **vi** та навпаки.
3. Позиціонування курсора у редакторі **vi**.
4. Позиціонування курсора у редакторі **nano**.
5. Збереження файлів у редакторах **vi** та **nano** та завершення їх роботи.

Лабораторна робота № 5. Розмежування прав доступу до файлів та каталогів

Операційні системи сімейства UNIX традиційно розраховані на багато користувачів системи. Щоб почати працювати, користувач повинен увійти в систему, увівши з вільного термінала своє реєстраційне ім'я і пароль. Реєстрацію нових користувачів зазвичай виконує адміністратор системи. Основними мінімальними даними, необхідними для реєстрації користувача в системі, є:

– ім'я користувача;
– назва групи, до якої відноситься користувач;
– пароль.

Системний файл реєстрації користувачів

В UNIX-системах реєстрація користувачів ведеться у файлі **`/etc/passwd`**. Вміст цього файлу є послідовністю текстових рядків. Кожен рядок відповідає одному зареєстрованому в системі користувачеві і містить 7 полів, розділених символами двокрапки. Це такі поля:

1) реєстраційне ім'я користувача;
2) зашифрований пароль;
3) значення UID (ідентифікатор користувача);
4) значення GID (ідентифікатор основної групи, до якої відноситься користувач);
5) коментар (може містити розширену інформацію про користувача, наприклад, ім'я, посаду, телефони та ін.);
6) домашній каталог;
7) командна оболонка користувача.

Приклад рядка з файлу `/etc/passwd` (зауважте, що поле коментаря в цьому випадку відсутнє):

`john:*:1004:101::/home/john:/bin/bash`

Файл ***`/etc/passwd`*** повинен бути доступним для читання всім користувачам, оскільки до нього повинні звертатися багато програм, що запускаються від імені рядового користувача (наприклад, щоб дізнатися відповідність UID реєстраційного імені). Але доступність для читання всіх зашифрованих паролів серйозно зменшує безпеку системи, тому що сучасні обчислювальні потужності дозволяють порівняно швидко підбирати паролі (особливо невдало обрані деякими користувачами).

Тому часто використовується схема тіньових паролів (shadow passwords), за якої поле пароля в ***/etc/passwd*** ігнорується, а реальний пароль береться з іншого файлу (наприклад, ***/etc/shadow***), доступного для читання тільки привілейованому користувачеві. Файл тіньових паролів часто містить й іншу важливу інформацію: термін, протягом якого допускається використання незмінного пароля, дата останньої зміни пароля та ін. У разі використання тіньових паролів друге поле в ***/etc/passwd*** зазвичай містить символ зірочки або будь-який інший довільний символ. Порожнім поле пароля в ***/etc/passwd*** залишати не можна, оскільки в цьому випадку система може порахувати, що цьому користувачеві пароль не потрібний.

Файл реєстрації груп користувачів

Інформація про групи, які відомі системі, міститься у файлі ***/etc/group***. Подібно файлу реєстрації користувачів, інформація в ***/etc/group*** представляє собою набір рядків, по одній для кожної зареєстрованої групи користувачів. Кожен рядок містить чотири поля, розділених двокрапкою:

1) реєстраційне ім'я групи;
2) пароль групи (зазвичай це поле пусте, оскільки групам зазвичай не призначають паролі);
3) значення GID, що відповідає цій групі;
4) розділений комами список користувачів, що входять у групу (може бути порожнім).

Зауважимо, що порожній список користувачів у записі ***/etc/group*** не означає, що в цій групі немає жодного користувача, бо GID основної групи користувача визначається у файлі ***/etc/passwd***.

Визначення ідентифікаторів користувачів і груп

Щоб визначити UID користувача, GID і ім'я його основної групи, а також список інших груп, до якого включений користувач, можна використовувати команду ***id***.

$ id [<користувач>]

У разі її використання без аргументів, команда виведе інформацію про поточного користувача. Якщо ж вказати як аргумент ім'я зареєстрованого користувача, результат виконання команди буде відповідати зазначеному користувачу.

Окремим випадком команди ***id*** є команда ***groups***, яка видає список імен всіх груп, в яких розташований поточний або вказаний користувач.

$ groups [<користувач>]

Перегляд і інтерпретація прав доступу до файлів

У UNIX базові права доступу до файлів включають 3 складові:

- дозвіл на читання (позначається літерою «***r***», від слова ***Read***);
- дозвіл на запис (літера «***w***», від слова ***Write***);
- дозвіл на виконання (літера «***x***», від слова ***eXecute***).

Нижче у таблиці наведені дії, які дозволяють виконувать відповідні дозволи для файлів і каталогів.

Дозвіл	***Дії з файлами***	***Дії з каталогами***
читання	читати вміст файлів	переглядати перелік імен файлів у каталозі (наприклад, командою ***ls***)
запис	змінювати вміст файлу	створювати у каталозі нові файли і каталоги або видаляти їх
виконання	запускати файли на виконання (як програми в машинному коді і командні файли)	переходити в цей каталог (тому для каталогів право виконання часто називають правом пошуку)

Якщо на файлі стоїть атрибут виконання, то незалежно від його імені він вважається програмою, яку можна запустити (на відміну від DOS або Windows, в UNIX можливість виконання файлу не залежить від розширення імені файлу, такого як .exe).

Відзначимо, що для каталогів біти читання і виконання (***r*** і ***x***) частіше використовуються в парі, тобто або присутні обидва, або відсутні.

В атрибутах доступу до файлів перераховані типи прав доступу можуть бути надані для 3 класів користувачів:

- власника (у кожного файлу в UNIX є один власник);
- групи (з кожним файлом пов’язана група користувачів цього файлу);
- всіх інших користувачів.

Набір прав доступу для конкретних файлів можна переглянути за допомогою команди ***ls -l***. Наприклад:

```
$ ls -l tmp/
drwxrwxr-x 10 john users 1024 Jan 13 newdir
-rw-r----- 1 john users 173727 Jan 13 23:48 archive-0113.zip
```

У цьому прикладі видно, що власником файлів є користувач ***john***, а групою власників є група ***users***. Набір літер і прочерків у лівій частині визначає тип файлу (перший символ) і права доступу до файлу (решта дев’ять символів).

У наведеному прикладі перший запис відноситься до каталогу (перша літера ***d***) і демонструє права доступу ***rwxrwxr-x***. Другий запис відноситься до звичайного файлу (прочерк на місці першого символу) і показує права ***rw-r-----***. Дев’ять символів прав доступу визначають можливість читання (***r***), записи (***w***) і виконання (***x***) для власника файлу (перші 3 символи), групи власника (наступні 3 символи) і всіх інших (останні 3 символи). Прочерки означають відсутність відповідних прав. Отже, в наведеному прикладі:

– ***john*** і всі користувачі групи ***users*** можуть переглядати і змінювати вміст каталогу ***newdir***, а також переходити в нього, а інші користувачі можуть читати і переходити в цей каталог, але не можуть створювати або видаляти в ньому нові файли;

– ***john*** може читати і змінювати файл ***archive-0113.zip***, користувачі групи users можуть тільки читати вміст цього файлу, а всі інші не мають до нього ніяких прав доступу.

Крім символьного подання прав доступу, часто використовується цифрова форма. У цифровому поданні права доступу складаються з 3 вісімкових цифр, кожна з яких визначає набір з трьох бітів повноважень ***rwx***. Щоб перевести права доступу з символьного представлення в числове, слід:

– представити набір прав в двійковому вигляді (наприклад, ***110100000*** для набору прав ***rw-r-----***);

– перевести отримане двійкове число в вісімкову систему числення (наприклад, вісімковим представленням двійкового числа ***110100000*** буде ***640***).

Права доступу так само можна у числовій формі шляхом підсумовування вісімкових значень окремих бітів прав доступу:

– ***400*** – власник має право на читання;

– ***200*** – власник має право на запис;

– ***100*** – власник має право на виконання;

– ***040*** – група має право на читання;

– ***020*** − група має право на запис;

– ***010*** – група має право на виконання;

– ***004*** – інші мають право на читання;

– ***002*** – інші мають право на запис;

– ***001*** – інші мають право на виконання.

Можна помітити, що для прав доступу ***rw-r-----*** отримаємо: ***400 + 200 + 040 = 640***.

Зміна власників файлів

Власником файлу стає користувач, який створив цей файл. Групою власника за замовчуванням стає основна група реєстрації користувача. Для зміни власників призначена стандартна команда ***chown*** (change owner). Однак у сучасних системах власника файлів може змінювати тільки привілейований користувач (root). У звичайного користувача існує можливість зміни тільки групи власників, і то лише в межах тих груп, в які входить сам користувач. Для зміни групи власників зручно використовувати команду ***chgrp*** (change group). Наприклад, щоб зробити групою власників каталогу ***newdir*** групу ***students***, можна ввести:

$ chgrp student newdir

Існує можливість рекурсивної зміни власників для всіх файлів і підкаталогів заданого каталогу. Для цього слід використовувати ключ ***-R***, наприклад:

$ chgrp -R student newdir

Зміна прав доступу до файлів

Змінити права доступу до файлу може або його власник, або привілейований користувач (***root***). Робиться це командою ***chmod*** (Change mode). Існує два формати використання цієї команди: з використанням символьного і числового представлення прав доступу.

Використання числового представлення дозволяє однією командою змінити повний набір прав доступу, наприклад:

$ chmod 770 newdir

Ця команда встановить права доступу в числове значення ***770***, тобто ***rwxrwx---***, що дасть повні права власнику і групі власника, і ніяких прав всім іншим.

Використання символьного представлення прав доступу в команді ***chmod*** може здатися дещо складнішим, але дозволяє маніпулювати окремими бітами прав доступу. Наприклад, щоб зняти біт запису для групи власника каталогу ***newdir***, достатньо ввести:

$ chmod g-w newdir

Умовний синтаксис цієї команди такий:

$ chmod {u, g, o, a} {+, -, =} {r, w, x} <файл ...>

Як аргумент команда приймає вказівку класів користувачів:

– «***u***» – власник-користувач (***user***);
– «***g***» – власник-група (***group***);

– «***o***» – інші користувачі (***others***);
– «***a***» – всі перераховані вище групи разом (***all***).

Операцію, яку необхідно зробити з правами доступу:

– «+» – додати;
– «-» – прибрати;
– «=» – привласнити.

Права доступу («***r***», «***w***», «***x***») можна призначити каталогам і файлам.

Як і у команді ***chgrp***, у ***chmod*** може використовуватися ключ ***-R***, що дозволяє рекурсивно обробляти вміст підкаталогів.

Встановлення права доступу за замовчуванням

Очевидно, що під час створення нових файлів і каталогів вони вже будуть мати певний набір прав доступу. Ці права доступу, що встановлюються за замовчуванням, визначаються значенням маски прав доступу, яка встановлюється командою ***umask***. У результаті введення цієї команди без аргументів вона виведе поточне значення маски, у разі використання вісімкового числа як аргумент буде встановлено нове значення.

Маска прав доступу визначає, які права мають бути видалені з повного набору прав, тобто маска прав доступу є в деякому роді зворотним значенням прав доступу. Наприклад, маска ***022*** призведе до скидання бітів запису для групи власника та інших користувачів. Зауважимо, що для звичайних файлів (НЕ каталогів) всі біти виконання (***x***) в правах за замовчуванням будуть скинуті незалежно від поточної маски.

Приклад, що демонструє ефект команди umask:

```
$ umask
002
$ mkdir dir1
$ ls -l
drwxrwxr-x 2 john users 1024 Apr 21 7:29 dir1
$ umask 072
$ umask
072
$ mkdir dir2
$ ls -l
drwxrwxr-x 2 john users 1024 Apr 21 7:29 dir1
drwx --- r-x 2 john users 1024 Apr 21 7:30 dir2
```

Завдання

1. Знайти запис у файлі ***/etc/passwd***, що відповідає вашому реєстраційному імені.
2. Визначити свій UID, дізнатися, до яких груп належить ваше реєстраційне ім'я, пояснити результати виконання команд ***id***, ***groups***.
3. Визначити межі файлового простору, де система дозволяє створювати власні файли і каталоги.
4. Перевірити, чи можливе втручання в приватний файловий простір іншого користувача системи:
   – який входить до тієї самої групи, що й ви;
   – який входить до будь-якої іншої групи.
5. Дізнатися, які права доступу мають новостворювані файли і каталоги (тобто створити новий файл і новий каталог і переглянути для них права доступу).
6. Визначити значення umask, за якого створювані файли і каталоги будуть недоступні для читання, запису і виконання всім, окрім власника.
7. Зробити свій домашній каталог видимим для всіх користувачів групи, до якої ви належите.
8. Створити в домашньому каталозі підкаталог ***tmp***, файли в якому зможе створювати, видаляти і перейменовувати будь-хто, що входить до групи вашої групи, при цьому вміст цього підкаталогу не має бути видимим всім іншим користувачам системи.

Контрольні питання

1. Як кодуються в атрибутах файлу і каталогу права доступу? Які формати записи прав бувають?
2. Кто може змінювати права доступу до файлів?
3. Які команди для зміни символьних кодів прав доступу ви знаєте? Перерахуйте і розкажіть про призначення кожної з команд.
4. У чому різниця в застосуванні команд chmod і umask?
5. Які команди обробки файлів дозволяють (або забороняють) права на читання, запис і виконання?
6. Які команди обробки каталогів дозволяють (або забороняють) ці ж права?
7. Що означає право на виконання стосовно каталогу?
8. Які правами треба володіти, щоб видалити файл або каталог?
9. Яке символьне значення запису прав доступу відповідає вісімковому значенню ***641***?
10. Яке вісімкове значення запису прав доступу відповідає символьному значенню ***rw-r------***?

Лабораторна робота № 6.
Використання регулярних виразів у Linux

Теоретичні відомості

Регулярним виразом є шаблон, за допомогою якого описується набір рядків. Регулярні вирази будуються аналогічно арифметичним виразам з використанням різних операторів, які об'єднуються в невеликі вирази.

Основними конструктивними елементами є регулярні вирази, які відповідають одиничному символу. Більшість символів, включаючи всі літери і цифри, є регулярними виразами, які відповідають самі собі. Можна скасувати спеціальне значення будь-якого метасимвола, якщо додати перед ним зворотний слеш.

Метасимволи регулярних виразів

За будь-яким регулярним виразом може слідувати один з поданих операторів повторення (метасимвол):

Таблиця 1.

Оператори регулярних виразів

Оператор	*Дія*
.	Відповідає будь-якому одиночному символу
?	Попередній елемент необов'язковий і може бути присутнім не більше одного разу
*****	Попередній елемент може бути присутнім нуль або більше число разів
+	Попередній елемент може бути присутнім один або більше число разів
{N}	Попередній елемент присутній рівно ***N*** разів
{N,}	Попередній елемент може бути присутнім ***N*** або більше число разів
{N, M}	Попередній елемент може бути присутнім принаймні ***N*** раз, але не більше ***M*** разів
-	За допомогою цього символу задається діапазон, якщо це не перший і не останній елемент у списку і не завершальне значення діапазону
^	Відповідає порожньому рядку на початку рядка введення, також представляє символи, що не підпадають в діапазон, зазначений у списку
$	Відповідає порожньому рядку в кінці рядка
\b	Відповідає порожньому рядку на межі слова

Закінчення таблиці

\B	Відповідає порожньому рядку не на межі слова
\<	Відповідає порожньому рядку на початку слова
\>	Відповідає порожньому рядку в кінці слова

Можна виконувати конкатенацію двох регулярних виразів; результуючий регулярний вираз буде відповідати будь-якому рядку, сформованому конкатенацією двох підрядків, які відповідно представляють з'єднані підвирази.

Два регулярних вирази можуть бути з'єднані бінарним оператором «**|**»; результуючий регулярний вираз буде відповідати будь-якому рядку, який відповідає якому-небудь одному зі з'єднаних підвиразів.

Операція повторення має більший пріоритет, ніж конкатенація, яка, у свою чергу, має більший пріоритет, ніж операція вибору варіанта. Підвираз можна укладати в дужки для того, щоб змінювати ці правила пріоритету.

У базових регулярних виразах символи «***?***» «***+***», «***{***», «**|**», «***(***» і «***)***» не є метасимволами; замість них використовуйте варіанти зі зворотним слешем «***\?***», «***\+***», «***\{***», «***\|***», «***\(***» та «***\)***».

Перевірте в документації до всієї системи, чи є команди, що дозволяють використовувати розширені регулярні вирази.

Команда grep і регулярні вирази

Команда ***grep*** використовується для пошуку у вхідних рядках відповідностей, що визначаються лише за вибраними шаблонами. Коли команда знаходить в рядку відповідність, цей рядок копіюється в стандартне виведення (дія, визначена за замовчуванням), або в будь-який інший вихідний потік, який ви можете задати.

Наведемо декілька прикладів:

```
$ grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

```
$ grep -n root /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
12:operator:x:11:0:operator:/root:/sbin/nologin
```

У першій команді відображаються рядки з файлу ***/etc/passwd***, в яких є рядок ***root***.

У другому прикладі відображаються, крім того, номери рядків, в яких є шуканий рядок.

Тепер відобразимо тільки ті рядки, які починаються з рядка «***root***»:

```
$ grep ^root /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Якщо ми хочемо побачити, в яких облікових записах командна оболонка взагалі не використовувалася, ми шукаємо рядки, що закінчуються символом «**:**»:

```
$ grep :$ /etc/passwd
news:x:9:13:news:/var/spool/news:
```

Щоб перевірити, чи експортується у файлі ***~/.bashrc*** змінна ***PATH***, спочатку потрібно вибрати рядки з «***export***», а потім знайти рядки, що починаються з рядка «***PATH***»; в такому разі не будуть відображатися ***MANPATH*** та інші можливі шляхи:

```
$ grep export ~/.bashrc | grep '\<PATH'
 export
PATH="/bin:/usr/lib/mh:/lib:/usr/bin:/usr/local/bin
:/ usr/ucb:/usr/dbin:$PATH"
```

Символьні класи

Виразом у квадратних дужках є список символів, укладених усередині символів «**[**» і «**]**». Він відповідає будь-якому одиночному символу, вказаному в цьому списку; якщо першим символом списку є «**^**», то він відповідає будь-якому символу, який відсутній у списку. Наприклад, регулярний вираз «***[0123456789]***» відповідає будь-якій одиночній цифрі.

Усередині виразу в квадратних дужках можна вказувати діапазон, що складається з двох символів, розділених дефісом. Тоді вираз відповідає будь-якому одиночному, який згідно з правилами сортування потрапляє всередину цих двох символів, включаючи і ці два символи; при цьому враховується послідовність упорядкування і набір символів, зазначених в локалі. Наприклад, коли за замовчуванням вказана локаль **C**, вираз «***[a-d]***» еквівалентний вислову «***[abcd]***». Є багато локалей, в яких сортування виконується в словниковому порядку, і в цих локалях «***[a-d]***», як правило, не еквівалентне «***[abcd]***», в них, наприклад, воно може бути еквівалентним вислову «***[aBbCcDd]***». Щоб використовувати традиційну інтерпретацію виразів, зазначених вище в квадратних дужках, ви можете скористатися локаллю **C**, встановивши для цього в змінній оточення ***LC_ALL*** значення «***C***».

Нарешті, є певним чином зазначені символьні класи, які вказуються всередині виразів у квадратних дужках. Додаткову інформацію про ці зумовлених виразах можна подивитись на сторінках ***man*** або в документації команди ***grep***.

```
$ grep [yf] /etc/group
sys:x:3:root,bin,adm
tty:x:5:
mail:x:12:mail,postfix
ftp:x:50:
nobody:x:99:
floppy:x:19:
xfs:x:43:
nfsnobody:x:65534:
postfix:x:89:
```

У прикладі відображаються всі рядки, що містять або символ «***y***», або символ «***f***».

Універсальні символи (метасимволи)

Можна використати «***.***» для пошуку відповідності будь-якому одиночному символу. Якщо ви хочете отримати список всіх англійських слів, взятих зі словника, що містять п'ять символів, що починаються з «***c***» і закінчуються «***h***»:

```
$ grep '\<c...h\>' /usr/share/dict/words
catch
clash
cloth
coach
couch
cough
crash
crush
```

Якщо ми хочемо відобразити рядки, в яких є символ точки у вигляді літерала, то потрібно вказати в команді ***grep*** параметр ***-F***.

Щоб подібним чином знайти слова, в яких між «***c***» і «***h***» може бути будь-яке число символів, потрібно використати зірочку. У наведеному нижче прикладі з системного словника вибираються всі слова, що починаються з «***c***» і закінчуються символом «***h***»:

```
$ grep '\<c.*h\>' /usr/share/dict/words
caliph
cash
```

```
catch
cheesecloth
cheetah
```

Якщо ви хочете знайти у файлі або в вихідному потоці літеральний символ «зірочка», використовуйте для цього одинарні лапки. Користувач в наведеному нижче прикладі спочатку намагається в файлі ***/etc/profile*** знайти «зірочку» без використання лапок, в результаті чого нічого не знаходиться. Коли використовуються лапки, у вихідний потік видається результат:

```
$ grep * /etc/profile

$ grep '*' /etc/profile
for i in /etc/profile.d/*.sh ; do
```

Символьні діапазони

Крім команди ***grep*** і регулярних виразів, у командній оболонці безпосередньо є ряд способів пошуку відповідності шаблоном без використання зовнішніх програм.

Як ви вже знаєте, зірочка (*******) і знак питання (***?***) відповідає будь-якому рядку або будь-якому одному символу, відповідно. Якщо укласти ці спеціальні символи в лапки, то під час пошуку відповідності буде розглядатися їх літеральне значення:

```
$ touch "*"

$ ls "*"
*
```

Але також можна використовувати квадратні дужки, щоб знайти відповідність будь-якого укладеного в них символу або діапазону символів, якщо пара символів розділяється дефісом. Наприклад:

```
$ ls -ld [a-cx-z]*
drwxr-xr-x 2 user user 4096 Jul 20 2014 app-defaults/
drwxrwxr-x 4 user user 4096 May 25 2014 arabic/
drwxr-xr-x 7 user user 4096 Sep 2 2015 crossover/
drwxrwxr-x 3 user user 4096 Mar 22 2014 xml/
```

У цьому списку відображаються всі файли, які починаються із символів «***a***», «***b***», «***c***», «***x***», «***y***» або «***z***» і розташовані в домашньому директорії користувача ***user***.

Якщо першим символом у квадратних дужках буде «***!***» або «***^***», то шукається відповідність будь-якому символу, який не вказаний

всередині дужок. Якщо потрібно знайти відповідність символу дефіса («**-**»), його потрібно вказати як перший або останній символ. Правила сортування залежать від поточної локалі і від значення змінної ***LC_COLLATE***, якщо вона встановлена. Якщо ви хочете бути впевненими, що використовується традиційна інтерпретація діапазонів, явно задайте саме таку інтерпретацію, присвоївши для цього значення «***C***» змінним ***LC_COLLATE*** або ***LC_ALL***.

Символьні класи

Усередині квадратних дужок можна вказувати символьні класи, використавши формат ***[:CLASS:]***, де ***CLASS*** визначається стандартом POSIX і має одне з таких значень:

«***alnum***», «***alpha***», «***ascii***», «***blank***», «***cntrl***», «***digit***», «***graph***», «***lower***», «***print***», «***punct***», «***space***», «***upper***», «***word***» або «***xdigit***».

Декілька прикладів:

```
$ ls -ld [[:digit:]]*
drwxrwxr-x 2 user user 4096 Apr 20  13:45  2/

$ ls -ld [[:upper:]]*
drwxrwxr-- 3 user user 4096 Sep 30 2014 Nautilus/
drwxrwxr-x 4 user user 4096 Jul 11 2002 OpenOffice.org1.0/
-rw-rw-r-- 1 cathy cathy 997376 Apr 18 15:39 Schedule.sdc
```

Завдання

1. Покажіть список всіх користувачів у вашій системі, які під час входу в систему за замовчуванням користуються командною оболонкою ***bash***.
2. Відобразіть у файлі ***/etc/group*** всі рядки, що починаються з рядка «***daemon***».
3. Відкрийте всі рядки з тих же файлів, в яких немає цього рядка.
4. Покажіть інформацію про ***localhost*** з файлу ***/etc/hosts***, покажіть номери рядків, в яких знайдено відповідність, і порахуйте кількість входжень шуканого рядка.
5. Покажіть список підкаталогів каталогу ***/usr/share/doc***, в яких міститься інформація про командні оболонки.
6. Підрахуйте, скільки файлів ***README*** знаходиться в цих директоріях.

7. За допомогою команди ***grep*** створіть список файлів з вашого домашнього каталогу, які були змінені менше 10 годин тому, не враховуйте при цьому підкаталоги.
8. Помістіть ці команди у сценарій, за допомогою якого буде створюватися зрозумілий вихідний потік.
9. Створіть сценарій, за допомогою якого можна перевіряти, чи існує користувач у файлі ***/etc/passwd***.
10. Покажіть конфігураційні файли, що знаходяться у каталозі ***/etc***, в іменах яких присутні числа.

