МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни “Операційні системи”

для студентів спеціальності 121
"Інженерія програмного забезпечення"

2022

Методичні вказівки до виконання лабораторних робіт з дисципліни “Операційні системи” для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад.: Сердюк С.М., Качан О.І. – Запоріжжя: НУ «Запорізька політехніка», 2022. – 75 с.

Укладачі: С.М. Сердюк, к.т.н., доцент кафедри ПЗ,
О.І. Качан, асистент кафедри ПЗ.

Рецензент: С.К. Корнієнко, к.т.н., доцент кафедри ПЗ,
О.О. Степаненко, к.т.н., доцент кафедри ПЗ.

Відповідальний
за випуск: С.О. Субботін, зав. каф. ПЗ, д.т.н., професор

Затверджено
на засіданні кафедри
"Програмні засоби"

Протокол № 9 від 15.05.2022 р.

ЗМІСТ

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Методичні вказівки представляють собою необхідний засіб для успішного вивчання теоретичного матеріалу та практичного освоєння предмету «Операційні системи».

Мета роботи - підвищити рівень знань студентів з методики написання програм, їхнього алгоритмічного представлення та відлагодження на мові низького рівня (Асемблер). Лабораторний курс складається з cmjми робіт, що охоплює велику область програмного забезпечення та зовнішніх пристроїв. Кожна робота виконується та здається студентом індивідуально. Студенти, що не підготовлені до роботи, а також, які не мають вірно оформленого звіту, до занять не допускаються.

Вимоги до оформлення та змісту звіту, а також контрольні запитання, представлені в кожній лабораторній роботі. Студент повинен знати мету роботи, теоретичні відомості, методику побудови та відлагодження необхідних програм.

Студент, який не здав попередньої роботи, не допускається до виконання наступних.

1 ЛАБОРАТОРНА РОБОТА №1

Функції операційної системи MS-DOS: визначення версії ОС

Мета роботи: Практичне вивчення роботи основних функцій операційної системи MS-DOS.

1.1 Короткі теоретичні відомості

Звернення до функцій DOS та BIOS здійснюється за допомогою програмних переривань (команда INT). Система переривань машин типу IBM PC в принципі не відрізняється від любої іншої системи векторизованих переривань. Самий початок оперативної пам'яті від адреси 0000h до 03FFh відводиться під вектори переривань – чотирьох байтові області, в яких зберігаються адреси програм обробки переривань (ПОП). У два старші байти кожного вектору записується сегментний адрес ПОП, а у два молодші - відносний адрес точки входу в ПОП у сегменті (зміщення). Вектори, як і відповідні їм переривання, мають номера, що називаються типами, причому вектор з номером 0 (вектор типу 0) розміщується з адреси 0, вектор типу 1- з адреси 4, вектор типу 2 - з адресу 8 і т.д. (від (N*4 до ...)). Всього у виділеній під вектори області пам'яті розміщується 256 векторів. Отримавши сигнал на виконання ПОП з визначеним номером, процесор зберігає в стеку програми, що виконується, слово флагу, а також сегментний та відносний адрес сегменту команд (CS та IP) та загружає CS та IP з відповідного вектора переривань, здійснюючи перехід на ПОП. ПОП звичайно закінчується командою повернення з переривання IRET, що виконує зворотні дії - загрузку IP та CS і регістру флагів із стеку, що приводить до повернення в основну програму у точку, де вона була перервана.

Вимоги на виконання векторів переривань можуть мати різну природу. Незалежно від джерела, ПОП завжди виконується однаково. Більша частина векторів переривань зарезервована для виконання визначених дій; частина з них автоматично заповнюється адресами системних програм при завантаженні програми.

У системі передбачено переривання, що повертає номер версії MS-DOS. Це число може використовуватися для перевірки виконання програми, програма може при старті видавати звістку про помилку, якщо їй потрібна інша версія MS-DOS.

Призначенням функції 3306h переривання 21h є отримання версії DOS. Функція повертає номер версії MS-DOS, що використовується.

При виклику:

```
AH=3306h
```

При поверненні:

```
BL= номер основної версії
BH= номер підверсії
DL= ревізія (біти 2-0, інші – 0)
DH= прапорці версії
Біт 3: DOS у ROM
Біт 4: DOS у HMA
AL=FFh якщо версія DOS < 5.0
```

Приклад 1.