Контрольні питання

1. Які символи відносяться до метасимволів для їх застосування у регулярних виразах?
2. Що означають вирази у квадратних дужках у регулярних виразах?
3. Що означає символ «***^***» у регулярних виразах?
4. Що означає символ «***$***» у регулярних виразах?
5. Які символьні класи POSIX можна застосувати у регулярних виразах?
6. Який символ у розширених регулярних виразах одначає збіг 0 або 1 раз?
7. Який символ у розширених регулярних виразах одначає збіг 0 або більше разів?
8. Який символ у розширених регулярних виразах одначає збіг з елементом один або більше разів?

Лабораторна робота № 7.
Обробка текстових даних.
Користувацьке оточення

Обробка текстових даних

В операційних системах текстові файли використовуються для зберігання даних різного призначення. Як приклади можна виділити файли конфігурації, системні журнали, файли з вихідним кодом програм та ін. Для роботи з цими даними розроблено велику кількість утиліт.

Для об'єднання вмісту декількох файлів і виведення його в стандартний канал виведення або в файл використовується команда ***cat***, яка вже була розглянута в минулих лабораторних роботах. Ключ ***-n*** цієї команди виробляє нумерацію рядків під час виведення.

Серед інших вже знайомих команд обробки тексту слід виділити такі:

– ***less*** – дозволяє організувати посторінкову роботу з великим набором даних;

– ***sort*** – виконує сортування даних, що поступають їй на вхід. Використання додаткових опцій дозволяє провести сортування за однією з полів згрупованих даних. Наприклад: ***$ ls -l ~ | sort -n -k 5.*** Ця команда виробляє числову (опція ***-n***) сортування отриманих від команди ls даних за п'ятим стовпчиком (опція ***-k***);

– ***uniq*** – видаляє сусідні повторювані рядки у файлі. Опції команди дозволяють також знайти неунікальні рядки і підрахувати кількість входжень кожного рядка. Цю команду часто використовують спільно з командою ***sort***;

– ***head*** і ***tail*** – використовуються для відображення обраного числа рядків на початку або в кінці файлу. За замовчуванням число рядків дорівнює 10. Змінити кількість виведених рядків можна за допомогою опції ***–n***;

– ***wc*** – використовується для підрахунку рядків, слів, байт і символів у файлі.

Розглянемо інші, ще незнайомі вам команди обробки текстових даних.

cut - виконує фільтрацію тексту за стовпцями. Як опцію команда приймає номер поля ***(-f)***, роздільник ***(-d)*** та ін. Наприклад:

$ cut -d: -f 1 file1

Наведена вище команда з множини стовпців, розділених символом «**:**» у файлі ***file1***, вибирає перший.

paste – використовується для об’єднання декількох файлів;

join – може розглядатися як команда, споріднена команді paste. Ця потужна утиліта дозволяє об’єднувати два файли за загальним полем, що є спрощеною версією реляційної бази даних.

Команда join оперує тільки двома файлами і об’єднує тільки ті рядки, які мають загальне поле (зазвичай числове), результат об’єднання виводиться на стандартний потік виведення. Файли, що об’єднуються, повинні бути відсортовані за ключовим полем.

Наприклад, існує 2 файли:

```
File: 1.data
100 Shoes
200 Laces
300 Socks
File: 2.data
100 $40.00
200 $1.00
300 $2.00
```

Виконання команди `join`:

```
$ join 1.data 2.data
100 Shoes $40.00
200 Laces $1.00
300 Socks $2.00
```

Порівняння файлів

cmp – порівнює вміст двох файлів побайтно:

$ cmp file1 file2

Якщо файли повністю збігаються, ця команда завершує свою роботу, а якщо файли розрізняються, видає номер рядка й номер байта в рядку, де має місце перша розбіжність. Звичайно, інформації, що видається командою cmp, обмаль для того, щоб прийняти, наприклад, рішення про те, який із двох файлів для нас важливіший.

diff – порівняння двох файлів з одержанням повної інформації про розбіжності у файлах. Для одержання інформації достатньо вказати команді, які саме файли порівнювати:

$ diff file1 file2

Інформація про виявлені розбіжності буде видана на стандартний потік виведення, але її можна перенаправити у файл:

$ diff file1 file2 > diff12

Середовище оточення

Під час роботи з командною оболонкою визначено набір змінних, що описують поточний сеанс взаємодії користувача з системою, що називається **оточенням (environment)**.

Змінні оточення доступні одночасно декільком процесам.

Завантаження змінних оточення з конфігураційних файлів відбувається у результаті запуску командного інтерпретатора. На додаток до змінних оточення, оболонка так само зберігає псевдоніми і функції оболонки.

Список всіх встановлених змінних можна отримати, використовуючи команди ***env*** або ***set*** без опцій і аргументів.

Установка нових та зміна значення існуючих змінних середовища оточення здійснюється шляхом експортування (поміщення у середовище):

```
$ export <змінна>=<значення>
```

Дізнатися значення конкретної змінної можна також за допомогою команди:

```
$ echo $<змінна>
```

Щоб видалити змінну, використовується команда ***unset***.

Таблиця 1.

Деякі стандартні змінні середовища оточення

І'мя	*Значення*
UID	Містить числовий ідентифікатор поточного користувача. Ініціалізується при запуску оболонки
HOME	Домашній каталог поточного користувача
PATH	Список каталогів, розділених двокрапкою, в яких командна оболонка виконує пошук файлу, в разі якщо в команді не заданий його шлях
PS1	Формат рядка-запрошення
PWD	Поточний каталог
TERM	Тип термінала, що використовується
HOSTNAME	Мережеве ім'я комп'ютера

Налаштування зовнішнього вигляду рядка-запрошення

За замовчуванням рядок запрошення до вводу має такий вигляд (у Debian, в інших дистрибутивах він може відрізнятись):

```
<Ім'я_користувача>@<Ім'я_хоста>:<Поточний_робочий_каталог>$
```

Форма запрошення до введення визначається в змінній оточення ***PS1*** (скорочено від prompt string 1 – рядок запрошення 1). Побачити вміст змінної ***PS1*** можна за допомогою команди ***echo***:

```
$ echo $PS1
\u@\h:\w\$
```

Символи, що екрануються слешем, є спеціальними символами. У табл. 2 наведений неповний список символів, які командна оболонка інтерпретує спеціальним чином у рядку запрошення.

Табллиця 2.

Екрановані послідовності, що використовуються у рядку запрошення

Послідовність	***Значення, що відображується***
\a	дзвінок. Змушує комп'ютер видавати звуковий сигнал
\d	поточна дата у форматі: день тижня, місяць, число; наприклад, «Mon May 26»
\h	ім'я хоста локальної машини мінус ім'я домена
\H	повне ім'я хоста
\j	число завдань, що діють в поточному сеансі
\l	ім'я поточного пристрою термінала
\n	символ переведення рядка
\r	повернення каретки
\s	ім'я програми командної оболонки
\t	поточний час в 24-годинному форматі
\T	поточний час в 12-годинному форматі
\u	ім'я користувача
\w	ім'я поточного робочого каталогу
\W	остання частина в імені поточного робочого каталогу
\$	виводить символ $, якщо користувач не є суперкористувачем, в іншому випадку виводить символ #
\!	номер поточної команди в історії
\#	число команд, введених в поточному сеансі командної оболонки

Маючи список спеціальних символів, можна спробувати змінити оформлення запрошення. Для початку можна зберегти вихідне визначення, щоб його можна було відновити пізніше. Для цього скопіюємо значення змінної ***PS1*** в іншу змінну:

```
$ ps1_old="$PS1"
```

Тут створюється нова змінна з ім'ям ***ps1_old***, і їй присвоюється значення змінної ***PS1.*** Це дозволить вам в будь-який момент відновити вихідне оформлення запрошення, виконавши зворотну процедуру:

$ PS1="$ps1_old"

Тепер можна спробувати змінити зовнішній вигляд рядка запрошення. Наведемо декалька прикладів:

– ***$ PS1=*** – порожній рядок запрошення;

– ***$ PS1="\$ "*** – рядок виводить тільки символ ***$*** (або ***#*** для суперкористувача);

– ***$ PS1="\a\$ "*** – звуковий сигнал під час кожного виведення рядка запрошення та символ ***$*** (або ***#***);

– ***$ PS1="\t \h \$ "*** – виводить час та ім'я хоста;

– ***$ PS1="<\u@\h \w>\$ "*** – виводить ім'я користовача, ім'я хоста, поточний каталог. Все це знаходиться у трикутних дужках. Завершується рядок символом ***$*** (або ***#***).

Оформлення кольору рядка запрошення

Кольором символів можна управляти, посилаючи емулятору термінала екрановані послідовності ANSI всередині потоку символів, призначених для виведення на екран. Екрановані послідовності не виводяться на екран; вони інтерпретуються терміналом як інструкції.

Для включення недрукованих символів використовуються послідовності ***\[*** і ***\]***. Екрановані послідовності ANSI починаються з вісімкового коду 033 (код, що генерується клавішею **ESC**), за яким іде необов'язковий атрибут символу і інструкція. Наприклад, ось як виглядає код, що визначає текст як простий (атрибут = 0), чорного кольору:

\033[0;30m.

У табл. 3 перераховані підтримувані кольори тексту.

Таблиця 3.

Екрановані послідовності, що використовуються для визначення кольору тексту

Послідовність	*Колір*
\033[0;30m	чорний
\033[0;31m	червоний
\033[0;32m	зелений
\033[0;33m	коричневий
\033[0;34m	синій

Закінчення таблиці

\033[0;35m	пурпуровий
\033[0;36m	бірюзовий
\033[0;37m	світло-сірий
\033[1;30m	темно-сірий
\033[1;31m	світло-червоний
\033[1;32m	світло-зелений
\033[1;33m	жовтий
\033[1;34m	світло-синій
\033[1;35m	світло-пурпуровий
\033[1;36m	світло-бірюзовий
\033[1;37m	білий

Наприклад, для того, щоб пофарбувати рядок запрошення у червоний колір, потрібно зробити таке:

```
$ PS1="\[\033[0;31m\]<\u@\h \W>\$ "
```

Однак у нашому випадку червоним кольором буде фарбуватись і весь текст, який буде введений з клавіатури. Для усунення цього ефекту потрібно додати ще одну екрановану послідовність в кінець визначення запрошення – цим ми повідомимо емулятору термінала, що той повинен відновити нормальний колір:

```
$ PS1="\[\033[0;31m\]<\u@\h \W>\$\[\033[0m\] "
```

Крім того, існує можливість змінити колір фону, для чого призначені екрановані послідовності, пераховані у табл. 4.

Таблиця 4.

Екрановані послідовності, що використовуються для визначення кольору фону

Послідовність	***Колір***
\033[0;40m	чорний
\033[0;41m	червоний
\033[0;42m	зелений
\033[0;43m	коричневий
\033[0;44m	синій
\033[0;45m	пурпуровий
\033[0;46m	бірюзовий
\033[0;47m	світло-сірий

Наприклад, щоб вивести запрошення на червоному фоні, достатньо змінити першу екрановану послідовність:

```
$ PS1="\[\033[0;41m\]<\u@\h \W>\$\[\033[0m\] "
```

А так запрошення буде відображуватись зеленим кольором на червоному фоні (першою повинна бути вказана послідовність, що визначає колір фону, а потім – послідовність, що визначає колір тексту):

```
$ PS1="\[\033[0;41m\033[1;32m\]<\u@\h \W>\$\[\033[0m\] "
```

Завдання

1. Ознайомтеся з роботою команд, які наведені в теоретичній частині лабораторної роботи. Подивіться для цих команд сторінки довідкового керівництва.
2. Відсортуйте виведення команди ***ls -l*** за датою зміни вмісту за місяцями.
3. Скопіюйте файл ***/etc/passwd*** у файл ***passwd_example*** та помістіть його у своєму домашньому каталозі.
4. З файлу ***passwd_example*** отримайте імена всіх користувачів, що містяться в першому полі кожного рядка, і помістіть відсортований у зворотному порядку результат у файл ***cut_result***.
5. За допомогою текстового редактора **vi** або **nano** змініть імена кількох користувачів у цьому файлі і збережіть результат у новий файл ***cut_result2***.
6. Порівняйте вміст файлів з іменами користувачів за допомогою програми ***diff***.
7. Додайте до вмісту файлу ***cut_result*** вміст файлу ***cut_result2***.
8. За допомогою команди uniq позбудьтеся від дублікатів у файлі ***cut_result***.
9. За допомогою однієї команди отримаєте домашній каталог користувача ***user*** з файлу ***passwd_example***.
10. Виведіть на екран значення всіх змінних середовища оточення. Проаналізуйте отримані результати і поясніть значення відомих вам змінних оточення.
11. Визначте тип використовуваного термінала.
12. Змініть вміст змінної ***PS1*** так, щоб у запрошенні замість дужок використовувалися символи «**<>**».
13. Зробіть, щоб запрошення виводилось текстом жовтого кольору на синьому фоні.
14. Поверніть старий вигляд рядка запрошення.

Контрольні питання

1. Які утиліти для роботи з текстом ви знаєте?
2. Яка опція використовується для зміни порядку сортування ***`sort`***?
3. За допомогою якої команди можна порівняти файли?
4. Що таке змінні оточення?
5. Як задати значення змінної оточення і як вивести його на екран?
6. Як змінити значення змінних оточення, видалити змінну?
7. Як змінити зовнішній вигляд рядка запрошення до введення команд?

Лабораторна робота № 8. Пошук та архівація файлів

Пошук файлів

Незважаючи на те, що існують угоди з організації файлової системи для Unix-подібних операційних систем, пошук необхідного файлу може бути досить трудомістким.

У цій лабораторній роботі ми розглянемо кілька утиліт, призначених для пошуку файлів у системі.

Команда ***find*** здійснює рекурсивний обхід дерева файлової системи і здійснює пошук файлу, ґрунтуючись на одному або декількох атрибутах.

Таблиця 1.

Приклади опцій команди *find*

Опція	*Опис*
-name	ім’я файлу
-type	тип файлу
-newer	файли з меншою датою модифікації, ніж у заданого файлу
-mtime	дата модифікації файлу
-size	розмір файлу
-exec	виконання над знайденими файлами зазначеної команди
-delete	видалення знайдених файлів

Деякі опції можуть приймати аргумент ***n*** зі знаком ***+n*** для значень, які більше ніж n, ***-n*** для значень, які менше ніж n.

```
$ find ~ -name "*.txt" -type f -mtime -3
```

Наведена вище команда зробить пошук всіх звичайних файлів у домашньому каталозі користувача, з розширенням txt і датою модифікації менше 3-х днів.

Команда ***find*** також підтримує об’єднання шаблонів пошуку в логічні вираження за допомогою опцій-операторів ***-or***, ***-and*** і ***-not***.

```
$ find ~ \(-size + 100M \) -and \ (-not -type d \)
```

Наведена вище команда дозволяє отримати список файлів, які не є директорією розміром більше 100 мегабайт.

Альтернативним способом знайти файл на ім’я є використання команди ***locate***.

```
$ locate <ім’я_файлу>
```

Пошук у цьому випадку відбувається не за деревом файлової системи, а за спеціальною базою імен файлів, яка періодично оновлюється. Для оновлення бази використовується команда **updatedb**. За замовчуванням **locate** не перевіряє, чи існують файли, знайдені в базі імен на цей момент.

Команда **whereis** дозволяє виконати пошук розташування виконуваних файлів, файлів з вихідним кодом і файлів довідкових сторінок для обраної команди. Форматом виведення команди можна керувати за допомогою спеціальних опцій.

$ whereis <команда>

Команда **which** виконує пошук шляху до виконуваного файлу для певної команди на основі вмісту змінної PATH.

$ which <команда>

Для пошуку рядків у файлі, що містять певний шаблон, можна використовувати команду **grep**. Якщо команді не передається список вхідних файлів, то пошук здійснюється в стандартному вхідному потоці.

$ cat /var/log/mylog | grep 'warning'

Таблиця 2.

Приклади опцій команди *grep*

Опція	*Опис*
-n	виведення номера рядка, що містить шаблон
-c	виведення кількості рядків, що містять зразок
-i	ігнорувати регістр символів
-v	виведення всіх рядків, що не містять шаблон

Утиліту **grep** часто використовують у зв'язці з іншими командами, передаючи їй дані на стандартний потік введення. Можливе застосування регулярних виразів у написанні шаблону для пошуку.

Команда **xargs** перетворює рядки, що надходять їй на вхід, в аргументи для заданої команди.

$ find ~/mydir -type f -name filename | xargs ls –l

Архівація

Для скорочення обсягу, займаного призначеними для довготривалого зберігання або передачі файлами, застосовують різні утиліти стиснення.

Утиліта **gzip** призначена для стиснення одного або декількох файлів. У результаті упаковки оригінальні файли замінюються файлами

з розширенням ***.gz***. Перенаправити закодовану інформацію на стандартне виведення зі збереженням вихідного файлу можна за допомогою опції ***-c***.

```
$ gzip file1 file2 file3
```

Розпакування файлів може здійснюватися з використанням опції ***-d*** або утилітою ***gunzip***. Отримати інформацію про ступінь стиснення файлу можна за допомогою опції ***-l***. За допомогою числових опцій – [***1-9***] можна регулювати ступінь стиснення.

```
$ gzip -l file1.gz
```

Існує схожа на gzip утиліта, яка використовує для стиснення перетворення Барроуза – Уілера, ***bzip2.*** Вона має схожий синтаксис і опції. Стислі файли в цьому випадку мають розширення ***.bz2***.

Одночасно зі стисненням, під час роботи з файлами застосовується операція об’єднання декількох файлів в один архів. Архівація часто застосовується у створенні резервних копій.

Утиліта ***tar*** призначена для упаковки безлічі файлів в архів, і їх вилучення. Як аргументи команда приймає файли, що архівуються.

Таблиця 3.

Приклади опцій команди *tar*

Опція	*Опис*
-r	додавання файлів до архіву
-c	створення нового архіву
--delete	видалення файлів з архіву
-t	виведення вмісту архіву
-x	витяг файлів з архіву
-f	використання архівного файлу
-v	виведення списку оброблюваних файлів

Створення нового архівного файлу:

```
$ tar -cvf myarchive.tar file1 file2 file3
```

Перегляд вмісту архівного файлу:

```
$ tar -tf myarchive.tar
```

Утиліта ***zip*** виконує одночасно функції стиснення та архівації. Підсумковий файл буде мати розширення ***.zip***. Для розміщення в архів директорій разом зі вкладеними файлами використовується прапор ***-r***.

```
$ zip -r archive.zip <ім’я_діректорії>
```

Для розпаковування архіву використовується утиліта ***unzip***. Для отримання додаткової інформації про видобуті файли використовується опція ***-l***.

$ unzip -l archive.zip <шлях_для_розпакування>

Завдання

1. Ознайомтесь з роботою команд, наведених у тексті лабораторної роботи. Отримайте для них сторінки довідкового керівництва.
2. За допомогою утиліт ***find*** і ***wc*** отримаєте інформацію про кількість файлів у домашньому каталозі користувача.
3. Отримайте імена всіх файлів, які не є символічними посиланнями або каталогами, і помістіть їх у файл ***filelist1.txt***.
4. За допомогою команд ***find***, ***xargs*** і ***ls*** отримайте повну інформацію про атрибути файлів домашнього каталогу, розмір яких перевищує 5 кілобайтів, і помістіть результат у файл ***filelist2.txt***.
5. За допомогою команди ***locate*** отримаєте список імен файлів, що містять у назві рядок "***bash***".
6. Для команд, які використовуються в попередніх підпунктах, отримаєте розташування файлів довідкових посібників.
7. З файлу ***passwd_example*** з минулої лабораторної роботи за допомогою утиліти ***grep*** отримайте записи користувачів з домашніми каталогами в папці ***home***, із зазначенням номерів рядків. Помістіть результат у файл ***filelist3.txt***.
8. Стисніть файл ***filelist1.txt*** зі збереженням вихідного файлу, утилітою ***gzip*** з різними ступенями стиснення. Для одержаних файлів дізнайтеся відсоток коефіцієнту стиснення.
9. Стисніть файл ***filelist1.txt*** зі збереженням вихідного файлу, утилітою ***bzip2*** з різними ступенями стиснення.
10. Порівняйте результати для утиліт ***gzip*** і ***bzip2***, подивившись розміри отриманих стиснених файлів.
11. Створіть архів ***tar***, що містить файли ***filelist1.txt***, ***filelist2.txt*** та ***filelist3.txt***.
12. Додайте до створеного архіву файл ***passwd_example***.
13. Створіть архів ***zip***, що містить файли ***filelist1.txt***, ***filelist2.txt***, ***filelist3.txt*** і ***passwd_example***.
14. Порівняйте розміри одержані архівів.
15. Розпакуйте архіви, створені командами ***tar*** та ***zip***.

Контрольні питання

1. Які утиліти для пошуку файлів ви знаєте?
2. Як дізнатися розташування бінарних файлів певної команди?
3. Де здійснює пошук файлів команда ***locate***?
4. Як отримати номери рядків у файлі, що не містять шуканого шаблона?
5. Як додати файли до архіву ***tar***, отримати список файлів в архіві?
6. Як витягти файли з архіву ***tar***, ***zip***?

Лабораторна робота № 9.
Знайомство з процесами.
Контроль ресурсів та планування задач

Поняття процесу

Операційна система Linux є багатозадачною (мультизадачною). Це означає, що одночасно в системі може бути присутня множина процесів, кожному з яких доступна певна кількість процесорного часу. Для користувача створюється «ілюзія» одночасного виконання процесів.

Процес – виконувана програма з її даними і контекстом.

Кожен процес має унікальний у будь-який момент часу ідентифікатор у системі – ***PID***. Перший підготовлений до запуску в системі процес ***init*** має ***pid = 1***.

Для опису процесів в операційній системі є список структур – дескрипторів, що містять інформацію про ідентифікатор процесу, пріоритет, стан процесу, інформацію про належність користувачеві і групі, займаних процесом ресурсах та ін.

Кожен процес у системі Linux запускається будь-яким іншим процесом. Запускаючий процес – батьківський, новий процес – дочірній. Процеси, які виконують одну задачу, об’єднуються в групи, які мають власний ідентифікатор. Процес всередині групи, ідентифікатор якого збігається з ідентифікатором групи процесів, вважається лідером групи процесів.

Усі запущені процеси умовно (залежно від виконуваної ними функції) можна розділити на три типи:

1. **Системні процеси** є частиною ядра і завжди розташовані в оперативній пам’яті. Вони часто не мають відповідних їм програм у вигляді виконуваних файлів і завжди запускаються особливим чином під час завантаження ядра системи.
2. **Процеси-демони** – це неінтерактивні процеси, які виконуються у фоновому режимі.
3. До **прикладних** відносяться всі інші процеси, що виконуються в системі.

Інтерактивні процеси пов’язані з визначеним терміналом і через нього взаємодіють з користувачем. Фонові процеси виконуються незалежно від користувача і (псевдо) паралельно.

Кожен процес в операційній системі Linux може перебувати в одному з чотирьох станів: **працездатний**, **сплячий** (або очікування), **зупинений** і **той, що завершився**.

Додаткові утиліти

Для отримання інформації про запущені процеси часто використовується команда ***ps***.

$ ps

Виведення запущеної без аргументів команди містить: інформацію про процеси поточного користувача і асоційовані з поточним терміналом, процесорний час, зайнятий цим процесом, і ім'я файлу. Управляти форматом виведення можна за допомогою додаткових опцій.

Таблиця 1.

Приклади опцій команди *ps*

Ключ	*Опис*
-a	видати всі процеси системи, включаючи лідерів сеансів
-d	видати всі процеси системи, виключаючи лідерів сеансів
-e	видати всі процеси системи
-x	видати процеси системи, що не мають контрольного терміналa
-o	визначає формат виведення у вигляді списку полів, розділених символом «,»
-u	видати процеси, що належать зазначеному користувачеві

Наприклад, можна отримати вибіркову інформацію про всі процеси в системі:

$ ps -eo s,pid,tty,command

Альтернативним способом дізнатися про стан процесів в реальному часі є використання команди ***top***. Висновком команди можна керувати за допомогою спеціальних комбінацій клавіш. Довідкову інформацію можна отримати, натиснувши клавішу «***h***».

Щоб отримати інформацію про запущені в системі процеси у вигляді дерева, можна використовувати утиліту ***pstree***.

Управління процесами

Щоб запустити програму, достатньо ввести її ім'я в командному рядку і натиснути «Enter». Однак не всі команди запускають єдиний процес.

Інтерактивні процеси, запущені в терміналі, займають термінальну сесію, і оболонка не виводить користувачеві рядок запрошення до тих пір, поки програма не завершиться.

Роботу деяких запущених у терміналі програм можна перервати за допомогою поєднання клавіш «**Ctrl + C**» у вікні терміналa. У цей момент програмі надсилається сигнал **INT (Interrupt)**.

Щоб запустити програму у фоновому режимі, необхідно завершити команду символом амперсанд «**&**». Після цього в термінал виводиться

інформація про запущений процес, включаючи номер завдання термінал, і запрошення користувачеві на введення нової команди.

```
$ top &
```

Використовуючи команду ***jobs***, ми можемо отримати список завдань, які запущені через термінал.

```
$ jobs
```

Щоб повернути запущений у фоні процес на перший план, використовується команда ***fg*** із зазначенням номера завдання у списку завдань.

```
$ fg %2
```

Якщо ми хочемо перевести процес у стан «**зупинений**», використовується поєднання клавіш «**Ctrl + z**». У цей момент програмі надсилається сигнал **TSTP (Terminal Stop)**.

Після цього ми можемо або перемістити завдання на перший план командою ***fg***, або продовжити його виконання у фоновому режимі командою ***bg***.

```
$ bg %2
```

Ще одним способом управляти виконанням процесів є використання утиліти ***kill***. Ця команда дозволяє послати певний сигнал процесу. Можливе завершення процесу як за іменем, так і за номером завдання або за ідентифікатором процесу (PID).

```
$ kill -SIGINT 124672
```

Отримати список сигналів можна за допомогою опції -l.

```
$ kill -l
```

Існує більше двадцяти різних сигналів. Перелічимо основні з них:

– **SIGCHLD** – сигнал про завершення дочірнього процесу;

– **SIGHUP** – сигнал звільнення лінії. Посилається всім процесам, підключеним до керуючого терміналу у разі відключення терміналу. Багато демонів під час отримання такого сигналу заново переглядають файли конфігурації і перезапускаються;

– **SIGINT** – сигнал посилається всім процесам сеансу, що пов’язані з терміналом у разі натискання користувачем клавіші переривання (CTRL-C);

– **SIGTERM, SIGKILL** – сигнали призводять до негайного припинення роботи процесу, що отримав сигнал. На відміну від сигналу SIGTERM, процес не може блокувати і перехоплювати сигнал SIGKILL;

– **SIGSEGV** – сигнал посилається процесу, якщо той намагається звернутися за неправильною адресою пам’яті;

– **SIGSTOP** – сигнал приводить до зупинки процесу. Для відправки сигналу SIGSTOP активного процесу поточного термінала можна скористатися комбінацією клавіш (CTRL-Z);
– **SIGCONT** – сигнал відновлює роботу зупиненого процесу;
– **SIGUSR1, SIGUSR2** – сигнали, що визначаються користувачем.

Послати сигнал декільком процесам можна за допомогою команди ***killall***.

$ killall gedit

Наведена вище команда завершить всі процеси поточного користувача з ім'ям **gedit**. За замовчуванням команда відправляє сигнал **TERM** (software termination signal).

Отримання інформації про систему

Однією з утиліт для відстеження інформації про продуктивність є ***vmstat***. Виведенням команди є звіт про стан системи, що отримується із заданим інтервалом часу. Наприклад, отримати звіт, що складається з 8 рядків, котрі містять статистику, зібрану з інтервалом у 2 секунди, можна командою:

$ vmstat 2 8

Виведення містить інформацію про готові до виконання і сплячі процеси, використання оперативної пам'яті, підкачки, дискові операції, статистику використання центрального процесора. Детальніше про формат виведення можна дізнатися на сторінках довідкового керівництва.

Інформацію про середню завантаженість системи за період 5 с, 10 с і 15 с, а також час безперервної роботи системи можна отримати за допомогою утиліти ***uptime***.

$ uptime

Отримати загальну інформацію про наявну фізичну і віртуальну пам'ять у системі можна з допомогу команди ***free***. Крім цього, виведення містить інформацію про розмір буферів системи.

$ free -h

Отримати інформацію про використання дискового простору в системі можна за допомогою утиліти ***df***. Виведення команди містить дані за усіма змонтованими на поточний момент файловими системами із зазначенням відсотка використаного простору і точки монтування кожної з них. Опція ***-h*** призводить форматування виведення до зручного для користувача вигляду.

$ df -h

Щоб дізнатися розмір не всього обсягу в цілому, а будь-якої директорії, можна застосувати утиліту ***du***. Початкове виведення команди містить розміри всіх вкладених директорій, є можливість управляти глибиною вкладеності. Також, як і у випадку з командою ***df***, може застосовуватися опція ***-h***.

$ du -h

Планування повторюваних завдань

Для більшості завдань, що стоять перед системними адміністраторами, характерне періодичне виконання. Для зручності складання розкладу для користувача завдань в операційній системі є служба **cron**.

Завдання планувальника **cron** за рядками перераховані в спеціальному **crontab**-файлі. Записи у файлі мають такий вигляд:

10 15 * * * /home/user/my_script.sh

Де п'ять полів, розділених проміжками, означають числові представлення хвилин, годин, днів місяця, місяців на рік і дня тижня відповідно. Символ «*****» відповідає будь-якому значенню. Символ «**/**» служить для вказівки додаткової періодичності завдання. Наприклад, «***/3**» в першому полі означає «кожні 3 хвилини». У наведеному вище прикладі для користувача скрипт **my_script.sh** буде виконуватися кожен день о 15 годині і 10 хвилинах.

Кожен користувач може мати свій файл **crontab**, щоб повідомити системі ім'я файлу, необхідно виконати команду:

$ crontab filename

Виведення наявних завдань виконується цією утилітою з опцією ***-l***. Очищення списку завдань виконується командою ***crontab*** з опцією ***-r***. Редагування наявного файлу завдань можливо текстовим редактором з використанням опції ***-e***, наприклад:

$ crontab -e

Після додавання файлу завдання або зміни наявного файлу відбувається перевірка синтаксису.

Завдання

1. Ознайомтеся з роботою команд, які наведені у лабораторній роботі, за допомогою довідкового керівництва.
2. Створіть файл ***proc1***, що містить список процесів користувача **root**, відсортоване за ідентифікатором батьківського процесу. Використовуйте команду ***ps*** і вивчені раніше команди.
3. Отримайте інформацію про процеси вашого користувача, що мають статус «працездатний».

4. Додайте до файлу ***proc1*** відомості про процес, що в цей момент споживає більшість ресурсів центрального процесора.
5. Запустіть утиліту ***top***. Вивчіть вміст інформаційних полів, що надаються утилітою. Отримайте інформацію про ступінь використання ресурсів системи, кількості користувачів, часу роботи системи.
6. Отримайте список завдань поточної сесії термінала.
7. Використовуючи команди ***fg*** і ***bg*** і поєднання клавіш «***Ctrl+Z***» і «***Ctrl+C***» навчитеся переміщувати завдання з фону на перший план і навпаки.
8. Отримайте список сигналів для команди ***kill***. Завершіть запущені процеси за допомогою сигналів ***SIGKILL*** і ***SIGTERM***.
9. Протягом 30 с з інтервалом в 3 с збирайте статистику про використання ресурсів системи. Отримайте інформацію про середню завантаженість процесора протягом останніх 15 с.
10. Опишіть поточний стан сторінок пам'яті та твердих дисків, доступних у вашій системі.
11. Отримайте інформацію про розмір вашого домашнього каталогу, отримайте список 3 найбільших каталогів у вашій домашній директорії за допомогою команд, вивчених раніше.
12. Створіть завдання, згідно з яким:

    – кожну хвилину у файл **~/memory/stat** буде додаватися інформація про поточний стан пам'яті, без урахування розміру підкачки і заголовка;

    – кожні 3 хвилини файл ***~/memory/stat*** буде упаковуватися в архів.
13. Після виконання роботи видаліть всі записи з вашого crontab-файлу.

Контрольні питання

1. Які способи отримання інформації про процеси в системі ви знаєте?
2. Як можна керувати виведенням утиліти ***top***?
3. Які сигнали відправляються поєднаннями клавіш «***Ctrl+Z***» і «***Ctrl+C***»?
4. Які типи процесів ви знаєте?
5. Як отримати інформацію про стан пам'яті?
6. Як отримати інформацію про доступний дисковий простір?
7. Як відбувається робота з файлами завдань планувальника **cron**?

Лабораторна робота № 10. Розробка сценаріїв мовою оболонки bash (частина 1). Змінні, командні файли, файли ініціалізації

Основи розробки сценаріїв оболонки

Взаємодія користувача з операційною системою здійснюється через оболонку, яка представляє собою зовнішню програму. Відразу після запуску оболонки проводиться її ініціалізація для установки ряду параметрів. При цьому оболонкою проводиться читання двох файлів: ***/etc/profile*** і ***~/.profile***. У першому з них містяться налаштування параметрів, що є загальними для всіх користувачів. У другому файлі кожен користувач може розмістити власні налаштування для роботи з оболонкою.

Призначена для користувача оболонка може бути запущена на виконання в 2 режимах – інтерактивному і не інтерактивному. Коли оболонка видає користувачеві запрошення, вона працює в **інтерактивному** режимі. Це означає, що оболонка приймає введення від користувача і виконує команди, які користувач вказує. Оболонка називається інтерактивною, оскільки вона взаємодіє з користувачем. Завершення роботи з оболонкою в цьому випадку відбувається по команді користувача.

У **не інтерактивному** режимі оболонка не взаємодіє з користувачем. Замість цього вона читає команди з деякого файлу і виконує їх. Коли буде досягнуто кінець файлу, оболонка завершиться. Запуск оболонки в не інтерактивному режимі можна здійснити таким способом:

$ /bin/bash <ім'я_файлу>

Тут *<ім'я файлу>* – ім'я файлу, що містить команди для виконання. Такий файл називається **сценарієм оболонки**. Він є текстовим файлом і може бути створений будь-яким доступним текстовим редактором.

Користувачеві може представляти незручність кожен раз вказувати ім'я програми-оболонки */bin/bash* для виконання сценарію. Для того, щоб мати можливість виконувати сценарій, набираючи тільки його ім'я, перш за все необхідно зробити його виконуваним. Для цього необхідно встановити відповідні права доступу до файлу за допомогою команди ***chmod*** у такий спосіб:

$ chmod u+x <ім'я_файлу>

Крім цього, необхідно явно вказати, яка оболонка повинна використовуватися для виконання такого сценарію. Це можна зробити, розмістивши в першому рядку сценарію послідовність символів ***#!/bin/bash***. Тут вказується, що для виконання сценарію необхідно використовувати оболонку ***/bin/bash***.

Сценарій може містити коментарі. Коментар – це оператор, який може розміщуватися в сценарії оболонки, але оболонка не виконується. Коментар починається з символу **#** і триває до кінця рядка. Нижче наведено приклад короткого сценарію:

```
#!/bin/bash
# Приклад короткого сценарію
date
who
```

Складені команди

Крім простих команд можна формувати складові команди. Вони являють собою упорядкований набір простих команд, пов’язаних між собою за допомогою спеціальних **операторів управління (ОУ)**. Складена команда має такий синтаксис:

```
<команда> <ОУ> <команда> <ОУ> ... <ОУ> <команда>
```

У таблиці 1 наведено найбільш часто використовувані оператори управління та наведені пояснення щодо виконання відповідних складених команд.

Таблиця 1.

Оператори управління у складених командах

Оператор управління в складеній команді	*Правило виконання*
команда1 ; команда2	«команда2» виконується після завершення виконання «команди1» у будь-якому разі
команда1 && команда2	«команда2» виконується, якщо «команда1» була виконана успішно
команда1 || команда2	«команда2» виконується, якщо «команда1» не була виконана успішно

Змінні

Змінна – це «слово», якому присвоєне значення. Оболонка дозволяє створювати і видаляти змінні, а також присвоювати їм значення. У більшості випадків розробник відповідальний за управління змінними в сценаріях. Використання змінних дозволяє створювати гнучкі сценарії, що легко адаптуються. Визначаються змінні таким чином (відсутність проміжків до і після символу «**=**» істотно!):

```
<ім’я_змінної>=<значення>
```

Наприклад, ***MY_NAME=Sergey*** визначає змінну з ім'ям ***MY_NAME*** і привласнює їй значення ***Sergey***. Імена змінних визначаються за тими ж правилами, що і в мові програмування С. У змінній можна зберігати будь-яке потрібне значення. Особливий випадок – коли це значення містить проміжки. Для правильного виконання такої дії вказане значення достатньо взяти в лапки.

Для того, щоб отримати значення змінної, перед її ім'ям необхідно поставити знак ***$***. У тому випадку, коли деяка змінна стає непотрібною, її можна видалити командою:

```
unset <ім'я_змінної>.
```

Нижче наведено приклад, який ілюструє роботу зі змінними в сценаріях.

```
#!/bin/bash
#приклад операцій над змінними
MY_NAME=Sergey
MY_FULL_NAME="Sergey B. Sidorov"
echo name = $MY_NAME and full name =
$MY_FULL_NAME
unset MY_NAME
```

У результаті виконання команди ***echo*** на термінал буде видано повідомлення:

```
name = Sergey and full name = Sergey B. Sidorov
```

Усі розглянуті вище приклади змінних – це приклади скалярних змінних, тобто таких, які можуть зберігати лише одне значення. Поряд з ними можна використовувати змінні-масиви. Доступ до елементів масиву здійснюється операцією індексування ***[]***. Мова програмування оболонки не вимагає попереднього оголошення змінної масиву із зазначенням його розмірності. При цьому окремі елементи масиву створюються в міру доступу до них. Нижче наведено приклад роботи з масивом ***NAME***.

```
#!/bin/bash
# Приклад використання масиву
NAME[0] = Сергій
NAME[10] = Катя
NAME[2] = Ліза
echo всі імена: ${NAME[*]}
echo ${NAME[10]} і ${NAME[2]} сестри
```

Якщо замість індексу елемента використовувати *, результатом виразу буде список значень всіх елементів масиву (в нашому випадку таких три).

Область видимості змінних

Кожна змінна має свою зону видимості. Припустимо, у сценарії визначена деяка змінна. Що буде з нею після завершення цього сценарію, чи доступна вона з інших сценаріїв, що викликаються вихідним сценарієм? У мові оболонки всі змінні діляться на три категорії: **локальні змінні**, **змінні оточення** і **змінні оболонки**.

Локальна змінна – це така змінна, яка існує тільки всередині конкретного екземпляра оболонки. Вона не доступна програмам, що запускаються на виконання з цієї оболонки. Усі розглянуті раніше змінні були локальними.

Змінна оточення – це змінна, яка доступна будь-якій програмі, запущеної з цієї оболонки. Деякі програми потребують певних змінних оточення. У такому випадку у сценарії їх можна визначити.

Змінна оболонки – це спеціальна змінна, яка встановлюється оболонкою і необхідна їй для коректної роботи. Деякі з них є змінними оточення, а деякі – локальними. У змінних з іменами ***1,2,3,…*** зберігаються параметри командного рядка. Тобто якщо якийсь сценарій ***script*** запустити у вигляді ***script something***, то в його оболонці ***$1 = something***.

Змінну можна розмістити в оточенні, виконавши команду експорту:

export <ім'я_змінної>.

У результаті виконання наступного сценарію на термінал будуть видані значення всіх змінних оточення оболонки і, в тому числі, змінної ***MY_NAME***.

```
#! /bin/bash
# Приклад розміщення змінної в оточенні
MY_NAME = Sergey; export MY_NAME;
set
```

Деякі стандартні змінні оточення та їх призначення вже розглядались у лабораторній роботі №6.

Засоби введення-виведення

Задачі введення-виведення можуть бути вирішені за допомогою використання спеціальних команд ***echo, printf, read***.

Команда ***echo*** видає на стандартний пристрій виведення значення всіх своїх параметрів. Команда ***printf*** подібна своєму аналогу – функції

printf у програмах мовою С. Першим параметром вказується форматний рядок, а далі перераховуються значення, що видаються.

Команда ***read <ім'я1> <ім'я2> ... <ім'яN>*** зчитує зі стандартного пристрою введення рядок та виділяє з нього окремі слова (групи символів, відокремлювані проміжками) і кожне слово заносить у зазначені як параметри відповідні змінні. При цьому якщо змінних менше, ніж виділених слів, то в останню з них буде занесена решта рядка.

Як приклад наведено сценарій, в якому від користувача запитується рядок, після чого він виводиться на термінал. Ключ ***–n*** команди ***echo*** забороняє перехід на наступний рядок після закінчення виведення, що дозволений за замовчуванням.

```
#!/bin/bash
#приклад використання засобів введення-виведення
echo –n Введіть рядок:
read INPUT
printf "%s\n" "$INPUT"
```

Підстановки

Виділяють такі види підстановок:

– **підстановка імен файлів** – дозволяє виконувати перетворення рядка, що містить шаблони, в список імен файлів;

– **підстановка змінної** – дозволяє управляти значенням змінної, ґрунтуючись на її стані;

– **підстановка команди** – дозволяє захопити виведення команди і підставити його в іншу команду;

– **арифметична підстановка** – дозволяє виконувати прості математичні обчислення, використовуючи оболонку.

Підстановка імен файлів ґрунтується на використанні різних шаблонів для завдання групи файлів у стислому вигляді замість повного їх перерахування. Шаблон * ототожнюється з будь-якою кількістю будь-яких символів в імені файлу. Шаблон ***?*** відповідає одному довільному символу, а шаблон ***[группа_символів]*** явно визначає набір символів, допустимих в місці його розташування. Нижче наведені приклади використання шаблонів:

****.cpp*** – список файлів, що закінчуються на **.cpp**

module?.h – список файлів з одним довільним символом замість ?

module[aA].cpp – **modulea.cpp** і / або **moduleA.cpp**

Механізм підстановки команди полягає у виконанні оболонкою вказаного набору команд і подальшої підстановки їх виведення замість

самих команд. Підстановка команди виконується під час запису команди у зворотних апострофах. Наприклад, у результаті виконання команди ***DATE = `date`*** змінна ***DATE*** прийме значення виведення команди ***date***.

Для обчислення значення арифметичного виразу ***expression*** необхідно використовувати команду такого вигляду:

```
$((expression))
```

При цьому у виразі можуть використовуватися операції додавання, віднімання, множення і цілочисельного ділення, а також круглі дужки для завдання порядку обчислень. Наприклад, у результаті виконання наступної команди змінна ***foo*** прийме значення **3**.

```
foo = $((((5+3*2)-4)/2))
```

Спочатку в командному інтерпретаторі Bourne була передбачена спеціальна команда, призначена для обробки математичних виразів. Команда **expr** дозволяла обробляти вираження, задані в командному рядку, але для цього застосовувалися надзвичайно складні конструкції:

```
$ expr 1+5
6
```

Командний інтерпретатор **bash** продовжує підтримувати команду **expr**, оскільки він повинен залишатися сумісним з командним інтерпретатором Bourne. Однак у **bash** передбачений більш простий спосіб виконання математичних обчислень. У командному інтерпретаторі **bash** у разі використання операції присвоювання результату математичних обчислень змінної можна включити математичний вираз у конструкцію, що складається зі знака долара зі квадратних дужок (***$[operation]***):

```
$ var1=$[1+5]
$ echo $var1
6
$ var2=$[$var1*2]
$ echo $var2
12
```

Кольорове виведення тексту у сценаріях bash

Для виведення у різних кольорах використовуються спеціальні послідовності, що визначають колір тексту та колір фону (див. ЛР № 6). У коді сценаріїв **bash** можливе використання даних послідовностей для виведення інформації у різних кольорах.

```
#!/bin/bash
red="\033[0;31m" #змінна, що визначає червоний колір тексту
blue="\033[0:34m" #змінна, що визначає синій колір тексту
```

normal="\033[0m" #змінна, що визначає звичайний колір тексту командного рядка

#оголошення змінної ***user***, що містить назву поточного користувача (змінна середовища оточення ***USER***), але червоного кольору
user="redUSER"

#оголошення змінної ***hello***, що містить рядок ***"Hello"*** синього кольору
#тут екранується назва змінної, оскільки здійснюється конкатенація значення змінної з літеральною константою
hello="${blue}Hello"

#виведення змінної та повернення до звичайного кольору тексту
echo -e "$hello, $user $normal"

Команда ***echo*** з ключем ***-e*** розпізнає escape-послідовності всередині рядків, завдяки чому текст у зазначеному вище прикладі буде виведений відповідним кольором.

Файли ініціалізації

Файли ініціалізації – це сценарії командної оболонки, які викликаються автоматично. Під час реєстрації користувача в bash запускається файл ***$HOME/.bash_profile***. Якщо ***.bash_profile*** відсутній, його роль виконує файл ***$HOME/.profile***. У кожній командній оболонці є файл ініціалізації самої оболонки ***$HOME/.bashrc***. У разі виходу із системи запускається файл ***$HOME/.bash_logout***.

Існують конфігурації файлів командної оболонки, які є загальними для всіх користувачів: ***/etc/bashrc, /etc/profile***.

Файли ініціалізації є текстовими файлами і, маючи відповідні права, можна їх редагувати.

$HOME/.bash_profile (***$HOME/.profile***) зазвичай містить команди, що формують інтерфейс. Наприклад, вітання, рядок запрошення, колірні настройки, значення глобальних змінних та ін.

$HOME/.bashrc містить команди, що розширюють або налаштовують командний інтерпретатор. Наприклад, виклик необхідних програм, оголошення псевдонімів тощо.

$HOME/.bash_logout містить команди, які виконуються під час виходу із системи. Наприклад, очистка екрана, рядок повідомлення і команда ***sleep***, яка задає паузу. Така пауза дозволяє прочитати виведені повідомлення.

Завдання

1. Створити сценарій, що видає таке повідомлення: «У моєму домашньому каталозі <n> підкаталогів: <підкаталоги>, та <m> файлів: <файли>».
2. Створити сценарій, що прочитує з екрана деяке слово і що виводить кількість символів у цьому слові.
3. Написати свій ***.bash_profile*** (***.profile***), що виконує такі функції:
   – виводить рядок вітання з вказівкою поточної дати;
   – дозволяє виконати програми, що зберігаються в ***myscripts*** як звичайні команди (за допомогою змінної середовища оточення PATH);
   – виводить рядок, що включає ім'я користувача, ім'я комп'ютера і поточний каталог, використовувати різні кольори.
4. Створити свій ***.bashrc***, в якому задаються такі команди:
   – команда резервного копіювання текстових файлів, створених власником за поточний день, окрім порожніх;
   – команда видалення всіх порожніх файлів з каталога.
5. Написати свій ***.bash_logout***, що виводить прощальне повідомлення.

Контрольні питання

1. Яке призначення командного інтерпретатора?
2. За що відповідає перший рядок файлу сценарію?
3. У чому різниця локальних і глобальних змінних?
4. Що представляють собою змінні командної оболонки?
5. Як у змінну записати результат виконання команди?
6. Які стандартні змінні ви знаєте? Їх призначення?
7. Як можна запустити скрипт на виконання?

Лабораторна робота № 11. Розробка сценаріїв мовою оболонки bash (частина 2). Аргументи командного рядка, управляючі структури, цикли. Перевірка умов командою test

Аргументи командного рядка

Передача аргументів у програму відбувається таким чином:

<ім'я програми> [<арг.1> <арг.2>…]

Усередині програми доступ до аргументів здійснюється за допомогою спеціальних змінних, званих позиційними:

$<n>, n – номер позиції аргументу;

$1 – перший;

$2 – другий.

Є ще додаткові можливості підстановки.

$0 – ім'я сценарію;

$* – всі аргументи одним рядком;

$@ – всі аргументи окремо;

$# – кількість аргументів;

$$ – ідентифікаційний номер поточного процесу;

$! – ідентифікаційний номер фонового завдання;

$? – ідентифікаційний номер виконуваної команди.

Команди shift і set

Команда ***shift*** зсуває номери аргументів.

Приклад:

```
echo "$0 має $# аргументів"
shift
echo "$0 has $# arguments"
```

Команда ***set*** встановлює позиційні параметри. Імена (ідентифікатори) позиційних параметрів складаються з однієї або декількох цифр, крім одиночного нуля. Значеннями позиційних параметрів є аргументи, які були задані під час запуску оболонки (перший аргумент є значенням позиційного параметра 1 та ін.). Змінити значення позиційного параметра можна за допомогою вбудованої команди ***set***.

Таким чином, позиційними параметрами можуть бути не обов'язково аргументи, а й виведення будь-якої команди.

Приклад:

```
echo "$1 $2 $3"
set `uname -a`
echo "$1 $2 $3"
```

Команди розгалуження

Мова оболонки містить два оператори розгалуження: ***if*** і ***case***.