Реалізація визначення версії MS-DOS.

```
.....
MOV AH,30h
INT 21h
CMP BL,5; перевірка на версію  5.x
JL Wrong_DOS; якщо версія менша 5.0, то видати
                  повідомлення
.....
```

Використане переривання змінює зміст регістрів BX та CX, в які повертається значення 0.

1.2 Домашнє завдання

Використовуючи конспект лекцій, рекомендовану літературу, вивчити роботу функції 3306h INT 21h, структуру та образ пам'яті програм типу .EXE та .COM, структуру та завантаження ОС MS DOS, основи програмування на мові асемблеру, роботу простих функцій вводу з клавіатури та виводу на екран текстової інформації. Ознайомитися зі змістом та порядком виконання роботи.

1.3 Хід виконання роботи

Написати та відлагодити програму, що визначає версію MS-DOS.

1.4 Зміст звіту

1.4.1 Сформульована мета роботи.
1.4.2 Алгоритм функціонування програми.
1.4.3 Лістінг готової програми.
1.4.4 Карти розподілу пам’яті.

1.5 Контрольні запитання

1.5.1 Структура та образ пам’яті програми *.EXE.
1.5.2 Структура та образ пам’яті програми *.COM.
1.5.3 Використання утиліти MS-DOS **EXE2BIN**. Порядок трансляції, компоновки (редагування зв'язків) та запуску на виконання.
1.5.4 Як відбувається звернення до функцій DOS та BIOS.
1.5.5 Пояснити склад карти пам’яті (файл типу *.MAP).
1.5.6 Пояснити роботу функції 3306h INT 21h. Принцип та джерело отримання необхідної інформації.
1.5.7 Структура PSP.
1.5.8 Визначення операційної системи. Завантаження ОС.

2 ЛАБОРАТОРНА РОБОТА №2

Використання звукових можливостей IBM PC

Мета роботи: вивчити способи генерації звука, навчитися використовувати звукові можливості IBM-сумісних ПЕОМ та практично реалізувати запропоновані способи генерації звука.

2.1 Короткі теоретичні відомості

Стандартні члени сімейства PC можуть, використовуючи таймер, що програмується, та вбудований у комп'ютер динамік, створювати найпростіші звуки. Звук - це просто регулярні пульсації чи вібрації тиску повітря. Динамік в комп'ютерах IBM PC вібрує за рахунок електричних імпульсів, що надсилає йому комп'ютер. Через те, що комп'ютери мають справу з двійковими числами, то й створюють вони або високу, або низьку напругу. При кожному переході напруги з одного стану в інший мембрана динаміка або виштовхується, або втягується.

Звук через динамік ми можемо генерувати двома способами. При одному способі можна записати програму, яка буде змінюючи стан двох розрядів динаміка в периферійному інтерфейсі, що програмується (PPI), включати та виключати динамік. При іншому способі для створення коливань в динаміку заданої частоти використовується вмонтований в PC таймер, що програмується. Другий спосіб має більше розповсюдження, за двох причин:

- так як частотою звука керує таймер, а не програма, ЦПП комп'ютера в цей час вільний;
- робота таймера не залежить від робочої швидкості ЦПП.

2.1.1 Керування звуком за допомогою таймера

Для створення звуку за допомогою таймера необхідно виконати два кроки:

- необхідно запрограмувати таймер для генерації деякої частоти;
- направити вихід з таймеру на динамік. Таймер по команді зчитує сигнал генератора тактових імпульсів (маючих частоту 1.193 МГц) до тих пір, поки їх число не співпаде з числом, заданим нами (лічильник). Фактично таймер ділить тактову частоту на наше число, та створює вихідну частоту. В результаті таймер генерує

сигнали, які створюють звук визначеної частоти, коли ми підєднуємо його до динаміка.

Лічильник і результуюча частота пов'язані між собою наступним відношенням:

частота = 1193180/лічильник.

Після обчислювання лічильника, що необхідний для тієї частоти звуку, яка нам потрібна (звукова), ми надсилаємо його в регістр таймера. Це робиться за допомогою трьох виводів у порти. Спочатку у порт 43H надсилається значення B6H, тим самим таймеру повідомляється, що йому надіслано лічильник. Потім в порт 42H надсилається спочатку молодший, потім старший байти лічильника.