Оператор ***if*** виконує дії залежно від істинності заданої умови і має такий синтаксис:

```
if <list1>
then
<commands1>
elif <list2>
then
<commands2>
else
<commands3>
fi
```

У наведеному записі як ***elif***, так і ***else*** можуть бути відсутні. Якщо тіло оператора ***if*** невелике, то зазвичай використовують запис в один рядок, такий як:

```
if <list1>; then <commands1>; else <commands2>; fi;
```

При цьому важливо правильно ставити ***;***.

Алгоритм виконання оператора ***if*** такий:

1) виконується ***<list1>***;

2) якщо результат виконання ***<list1>*** – істина, то виконується ***<commands1>*** і оператор ***if*** завершується;

3) інакше виконується ***<list2>*** і перевіряється результат його виконання;

4) якщо результат виконання ***<list2>*** – істина, то виконується ***<commands2>*** і оператор ***if*** завершується;

5) якщо результат виконання ***<list2>*** – хиба, то виконується ***<commands3>***.

Як умови ***<list1>*** і ***<list2>*** можуть використовуватися звичайні команди, але переважно – це виклик однієї або декількох команд ***test*** у вигляді ***test <вираз>*** або більш стисло ***test [<вираз>]***. Детально команда ***test*** розглянута нижче.

Нижче наведено приклад використання оператора ***if***:

```
#!/bin/bash
# Виконання програми з контролем її існування
if [ -x $HOME/script ]; then $HOME/script; fi;
```

У цьому прикладі перевіряється, чи існує в домашньому каталозі користувача файл з ім'ям ***script*** і чи є він виконуваним (атрибут **'x'**). Якщо це так, то сценарій запускає ***script*** на виконання.

Якщо необхідно зробити вибір з багатьох альтернатив, то зручніше використовувати оператор ***case***, що має такий синтаксис:

```
case значння in
шаблон1) список1 ;;
шаблон2) список2 ;;
…
esac
```

Оператор ***case*** є аналогом оператора switch у мові С і працює схожим чином. Рядок значення порівнюється з кожним із зазначених шаблонів до тих пір, поки не буде виявлена відповідність. Після виявлення відповідності виконується список, зазначений після шаблону. Два символи «**;;**» після списку мають таке ж значення, що і оператор ***break*** у програмах мовою С. Якщо ніяких відповідностей не виявлено, то оператор case завершується без виконання будь-яких дій.

Нижче наведено приклад використання оператора ***case***. Сценарій очікує як параметр назву фрукта і видає його опис.

```
#!/bin/bash
# Друк інформації про фрукти
if [ -z "$1" ]
then
echo "Помилка: пропущений параметр"
exit 1
fi

case "$1" in
apple) echo "Яблука дуже корисні" ;;
banana) echo "Банани дуже смачні" ;;
*) Echo "Нічого не можу сказати про" $1 ;;
esac
```

Синтаксис команди test

Команда **`test`** перевіряє зазначений вираз і закінчує свою роботу з кодом завершення 0 (істина) або 1 (хиба). Як вираз можуть використовуватися вирази 3 типів:

– перевірки характеристик файлів;
– порівняння рядків;
– числові порівняння.

Основний синтаксис для перевірки характеристик файлів такий:

test <опція> <ім'я_файлу>.

У наведеній нижче таблиці зведені можливі опції команди ***test*** для перевірки характеристик файлів.

Таблиця 1.

Опція	*Опис*
-b	Істина, якщо файл існує і є блоковим спеціальним файлом
-c	Істина, якщо файл існує і є символьним спеціальним файлом
-d	Істина, якщо файл існує і є каталогом
-e	Істина, якщо файл існує
-f	Істина, якщо файл існує і є звичайним файлом
-h	Істина, якщо файл існує і є символічним посиланням
-r	Істина, якщо файл існує і доступний на читання
-s	Істина, якщо файл існує і має ненульовий розмір
-S	Істина, якщо файл існує і є сокетом
-w	Істина, якщо файл існує і доступний за попереднім записом
-x	Істина, якщо файл існує і доступний на виконання

У наведеній нижче таблиці зведені можливі способи вказівки використання команди ***test*** для перевірки рядків.

Таблиця 2.

Опція	*Опис*
-z <рядок>	Істина, якщо рядок має нульову довжину
-n <рядок>	Істина, якщо рядок має ненульову довжину
<рядок1> = <рядок2>	Істина, якщо рядки збігаються
<рядок1>! = <рядок2>	Істина, якщо рядки різні

У разі використання числових порівнянь команда ***test*** має такий синтаксис:

test <число1> <оператор> <число2>

У наведеній нижче таблиці зведені можливі значення операторів у числових порівняннях.

Таблиця 3.

Оператор	*Опис*
-eq	Істина, якщо числа рівні
-ne	Істина, якщо числа не рівні
-lt	Істина, якщо менше
-le	Істина, якщо менше або дорівнює
-gt	Істина, якщо більше
-ge	Істина, якщо більше або дорівнюють

Організація циклів

Мова оболонки дозволяє організовувати циклічне виконання команд. У розпорядження користувачеві пропонується кілька варіантів циклів:

– ***while;***
– ***for;***
– ***select.***

Цикл while

Оператор циклу ***while*** має такий синтаксис:

```
while <команда>
do
<список>
done
```

Під час виконання циклу спочатку виконується ***<команда>***. Якщо результат ненульовий, то відбувається вихід з циклу, в іншому випадку виконується тіло циклу ***<список>*** і відбувається перехід на наступну ітерацію. Хоча як умову команда може використовувати будь-яку допустиму в GNU/Linux команду, зазвичай це команда ***test***. Нижче наведено приклад виведення на термінал десяти послідовних чисел від 0 до 9.

```
#! /bin/bash
# Арифметичні обчислення
x = 0
while [$x -lt 10]   # значення змінної x менше 10?
do
echo $x
x = `expr $x + 1`  # збільшення x на 1
done
```

Цикл for

Цикл ***for*** виконує команди з список для кожного елемента зі списку і має такий синтаксис:

```
for <ім'я> in <елемент1> <елемент2> ... <елементN>
do
<список>
done
```

Як елементи можна використовувати різні шаблони. В наведеному нижче прикладі всі файли з домашнього каталогу користувача, які закінчуються на ***.bash***, копіюються в каталог ***scripts*** і робляться доступними на читання всім користувачам.

```
#!/bin/bash
# виконання операцій над групою файлів
for FILE in $HOME/*.bash
do
cp $FILE ${HOME}/scripts
chmod a+r ${HOME}/scripts/${FILE}
done
```

Мова програмування оболонки містить оператори, які порушують нормальне виконання циклу. Ці оператори мають назви ***break*** і ***continue***. Вони працюють точно так само, як і їх аналоги в мові С.

Цикл select

Цикл ***select*** дозволяє створювати зручні меню. Він корисний, коли необхідно, щоб користувач вибрав один елемент із запропонованого списку. Оператор ***select*** має такий же вигляд, як і оператор ***for***, за винятком ключового слова.

Під час виконання цього оператора циклу всі елементи зі списку висвічуються на екрані разом з їх порядковими номерами, після чого з'являється спеціальне запрошення для введення. Зазвичай воно має вигляд ***#?***. Введений користувачем номер пункту меню записується в змінну ***REPLY***. Якщо ***$REPLY*** містить номер пункту меню, то в змінну ***<ім'я>*** заноситься значення відповідного елемента зі списку. В іншому випадку список буде виведений заново.

Після того, як користувачем буде зроблений допустимий вибір, виконується команди зі ***<список>***, після чого виконання циклу повторюється з самого початку (висвітиться запрошення для введення та ін.). Для повторного відображення меню у відповідь на запрошення введення слід натиснути клавішу Enter. Вихід з циклу здійснюється тими ж засобами, що і для ***while*** та ***for***.

Нижче наводиться приклад використання оператора ***select***. У користувача запитується тип пристрою «миша», і залежно від зробленого вибору виконуються певні дії. В цьому випадку це просто виведення повідомлення, що підтверджує зроблений вибір. У наведеному сценарії звернемо увагу на другий рядок. У більшості випадків діючий за замовчуванням вигляд запрошення не влаштовує. Визначаючи змінну PS3, можна задати необхідний текст запрошення.

```
#!/bin/bash
# «Конфігурація» пристрою «миша»
PS3 = "Оберіть тип пристрою «миша»:"
select ITEM in Microsoft Logitech ps2 none
do
case $ITEM in
Microsoft) echo "дії по «миші» Microsoft" ;;
Logitech) echo "дії по «миші» Logitech" ;;
ps2) echo "дії по «миші» ps2" ;;
none) echo "вибрано – немає «миші»" ;;
esac
break
done
```

Завдання

1. Написати сценарій, що приймає три аргументи (***a***, ***b***, ***c***) і виводить значення ***(a+b)/c***.
2. Написати сценарій, що приймає два числові аргументи і виводить найбільший з них. У тому випадку, якщо аргументів більше 2, треба вивести повідомлення про помилку.
3. Потрібно перевірити, чи є файл звичайним або він є каталогом (аргументом є назва файлу). Якщо це звичайний файл, то сценарій повинен виводити ім’я файлу і його розмір. У разі, якщо розмір файлу перевищує кілобайт, то розмір повинен виводитися в кілобайтах. Якщо розмір перевищує мегабайт – у мегабайтах.
4. Написати сценарій, який виводить посекундно в циклі імена файлів поточного каталогу і їх порядковий номер (використовувати команду ***sleep <секунди>***).
5. Написати сценарій, який генерує 100 файлів ***1.txt ... 100.txt***, і в кожен файл записує поспіль 100 чисел N, де N = порядковий номер файлу. Потім скрипт повинен з’єднати в один файл всі файли з парними номерами (***even.txt***) і в інший файл – всі файли з непарними номерами (***odd.txt***).

Контрольні питання

1. Яким чином у програму можна передати аргументи?
2. Як у програмі можна отримати доступ до аргументів, з якими визивалася програма?
3. Які спеціальні змінні можна використовувати в скриптах командного інтерпретатора?
4. Яким чином можна запускати програми залежно від результату виконання інших програм?
5. Які оператори розгалуження існують у bash? Наведіть їх синтаксис.
6. Для чого призначена та який синтаксис має команда ***`test`***?
7. Які види циклів існують у bash? Наведіть їх синтаксис.

Лабораторна робота № 12. Розробка сценаріїв мовою оболонки bash (частина 3). Використання функцій у сценаріях

Створення функцій

Під час написання сценаріїв командного інтерпретатора часто виникає необхідність використовувати один і той же код у декількох місцях. Якщо це лише невеликий фрагмент коду, то труднощів не виникне. Але якщо в сценарії командного інтерпретатора необхідно знову і знову вводити великі шматки однакового коду, то істотно зростають витрати часу на написання програми.

У командному інтерпретаторі **bash** передбачений спосіб дії в подібних ситуаціях, який заснований на застосуванні функцій, визначених користувачем. Повторювані фрагменти коду сценарію командного інтерпретатора можна оформляти у вигляді функції, а потім використовувати вже саму функцію в сценарії стільки разів, скільки буде потрібно.

Для створення функції в сценаріях командного інтерпретатора **bash** можна скористатися одним із двох форматів. У першому форматі використовується ключове слово ***function*** поряд з ім’ям функції, яким позначається блок коду:

```
function <ім’я>
{
<команди…>
}
```

Атрибут ***<ім’я>*** визначає унікальне ім’я, присвоєне функції. Кожна функція, яка визначається в сценарії, повинна отримати унікальне ім’я.

У цьому визначенні ***<команди…>*** – це одна або кілька команд командного інтерпретатора **bash**, з яких складається ця функція. Після виклику функції командний інтерпретатор **bash** виконує кожну з цих команд в тому порядку, в якому вони присутні у визначенні функції, точно так, як і під час виконання звичайного коду сценарію.

Другий формат визначення функції в сценарії командного інтерпретатора **bash** більшою мірою нагадує формат, який застосовується для визначення функцій в інших мовах програмування:

```
<ім’я> ()
{
<команди…>
  }
```

Порожні круглі дужки після імені функції вказують на те, що далі йде визначення функції. На цей формат поширюються такі ж правила іменування функцій, як і на вихідний формат функцій сценаріїв командного інтерпретатора.

Використання функцій

Щоб скористатися функцією в сценарії, необхідно вказати в рядку коду ім'я функції за таким же принципом, як відбувається виклик будь-якої іншої команди командного інтерпретатора:

Приклад 1. Файл example1.sh

```
#!/bin/bash
# використання функції у сценарії

func1()
{
echo "Приклад функції"
}

count=1
while [ $count -le 5 ]
do
func1
count=$[ $count + 1 ]
done
echo "Кінець циклу"
func1
echo "Кінець сценарію"
```

Виконання сценарію

```
$ ./example1
Приклад функції
Приклад функції
Приклад функції
Приклад функції
Приклад функції
Кінець циклу
Приклад функції
Кінець сценарію
```

У цьому випадку за кожного посилання на ім'я функції ***func1*** командний інтерпретатор **bash** звертається до визначення ***func1()*** і виконує всі команди, передбачені у цьому визначенні.

Визначення функції не обов’язково має бути приведене у сценарії командного інтерпретатора в першу чергу, однак необхідно дотримуватися обережності. **За спроби використання функції до її визначення з’являється повідомлення про помилку.**

Необхідно також дотримуватися певних правил, що стосуються імен функцій. Кожне ім’я функції має бути унікальним, оскільки в іншому випадку виникає проблема. Якщо в сценарії трапляється ще одне визначення однієї і тієї ж функції, то відбувається так зване перевизначення, і нова версія функції перекриває її вихідну версію, не викликаючи жодних повідомлень про помилку.

Приклад 2. Файл example2.sh

```
#!/bin/bash
# перевірка використання повторюваного імені функції

func1()
{
echo "Перше визначення імені функції"
}

func1

func1()
{
echo "Повторення одного і того ж імені функції"
}

func1
echo "Кінець сценарію"
```

Виконання сценарію

```
$ ./example2
Перше визначення імені функції
Повторення одного і того ж імені функції
Кінець сценарію
```

Початкове визначення функції ***func1*** діяло цілком задовільно, але після обробки другого визначення функції ***func1*** у всьому подальшому коді сценарію виклик функції призводить до застосування другого визначення.

Повернення значення з функції

У командному інтерпретаторі **bash** функції розглядаються як свого роду мінісценарії, зі своїм статусом виходу. Передбачено три способи, за допомогою яких можна формувати статус виходу для користувацьких функцій.

1. Статус виходу, що заданий за замовчуванням. За замовчуванням статус виходу функції визначається як статус виходу, повернутий останньою командою у функції. Після завершення виконання функції можна скористатися стандартною змінною ***$?*** для визначення статусу виходу функції.

Приклад 3. Файл example3.sh

```
#!/bin/bash
# перевірка статусу виходу функції

func3()
{
echo "Намагання відобразити неіснуючий файл"
ls -l badfile
}

echo "Тестування функції:"
func3
echo "Статус виходу: $?"
```

Виконання сценарію

```
$ ./example3
Тестування функції:
Намагання відобразити неіснуючий файл
ls: badfile: No such file or directory
Статус виходу: 1
```

У цьому випадку статус виходу функції дорівнює 1, оскільки виконання останньої команди у функції закінчилося невдачею. Але це не дозволяє дізнатися, чи закінчилось виконання всіх інших команд у функції успішно. Змінимо тіло функції ***func3*** таким чином:

```
func3()
{
ls -l badfile
echo " Намагання відобразити неіснуючий файл"
}
```

Цього разу останньою інструкцією до функцій був виклик команди відлуння, який завершився успішно, тому функція має статус виходу 0, незважаючи на те, що виклик однієї з команд у функції закінчився невдачею. Таким чином, підхід, який передбачає використання заданого за замовчуванням статусу виходу функції, не завжди виправданий. На щастя, передбачені інші способи формування статусу виходу функції.

2. Використання команди *return*. У командному інтерпретаторі **bash**, для виходу з функції з конкретним статусом може застосовуватися команда ***return***, яка дозволяє задати одне цілочисельне значення для визначення статусу виходу функції, що може служити простим способом завдання статусу виходу функції програмним шляхом.

Приклад 4. Файл example4.sh

```
#!/bin/bash
# використання команди return у функції

func4()
{
Echo "Введіть значення: "
read value
echo "Подвоєння значення"
return $[$value*2]
}

func4
echo "Нове значення дорівнює $?"
```

Функція ***func4*** подвоює значення, що міститься у змінній ***$value***, отримане в рядок введення даних користувачем. Потім у цій функції відбувається повернення сформованого результату за допомогою команди ***return***, а повернене значення відображається в сценарії з використанням змінної ***$?***.

Однак під час використання цього способу повернення значення з функції слід дотримуватись обережності. Щоб уникнути проблем, необхідно керуватися двома рекомендаціями:

– обов’язково виконувати вибірку значення, що повертається, відразу після завершення функції;

– не забувати про те, що статус виходу повинен знаходитися в межах від 0 до 255.

3. Використання виведення з функції. За аналогією з тим, що можна перехоплювати виведення команди за допомогою змінної командного інтерпретатора, можна також перехоплювати за допомогою змінної командного інтерпретатора виведення функції. Цей спосіб може використовуватися для отримання висновку будь-якого типу з функції для присвоювання його змінній:

result = 'dbl'

Ця команда присвоює висновок функції ***dbl*** змінної командного інтерпретатора ***$result***. Нижче наведено приклад використання цього способу в сценарії.

Приклад 5. Файл example5.sh

```
#!/bin/bash
# використання команды echo для повернення значення

func5() {
echo "Введіть значення: "
read value
echo $[$value*2]
}

result=`func5`
echo "Нове значення дорівнює $result"
```

Виконання сценарію

```
$./example5
Введіть значення: 200
Нове значення дорівнює 400

$./example5
Введіть значення: 1000
Нове значення дорівнює 2000
```

У новій версії функції подвоєння значення тепер застосовується інструкція ***echo*** для відображення результату обчислення. У сценарії просто відбувається перехоплення виведення функції ***func5***, а не перегляд статусу виходу для отримання відповіді.

Передача параметрів у функцію

У командному інтерпретаторі **bash** функція розглядається як свого роду мінісценарій. Це, зокрема, означає, що у функцію можна передавати параметри, як і у звичайний сценарій.

У функціях можна використовувати стандартні змінні середовища параметрів для представлення будь-яких параметрів, переданих у функцію в командному рядку. Наприклад, ім'я функції визначено в змінній ***$0***, а всі параметри в командному рядку функції визначаються з використанням змінних ***$1, $2*** та ін. Крім того, для визначення кількості параметрів, що передаються у функцію, можна використовувати спеціальну змінну ***$#***.

Задаючи функцію у сценарії, необхідно приводити параметри в тому ж командному рядку, як і функцію, приблизно так:

func1 $value1 10

Потім у функції можна здійснити вибірку значень параметрів з використанням змінних середовища параметрів. Нижче наведено приклад застосування такого способу для передачі значень у функцію.

Приклад 6. Файл example6.sh

```
#!/bin/bash
# передача параметрів у функцію

addem()   #функція додавання 2 чисел
{
if [$# -eq 0] || [$# -gt 2]
then
echo -1
elif [$# -eq 1]
then
echo $[$1+$1]
else
echo $[$1+$2]
fi
}

echo "Сума 10 та 15: "
value='addem 10 15'
echo $value
echo "Спробуємо знайти суму тільки одного числа: 
"
value='addem 10'
echo $value
```

```
echo "Тепер спробуємо знайти суму жодного числа: "
value='addem'
echo $value
echo "Наприкінці спробуємо знайти суму трьох чисел: "
value='addem 10 15 20'
echo $value
```

Виконання сценарію

```
$./example6
Сума 10 та 15: 25
Спробуємо знайти суму тільки одного числа: 20
Тепер спробуємо знайти суму жодного числа: -1
Наприкінці спробуємо знайти суму трьох чисел: -1
```

У сценарії ***example6*** функція ***addem*** спочатку перевіряє кількість параметрів, переданих зі сценарію. Якщо параметри не задані або кількість параметрів більше двох, функція ***addem*** повертає значення -1. Якщо заданий тільки один параметр, функція ***addem*** складає значення цього параметра з самим собою для отримання результату. Якщо задані два параметри, функція ***addem*** для формування результату складає отримані значення.

Завдання

1. Написати функцію, яка:
   - вводить символьний рядок, що містить маршрутне ім’я деякого файлу;
   - перевіряє введене маршрутне ім’я, якщо воно починається з символу /, на збіг його першої частини з маршрутним ім’ям домашнього каталогу користувача;
   - якщо введене маршрутне ім’я містить маршрутне ім’я домашнього каталогу або є відносним, то перевіряє існування зазначеного файлу, в іншому випадку виводить на екран повідомлення про помилку;
   - якщо файл існує, то виводить на екран його вміст;
   - якщо файл не існує, то створює його і записує в нього рядок, переданий як параметр;
2. Написати функцію, яка:
   - у заданому першим параметром каталозі знаходить всі прості файли, в яких містяться задані другим і третім параметрами символьні рядки;

– у знайдених файлах видаляє всі повторювані рядки;
– виводить на екран імена всіх отриманих файлів.

3. Написати функцію, яка:
   – у каталозі, ім’я якого передається першим параметром, знаходить всі прості файли розміром більше заданого другим параметром;
   – створює в зазначеному каталозі 3 нових каталоги;
   – поміщує в створені каталоги файли з вихідного каталогу: у перший – файли, що містять один рядок із заданим словом, у другій – файли з двома такими рядками, в третій – з трьома;
   – імена всіх файлів, які не включені в нові каталоги, виводить на екран.

Контрольні питання

1. Синтаксис оголошення функції у bash.
2. Які існують способи повертання значення з функції?
3. Як передаються параметри у функцію?

Лабораторна робота № 13. Розробка сценаріїв bash для графічних робочих столів. Використання пакета zenity

Графічні оболонки Linux

Графічна оболонка середовища, являє собою оболонку операційної системи, яка забезпечує зв'язок користувача з різними функціями системи, тобто, просто кажучи, дозволяє управляти нею.

У **Windows**, по суті, всього одна графічна оболонка, тобто користувач може змінювати графічну тему, змінювати налаштування деяких графічних елементів (наприклад, змінити іконку у папки), але сам графічний інтерфейс буде залишатись тим же.

У **Linux** же таких графічних оболонок кілька і користувач може встановити їх всі, а під час входу в систему вибрати ту, яка йому більше подобається. Найбільш поширені середовища в Linux – це **KDE** та **GNOME**. Вони являють собою два конкуруючих середовища, тобто якщо користувач має більш-менш потужний комп'ютер, то вибір графічної оболонки зупиниться на цих двох середовищах, а вже яку встановити – буде залежати тільки від нього.

Обираючи графічне середовище, користувач обирає набір програм, з якими буде працювати. Середовище **KDE** використовує для роботи бібліотеку **Qt**, а **GNOME** – **GTK**. Якщо був обраний KDE, то будуть встановлені програми, що працюють на бібліотеці Qt, якщо в GNOME, то відповідно встановлюються програми, засновані на бібліотеці GTK. Наприклад, як файловий менеджер під час вибору KDE буде встановлено Dolphin, а у результаті вибору GNOME – Nautilus. Якщо дозволяє дисковий простір, то звичайно ж можна встановити їх разом і у результаті завантаження операційної системи можна вибирати, в якій графічній оболонці ми сьогодні захочемо працювати, але запустити програму під невластивим їй середовищем не вийде.

Графічне середовище XFCE

XFce є графічною оболонкою, що побудована на основі інструментального пакета **GTK+**, який використовується в **Gnome**, але набагато легша і призначена для тих, кому потрібен простий, ефективно працюючий стіл, який легко використовувати і налаштовувати. Переваги **XFce**:

– простий, легкий у користуванні робочий стіл, який повністю налаштовується за допомогою миші, з інтерфейсом «drag and drop»;

– інтегрований віконний менеджер, менеджер файлів, управління звуком, модуль сумісності з **Gnome** та інше;

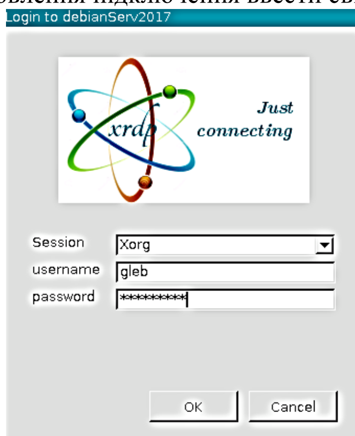
– стандартизовані меню і панелі інструментів, комбінації клавіш, колірні схеми та ін.;

– можливість використання тем, оскільки використовує **GTK+**;

– **XFce** швидка, легка і ефективна: ідеальна для застарілих / слабких машин або машин з обмеженою пам'яттю.

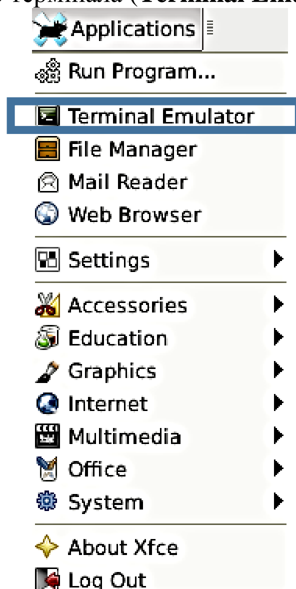
На сервері Debian в університеті встановлене графічне середовище **XFce**. Кожен з користувачів, зареєстрованих у системі, може віддалено під'єднатись до нього за допомогою оснастки **mstsc** (підключення до віддаленого робочого столу), увівши IP-адресу сервера Linux,

а потім після встановлення підключення ввести свій логін та пароль.



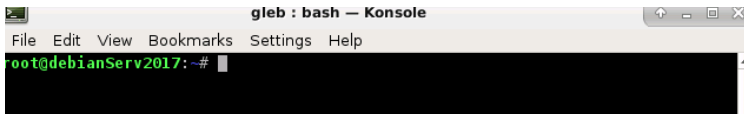
Після введення необхідних даних стане можливою робота на віддаленому робочому столі **XFce**.

Подібно до меню «Пуск» у Windows, у графічній оболонці **XFce** існує меню «Applications», що дозволяє відкрити певні програми, які розподілені за категоріями. Для виконання цієї лабораторної роботи знадобиться емулятор термінала (**Terminal Emulator**).



Відкрити емулятор термінала також можна і за допомогою меню швидкого доступу внизу робочого стола **XFce**.

Після відкриття емулятора термінала ми бачимо звичний інтерфейс командного рядка.



Написання сценаріїв з використанням візуальних елементів графічних середовищ

Колись сценарії командного інтерпретатора представляли собою один з найбільш поширених способів організації роботи в системі, але з часом вони стали сприйматися як похмурі і нудні. Але так не має бути, якщо сценарій призначений для виконання в графічному середовищі. Передбачено широкий спектр можливостей забезпечення взаємодії з користувачем сценарію, не базованих виключно на інструкціях ***read*** і ***echo***.

Утиліта ***dialog*** – одна з небагатьох, які допомагають створювати доброзичливі до користувача (в плані призначеного для користувача інтерфейсу) сценарії і програми. Вона може конструювати різні діалогові вікна, не потребуючи при цьому запущений графічний інтерфейс – достатньо лише консолі, адже для малювання використовується бібліотека **ncurses**. Як наслідок, ***dialog*** добре пристосований для використання у сценаріях **bash**, в яких він і застосовується в більшості випадків.

Варто відзначити, що існують аналоги dialog'а, спеціально розраховані на використання в графічному оточенні – універсальний ***Xdialog*** (повністю сумісний з dialog), ***gtkdialog*** (вікна малюються за допомогою графічної бібліотеки GTK), ***kdialog*** (для роботи у графічному середовищі **KDE**) і ***zenity*** (призначений для використання разом з графічним середовищем **Gnome**). Слід підкреслити, що, принаймні, Xdialog, крім стандартних діалогових вікон з dialog'а, має ще й трохи своїх: деревовидні списки, поле для редагування багаторядкових текстових даних та ін.

У команді ***dialog*** використовуються параметри командного рядка для визначення того, який графічний елемент Windows повинен бути

сформований. Термін «графічний елемент» (widget) застосовується в пакеті dialog для позначення графічного об’єкта, приблизно відповідного одному з об’єктів Windows.

Ось короткий список діалогових вікон, які здатний малювати dialog:

– вікна з кнопками Так / Ні;
– меню;
– вікна з полями введення;
– інформуючі вікна і просто вікна з текстом;
– поля з радіокнопками;
– вікна для вибору файлів;
– вікна для вибору необхідного часу, дати;
– спеціальні вікна для запиту пароля;

і багато іншого.

Огляд утиліти zenity

Zenity – це засіб створення діалогових вікон в режимі командного рядка. Слід зазначити, що насправді діалогові вікна створюються засобами **Gtk+**, тому в системі повинні бути встановлені відповідні бібліотеки. Від аналогічних програм ***zenity*** відрізняють більш витончені засоби реалізації GUI-елементів.

Завантажити актуальну версію цієї утиліти можна на Web-сайті розробників [**http://library.gnome.org/users/zenity/**]. Також цю програму можна знайти в репозиторіях деяких дистрибутивів Linux.

Застосування ***zenity*** для окремих команд в інтерактивному режимі не настільки ефективне, як під час написання сценаріїв. У сценаріях командної оболонки часто потрібна взаємодія з користувачем, щоб повідомити якусь інформацію, наприклад, про виникнення «нестандартної» ситуації. Також потрібно відображати інформацію про стан виконання операції, що триває досить довго. Крім цього, іноді сценарієм необхідно отримати деяку інформацію від користувача: вибір варіанта відповіді на поставлене запитання, вибір файлу із запропонованого списку та ін. Все це можна організувати за допомогою ***zenity***.

Необхідно уточнити, що після закриття діалогового вікна, ***zenity*** повертає числовий код завершення операції:

– 0 – означає, що користувач натиснув в діалоговому вікні кнопку **«OK»** або **«Закрити» (Close)**;

– 1 – означає, що користувач натиснув кнопку **«Скасувати» (Cancel)** або скористався функціями (кнопками) вікна, щоб закрити його;

– -1 – повідомляє про те, що операція завершилася з помилкою;

– 5 – діалогове вікно було закрито після закінчення інтервалу часу очікування.

Створення діалогового вікна для виведення повідомлень

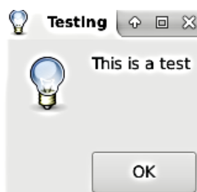
Щоб почати використовувати ***zenity***, на практиці не потрібно володіти особливими знаннями або вміннями, досить просто познайомитися з різними опціями (ключами), що дозволяють сповна використовувати можливості цієї програми.

У ***zenity*** визначено чотири типи діалогових вікон для виводу повідомлень:

- помилка (ключ ***--error***);
- інформація (ключ ***--info***);
- питання (ключ ***--question***);
- попередження (ключ ***--warning***).

Просте повідомлення визначається такою командою:

$ zenity --info --title="Testing" --text="This is a test"



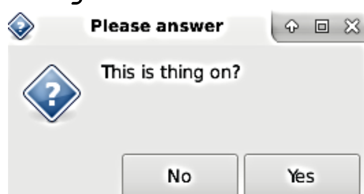
Якщо застосовуваний емулятор термінала підтримує мишу, то з'являється додаткова можливість клацати на кнопці ОК для закриття діалогового вікна. Для моделювання натиснення можна також використовувати команди клавіатури; в нашому випадку достатньо натиснути клавішу <Enter>.

Створення діалогового вікна для вибору варіанта відповіді

Діалогове вікно з питанням являє собою більш складний варіант графічного елемента повідомлення у ***zenity***, який дозволяє користувачеві відповісти на питання, що відображене у вікні, позитивно або негативно. Цей елемент створює в нижній частині вікна дві кнопки, на однією з яких є напис «Yes», а на іншій – «No». Користувач може переходити від однієї кнопки до іншої за допомогою миші, клавіші табуляції або клавіш зі стрілками. Щоб вибрати кнопку, користувач може натиснути клавішу пробілу або <Enter>. Для створення такого діалогового вікна до команді ***zenity*** потрібно вказати ключ ***--question***.

Нижче наведено приклад використання графічного елемента yesno.

```
$ zenity --question --title="Please answer" –
text="Is this thing on?"
```




```
$ echo $?
1
$
```

(була натиснута кнопка «No»)

Статус виходу команди ***zenity*** встановлюється залежно від того, яку кнопку обрав користувач. Якщо обрана кнопка **No**, статусом виходу є 1, а якщо обрана кнопка **Yes**, статус виходу набуває значення 0.

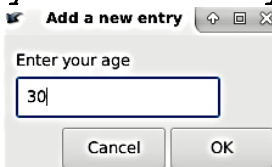
Приклад. Файл yesno.sh

```
#!/bin/bash

zenity --question \
--title="Displays the contents of the current
directory" \
--text="Do you want to use a long format?"

case $? in
   '0')
    ls -l;;
   '1')
    ls;;
   '-1')
    echo "There was an error!"
    exit 1;;
esac
```

Введення та редагування тексту

Під час взаємодії з користувачем неможливо обійтися без засобів введення текстової інформації. Звісно, ***zenity*** надає і ці засоби. Для цього разом з командою потрібно використати ключ ***--entry***. При цьому графічний елемент створює дві кнопки – **OK** і **Cancel**. Якщо обрана кнопка Cancel, то статусом виходу команди є 1; в іншому випадку статус виходу дорівнюватиме 0, а введені користувачем дані при цьому виводяться у стандартний потік **STDOUT**.

```
$ zenity --entry --text="Enter your age"
```




```
30
$ echo $?
```
(була натиснута кнопка «ОК»)
```
0
$
```

Також можна присвоїти результат виконання команди ***kdialog*** певній змінній.

```
$ age=`zenity --entry --text="Enter your age"`
$ echo $age
30
```

Після заголовку (перший параметр) можна задати другий параметр, який представляє значення за замовчуванням:

```
$ age=`kdialog –inputbox «Enter your age:» 30`
```

У полі введення може міститись слово, визначене за замовчуванням (ключ ***--entry-text***). Також введені в поле дані можна приховати, використавши ключ ***--hide-text***.

Приклад. Файл inputPassword.sh

```
#!/bin/bash

if zenity --entry --title="Verify access rights" \
     --text=" Input your password:" \
     --entry-text="password" \
     --hide-text
then
  echo $?
else
  zenity --error --text="Password not entered"
fi
```

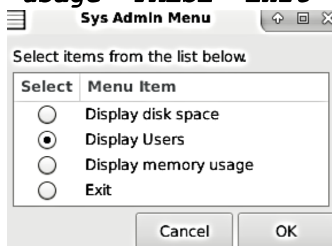
Списки

У ***zenity*** не залишені без уваги і такі важливі елементи інтерфейсу, як списки (ключ ***--list***). Поряд зі звичайним текстовим списком є можливість створення списків з перемикачами і радіокнопками. Елементи для діалогового вікна, що містить список, повинні бути вказані в такому порядку: стовпці формують рядок, далі символ нового рядка, після чого вводяться стовпці, що становлять наступний рядок та ін. Рядки елементів списку можуть бути передані в діалогове вікно через потік стандартного введення. Новий стовпець вказується за допомогою ключа ***--column***.

Перемикачі (ключ ***--checklist***) і радіокнопки (***--radiolist***) можуть розташовуватися тільки в першому стовпці, тому в рядках для цих варіантів списку найпершим елементом обов'язково повинен бути рядок «TRUE» або «FALSE».

За допомогою списків з радіокнопками у першому стовпці можна реалізувати інтерактивне меню.

```
$ zenity --list --radiolist --title «Sys Admin Menu» --column "Select" --column "Menu Item" FALSE "Display disk space" FALSE "Display users" FALSE "Display memory usage" FALSE "Exit"
```




```
Display users
$
```

Приклад. Файл menu.sh

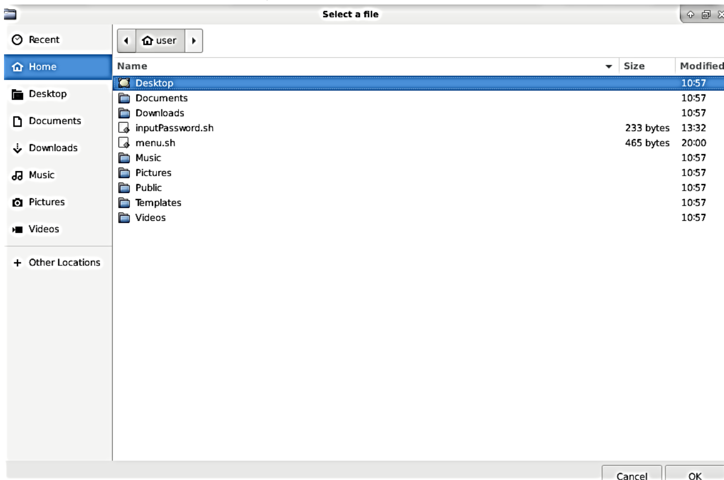
```
#!/bin/bash
command=` zenity --list --radiolist --title «Sys Admin Menu» \
          --column "Select" --column "Menu Item" \
          FALSE "Display disk space" \
          FALSE "Display users" \
          FALSE "Display memory usage" \
          FALSE "Exit"`
if [ $? -eq 0 ]
```

```
then
   case $command in
    1)
     df ;;
    2)
     who ;;
    3)
     vmstat ;;
    4)
     exit ;;
   esac
fi
```

Діалогове вікно вибору файлу

Команда ***zenity*** дозволяє скористатися ще кількома привабливими вбудованими графічними елементами. Під час роботи з іменами файлів надзвичайно зручним є графічний елемент ***--file-selection***, який дозволяє позбутися від необхідності введення імен використовуваних файлів, оскільки з його допомогою можна перейти безпосередньо до того каталогу, де знаходиться файл, і зробити вибір.

```
$ zenity --file-selection --title="Select a file" --filename=" $HOME/"
```



Додатковий ключ ***--filename*** задає місце розташування каталогу, з якого починається перехід по каталогам у вікні, або файл, який позначається вибраним за замовчуванням. Вікно графічного

елемента ***--file-selection*** містить лістинг каталогів, що знаходиться зліва, та лістинг файлів, який представлений справа. Повне ім'я обраного файлу у результаті виводиться до стандартного потоку виведення.

Нижче наведений приклад сценарію, який містить вікно вибору файлу, який потрібно видалити.

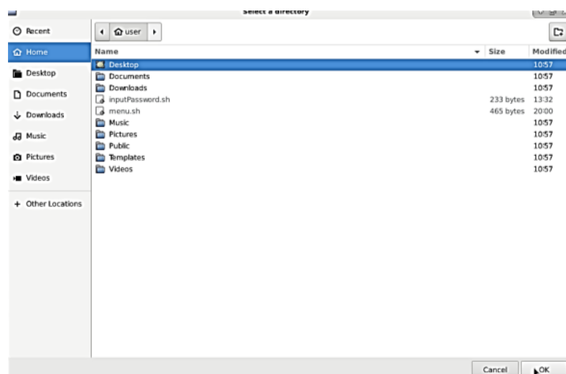
Приклад. Файл Deletefile.sh

```
#!/bin/bash

delete_file()
{
   local f=$1
   local m="$0: file $f failed to delete."
   if [ -f $f ]
   then
         rm $FILE && m="$0: $f file deleted."
   else
         m="$0: $f is not a file."
   fi

   zenity --info --title=" Remove file" --
text="$m"
}

file=`zenity --file-selection --title="Select a
file" --filename=" $HOME/"`

if [ ! -z $file ] then
   delete_file $file
```

Вікно вибору каталогу

Подібно до діалогового вікна вибору існуючого файлу, у ***zenity*** передбачена опція вибору існуючого каталогу. Для цього під час створення діалогового вікна вибору файлу необхідно вказати додатковий ключ ***--directory***. Оброблюється таке діалогове вікно так само, як і ділогове вікно вибору файлу.

```
$ kdialog –title «Select a file» --
getexistingdirectory ~
```



Завдання

1. Написати сценарій для графічних робочих столів, який шукає заданий користувачем у діалоговому вікні рядок у всіх файлах заданого за допомогою діалогового вікна вибору певного каталогу, що знаходиться у вашому домашньому каталозі. Передбачити перевірку, чи є елемент з обраним іменем каталогом. Якщо це каталог, вивести у інформаційному діалоговому вікні повний шлях і імена файлів, у вмісті яких присутній заданий рядок, і їх розмір, якщо ні, вивести повідомлення, що це не каталог.
2. Написати сценарій для графічних робочих столів, який приймає від користувача ім'я файлу (за допомогою діалогового вікна вибору файлу) та надає користувачеві обрати одну з дій над ним (переглянути, скопіювати, перемістити, видалити), використовуючи інтерактивне меню. Під час вибору користувачем певної команди передбачити необхідні параметри для її виконання (назва файлу або каталогу у разі копіювання або переміщення), а також вивести для користувача вікно для додаткового підтвердження, чи дійсно він бажає виконати цю дію.

Контрольні питання

1. Які елементи графічного інтерфейсу можна використати за допомогою команди kdialog?
2. Як обробити введення даних користувачем у діалозі?
3. Як додати до сценарію вікно з кнопками Yes / No?
4. Як додати до сценарію поле для введення даних?
5. Як додати до сценарію інтерактивне меню?
6. Як додати до сценарію вікно для вибору файлу та вікно для вибору каталогу?

Лабораторна робота № 14. Мережеві засоби Linux

Теоретичні відомості

Під час роботи в мережі за допомогою протоколу TCP\IP комп'ютеру призначаються такі параметри:

- IP-адреса вашого комп'ютера в мережі;
- широкомовна IP-адреса;
- ім'я домену, в який включений комп'ютер;
- маска підмережі;
- IP-адреса маршрутизатора (router);
- IP-адреса сервера імен (DNS-сервера).

Основні файли:

– ***/etc/sysconfig/network*** – загальна мережева інформація;

– ***/etc/sysconfig/network-scripts*** – каталог з файлами налаштувань і сценарії для роботи з різними типами мережевих приладів і підключень. Наприклад, ***файл /etc/sysconfig/ network-scripts/ifcfg-eth0*** містить інформацію про налаштування мережевої Ethernet-карти з інтерфейсом **eth0**;

– ***/etc/ppp*** – інформація протоколу **ppp**;

– ***/etc/init.d/*** – каталог з різними ініціалізаційними скриптами, серед яких сценарії **network**, **firewall** і деякі інші відповідають за налагодження мережі в момент завантаження і вимкнення комп'ютера.

Файл ***network*** у форматі **shell** містить основні налаштування мережі: ім'я комп'ютера і домену, а також маршрутизатор за замовчуванням.

Каталог ***/etc/sysconfig/network-scripts*** містить множину сценаріїв на всі випадки мережевого життя. У файлі ***README*** описано, для чого який сценарій потрібний і що означають поля в кожному з них (часто цей файл зберігається в ***/usr/share/doc/net-scripts***, а не в ***/etc***).

Наприклад, налаштування інтерфейсу **eth0** міститься у файлі ***ifcfg-eth0***

```
DEVICE=eth0
BOOTPROTO=static
IPADDR=192.168.0.125
NETMASK=255.255.255.0
```

```
NETWORK=192.168.0.1
BROADCAST=192.168.0.255
ONBOOT=yes
```

Мережеві утиліти

1. ifconfig

Команда ***ifconfig*** використовується для конфігурації мережевих інтерфейсів ядра. Вона використовується на етапі завантаження для налаштування інтерфейсів за необхідності. Після цього вона, зазвичай, використовується тільки у разі відлагодження або налаштування швидкодії системи.

Якщо аргументи не передані, ***ifconfig*** видає інформацію про стан активних інтерфейсів. Якщо вказаний один аргумент (інтерфейс), видається інформація тільки про стан цього інтерфейсу; якщо вказаний один аргумент -a, видається інформація про стан всіх інтерфейсів, навіть відключених. Інакше команда конфігурує вказаний інтерфейс.

У виведенні команди за кожним інтерфейсом можна проглянути такі дані:

inet addr:ip_address – адреса;

Bcast:broadcast_address – широкомовні адреси;

Mask: network_mask – маска підмережі.

Інформація щодо трафіку зберігається в таких рядках:

RX packets – число отриманих пакетів;

TX packets – число відправлених пакетів;

errors – число помилок при прийомі \ передачі пакетів;

dropped – число відкинутих при прийомі \ передачі пакетів;

collisions – число колізій в мережі;

RX <bytes> – число отриманих байтів;

TX <bytes> – число відправлених байтів.

2. netstat

Команда ***netstat*** показує вміст різних структур даних, пов’язаних з мережею, в різних форматах залежно від вказаних опцій.

Опції:

-a – показувати стан всіх сокетів; зазвичай сокети, які використовуються серверними процесами, не показуються;

-A – показувати адреси будь-яких управляючих блоків протоколу, пов’язаних з сокетами; використовується для відлагодження;

-i – показувати стан автоматично конфігурованих (auto-configured) інтерфейсів. Інтерфейси, статично конфігуровані в системі, але не знайдені під час завантаження, не показуються;

-n – показувати мережеві адреси як числа; ***netstat*** звичайно показує адреси як символи. Цю опцію можна використовувати з будь-яким форматом показу;

-r – показати таблиці маршрутизації. У разі використання з опцією ***-s*** показує статистику маршрутизації;

-s – показати статистичну інформацію за протоколами. Під час використання з опцією ***-r*** показує статистику маршрутизації;

-f <сімейство_адрес> – обмежити показ статистики або адрес управляючих блоків тільки вказаним сімейством_адрес, у ролі якого можна указувати inet для сімейства адрес **AF_INET** або **unix** для сімейства адрес **AF_UNIX**;

-I <інтерфейс> – виділити інформацію про вказаний інтерфейс в окремий стовбець; за замовчуванням (для третьої форми команди) використовується інтерфейс з найбільшим об'ємом переданої інформації з моменту останнього перезавантаження системи. Як інтерфейс можна указувати будь-який з інтерфейсів, перерахованих у файлі конфігурації системи, наприклад **emd1** або **lo0**;

-p <ім'я_протоколу> – обмежити показ статистики або адрес управляючих блоків тільки протоколом з вказаним іменем_протоколу, наприклад, **tcp**.

Програма ***netstat*** дозволяє ознайомитись і з таблицею маршрутизації:

$ netstat -nr

Можливості ***netstat*** не обмежуються локальною мережею або автономною системою, за допомогою її можна одержати деяку інформацію про віддалені ЕОМ або маршрутизатори. Наприклад:

$ netstat -r 194.85.112.34

3. ping

Для перевірки того, чи з'єднується ваш комп'ютер з мережею, використовується утиліта ***ping***

ping <опції> <адреса>

Можна скористатися IP-адресою:

$ ping 192.168.0.2

або (тут ми одночасно перевіряємо і роботу служби DNS)

$ ping pc1

Утиліта відправляє на задану машину запити за протоколом **icmp** і чекає відповідей.

У виведенні команди відображається така інформація:

packets transmitted – число відправлених пакетів

received – число прийнятих пакетів

% packet loss – відсоток загублених пакетів

rtt min/avg/max/mdev – мінімальний / середній / максимальний час відповіді по мережі.

4. telnet – підключення до віддаленої системи з використанням протоколу **TELNET**. Після підключення до віддаленої системи запрошення командного рядка заміниться на запрошення команди ***telnet***, і ми можемо вводити команди, що управляють з'єднанням. ***telnet*** також підтримує режим безпосереднього введення, в якому символи, що вводяться нами, передаються на віддалену систему.

Формат виклику команди:

telnet <ім'я_системи> <порт>

Аргументи:

<ім'я_системи> – мережеве ім'я або адреса віддаленої системи

<порт> – номер порту, використовуваного для підключення до віддаленої системи (цей параметр може бути опущений)

5. wall – відправка повідомлення всім користувачам, підключеним до системи. Після введення команди потрібно набрати текст з клавіатури і натиснути ***<Ctrl-D>***. Ця команда звичайно використовується системним адміністратором для того, щоб попередити користувачів про заплановане перезавантаження системи.

Приклад:

$ wall

WARNING: After 5 minutes, the system will be restarted

<Ctrl-D>

Ця команда відправить введене повідомлення всім користувачам, підключеним до системи в момент виконання команди.

6. write – відправка повідомлення користувачу, підключеному до системи. Для завершення вводу повідомлення потрібно натиснути ***<Ctrl-D>***.

Приклад:

$ write student Hello! <Ctrl-D>

Ця команда відправить повідомлення ***"Hello!"*** користувачу ***student***.

7. ssh

Використовує протокол SSH. Служить для взаємодії між віддаленими комп'ютерами. За допомогою цього протоколу можливо зайти на віддалений комп'ютер або виконати на ньому команди. У роботі існує клієнт і сервер. Клієнти під'єднуються до сервера і у разі успішної авторизації дістають доступ до ресурсів віддаленого комп'ютера.

Приклад заходу на віддалений комп'ютер:

$ ssh <user>@<remote>, де:

<user> – ім'я користувача на віддаленому комп'ютері;

<remote> – ip-адреса або доменне ім'я віддаленого комп'ютера.

Після успішної авторизації робота на віддаленій машині відбувається аналогічно роботі на локальній.

Можливо виконати команду на віддаленому комп'ютері з виведенням результату на локальній:

$ ssh <user>@<remote> <command>

Можлива авторизація на віддаленій машині не тільки за паролем, але і на основі ключів. Для цього на клієнті генерується публічний і приватний ключ за допомогою утиліти ***ssh-keygen***. Після цього в директорії ***~/.ssh*** з'являються файли ***id_rsa.pub*** – публічний ключ і ***id_rsa*** – приватний ключ. Назви файлів можуть мінятися залежно від алгоритму шифрування.