Після того, як ми запрограмували таймер, необхідно задіяти схеми динаміка. Динамік керується за допомогою PPI та використовує порт 61H. Динаміком використовується тільки два з восьми розрядів порту: розряди 0 та 1. Останні розряди використовуються для інших цілей, тому при роботі з динаміком ми не повинні змінювати їх.

Розряд 0 керує сигналом таймера, що використовується для роботи динаміка. Для того, щоб динамік працював від таймеру, обидва ці розряди повинні бути встановлені в 1.

Приклад 1.

Генерація звуку за допомогою таймера:

```
STAK SEGMENT PARA STACK 'stack'
     dw 125 dup(?)
STAK ENDS
DAT SEGMENT PARA 'data'
     m_freq dw 165, 156, 165, 0FFFF
     m_time dw 3 dup (100)
DAT ENDS
COD SEGMENT PARA 'CODE'
ASSUME CS:COD, DS:DAT, SS:STAK
PROGRAM PROC FAR
      ..................
     LEA SI,m_freq
     LEA BP, DS:m_time
      ..................
FREQ:
     MOV DI,[SI]
     CMP DI,0FFFFH
     JE END_PL
     MOV BX, DS:[BP]
```

```
        MOV AL, 0B6H
        OUT 43H, AL
        MOV DX, 14H
        MOV AX, 4F38H
        DIV DI
        OUT 42H, AL
        MOV AL, AH
        OUT 42H, AL
        IN AL, 61H
        MOV AH, AL
        OR AL,3
        OUT 61H, AL
        ……….
        MOV AL,AH
        OUT 61H, AL
        ADD SI, 2
        ADD BP, 2
        JNZ FREQ
END_PL:
            ……….
PROGRAM ENDP
COD ENDS
END PROGRAM
```

2.1.2 Пряме керування динаміком

Таймер керує динаміком, надсилаючи періодичні сигнали, що примушують коливатися мембрану динаміка. Теж саме ми можемо зробити і за допомогою програми, яка надсилає до динаміка сигнали, що вмикають, та вимикають його. Встановивши в нуль розряд 0 порту 61H і тим самим відєднавши динамік від таймера, а потім встановивши та скинувши розряд 1, ми змусимо динамік коливатися. При використанні цього методу швидкість програми визначає частоту звуку: чим швидше буде виконуватися програма - тим вище буде тон.

2.2 Домашнє завдання

Використовуючи конспект лекцій, рекомендовану літературу, вивчити методи генерації звуку. Ознайомитися зі змістом та порядком виконання роботи.

2.3 Хід виконання роботи

2.3.1 Написати та відлагодити програму на Асемблері, що реалізує генерацію звуку за допомогою таймера.

2.3.2 Написати та відлагодити програму на Асемблері, що реалізує генерацію звуку за допомогою прямого керування динаміком.

2.4 Зміст звіту

2.4.1 Сформульована мета роботи.

2.4.2 Алгоритм функціонування програми.

2.4.3 Лістинг програм.

2.5 Контрольні запитання

2.5.1 Що таке звук?

2.5.2 В чому полягає основна ідея отримання звуку на IBM PC?

2.5.3 Спосіб генерації звуку за допомогою таймеру.

2.5.4 Спосіб безпосереднього керування динаміком.

2.5.5 Порівняльний аналіз розглянутих методів генерації звуку.

2.5.6 Принцип частотної модуляції звуку.

2.5.7 Звук як частина інформації, що сприймається користувачем.

3 ЛАБОРАТОРНА РОБОТА №3

Функції припинення процесу

Мета роботи: вивчити функції припинення обчислювального процесу, навчитися їх практичному використанню.

3.1 Короткі теоретичні відомості

Завершити роботу програми можна різними способами.

Переривання 20h, використовується для завершення програм та передачі управління ОС DOS. При завершенні виконування програми переривання 20h не закриває автоматично всі відкриті програмою файли, тому перед його викликом завжди слід закривати файли, що змінилися. Якщо програмою змінений файл не було закрито, то в каталог не буде записана його нова довжина.

Функція 0h переривання 21h. Функціонально вона ідентична перериванню 20h.