Публічний копіюється на SSH-сервер у файл ***~/.ssh/authorized_keys***.

Після цього по команді ***ssh <user>@<remote***> відбувається логін на віддалену машину без запиту пароля.

8. ftp – проста команда, що дозволяє пересилати файли між машинами протоколом TCP/IP:

$ ftp hostname
User name: debianuser
Passwd:

Віддалена машина запитує у вас логін і пароль користувача, під якими він зареєстрований на віддаленій машині. Після цього ***ftp*** переходить в командний режим. У цьому режимі можна «переміщуватись» по каталогам віддаленої машини, по каталогам своєї машини, проглядати їх зміст і забирати файли звідти до себе або класти їх від себе туди.

ftp> quit – завершити роботу;

ftp> bin – встановити режим пересилання бінарних файлів;

ftp> cd dir1 – перейти в каталог ***dir1*** на віддаленій системі;

ftp> ls – вивести вміст каталогу у віддаленій системі;

ftp> !cd dir2 – перейти в каталог ***dir2*** у локальній системі;

ftp> !ls -al – вивести вміст каталогу у локальній системі;

ftp> get fileRemoteName [fileLocalName] – взяти файл з віддаленої системи;

ftp> put fileLocalName [fileRemoteName] – покласти файл у віддалену систему.

Завдання

1. За допомогою команди ***ifconfig*** з'ясувати, які мережеві інтерфейси в наявності. З'ясувати IP-адресу, широкомовну адресу, маску мережі.
2. За допомогою команди ***ping*** визначити:
   - доступність іншої локальної машини у мережі (за наявності);
   - доступність віддаленого сервера;
   - втрати пакетів;
   - середній час відповіді по мережі.
3. Зайти по **ssh** на іншу машину з ОС Linux і підрахувати кількість процесів, запущених на комп'ютері.
4. Написати сценарій, який кожну хвилину записує у файл поточний час і кількість байтів, прийнятих / відправлених за останню хвилину.

Контрольні питання

1. У яких файлах зберігається основна мережева інформація?
2. Які мережеві утиліти ви знаєте? Для чого призначена кожна з них?
3. Які команди Linux призначені для відправлення повідомлень? Чи вони відрізняються одна від одної?
4. Який синтаксис має команда ***ssh***?
5. Як працювати на віддаленій машині за допомогою команди ***ftp***?

Лабораторна робота № 15. Робота з потоковим редактором sed

Теоретичні відомості

Потоковий редактор **Stream EDitor** використовується для виконання базових перетворень у тексті, який зчитується з файлу або з конвеєра. Результат відправляється на стандартне виведення. Синтаксис команди ***sed*** не дозволяє вказати вихідний файл, але результат можна зберегти в файлі за допомогою перенаправлення виведення. Редактор не дозволяє змінити вихідний потік.

Редактор ***sed*** відрізняється від інших редакторів тим, що з його допомогою можна фільтрувати текст, що подається з конвеєра. Вам не потрібно управляти редактором під час його роботи, тому редактор ***sed*** іноді називається редактором потокової обробки. Ця особливість дозволяє вказувати команди редагування всередині скриптів, що значно полегшує виконання завдань, що повторюються. Коли у великій кількості файлів потрібно виконати текстову заміну, ***sed*** стає гарною підмогою.

Команди sed

За допомогою програми ***sed*** можна виконувати текстові заміни і видалення за зразком, а також можна використовувати точно такі ж регулярні вирази, що і з командою ***grep***.

Команди редагування аналогічні тим, що використовуються в редакторі ***vi***.

Таблиця 1.

Команди редагування *sed*

Команда	*Результат*
a	Додавання тексту під поточним рядком
c	Заміна тексту в поточному рядку новим текстом
d	Видалення тексту
i	Вставка тексту над поточним рядком
p	Видача тексту
r	Читання файлу
s	Пошук і заміна тексту
w	Запис у файл

Крім команд редагування, можна в ***sed*** вказувати додаткові параметри. Їх огляд наведено в таблиці нижче.

Таблиця 2.

Параметри редактора *sed*

Параметр	*Дія*
-e SCRIPT	До набору команд, які повинні бути запущені під час обробки вхідного потоку, додаються команди, які вказуються в параметрі ***SCRIPT***
-f	До набору команд, які повинні бути запущені у процесі обробки вхідного потоку, додаються команди, що знаходяться в файлі ***SCRIPT-FILE***
-n	Режим мовчання (у стандартний потік виведення нічого не видається)
-V	Видається інформація про версії і відбувається вихід з програми

У документації по ***sed*** міститься більше інформації, тут ми перерахували тільки найбільш часто використовувані команди і параметри.

Інтерактивне редагування
Видача рядків, що містять шаблон

Подібне, звичайно, можна зробити за допомогою команди ***grep***, але з її допомогою ми не зможемо виконати дію «Знайти і замінити». З цього ми тільки почнемо.

Нижче наведено наш текстовий файл ***example***:

```
$ cat -n example
1 This is the first line of an example text.
2 It is a text with erors.
3 Lots of erors.
4 So much erors, all these erors are making me sick.
5 This is a line not containing any errors.
6 This is the last line.
```

Ми хочемо за допомогою редактора ***sed*** знайти всі рядки, що містять наш шаблон пошуку, в цьому випадку «***erors***». Щоб знайти результат, ми використовуємо ***p***:

```
$ sed '/erors/p' example
This is the first line of an example text.
It is a text with erors.
It is a text with erors.
Lots of erors.
Lots of erors.
```

```
So much erors, all these erors are making me sick.
So much erors, all these erors are making me sick.
This is a line not containing any errors.
This is the last line.
```

Як можна помітити, ***sed*** видає весь файл, а рядки, що містять шуканий рядок, виводяться двічі. Це не те, що нам потрібно. Щоб виводити тільки ті рядки, які відповідають нашому шаблону, потрібно використати параметр ***-n***:

```
$ sed -n '/erors/p' example
It is a text with erors.
Lots of erors.
So much erors, all these erors are making me sick.
```

Видалення рядків, що містять шаблон

Ми використовуємо той же текстовий файл ***example***. Тепер ми хочемо бачити лише рядки, в яких немає шуканого рядка:

```
$ sed '/erors/d' example
This is the first line of an example text.
This is a line not containing any errors.
This is the last line.
```

Команда ***d*** вказує, що знайдені рядки не показуються.

Рядки, що починаються заданим шаблоном і закінчуються другим шаблоном, показуються в такий спосіб:

```
$ sed -n '/^This.*errors.$/p' example
This is a line not containing any errors.
```

Відзначимо, що насправді для останньої крапки потрібно вказати, що це не спецсимвол, і не шукати з нею збіги. У нашому прикладі відповідність шукається для будь-якого останнього символу, в тому числі і крапки.

Діапазони рядків

На цей раз ми хочемо отримати рядки, в яких є помилки. У прикладі такими є рядки з ***2*** по ***4.*** Позначимо цей діапазон адрес і команду ***d***:

```
$ sed -n '2,4d' example
This is the first line of an example text.
This is a line not containing any errors.
This is the last line.
```

Щоб видати тільки початкові рядки з файлу і виключити рядки, починаючи з певного рядка і до кінця файлу, потрібно використати команду, подібну до поданої:

```
$ sed '3,$d' example
This is the first line of an example text.
It is a text with erors.
```

У результаті будуть виведені тільки перші два рядки файлу ***example***.

Наступна команда видає рядки, починаючи з першого, в якій виявлений шаблон «***a text***», і до того рядка, в якій виявлений шаблон «***a line***»:

```
$ sed -n '/a text/,/This/p' example
It is a text with erors.
Lots of erors.
So much erors, all these erors are making me sick.
This is a line not containing any errors.
```

Пошук і заміна за допомогою sed

Тепер ми будемо у файлі ***example*** шукати помилки і замінювати їх, а не тільки шукати рядки, що містять (або не містять) шуканий рядок.

```
$ sed 's/erors/errors/' example
This is the first line of an example text.
It is a text with errors.
Lots of errors.
So much errors, all these erors are making me sick.
This is a line not containing any errors.
This is the last line.
```

Як бачимо, це не зовсім те, що нам потрібно: в рядку 4 заміна була виконана тільки для першого входження, і там ще залишилася рядок «***eror***». Потрібно використати команду ***g***, щоб вказати ***sed***, що він повинен перевіряти весь рядок, а не зупинятися після того, як виявить перше входження відповідного рядка:

```
$ sed 's/erors/errors/g' example
This is the first line of an example text.
It is a text with errors.
Lots of errors.
So much errors, all these errors are making me
sick.
This is a line not containing any errors.
This is the last line.
```

Вставка рядка на початку кожного рядка:

```
$ sed 's/^/> /' example
> This is the first line of an example text.
> It is a text with erors.
> Lots of erors.
> So much erors, all these erors are making me sick.
> This is a line not containing any errors.
> This is the last line.
```

Вставка рядка у кінці кожного рядка:

```
$ sed 's/$/EOL/' example
This is the first line of an example text.EOL
It is a text with erors.EOL
Lots of erors.EOL
So much erors, all these erors are making me sick.EOL
This is a line not containing any errors.EOL
This is the last line.EOL
```

Можна задати відразу кілька команд пошуку і заміни, випереджаючи кожну параметром ***-e***:

```
$ sed -e 's/erors/errors/g' -e 's/last/final/g' example
This is the first line of an example text.
It is a text with errors.
Lots of errors.
So much errors, all these errors are making me sick.
This is a line not containing any errors.
This is the final line.
```

Треба мати на увазі, що за замовчуванням ***sed*** видає свої результати в стандартний потік виведення, тобто, як правило, у термінальне вікно. Якщо ми хочемо зберегти результат у файлі, то потрібно виконати перенаправлення.

Неінтерактивне редагування

Читання команд sed з файлу

Кілька команд ***sed*** можна об’єднати у файл і виконати файл з використанням параметра ***-f***. Коли створюється такий файл, переконайтеся в такому:

– у кінці рядків не повинно бути пробілів;

– не повинні використовуватися лапки;

– коли вводиться текст для додавання або заміни, перед усіма символами кінця рядків, крім останнього, повинен додаватися зворотний слеш.

Запис вихідних файлів

Запис результату, що виводиться, виконується за допомогою оператора перенаправлення **>**. Нижче наведено приклад скрипта, який використовується для створення дуже простого файлу HTML зі звичайного текстового файлу.

```
$ cat script.sed
<html>
<head><title>sed generated html</title></head>\
<body bgcolor="#ffffff">\
<pre>
$a\
</pre>\
</body>\
</html>
```

```
$ cat txt2html.sh
#!/bin/bash

echo "converting $1..."
SCRIPT="/home/sandy/scripts/script.sed"
NAME="$1"
TEMPFILE="/var/tmp/sed.$PID.tmp"
sed "s/\n/^M/" $1 | sed -f $SCRIPT | sed
"s/^M/\n/" > $TEMPFILE
mv $TEMPFILE $NAME
echo "done."
```

У змінній ***$1*** зберігається перший аргумент цієї команди, в такому випадку це ім’я файлу, використовуваного у перетворенні:

```
$ cat test
line1
line2
line3
```

Виклик сценарію:

```
$ txt2html.sh test
converting test...
done.
```

```
$ cat test
<html>
<head><title>sed generated html</title></head>
<body bgcolor="#ffffff">
<pre>
line1
line2
line3
</pre>
</body>
</html>
```

Потужні редактори, що підтримують підсвічування синтаксису, можуть розпізнавати синтаксис ***sed***. Це може бути великою підмогою для тих, хто забуває вказувати зворотний слеш та ін.

Завдання

1. Створіть список файлів, що є в директорії ***/usr/bin***, в яких як другий символ використовується буква «***a***». Помістіть результат в тимчасовий файл.
2. Видаліть перші три рядки кожного тимчасового файлу.
3. Видайте в стандартне виведення тільки ті рядки, в яких є шаблон «***an***».
4. Створіть файл з командами ***sed***, які виконують два попередні завдання. Додайте в цей файл ще одну команду, яка додає рядок, наприклад, ***"*** This might have something to do with man and man pages ***"*** перед кожним входженням рядка ***«man»***. Перевірте результати.
5. Створіть довгий список файлів, використавши як вхід кореневий каталог ***"/"***. Створіть файл з командами ***sed***, які перевіряють, чи є файл символічним посиланням або звичайним файлом. Якщо файл є символічним посиланням, поставте перед ним рядок ***"--This is a symlink--"***. Якщо файл є звичайним файлом, додайте в той самий рядок, наприклад, коментар ***"<--- this is a plain file"*** (.
6. Створіть скрипт, який показує у файлі рядки, що містять проміжки в кінці рядка. В цьому сценарії повинен використовуватися ***sed*** і видаватися зрозумілий користувачам результат.

Контрольні питання

1. Який формат має команда ***sed***?

2. Яке призначення команди «***s***» у редакторі ***sed***?
3. Як прочитати команди редактора ***sed*** з файлу?
4. Як замінити символ за допомогою потокового редактора ***sed***?
5. Для чого призначений ключ ***–e*** у команді ***sed***?
6. Як додати текст під поточним рядком за допомогою редактора ***sed***?
7. Як додати текст над поточним рядком за допомогою редактора ***sed***?
8. Як у редакторі ***sed*** вказати діапазон рядків?

Лабораторна робота № 16. Мова обробки вхідного потоку та пострококового розбору gawk у Linux

Теоретичні відомості

Утиліта ***gawk*** призначена для знаходження інформації в текстових файлах за заданими критеріями вибору. Вона забезпечена потужними засобами обробки тексту в великих текстових файлах. Утиліту ***gawk*** можна віднести до програмованих фільтрів, налаштованих на виконання конкретного завдання.

Gawk одночасно є утилітою, програмованим фільтром і середовищем програмування, за допомогою якої можна створювати інші фільтри. Тому вона посідає особливе місце в операційних системах UNIX і Linux і є потужним та гнучким інструментом; ***gawk*** є версією утиліти ***awk***, використовуваної в UNIX і розробленої одним із творців мови С Брайеном Керніганом та іншими.

Будь-яке середовище програмування має свою мову програмування, це стосується і до утиліти ***gawk***. Багато операторів взяті з мови програмування С і мають такий же синтаксис. Фільтри, що створюються за допомогою мови програмування ***gawk***, можуть вико-ристовуватися в середовищі будь-якого інтерпретатора UNIX і Linux, наприклад, в **bash**, **shell**, **tcsh**. Створені фільтри зчитують інформацію з файлу або стандартного потоку введення, аналізують, виділяють за певними заданими критеріями або змінюють цю інформацію, зберігають вихідні дані або направляються в стандартний потік виведення.

Таким чином, можна створювати не тільки фільтри, але і власні команди UNIX і Linux, що особливо актуально у великих системах розподілених ресурсів із нестандартною структурою організації. Утиліту ***gawk*** можна використовувати для створення звітів, пошуку заданих текстових фрагментів, виконання обчислень на основі введених даних, для роботи зі структурами, що нагадують бази даних, тобто що складаються з записів, що підрозділяються на поля, розділені спеціальними символами.

Утиліту ***gawk*** можна викликати безпосередньо з командного рядка або з shell-сценарію за допомогою ключового слова ***gawk***. Якщо скрипт включає ***gawk*** як складову, то його можна розглядати як новий фільтр, утиліту або команду.

Синтаксис команди ***gawk*** такий:

$ gawk '<Шаблон або умова {дія}>' <імена_файлів>

Командою ***gawk*** використовується такий шаблон пошуку, який застосовується у команді фільтрі ***grep*** та її похідних. Шаблон виділяється символами косої риски (знаками слеш).

Приклад 1.

$ gawk '/інтерфейс Gnome/ {print}' /home/info/*

Тут шукаються всі файли в каталозі ***/home/info***, в яких трапляється послідовність зі слів «***інтерфейс Gnome***», і результати направляються на стандартний пристрій виведення за допомогою дії ***{print}***. У результатах виводяться імена файлів і рядки, що містять шаблон.

Дія перенаправлення на стандартний пристрій виведення ***{print}*** зазвичай задається як дія за замовчуванням, тому команду у цьому прикладі можна переписати таким чином:

$ gawk '/інтерфейс Gnome/' /home/info/*

Результат при цьому не зміниться.

Приклад 2. Знайти за зазначеним шляхом файли вихідних текстів на мові С, що включають слова, що позначають ім'я змінної, наприклад, ***SHELLDATA*** або ***SHDATA***:

$ gawk '/SHELLDATA|SHDATA/' /home/include/*

Змінні і константи

Утиліта ***gawk*** дозволяє визначати **змінні і константи**. Існує три типи змінних: змінні для позначення полів, спеціальні змінні і змінні. Змінні визначає користувач, інші утиліта визначає автоматично. Існує два типи констант: арифметичні та рядкові. Арифметичні константи складаються з цифр, рядкові константи вписують у подвійні лапки і складаються з будь-яких символів, включаючи символи цифр. **Змінні визначаються після їх першого використання.**

Поля

Утиліта ***gawk*** може працювати зі змінними, що називаються полями. Поле являє будь-який набір символів, обмежений роздільниками полів (перше і останнє поля можуть мати по одному розділювачу). За замовчуванням як розділювач використовується проміжок або знак табуляції. Утиліта ***gawk*** нумерує поля рядків текстового

файлу, починаючи з 1, а також автоматично визначає змінну для кожного поля файлу. Ім'я змінної, що позначає поле, складається зі знака ***$*** і номера поля. Наприклад: ***$1***, ***$2***, ***$3*** та ін. Змінна ***$0*** – спеціальна змінна, що визначає весь рядок.

Приклад 3. Припустимо, що у нас є текстовий файл ***/home/tabl/list_students***, відсортований за алфавітом, кожен рядок якого включає: прізвище, ім'я, факультет, курс, рейтингову оцінку, і складається з таких рядків-записів:

```
Арінин Іван МП 1 4
Бетінов Євген ЕКТ 2 5
Кошин Леонід МП 1 4
Кошкін Володимир ЕКТ 1 5
Ліпін Федір МП 2 3
Пенов Микола МП 1 4
Яшин Петро ЕКТ 2 4
```

Потрібно вивести список, що складається з прізвищ, імен та рейтингових оцінок.

Тобто потрібно вивести **1**, **2** та **5** поля. Поля перераховуються через кому і проміжок, а команда виглядатиме таким чином:

```
$ gawk '{print $1, $2, $5}' /home/tabl/list_students
```

У результаті виконання на екран виводиться така таблиця, поля якої будуть вирівняні.

```
Арінин      Іван          4
Бетінов     Євген         5
Кошин       Леонід        4
Кошкін      Володимир     5
Ліпін       Федір         3
Пенов       Микола        4
Яшин        Петро         4
```

Приклад 4. Тепер роздрукуємо весь файл у вигляді таблиці так, щоб всі поля були вирівняні. Для цього використовуємо змінну ***$0***, інакше виведені дані не вирівняються і будуть погано читабельними:

```
$ gawk '{print $0}' /home/tabl/list_students
```

Результат виведення на екран буде таким:

```
Арінин     Іван        МП      1     4
Бетінов    Євген       ЕКТ     2     5
Кошин      Леонід      МП      1     4
```

```
Кошкін    Володимир    ЕКТ    1    5
Ліпін     Федір        МП     2    3
Пенов     Микола       МП     1    4
Яшин      Петро        ЕКТ    2    4
```

Приклад 5. Виберемо з вихідного файлу ***/home/tabl/list_students*** студентів з рейтинговою оцінкою 5 і роздрукуємо все значення полів для обраних записів:

```
$ gawk '/5/ {print $0}' /home/tabl/list_students
Бетінов   Євген        ЕКТ    2    5
Кошкін    Володимир    ЕКТ    1    5
```

У разі, якщо 5 трапляється в інших полях, наприклад, курс 5, то умови вибірки недостатні.

Оператори порівняння і логічні операції

Для формування логічних виразів, що виконують перевірку умов, використовуються **оператори порівняння**. В останньому прикладі для пошуку потрібно використовувати в шаблоні оператор порівняння, а саме – оператор перевірки на рівність (**==**):

```
$ gawk '/$5==5/ {print $0}' /home/tabl/list_students
```

Інший оператор порівняння називається оператор перевірки на нерівність (**!=**). І як приклад може бути використаний для пошуку записів з рейтинговою оцінкою, відмінною від 5:

```
$ gawk '/$5!=5/ {print $0}' /home/tabl/list_students
```

Утиліта ***gawk*** використовує оператори порівняння такі ж, як в мові С: **<**, **>**, **>=**, **<=**, але, на відміну від С, може порівнювати рядкові значення. Порівняння виконується відповідно до алфавіту. Утиліта ***gawk*** використовує логічні оператори:

– ***&&*** – логічне І (AND);
– **||** – логічне АБО (OR);
– ***!*** – логічне заперечення (NOT).

Вони дають можливість використовувати складні логічні умови, при цьому умови записуються в дужках.

Приклад. Вибрати зі списку студентів 1 курсу факультету ЕКТ:

```
$ gawk '($3 == "ЕКТ") && ($4 == 1) {print}' list_student
```

Функція length

В умові можна використовувати функцію ***length***, що визначає довжину поля. Наприклад, потрібно вибрати записи студентів факультету ЕКТ. Можна записати так:

```
$ gawk '/ЕКТ/ {print}' /home/tabl/list_students
```

Використовуючи функцію ***length*** і помічаючи, що третє поле (факультет) в наших даних складається з 2 або 3 символів, а потрібно вибрати значення, що складається з трьох символів, можна записати так:

```
$ gawk '/length ($3 == 3) / {print}' /home/tabl/list_students
```

Спеціальні змінні мови утиліти gawk

В утиліті ***gawk*** визначаються спеціальні змінні:

– ***NR*** – зберігає номер запису, що переглядається або оброблюється, з її допомогою зручно нумерувати рядки;

– ***NF*** – кількість полів у поточному записі;

– ***$0*** – всі поля поточного запису;

– ***$N*** – номер поля (див. Розділ «Поля»);

– ***FS*** – роздільник поля введення;

– ***FILENAME*** – ім'я файлу, в якому працюємо.

Припустимо, нам потрібно вивести дані вихідного файлу у вигляді таблиці і пронумерувати рядки:

```
$ gawk '{print NR, $0}' /home/tabl/list_students
1 Арінин    Іван       МП   1  4
2 Бетінов   Євген      ЕКТ  2  5
3 Кошин     Леонід     МП   1  4
4 Кошкін    Володимир  ЕКТ  1  5
5 Ліпін     Федір      МП   2  3
6 Пенов     Микола     МП   1  4
7 Яшин      Петро      ЕКТ  2  4
```

Використання слів BEGIN і END

Для опису дій до або після перегляду і обробки всіх записів використовуються ключові слова ***BEGIN*** і ***END*** відповідно.

Наприклад, якщо перед друком всіх пронумерованих записів ми хочемо поставити заголовок, то можна написати так (вважаємо, що ми знаходимося в ***/home/tab***):

```
$ gawk 'BEGIN {print "student's list"} {print NR, $0}' list_students
```

Спочатку буде виведений заголовок ***student's list***, а далі пронумерований за рядками текст таблиці даних.

Арифметичні оператори і функції

Gawk підтримує повний набір арифметичних операторів, вони такі ж, як в мові С: *******, ***/***, ***+***, ***–***, ***%***.

Арифметичні обчислення в ***gawk*** виконуються над числовими шаблонами. Числовий шаблон являє собою послідовність цифр. Це може бути арифметична константа, змінна або поле, які складаються з цифр. Сюди входять вбудовані числові змінні ***NR*** і ***NF***.

Приклад 1. Вивести всі парні записи текстового файлу.

```
$ gawk ' (NR % 2) {print NR, $0}' /home/tabl/list_students
```

Приклад 2. Порахувати сумарний бал оцінок студентів (знаходимося в ***/home/tabl***). Вивести на екран таблицю і сумарний підсумок.

```
$ gawk '{print; sum += $5} END {print "summa"=, sum}' list_students
```

Приклад 3. Вивести на екран заголовок таблиці, таблицю, кількість записів і середнє значення. Кількість записів визначається змінною ***NR*** після перегляду всіх записів. У процесі обробки ***NR*** збільшується на 1 під час звернення до чергового запису файлу. Нумерація починається з 1.

```
$ gawk 'BEGIN {print "exam results" }{print; sum += $5} END {print NR, "average"=, sum}' list_students
```

Приклад 4. Нехай є файл ***/home/plant***, що складається із записів з такими полями: ім'я підприємства, дохід по **1**, **2**, **3**, **4** кварталах. Знайти підсумковий дохід **d** для кожного підприємства і роздрукувати записи, пронумерувавши їх і вказавши підсумковий дохід останнім полем кожного запису.

```
$ gawk '{ d = $1 + $2 + $3 + $4; print NR, $0, d}' /home/plant
```

Пошук за шаблоном

Для виконання пошуку за шаблоном в полях поточного запису застосовуються спеціальні символи: ***~***, ***!~***. За допомогою операції ~ можна перевірити, чи присутній конкретний шаблон в певному полі (шаблоном тут є частина поля або все поле). За допомогою операції ***!~*** Можна перевірити, в яких записах відсутній конкретний шаблон в певному полі.

Приклад 1. Чи присутнє прізвище Леонов у списку файлу ***list_students***?

$ gawk '($1 ~ /Леонов/) {print}' /home/tabl/list_students

Приклад 2. Є запис студента Іванова в списку файлу ***list_students***? Причому ми не впевнені, з великої чи малої літери її могли написати.

$ gawk '($1 !~ /[Ii]ванов/) {print}' /home/tabl/list_students

Оформлення інструкцій gawk окремим файлом

Якщо інструкція ***gawk*** велика в написанні, то її можна оформити окремим файлом, наприклад, зазначений вище приклад:

$ gawk 'BEGIN {print "exam results" }{print; sum += $5} END {print NR, "average"=, sum}' list_students

можна записати так. Створюємо окремий файл, назвемо його, наприклад, ***f_instr***:

```
BEGIN
{print "exam results"}
{print; sum += $5}
 END {
print NR, "average"=, sum}
```

Далі виконуємо команду ***gawk*** з опцією ***-f***, яка дає можливість читати інструкції із зазначеного файлу, а не з командного рядка. Тоді загальна структура команди така:

$ gawk -f <ім'я_файлу_інструкції> <ім'я_оброблюваного_файлу>

Приклад.

$ gawk -f f_instr list_students

Іноді файли інструкцій для зручності записують з розширенням ***gawk***, тобто ім'я файлу тоді виглядає так: ***f_instr.gawk***, а команда так:

$ gawk -f f_instr.gawk list_students

Завдання

1. Вивести у файл список файлів і каталогів поточного каталогу (з опцією ***-1***). Скласти сценарій, який виводить на друк імена файлів або каталогів із зазначенням їх власників.

2. Створити сценарій **bash**, який використовує команду gawk і стандартні команди Linux, який буде показувати 3 користувачів, що використовують найбільший обсяг дискового простору файлової системи /home Спочатку виконайте команди в командному рядку. Потім помістіть їх у сценарій. Сценарій повинен формувати зрозумілий для читання результат.

Контрольні питання

1. Для чого призначена програма ***gawk***?
2. Який формат має команда ***gawk***?
3. Як вивести певні поля у тексті за допомогою ***gawk***?
4. Яким чином можна використати декілька команд у програмному сценарії ***gawk***?
5. Як за допомогою ***gawk*** прочитати програму з файлу?
6. Для чого у ***gawk*** призначені слова ***BEGIN*** та ***END***?

Лабораторна робота № 17. Розробка програм мовою C/C ++ в ОС Linux

Етапи створення програми мовою C/C++

На рис. 1 представлена послідовність розробки програми мовою С. В ОС Linux один з популярних наборів компіляторів – **GNU Compiler Collection**, або **GCC**. Він включає компілятори з мов **C**, **C++**, **Java**, **Fortran** і ін. Ім'я компілятора мови **C** – ***gcc***, мови **C++** – ***g++***. Для мови **C** також можна використовувати компілятор **cc**, наявний у багатьох ОС сімейства UNIX. Для мови **C** розширення вихідних файлів – ***.c*** і ***.h***, для мови ***C++*** – ***.cpp***, ***.hpp***, ***.cxx***, ***.hxx***, ***.C*** і ***.H***. Розширення об'єктних файлів – ***.o***, бібліотек об'єктних файлів – ***.a***. Виконувані файли в Linux зазвичай не мають розширення.

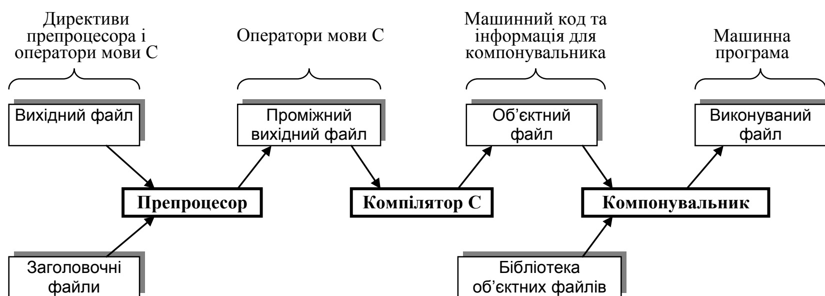


Рис. 1. – Послідовність розробки програми мовою С

Послідовність команд для створення програми

1. Програма складається з одного вихідного файлу

Дія	*Приклад*
1. У текстовому редакторі (**nano**, **vi**, **gedit**, **emacs** та ін.) створити і зберегти вихідний текстовий файл (наприклад, **myprog.c**)	***$ nano*** ……………… робота з nano
2. Відкомпілювати і скомпонувати програму. Результат – виконуваний файл (наприклад, **myprog**)	***$ gcc -o myprog myprog.c***
3. Виконати програму	***$ myprog*** або ***$ myprog param1 param2***

2. Програма складається з декількох вихідних файлів

Дія	***Приклад***
1. У текстовому редакторі створити і зберегти кожен з вихідних файлів	***$ nano*** ……………… робота з nano
2. Відкомпілювати окремо кожен текстовий файл. Результат – об'єктні файли (**module1.o**, **module2.o**, **module3.o**)	***$ gcc -c module1.c*** ***$ gcc -c module2.c*** ***$ gcc -c module3.c***
3. Скомпонувати об'єктні файли. Результат – здійсненний файл	***$ gcc -o myprog module1.o module2.o module3.o***
4. Виконати програму	***$myprog*** або ***$myprog param1 param2***

Компілятор за замовчуванням шукає заголовні файли в поточному каталозі і в каталогах, в яких встановлені заголовки для стандартних бібліотек. Якщо заголовки знаходяться в будь-якому іншому місці, то слід використовувати команду для виклику компілятора ***gcc***/***g++*** з опцією ***-I***. Нехай, наприклад, потрібно включити в програму файл ***obr.h*** з каталогу ***/home/ivanov/incl*** і нехай каталог ***/home/ivanov/texts*** є поточним. Тоді слід дати команду:

```
$ gcc -c -I ../incl obr.c
```

Або можна вказати абсолютне ім'я для каталогу ***incl***:

```
$ gcc -c -I /home/ivanov/incl obr.c
```

При цьому в програмі директива ***include*** повинна мати вигляд:

```
#include "obr.h"
```

Стандартна бібліотека **C** (що містить, зокрема, функцію ***printf*** та ін.) компонується в виконуваний файл автоматично. Для включення нестандартної бібліотеки (наприклад, бібліотеки ***libpthreads.a***, яка містить функції стандарту POSIX для роботи з потоками) слід скомпонувати програму з опцією ***-l***, наприклад:

```
$ gcc -o obr main.o obr.o –l pthreads
```

За цією командою в програму буде включена бібліотека ***libpthreads.a***, при цьому автоматично до імені бібліотеки додасться префікс ***lib*** і суфікс ***a***. Компонувальник буде шукати бібліотеки в ряді стандартних каталогів, включаючи каталоги ***/lib*** і ***/usr/lib***. Якщо ж бібліотека, яку необхідно включити, знаходиться

в якому-небудь іншому каталозі, то в командному рядку слід використовувати опцію ***-L*** спільно з опцією ***-l***, наприклад:

$ gcc -o obr main.o obr.o -L /usr/local/libs -l pthreads

За цією командою бібліотека ***libpthreads.a*** буде включена з каталогу ***/usr/local/libs***. Якщо цей каталог є поточним, то команду можна задати в такій формі:

$ gcc -o obr main.o obr.o -L . -l pthreads

Використання утиліти Make

Утиліта ***make*** використовується для автоматизації розробки програм. Для цього в текстовому файлі з ім'ям ***makefile*** вказується така інформація:

– цільові файли (цілі), які необхідно побудувати;
– правила для їх побудови;
– залежності, що визначають, коли цю мету необхідно перебудувати заново.

Так **makefile** має такий вигляд:

```
CFLAGS=-c
CC=gcc
obr: main.o obr.o
 <Tab> $(CC) -o obr main.o obr.o
main.o: main.c
<Tab> $(CC) $(CFLAGS) main.c
obr.o: obr.c
<Tab> $(CC) $(CFLAGS) obr.c
```

Тут зліва від «***:***» вказана мета, справа – її залежності. Правило для побудови цілі вказано на наступному рядку, яка повинна починатися з символу табуляції. ***CC*** і ***CFLAGS*** – змінні утиліти ***make*** (в цьому прикладі ***CC*** задає ім'я компілятора, ***CFLAGS*** – опцію компілятора). Значення для змінної може бути задано у файлі ***makefile*** (як в прикладі) і / або в командному рядку (наприклад, ***$ make CC=g++***).

Після підготовки файлу ***makefile*** для створення програми, що виконується, достатньо ввести команду:

$ make

або

$ make імя_make_файлу

(якщо ***імя_make_файлу*** відмінне від ***makefile***)

За цією командою утиліта ***make*** виконає файл ***makefile*** з поточного каталогу та автоматично перетранслює тільки ті файли, які необхідно.

Доступ до параметрів командного рядка

При запуску програми в командному рядку після імені програми можуть зазначатися параметри, що називаються параметрами (аргументами) командного рядка. Вони поділяються проміжками або, якщо параметри самі містять проміжки, поміщуються в апострофи. Для доступу до цих параметрів у програмі мовою **C** заголовок функції ***main*** повинен містити два параметра: кількість параметрів (***argc***) і масив вказівників на рядки (***argv***), що закінчується покажчиком NULL. І-ий рядок є значенням і-го параметра, при цьому рядок з номером 0 містить ім'я програми.

Приклад 1. Друк параметрів командного рядка

```
#include <stdio.h>
int main (int argc, char *argv[])
{
 if ( argc < 2 )
 {
printf( "Usage : %s parameter\n", argv[0] ) ;
   return 1;
 }
  printf("Starting program %s \n", argv[0]) ;
  printf("with %d parameter(s)\n", argc-1) ;
  printf("First parameter is %s\n", argv[1]) ;
  return 0;
}
```

Змінні оточення

Будь-яка програма, що виконується, має своє оточення, що представляє собою набір строкових пар виду ***змінна=значення***. Відповідно до угоди, імена змінних пишуться великими літерами.

У програмі можна отримати значення змінної оточення за допомогою функції ***getenv*** з ***<stdlib.h>***. Ця функція приймає ім'я змінної і повертає її значення у вигляді рядка символів або ***NULL***, якщо змінна не визначена в оточенні. Для установки і очищення змінних оточення використовуються функції ***setenv*** і ***unsetenv*** відповідно. Для доступу до всіх змінних оточення використовується спеціальна глобальна змінна ***environ***. Ця змінна має тип ***char***** і є масивом покажчиків на символьні рядки, що закінчується вказівником ***NULL***.

Приклад 2. Друк змінних середовища оточення

```
#include <stdio.h>
extern char** environ;
int main ()
{
```

```
    char** var;
    for (var = environ; *var != NULL; ++var)
      printf ("%s\n", *var);
    return 0;
  }
```

Обробка помилок в системних викликах

Більшість системних викликів повертає нуль, якщо операція виконана успішно, і ненульове значення в іншому випадку. У разі помилок в глобальну змінну ***errno*** записується додаткова інформація – ціле число, яке ідентифікує причину помилки. Якщо включити в програму ***<errno.h>***, можна посилатися на помилки за їх символічними іменами, наприклад, ***EACCES*** або ***EINVAL***.

Отримати в програмі текстове повідомлення про помилку можна двома способами:

1. За допомогою функції ***strerror*** із ***<string.h>***; функція приймає ***errno*** і повертає рядок з описом помилки.
2. За допомогою функції ***perror*** з ***<stdio.h>***; функція приймає рядок, яку вона виводить як префікс перед повідомленням про помилку (наприклад, це може бути ім'я файлу, де сталася помилка) і виводить опис помилки в потік ***stderr***.

Приклад 3. У програмі робиться спроба відкрити файл. Системний виклик ***open*** повертає дескриптор файлу або ***-1*** у випадку помилки.

Варіант 1

```
fd = open ("inputfile.txt", O_RDONLY);
if (fd == -1)
{
 fprintf (stderr, "error opening file: %s\n",
strerror (errno));
 exit (1);
}
```

Варіант 2

```
fd = open ("inputfile.txt", O_RDONLY);
if (fd == -1)
{
 perror("inputfile.txt");
 exit (1);
}
```

Зверніть увагу, що успішне виконання системного виклику не приводить до скидання змінної ***errno***.

Завдання

1. Знайдіть у довіднику ***man*** призначення основних опцій компілятора ***gcc***: ***-c***, ***-S***, ***-E*** та ***-o***.
2. Створіть у вашому каталозі каталог ***lab2*** для створення у ньому програм мовою **С** і зробіть його поточним для цієї лабораторної роботи.
3. Напишіть мовою **С/С++** і виконайте програму ***Hello***, яка виводить рядок ***"Hello, world"***.
4. Напишіть мовою **С/С++** і виконайте програму ***obr***, що складається з двох модулів:
   - модуль ***obr.c*** містить функцію ***double obr(int i)***, яка повертає число, зворотне числу ***i***;
   - модуль ***main.c*** містить функцію ***main()***, яка запитує у користувача ціле число ***i*** та виводить значення ***obr(i)***.
5. Створіть файл ***makefile*** для програми із завдання 4, попередньо видаливши файли ****.o*** і ****.obr*** з поточного каталогу. Потім виконайте команду ***make***.
6. Виконайте **Приклад 1** з цієї лабораторної роботи та змініть його таким чином, щоб у ролі параметрів командного рядка були цілі числа. Для перетворення рядка в ціле число використовуйте функцію ***atoi***.
7. Виведіть всі змінні середовища оточення програмним шляхом мовою **С/С**++.

Контрольні питання

1. Як отримати проміжний (після препроцесора) вихідний текст програми?
2. Як створити програму з декількох модулів?
3. Як додати в програму бібліотеку об’єктних модулів?
4. Як використовувати утиліту ***make*** для створення програм?
5. Як у програмі отримати доступ до параметрів командного рядка і змінним оточення?
6. За допомогою яких функцій можна вивести повідомлення про помилку в системному виклику?

Лабораторна робота № 18.
Робота з файлами та каталогами Linux мовою С

Теоретичні відомості

Для виконання операцій запису і читання даних в існуючому файлі програмним шляхом мовою програмування **С/С++** його слід відкрити за допомогою виклику функції ***open()***.

```
int open(const char *pathname, int flags, [mode_t mode]);
int fopen(const char *pathname, int flags, [mode_t mode]);
```

Другий аргумент ***flags*** у системному виклику ***open()*** має цілочисельний тип і визначає метод доступу. Параметр ***flags*** приймає одне із значень, заданих постійними в заголовки ***fcntl.h***. У файлі визначені три константи:

- ***O_RDONLY*** – відкрити файл лише для читання;
- ***O_WRONLY*** – відкрити файл лише для запису;
- ***O_RDWR*** – відкрити файл для читання і запису;

або ***"r"***, ***"w"***, ***"rw"*** для ***fopen()***.

Третій параметр ***mode*** встановлює права доступу до файлу і є необов’язковим, він використовується тільки разом з прапором ***O_CREAT***. Приклад створення нового файлу:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int Fd1;
FILE *F1;
F1=fopen("Myfile2.txt", "w", 644);
Fd1=open("Myfile1.txt", O_CREAT, 644);
```

Системні виклики ***stat*** і ***fstat*** дозволяють процесу визначити значення властивостей в існуючому файлі.

```
#include <sys/types.h>
#include <sys/stat.h>
int stat(const char *pathname, struct stat *buf);
int fstat(int filedes, struct stat *buf);
```

Приклад: ***stat("1.exe", &st1);***

Аргументи функції ***stat***:

– ***pathname*** – повне ім’я файлу;

– ***buf*** – структура типу ***stat***, яка після успішного виклику буде містити пов’язану з файлом інформацію.

У свою чергу структура ***stat*** включає такі елементи:

```
struct stat {
dev_t st_dev; /* логічний пристрій, де знаходиться файл */
ino_t st_ino; /* номер індексного дескриптора */
mode_t st_mode; /* права доступу до файлу */
nlink_t st_nlink;/* кількість жорстких посилань на файл */
uid_t st_uid; /* ID користувача-власника */
gid_t st_gid; /* ID групи- власника */
dev_t st_rdev; /* тип пристрою */
off_t st_size; /* загальний розмір в байтах */
unsigned long st_blksize; /* розмір блоку введення-виведення */
unsigned long st_blocks; /* кількість блоків, що займає файл */
time_t st_atime;/* час останнього доступу */
time_t st_mtime;/* час останньої модифікації */
time_t st_ctime;/* час останньої зміни */
};
```

Призначення прав доступу в Linux

Права доступу до файлів представлені у вигляді послідовності бітів, де кожен біт означає дозвіл на **запис (w)**, **читання (r)** або **виконання (x)**. Права доступу записуються для власника (створювача) файлу (**owner**); групи, до якої належить власник файлу (**group**); і всіх інших (**other**) (див. Лабораторну роботу № 5). Для перегляду прав доступу можна використати функцію ***stat***.

Для запису прав доступу служить функція ***chmod***:

```
#include <sys/types.h>
#include <sys/stat.h>
int chmod(const char *pathname, mode_t mode);
```

Приклад: ***chmod("1.exe", 0777);***

Робота з каталогами

Каталоги в **Linux** – це особливі файли. Для відкриття або закриття каталогів існують виклики:

```
#include <dirent.h>
```

```
DIR *opendir(const char *dirname);
int closedir(DIR *dirptr);
```

Для роботи з каталогами існують системні виклики:

– ***int mkdir(const char *pathname, mode_t mode)*** – створення нового каталогу;

– ***int rmdir(const char *pathname)*** – видалення каталогу.

Перший параметр функції ***mkdir*** – ім'я створюваного каталогу, другий – права доступу.

Приклади:

```
retval=mkdir("/home/gleb/test",0777);
retval=rmdir("/home/gleb/test");
```

Зауважимо, що виклик ***rmdir("/home/gleb/test")*** буде успішним, тільки якщо каталог, що видаляється, є порожнім, тобто містить записи «крапка» (***.***) і «подвійна крапка» (***..***). Для читання записів каталогу існує виклик:

Для читання записів каталогу існує виклик:

```
struct dirent *readdir(DIR *dirptr);
```

Структура ***dirent*** є такою:

```
struct dirent {
  long d_ino;
  off_t d_off;
  unsigned short d_reclen;
  char d_name [1];
};
```

Поле ***d_ino*** – це число, яке унікальне для кожного файлу у файловій системі. Значенням поля ***d_off*** служить зсув цього елемента в реальному каталозі. Поле ***d_name*** є початком масиву символів, що задає ім'я елементу каталогу. Таке ім'я обмежене нульовим байтом і може містити не більше ***MAXNAMLEN*** символів. Тим самим описувана структура має змінну довжину, що зберігається в поле ***d_reclen***.

Приклад виклику:

```
DIR *dp;
struct dirent *d;
d=readdir(dp);
```

Під час першого виклику функція ***readdir*** в структуру ***dirent*** буде зчитаний перший запис каталогу. Після прочитання всього каталогу в результаті зазначених викликів ***readdir*** буде повернуто

значення ***NULL***. Для повернення покажчика в початок каталогу на перший запис існує виклик:

```
void rewindir(DIR *dirptr);
```

Щоб отримати ім'я поточного робочого каталогу, існує функція:

```
char *getcwd(char *name, size_t size);
```

Завдання

1. Написати програму введення символів з клавіатури і запису їх у файл (як аргумент під час запуску програми вводиться ім'я файлу). Для читання або запису файлу використовувати тільки функції посимвольного введення-виведення (***getc()***, ***putc()***, ***fgetc()***, ***fputc()***. Передбачити вихід після введення певного символу та контроль помилок відкриття / закриття / читання файлу.
2. Написати програму виведення вмісту текстового файлу на екран (як аргумент під час запуску програми передається ім'я файлу, другий аргумент (***N***) встановлює виведення по групах рядків (по ***N*** рядків) або суцільним текстом (***N = 0***)). Для виведення чергової групи рядків необхідно очікувати натискання користувачем будь-якої клавіші. Для читання або запису файлу використовувати тільки функції посимвольного введення-виведення. Передбачити контроль помилок відкриття / закриття / читання / запису файлу.
3. Написати програму копіювання одного файлу в інший. Як параметри під час виклику програми передаються імена першого і другого файлів. Для читання або запису файлу використовувати тільки функції посимвольного введення-виведення. Передбачити копіювання прав доступу до файлу і контроль помилок відкриття / закриття / читання / запису файлу.
4. Написати програму виведення на екран вмісту поточного та кореневого каталогів. Передбачити контроль помилок відкриття / закриття / читання каталогу.

Контрольні питання

1. Яка інформація визначена у структурі ***stat***?
2. Які системні виклики використовуються для зміни доступу до файлів?
3. Як програмним чином отримати поточний робочий каталог?
4. Як програмним чином змінити поточний робочий каталог?
5. Як програмним чином створити та видалити каталоги?
6. Як програмним чином прочитати вміст каталогу?

Лабораторна робота № 19. Робота з процесами Linux програмним шляхом мовою С. Системний виклик fork()

Теоретичні відомості

В ОС Linux для створення процесів використовується системний виклик ***fork()***:

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

У результаті успішного виклику ***fork()*** ядро створює новий процес, який є майже точною копією процесу, який його викликає. Іншими словами, новий процес виконує копію тієї ж програми, що і процес, який створив його, при цьому всі його об'єкти даних мають ті ж самі значення, що і в процесі, що викликає. Створений процес називається **дочірнім процесом**, а процес, який здійснив виклик ***fork()***, називається **батьківським**. Після виклику батьківський процес і його новостворений нащадок виконуються одночасно, при цьому обидва процеси продовжують виконання з оператора, який слід відразу ж за викликом ***fork()***. Процеси виконуються в різних адресних просто-рах, тому прямий доступ до змінних одного процесу з іншого процесу неможливий.

Приклад. Наступна коротка програма більш наочно показує роботу виклику ***fork()*** і використання процесу:

```
#include <stdio.h>
#include <unistd.h>
int main ()
{
  pid_t pid;       /*ідентифікатор процесу*/
  printf ("Поки всього один процес\n");
  pid = fork();     /*створення нового процесу*/
  printf ("Вже два процеси\n");
  if (pid == 0)
  {
   printf("Це дочірній процес, його pid=%d\n",
getpid());
   printf("А pid його батьківського процесу=%d\n",
getppid());
```

```
    }
    else if (pid > 0)
    printf ("Це батьківський процес, його pid=%d\n", getpid());
    else
     printf ("Помилка виклику fork, нащадок не створений\n");
 }
```

Для коректного завершення дочірнього процесу в батьківському процесі необхідно використовувати функцію ***wait()*** або ***waitpid()***:

```
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

Функція ***wait()*** призупиняє виконання батьківського процесу до тих пір, поки дочірній процес не припинить виконання або до появи сигналу, який або завершує поточний процес, або вимагає викликати функцію-обробник. Якщо дочірній процес до моменту виклику функції вже завершився (так званий «зомбі»), то функція негайно повертається. Системні ресурси, пов'язані з дочірнім процесом, звільняються.

Функція ***waitpid()*** призупиняє виконання батьківського процесу до тих пір, поки дочірній процес, зазначений в параметрі ***pid***, що не завершить виконання, або поки не з'явиться сигнал, який або завершує батьківський процес, або вимагає викликати функцію-обробник. Якщо вказаний дочірній процес до моменту виклику функції вже завершився (так званий «зомбі»), то функція негайно повертається. Системні ресурси, пов'язані з дочірнім процесом, звільняються. Параметр ***pid*** може приймати кілька значень:

– **pid<-1** означає, що потрібно чекати будь-якого дочірнього процесу, чий ідентифікатор групи процесів дорівнює абсолютному значенню ***pid***;

– **pid=-1** означає очікувати будь-якого дочірнього процесу; функція ***wait()*** поводиться точно так само;

– **pid=0** означає очікувати будь-якого дочірнього процесу, чий ідентифікатор групи процесів дорівнює такому у пов'язаних з поточною діяльністю;

– **pid>0** означає очікувати дочірнього процесу, чий ідентифікатор дорівнює ***pid***.

Значення ***options*** створюється шляхом бітової операції АБО над такими константами:

– ***WNOHANG*** – означає повернути управління негайно, якщо жоден дочірній процес не завершив виконання.