Функція 4Ch переривання 21h, завершує роботу програми та передає тому, хто її визивав, код повернення. Якщо програма була викликана як підпрограма, то програма, що її визивала може отримати код повернення за допомогою функції 41h. Якщо програма викликалась програмою ОС, то код повернення може бути перевірений в командному файлі за допомогою команди ERRORLEVEL. Код повернення повертається в регістр AL. При виконанні цієї функції ОС DOS автоматично закриває всі файли, відкриті за допомогою функції 3Dh, або, можливо, функції 3Ch.

Команда асемблера RET.

Переривання 27h, яке забезпечує завершення роботи програми, але залишає задану її частину в пам'яті залишаючи її резидентною. Ця резидентна частина стає розширенням ОС DOS та інші програми не можуть накладатися на неї. Частіше програмі необхідно залишити тільки якусь свою частину, відкідаючи команди ініціалізації. Ця резидентна частини повинна розміщуватися на початку програми. Тоді при виклику переривання 27h програма може вказати в регістрі DX зміщення в сегменті команд, адрес першого байту, що слідує за її резидентною частиною. Резидентна програма звичайно (але не обов'язково) складається із наступних частин:

а) секції, що перепризначує адреса в таблиці векторів переривань;

б) процедури, що виконуються тільки один раз при завантаженні програми та призначеної для:

1) зміни адреси в таблиці векторів переривань на власний адрес;

2) встановлення розміру частини програми, що повинна стати резидентною;

3) використання переривань DOS для завершення програми та створення резидентної частини по вказаному розміру.

в) процедури, яка залишається резидентною та активується, наприклад, по уводу з клавіатури або по сигналу таймера.

Процедура ініціалізації повинна створювати необхідні умови, для того, щоб забезпечити роботу резидентної програми, а потім дозволити собі бути стертою з пам'яті.

Приклад 1.

Резидентна програма (активується при натисненні Alt+Left Shift)

```
INTTAB SEGMENT AT 0H ;таблиця векторів переривання
     ORG 9H*4        ; адреса для int 9h
     KBADDR LABEL DWORD; подвоєне слово
INITTAB ENDS
;_____________________________________________________
ROMAREA SEGMENT AT 400H    ; область параметрів BIOS
     ORG 17H               ; адреса флагів консолі
     KBFLAG DB ?           ; стан Alt+Left Shift
ROMAREA ENDS
;_____________________________________________________
CSEG SEGMENT PARA
     ASSUME CS:CSEG
     ORG 100H
BEGIN: JMP INITZE
     KBSAVE DD ?           ;для адреси int 9h BIOS
MAIN PROC NEAR
      ...
     CALL KBSAVE           ; обробка переривань
     ASSUME DS:ROMAREA
     MOV AX, ROMAREA       ; установити DS для доступу
     MOV DS, AX            ; до стану
     MOV AL, KBFLAG
     CMP AL, 00001010B     ; Alt+LeftShift натиснуті?
```

```
     JNE EXIT              ; ні – вийти
...
EXIT:

     IRET
MAIN ENDP
     INITZE PROC
     ASSUME DS:INTTAB
     PUSH DS
     MOV AX, INTTAB
     MOV DS,AX
     CLI              ; заборонити переривання
; зміна адресу обробника
     MOV AX, WORD PTR KBADDR
     MOV WORD PTR KBSAVE, AX
     MOV AX, WORD PTR KBADDR+2
     MOV WORD PTR KBSAVE+2, AX
     MOV WORD PTR KBADDR, OFFSET MAIN
     MOV WORD PTR KBADDR+2, CS
     STI              ; дозволити переривання
     MOV DX, OFFSET INITZE    ; розмір програми
     INT 27H
INITZE ENDP
CSEG ENDS
END BEGIN
```

Функція 31h переривання 21h є покращеною версією переривання 27h. Як додаток до завершення роботи функції 31h , вона надає можливість програмі, що завершується повідомити код повернення , який розмішується в регістрі Al та може бути перевірений в командних файлах ОС DOS. В регістр DX заноситься обсяг пам'яті, яка залишається резидентною, у параграфах (16 Б).