– ***WUNTRACED*** – означає повертати управління також для зупинених дочірніх процесів, про чий статус ще не було повідомлено.

Кожний дочірній процес під час завершення роботи посилає свому процесу-батькові спеціальний сигнал ***SIGCHLD***, на який у всіх процесів за замовчуванням встановлена реакція «ігнорувати сигнал». Наявність такого сигналу спільно з системним викликом **waitpid()** дозволяє організувати асинхронний збір інформації про статус завершених породжених процесів процесом-батьком.

Для перевантаження виконуваної програми можна використовувати функції сімейства ***exec***. Основна відмінність між різними функціями в сімействі полягає в способі передачі параметрів.

```
int execl(char *pathname, char *arg0, arg1, ..., argn, NULL);
int execle(char *pathname, char *arg0, arg1, ..., argn, NULL, char **envp);
int execlp(char *pathname, char *arg0, arg1, ..., argn, NULL);
int execlpe(char *pathname, char *arg0, arg1, ..., argn, NULL, char **envp);
int execv(char *pathname, char *argv[]);
int execve(char *pathname, char *argv[],char **envp);
int execvp(char *pathname, char *argv[]);
int execvpe(char *pathname, char *argv[],char **envp);
```

Основна відмінність між різними функціями в сімействі полягає в способі передачі параметрів. Як видно з рис. 1, всі ці функції виконують один системний виклик ***execve***.

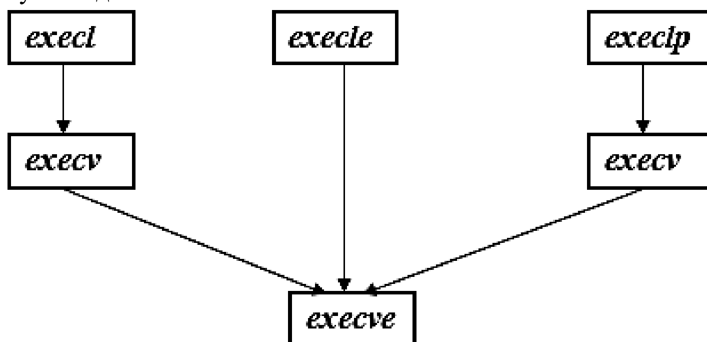


Рис. 1. – Дерево сімейства викликів ***exec***

Завдання

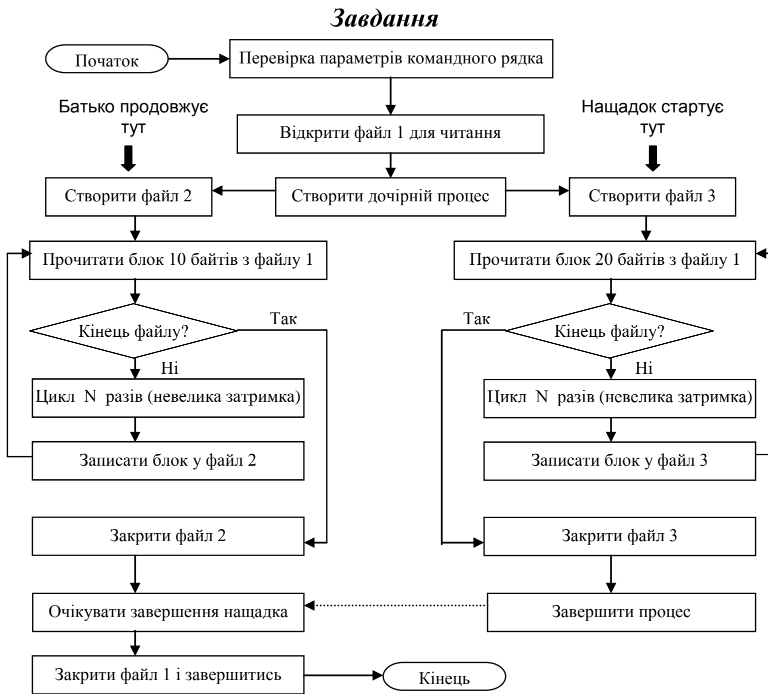


Рис. 2. – Схема програми ***prog.c***

1. Напишіть програму ***prog.c*** відповідно до схеми, представленої на рис. 2.
   Батьківський процес відкриває існуючий текстовий файл 1 для читання; потім створює дочірній процес. Після цього батьківський процес створює новий файл 2 для запису і копіює в нього вміст файлу 1 блоками по 10 байтів кожний.
   Одночасно з цим дочірній процес створює новий файл 3 для запису і копіює в нього вміст файлу 1 блоками по 20 байтів кожний.
   Після завершення копіювання обидва процеси завершуються. Програма повинна запускатися з трьома параметрами:

$prog file1 file2 file3

2. Запустіть програму кілька разів для заданого значення ***N*** з одними і тими ж іменами файлів і порівняйте розмір та вміст файлів 2 і 3 з розміром і вмістом файлу 1. Поясніть результати.
3. Повторіть попередній пункт для ***N = 10***, ***5000***, ***10000*** та ***100000***.

Контрольні питання

1. Які існують способи програмно запустити новий процес? Чи вони відрізняються один від одного?
2. Які функції входять до сімейства exec?
3. Що відбувається при системному виклику fork()?
4. Як програмним чином завершити процес?
5. Яка функція призначена для очікування завершення дочірніх процесів?
6. Чим функція waitpid() відрізняється від функції wait()?

Лабораторна робота № 20.
Використання сигналів у ОС Linux програмним шляхом мовою C

Теоретичні відомості

Сигнали не можуть безпосередньо переносити інформацію, що обмежує їх придатність як загального механізму взаємодії між процесами. Проте кожному типу сигналів присвоєно мнемонічне ім'я (наприклад, ***SIGINT***), яке вказує, для чого зазвичай використовується сигнал цього типу. Імена сигналів визначені в стандартному заголовному файлі ***<signal.h>*** за допомогою директиви препроцесора ***#define***.

Як і слід було очікувати, ці імена відповідають невеликим позитивним цілим числам. З точки зору користувача, отримання процесом сигналу виглядає як виникнення переривання. Процес перериває виконання, і управління передається функції-обробнику сигналу. Після закінчення обробки сигналу процес може відновити регулярне виконання. Типи сигналів прийнято задавати спеціальними символьними константами. Системний виклик ***kill()*** призначений для передачі сигналу одного або декількох спеціалізованих процесів у рамках повноважень користувача.

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int signal);
```

Послати сигнал (не маючи повноважень суперкористувача) можна тільки процесу, у якого ефективний ідентифікатор користувача збігається з ефективним ідентифікатором користувача для процесу, що посилає сигнал. Аргумент ***pid*** вказує процес, якому посилається сигнал, а аргумент ***signal*** – який сигнал посилається. Залежно від значення аргументів:

– ***pid > 0*** сигнал посилається процесу з ідентифікатором pid;

– ***pid = 0*** сигнал посилається всім процесам у групі, до якої належить посилає процес;

– ***pid = -1*** і посилаючий процес не є процесом суперкористувача, то сигнал посилається всім процесам в системі, для яких ім'я користувача збігається з ефективним ідентифікатором користувача процесу, що посилає сигнал;

– ***pid = -1*** і посилаючий процес є процесом суперкористувача, то сигнал посилається всім процесам в системі, за винятком системних процесів (зазвичай всім, крім процесів з ***pid = 0*** і ***pid = 1***);

– ***pid < 0***, але не ***-1***, то сигнал посилається всім процесам з групи, ідентифікатор якої дорівнює абсолютному значенню аргументу ***pid*** (якщо дозволяють привілеї);

– якщо ***sig = 0***, то проводиться перевірка на помилку, а сигнал не посилається. Це можна використовувати для перевірки правильності аргументу pid (чи є в системі процес або група процесів з відповідним ідентифікатором).

Системні виклики для установки власного обробника сигналів:

```
#include <signal.h>
void (*signal (int sig, void (*handler)
(int)))(int);
int sigaction(int sig, const struct sigaction
*act, struct sigaction *oldact);
```

Структура ***sigaction*** має такий вигляд:

```
struct sigaction {
  void (*sa_handler)(int);
  void (*sa_sigaction)(int, siginfo_t *, void
*);
  sigset_t sa_mask;
  int sa_flags;
  void (*sa_restorer)(void);
}
```

Системний виклик ***signal*** служить для зміни реакції процесу на будь-якої сигнал. Параметр ***sig*** – це номер сигналу, обробку якого належить змінити. Параметр ***handler*** описує новий спосіб обробки сигналу – це може бути вказівник на призначену для користувача функцію-обробник сигналу, спеціальне значення ***SIG_DFL*** (відновити реакцію процесу на сигнал ***sig*** за замовчуванням) або спеціальне значення ***SIG_IGN*** (ігнорувати надійшовший сигнал ***sig***). Системний виклик повертає покажчик на старий спосіб обробки сигналу, значення якого можна використовувати для відновлення старого способу в разі потреби.

Приклад користувальницької обробки сигналу ***SIGUSR1***.

```
void *my_handler(int nsig) {
 //код функції-обробника сигналу
}
```

```
int main()
{
  (void) signal(SIGUSR1, my_handler);
}
```

Системний виклик ***sigaction*** використовується для зміни дій процесу під час отримання відповідного сигналу. Параметр ***sig*** задає номер сигналу і може дорівнювати будь-якому номеру. Якщо параметр ***act*** не дорівнює нулю, то нова дія, не пов'язана з сигналом ***sig***, встановлюється відповідно ***act***. Якщо ***oldact*** не дорівнює нулю, то попередня дія записується в ***oldact***.

Більшість типів сигналів **UNIX** призначені для використання ядром, хоча є кілька сигналів, які надсилаються від процесу до процесу:

SIGALRM – сигнал таймера (**alarm clock**). Надсилається процесу ядром під час спрацьовування таймера. Кожен процес може встановлювати не менше трьох таймерів. Перший з них вимірює минулий час. Цей таймер встановлюється самим процесом за допомогою системного виклику ***alarm()***.

SIGCHLD – сигнал зупинки або завершення дочірнього процесу (**child process terminated or stopped**). Якщо дочірній процес зупиняється або завершується, то ядро повідомить про це батьківському процесу, пославши йому такий сигнал. За замовчуванням батьківський процес ігнорує цей сигнал, тому якщо в батьківському процесі необхідно отримувати відомості про завершення дочірніх процесів, то потрібно перехоплювати цей сигнал.

SIGHUP – сигнал звільнення лінії (**hangup signal**). Надсилається ядром всім процесам, підключеним до керуючого термінала (**control terminal**) у разі відключення термінала. Він також посилається всім членам сеансу, якщо завершує роботу лідер сеансу (зазвичай процес командного інтерпретатора), пов'язаного з керуючим терміналом;

SIGINT – сигнал переривання програми (**interrupt**). Надсилається ядром всіх процесів сеансу, пов'язаного з терміналом, коли користувач натискає кнопку переривання. Це також звичайний спосіб зупинки виконання програми.

SIGKILL – сигнал знищення процесу (**kill**). Це досить специфічний сигнал, який посилається від одного процесу до іншого і призводить до негайного припинення роботи отримує сигнал процесу.

SIGPIPE – сигнал про спробу запису в канал або сокет, для яких приймає процес, вже завершений.

SIGPOLL – сигнал про виникнення одного з опитуваних подій (**pollable event**). Цей сигнал генерується ядром, коли деякий відкритий дескриптор файлу стає готовим для введення або виведення.

SIGPROF – сигнал профілюючого таймера (**profiling time expired**). Як було згадано для сигналу ***SIGALRM***, будь-який процес може встановити не менше трьох таймерів. Другий з цих таймерів може використовуватися для вимірювання часу виконання процесу в призначеному для користувача і системному режимах. Цей сигнал генерується, коли закінчується час, встановлений у цьому таймері, і тому може бути використаний засобом профілювання програми.

SIGQUIT – сигнал про вихід (**quit**). Дуже схожий на сигнал ***SIGINT***, цей сигнал посилається ядром, коли користувач натискає кнопку виходу використовуваного термінала. На відміну від ***SIGINT***, цей сигнал призводить до аварійного завершення і скидання образу пам’яті.

SIGSTOP – сигнал зупинки (**stop executing**). Це сигнал управління завданнями, який зупиняє процес. Його, як і сигнал ***SIGKILL***, не можна проігнорувати або перехопити.

SIGTERM – програмний сигнал завершення (**software termination signal**). Програміст може використовувати цей сигнал для того, щоб дати процесу час для «наведення порядку», перш ніж посилати йому сигнал ***SIGKILL***.

SIGTRAP – сигнал трасування переривання (trace trap). Це особливий сигнал, який в поєднанні з системним викликом ptrace використовується відлагоджувачами, такими як **sdb**, **adb**, **gdb**.

SIGTSTP – термінальний сигнал зупинки (**terminal stop signal**). Він формується у результаті натискання спеціальної клавіші зупинки.

SIGTTIN – сигнал про спробу введення з термінала фоновим процесом (**background process attempting read**). Якщо процес виконується у фоновому режимі і намагається виконати читання з керуючого термінала, то йому надсилається цей сигнал. Дія сигналу за замовчуванням – зупинка процесу.

SIGTTOU – сигнал про спробу виведення на термінал фоновим процесом (**background process attempting write**). Аналогічний сигналу ***SIGTTIN***, але генерується, якщо фоновий процес намагається виконати запис у керуючий термінал. Дія сигналу за замовчуванням – зупинка процесу.

SIGURG – сигнал про надходження в буфер сокета термінових даних (**high bandwidth data is available at a socket**). Він повідомляє процесу, що з мережевого з’єднання отримані термінові позачергові дані.

SIGUSR1 і ***SIGUSR2*** – призначені для користувача сигнали (**user defined signals 1 and 2**). Так само, як і сигнал ***SIGTERM***, ці сигнали ніколи не надсилаються ядром і можуть використовуватися для будь-яких цілей за вибором користувача.

SIGVTALRM – сигнал віртуального таймера (virtual timer expired). Третій таймер можна встановити так, щоб він вимірював час, яке процес виконує в призначеному для користувача режимі.

Набори сигналів визначаються за допомогою типу ***sigset_t***, який визначений в заголовки ***<signal.h>***. Вибрати певні сигнали можна, почавши або з повного набору сигналів і видаливши непотрібні сигнали, або з порожнього набору, включивши в нього потрібні. Ініціалізація порожнього і повного набору сигналів виконується за допомогою процедур ***sigemptyset*** і ***sigfillset*** відповідно. Після ініціалізації з наборами сигналів можна оперувати за допомогою процедур ***sigaddset*** і ***sigdelset***, відповідно додають і видаляють зазначені вами сигнали.

Опис даних процедур:

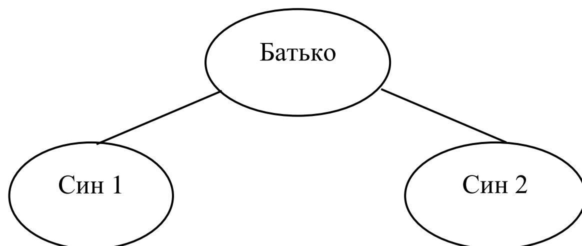
```
#include <signal.h>
/* Ініціалізація */
int sigemptyset (sigset_t *set);
int sigfillset (sigset_t *set);
/* Додавання та видалення сигналів */
int sigaddset sigset_t *set, int signo);
int sigdelset (sigset_t *set, int signo);
```

Процедури ***sigemptyset*** і ***sigfillset*** мають єдиний параметр – покажчик на змінну типу ***sigset_t***. Виклик sigemptyset ініціалізує набір set, виключивши з нього всі сигнали. І, навпаки, виклик ***sigfillset*** ініціалізує набір, на який вказує ***set***, включивши в нього всі сигнали. Додатки повинні викликати ***sigemptyset*** або ***sigfillset*** хоча б один раз для кожної змінної типу ***sigset_t***.

Процедури ***sigaddset*** і ***sigdelset*** приймають як параметри покажчик на ініціалізований набір сигналів і номер сигналу, який повинен бути доданий або вилучений. Другий параметр, ***signo***, може бути символічним ім’ям константи, таким як ***SIGINT***, або справжнім номером сигналу, але в останньому випадку програма виявиться системно-залежною.

Завдання

Організувати функціонування процесів наступної структури:



Процеси визначають свою роботу висновком повідомлень виду:

N pid ppid поточний час (мсек) (N – поточний номер повідомлення) на екран. «Батько» одночасно посилає сигнал ***SIGUSR1*** «синам». «Сини», отримавши цей сигнал, посилають у відповідь «Батькові» сигнал ***SIGUSR2.*** «Батько», отримавши сигнал ***SIGUSR2***, через час t = 100 мсек одночасно, посилає сигнал ***SIGUSR1*** «синам». І так продовжується далі... Написати функції-обробники сигналів, які під час отримання сигналу виводять повідомлення про отримання сигналу на екран. Під час отримання / посилки сигналу вони виводять відповідне повідомлення:

N pid ppid поточний час (мсек) син такий-то get / put SIGUSRm.

Передбачити механізм для визначення «Батьком», від кого з «синів» був прийнятий сигнал.

Контрольні питання

1. Як програмним чином відправити сигнал певному процесу?
2. Які набори сигналів визначені у стандарті POSIX?
3. Які можливості для управління сигналами надає системний виклик ***sigaction***?
4. Які сигнали призначені для дій користувача?
5. Які сигнали у Linux не можуть бути проігноровані?

Лабораторна робота № 21. Програмування потоків у Linux

Теоретичні відомості

Основна мета використання потоків – це поділ програми на підзадачі, які можуть виконуватись паралельно. Порівняно з процесами взаємодія і синхронізація потоків вимагає менше часу, оскільки потоки одного процесу виконуються в одному адресному просторі.

В ОС UNIX/Linux є API для потоків стандарту POSIX (Portable Operating System Interface) – ***pthreads*** («P» - від POSIX). Прототипи функцій роботи з потоками і необхідні типи даних містяться в заголовку ***<pthread.h>***. Ці функції не включені в стандартну бібліотеку мови **C**, вони знаходяться в бібліотеці ***libthread***. Тому в командний рядок для компонування необхідно додати опцію (див. Лабораторну роботу № 14) ***–lpthread***.

Створення та завершення потоків

Створення потоку. Потік створюється функцією ***pthtead_create***:

```
#include <pthread.h>
 int pthread_create( pthread_t *thread, const
pthread_attr_t *attr, void *(*start_routine)(
void*), void *arg);
```

Ця функція має 4 параметри:

1. Вказівник на змінну типу ***pthread_t***, в неї буде записаний ID нового потоку.
2. Вказівник на об’єкт-атрибут потоку. Цей об’єкт управляє деталями взаємодії потоку з іншою програмою. Якщо параметр дорівнює ***NULL***, то потік буде створений з атрибутами за замовчуванням.
3. Вказівник на функцію потоку. Це звичайний вказівник на функцію типу ***void * (*) (void *)***, тобто функція потоку приймає один параметр типу вказівник на ***void*** і повертає значення типу вказівник на ***void***.
4. Значення атрибута потоку типу ***void **. Це значення передається потоку як аргумент у функцію потоку. Через нього можна передати новому потоку параметри.

Після створення кожний потік виконує функцію потоку – звичайну функцію в програмі користувача. У разі завершення цієї функції потік завершується.

Повернення з функції ***pthread_create*** відбувається негайно, і вихідний потік продовжує виконання команд, що настають за викликом ***pthread_create***. Одночасно новий потік починає виконання функції потоку. Функція ***pthread_create*** повертає нуль у випадку успіху або нуль у разі помилки.

За нормальних умов потік завершується двома способами:

1. Звичайне повернення з функції потоку. Змінна, яка повертається у ***return***, буде значенням, що повертається потоком.
2. Повернення за допомогою функції ***pthread_exit***. Вона може бути викликана з будь-якої функції цього потоку. Аргумент цієї функції буде значенням, що повертається потоком.

Приклад 1. Програма створює потік, який безперервно друкує ***'x'*** на стандартний потік помилок. Після створення потоку головний потік безперервно друкує ***'o'*** на стандартний потік помилок. Призупинити виконання програми можна за допомогою <***Ctrl***>-<***S***>; відновити – будь-якою клавішею. Перервати програму можна за допомогою <***Ctrl***>-<***C***>.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
void* print_xs (void* unused)
{
  while (1)
   fputc ('x', stderr);
  return NULL;
}

int main ()
{
  int p;
  pthread_t thread_id;
  p     =     pthread_create(&thread_id,     NULL,
&print_xs, NULL);
  if (p != 0)
 {
perror("Thread problem");
exit(1);
}
  while (1)
   fputc ('o', stderr);
  return 0;
}
```

Передача даних у потік

Оскільки типом аргументу, переданого в потік, є ***void ****, то для передачі одного параметра типу ***int*** його слід перетворити до типу ***(void *)*** Для передачі більшої кількості параметрів аргумент потоку повинен бути вказівником на структуру або область даних, що містить передані параметри.

Необхідно, щоб дані, що передаються новому потоку, були доступні потоку, при цьому не слід передавати стекові змінні.

Приклад 2. Програма створює два нових потоки: один друкує ***'x'***, інший ***'o'*** на стандартний потік помилок. Кожний потік друкує певну кількість символів і потім завершується поверненням з функції потоку. Обидва потоки використовують одну і ту ж функцію, ***char_print***, але викликають її з різними значеннями параметрів.

```
#include <pthread.h>
#include <stdio.h>

struct char_print_parms
{
char character; /* символ, який друкувати */
int count;  /* скільки разів друкувати символ */
};

void* char_print (void* parameters)
{
/* перетворити вказівник до потрібного типу */
char_print_parms* p = (char_print_parms*) parameters;
int i;
for (i = 0; i < p->count; ++i)
fputc (p->character, stderr);
return NULL;
}

int main ()
{
 pthread_t thread1_id;
  pthread_t thread2_id;
  char_print_parms thread1_args;
  char_print_parms thread2_args;

  thread1_args.character = 'x';
  thread1_args.count = 30;  /* друкувати 'x' 30 разів */
```

```
    pthread_create (&thread1_id, NULL, &char_print,
&thread1_args);
    thread2_args.character = 'o';
    thread2_args.count = 20;  /* Друкувати 'o' 20 разів */
    pthread_create (&thread2_id, NULL, &char_print,
&thread2_args);
  return 0;
  }
```

Об'єднання потоків

У разі потоків аналогом функції ***wait*** є функція ***pthread_join***: потік, який викликав цю функцію, буде очікувати завершення зазначеного потоку. Функція повертає нуль у випадку нормального виконання, і не нуль у разі помилки. Функція має два параметри: ID потоку, завершення якого слід очікувати, і змінну типу вказівник на ***void***, куди буде записано значення, що повертається потоком. Якщо це значення не потрібно, то другий параметр функції ***pthread_join*** може бути ***NULL***.

```
  #include <pthread.h>
  int pthread_join (pthread_t thread, void
**status_addr);
```

Завдання

1. Написати програму, яка створює два дочірніх потоки. Батьківський процес і два дочірніх потоки повинні виводити на екран свій ***id*** і ***pid*** батьківського процесу і поточний час у форматі: година: хвилини: секунди: мілісекунди.
2. Знайдіть помилку у Прикладі 2 лабораторної роботи. Для налагодження програми додайте у функцію потоку друк переданих параметрів. Виправте помилку і поясніть її причину.
3. Модифікуйте Приклад 2 таким чином:
   – додайте в головний потік виклик функцій ***pthread_join*** для очікування завершення обидвох дочірніх потоків;
   – поверніть з дочірніх потоків якісь значення (різні!) і роздрукуйте їх в головному потоці.

Контрольні питання

1. Як програмним чином створити новий потік?
2. Як програмним чином завершити потік?
3. Як програмним чином приєднати потік?
4. Як програмним чином від'єднати потік?

Для нотаток

Для нотаток

Навчальне видання

Гліб Валентинович
Горбань

ОПЕРАЦІЙНІ СИСТЕМИ: підготовка до виконання лабораторних робіт

Методичні вказівки

Випуск 368

Редактор *А. Бурмус.*
Технічний редактор *О. Петроченко.*
Комп'ютерна верстка *Д. Кардаш.*
Друк *С. Волинець*, фальцювально-палітурні роботи *О. Мішалкіна.*

Підп. до друку 02.09.2021
Формат 60x84 $^{1}/_{16}$. Папір офсет.
Гарнітура «Times New Roman». Друк ризограф.

Ум. друк. арк. 8,60. Обл.-вид. арк. 5,24.
Тираж 5 пр. Зам. № 6319.

Видавець і виготовлювач: ЧНУ ім. Петра Могили.
54003, м. Миколаїв, вул. 68 Десантників, 10.
Тел.: 8 (0512) 50–03–32, 8 (0512) 76–55–81, e-mail: rector@chmnu.edu.ua.
Свідоцтво суб'єкта видавничої справи ДК № 6124 від 05.04.2018.