На жаль, в MS-DOS найважливіший обробник переривань в системі – обробник INT 21h, не є повторно вхідним. На відміну від переривань BIOS, обробники яких використовують стек перерваної програми, обробник системних функцій DOS записує в SS:SP адресу дна одного з трьох внутрішніх стеків DOS. Якщо функція була перервана апаратним перериванням, обробник якого викликав іншу функцію DOS, вона буде користуватись тим же стеком, затираючи все, що туди помістила перервана функція. Коли керування повернеться до перерваної функції, у стеці буде сміття і виникне помилка.

Функції BIOS також часто виявляються не повторно вхідними. Зокрема, цим відрізняються обробники переривань 5, 8, 9, 0Bh, 0Ch, 0Dh, 0Eh, 10h, 13h, 14h, 16h, 17h.

3.2 Домашнє завдання

Використовуючи конспект лекцій, рекомендовану літературу, вивчити функції припинення процесу. Ознайомитися із змістом та порядком виконання роботи.

3.3 Хід виконання роботи

Написати та відлагодити програму на Асемблері, що реалізує різні способи виходу із неї.

3.4 Зміст звіту

3.4.1 Сформульована мета роботи.
3.4.2 Алгоритм та текст програми згідно п.3.1.

3.5 Контрольні питання

3.5.1 Назвіть основні способи завершення програм.
3.5.2 Дайте порівняльну характеристику функціям завершення процесу.
3.5.3 Поняття резидентності.
3.5.4 Назвіть основні частини резидентної програми, їх призначення.
3.5.5 Наведіть приклади способів активізації резидентних програм.

4 ЛАБОРАТОРНА РОБОТА №4

Основи Windows. Графічний інтерфейс користувача. Визначення параметрів жорсткого диску

Мета роботи: вивчити принципи створення графічного інтерфейсу в ОС Windows, навчитися використовувати функції WinAPI для визначення параметрів логічних дисків системи.

4.1 Короткі теоретичні відомості

4.1.1 Загальні відомості про жорсткі диски

Кожний сектор диску складається з поля даних та поля службової інформації. Фізичний адрес сектора на диску визначається тріадою [t, h, s], де:

t - номер доріжки або циліндру;

h - номер головки або поверхні;

s - номер сектора.

Логічний дисковий простір любого логічного диску ділиться на дві області: системну та даних. Системна область створюється та ініціалізується при форматуванні, а в послідуючим відновляється при маніпуляції файловою структурою. Системна область кожного логічного диску складається із наступних компонентів:

- Boot Record;
- FAT;
- Root Directory;

Область даних містить файли та каталоги, що підпорядковуються кореневому.

4.1.2 Функції для роботи з дисковими накопичувачами

Для визначення інформації щодо вільного місця на певному диску можна використовувати функції WinAPI GetDiskFreeSpace і GetDiskFreeSpaceEx. Для використання обох функцій необхідно підключити до своєї програми модуль Windows.h, якщо він ще не був підключений.

Розглянемо більш детально роботу цих функцій.

Функція

```
BOOL GetDiskFreeSpace
        (LPCTSTR lpRootPathName,
```

```
        LPDWORD lpSectorsPerCluster,
        LPDWORD lpBytesPerSector,
        LPDWORD lpNumberOfFreeClusters,
        LPDWORD lpTotalNumberOfClusters);
```

повертає інформацію про кількість секторів на один кластер (lpSectorsPerCluster), розмір кластера в байтах (lpBytesPerSector), кількість вільних кластерів (lpNumberOfFreeCluster), загальну кількість кластерів (lpTotalNumberOfClusters). В якості вхідного параметра функція приймає покажчик на нуль-термінований рядок, який визначає кореневий каталог диску, інформацію про який необхідно повернути. Визначення диску має завершуватися зворотнім слешем (наприклад, C:\). В разі успіху функція повертає ненульове значення.

Слід відзначити, що ця функція повертає некоректну інформацію для дисків обсягом більше ніж 2 ГБ. Для отримання коректної інформації про диски великого обсягу слід використовувати функцію

```
BOOL GetDiskFreeSpaceEx
        (LPCTSTR lpDirectoryName,
         PULARGE_INTEGER lpFreeBytesAvailable,
         PULARGE_INTEGER lpTotalNumberOfBytes,
         PULARGE_INTEGER lpTotalNumberOfFreeBytes);
```

Ця функція повертає кількість вільних і доступних користувачеві байт (lpFreeBytesAvailable), загальну кількість доступних користувачеві байт (lpTotalNumberOfBytes), загальну кількість вільних байт (lpTotalNumberOfFreeBytes). В якості вхідного параметра lpDirectoryName функція може приймати не тільки кореневий каталог, але й будь-який каталог на диску.

Необхідно відмітити, що значення, які повертаються функціями GetDiskFreeSpace і GetDiskFreeSpaceEx, можуть бути відмінними від фактичних значень, якщо використовуються дискові квоти.

Також слід відзначити, що функція GetDiskFreeSpaceEx повертає 64-бітні значення, тому треба бути уважними при роботі з ними, щоб не усікти до 32-бітних значень.

Для визначення типу диску може використовуватись функція

```
UINT GetDriveType
        (LPCTSTR lpRootPathName);
```

Вона приймає, аналогічно попереднім функціям, визначення кореневого каталогу диску, і повертає одне з наступних значень:

DRIVE_UNKNOWN – якщо тип диску не може бути визначений;
DRIVE_NO_ROOT_DIR – вказаний корень не є коректним;
DRIVE_REMOVABLE – диск знімний;
DRIVE_ FIXED – диск незнімний;
DRIVE_REMOTE – диск в мережі;
DRIVE_CDROM – диск є CD-ROM-диском;
DRIVE_RAM – диск є RAM-диском.

Для визначення мітки диску і його серійного номеру використовують функцію GetVolumeInformation.

Перелік дисків, наявних у системі, можна отримати за допомогою однієї із функцій GetLogicalDrives або GetLogicalDrivesStrings.

4.1.3 Створення графічного інтерфейсу Windows

Найсуттєвіша відміна програм під Windows від програм, які були написані під інші операційні системи, - це повідомлення. Більшість програм під DOS періодично опитують пристрої вводу, такі, як клавіатура і миша, і таким чином відслідковують стан цих пристроїв. У Windows, це відбувається через так звані повідомлення. По суті, повідомлення – це спосіб взаємодії операційної системи з програмою. За допомогою повідомлень програми отримують необхідну їм інформації щодо подій, які відбуваються в системі.

Життєвий цикл віконної програми для ОС Windows складається з початкового створення вікон, ініціалізації необхідних даних, і головного циклу, в якому оброблюються повідомлення, що надходять до вікна програми.

Створення графічного інтерфейсу Windows-програми і забезпечення обробки повідомлень є доволі трудомістким. Для спрощення цього процесу існують розроблені різними фірмами бібліотеки класів, які інкапсулюють роботу з компонентами графічного інтерфейсу Windows. Серед цих бібліотек можна виділити MFC (Microsoft Foundation Classes) від Microsoft і VCL (Visual Components Library) від Borland.

Розглянемо принцип створення графічного інтерфейсу програми у середовищі MS Visual Studio 2005, використовуючи бібліотеку MFC. Найкращих результатів при використанні цієї бібліотеки можна досягнути, використовуючи технологію Документ/Представлення

(Document/View), але для реалізації елементарного інтерфейсу можна обмежитися створенням діалогового вікна.

Для створення проекту діалогового вікна у MS Visual Studio 2005 необхідно викликати пункт меню File – New – Project, і у вікні, що з'явилося, обрати Visual C++ - MFC – MFC Application, і ввести ім'я майбутнього проекту. Після цього буде викликано майстер настройки нового проекту. У розділі Application Type слід обрати Dialog Based, також там можна обрати, яким чином використовувати бібліотеку MFC (як загальну DLL або як статичну бібліотеку), і чи треба включати в програму підтримку кодування Unicode. В розділі Unit Interface Features підстроюється вид майбутнього вікна. На цьому попередню настройку можна завершити, натиснувши кнопку Finish.

В результаті буде згенеровано:

– файли, які містять клас додатку (звичайно носять ім'я проекту), успадкований від класу MFC CWinApp;

– файли, які містять клас діалогового вікна (звичайно носять ім'я проекту плюс Dlg), успадкований від класу MFC CDialog;

– файли ресурсів Resource;

– файли прекомпільованих заголовків stdafx.

Навігацію цими файлами можна здійснювати за вікна Solution Explorer (CTRL-ALT-L).

Налаштування інтерфейсу діалогового вікна зберігаються у файлі ресурсів. Для створення його дизайну слід перейти до вікна Resource View (CTRL-SHIFT-E) і в розділі Dialog обрати відповідне діалогове вікно. В результаті з'явиться поточний вигляд вікна. Додавати компоненти до нього можна за допомогою панелі компонентів Toolbox (CTRL-ALT-X).

Розмістимо на діалоговому вікні компонент Combo Box (поле зі списком). Настроїти його параметри можна, виділивши його за допомогою миші і відкрив панель Properties (F4). На цій панелі розміщується список всіх компонентів, які розміщені на діалоговому вікні. Нижче розміщені кнопки, які дозволяють переключати тип сортування властивостей/подій об'єкта (категорізоване або за алфавітом), а також переключатися між відображенням властивостей і подій.

Виставимо у відкритій панелі Properties у розділі Properties властивість Type у значення Drop List. Такий стиль дозволить лише

обирати елементи із вже існуючих у списку, без можливості вводити їх у полі для вводу.

Додамо до створеного списку функцію, що оброблює подію зміни поточного вибраного елементу списку. Для цього необхідно переключитися на відображення в панелі Properties подій (Control Events), знайти назву події CBN_SELCHANGE і, натиснувши у відповідному до нього полі на стрілку, обрати <Add>. Після цього у класі діалогового вікна з'явиться нова функція-член, в яка буде викликатися при зміні обраного в списку елементу.

З більшістю компонентів діалогового вікна можна пов'язувати змінні-члени. Для цього слід викликати контекстне меню відповідного компоненту і обрати пункт Add Variable. У вікні, що з'явилося, проставляється ім'я нової змінної (Variable Name), модифікатор доступу (Access), категорію змінної (якщо треба маніпулювати лише значенням компоненту – обирають категорію Variable, якщо ж необхідно виконувати інші дії над ним – категорію Control). Пов'яжемо створений список зі змінною m_ComboBox категорії Control. В клас діалогового вікна буде додано змінну-член класу CComboBox.

Додамо на діалогове вікно компонент Edit Control (поле для редагування). Пов'яжемо його зі змінною m_Edit категорії Value. В клас діалогового вікна буде додано змінну-член класу CString.

У функцію-член OnInitDialog класу діалогового вікна додамо код, який ініціалізує поле зі списком:

```
m_ComboBox.AddString(L"Text 1");
m_ComboBox.AddString(L"Text 2");
m_ComboBox.AddString(L"Text 3");
```

Повернемося до функції, яка оброблює зміну поточного елементу поля зі списком, внесемо туди наступний код:

```
m_ComboBox.GetLBText(m_ComboBox.GetCurSel(), m_Edit);
UpdateData(false);
```

Функція-член класу CComboBox GetLBText заносить у змінну m_Edit значення рядку зі списку з номером m_ComboBox.GetCurSel() (поточний обраний елемент). Функція-член класу CDialog UpdateData обновлює дані, пов'язані з компонентами діалогового вікна. При цьому її параметр вказує, в якому напрямку виконувати обновлення – зі змінних до компонентів (false), або навпаки (true).

Тепер при зміні елементу поля зі списком новий обраний елемент буде відображатися у полі для редагування.

4.2 Домашнє завдання

Використовуючи конспект лекцій, рекомендовану літературу, вивчити роботу функцій WinAPI для GetDiskFreeSpace, GetDiskFreeSpaceEx, GetDriveType, GetVolumeInformation, GetLogicalDrives, GetLogicalDrivesStrings. Вивчити структуру фізичного диску, методи його обслуговування. Ознайомитися з засобами проектування графічного інтерфейсу для програм Windows.

4.3 Хід виконання роботи

Реалізувати програму для Windows, яка за допомогою графічного інтерфейсу надає відомості про всі диски системи: тип диску, загальний обсяг диску, обсяг вільної пам'яті на диску, число секторів у кластері, кількість байтів в секторі, мітка диску, його серійний номер і тип файлової системи.

4.4 Зміст звіту

4.4.1 Сформульована мета роботи.
4.4.2 Алгоритм функціонування програми.
4.4.3 Лістинг готової програми.

4.5 Контрольні питання

4.5.1 Пояснити відміну в роботі функцій GetDiskFreeSpace і GetDiskFreeSpaceEx.
4.5.2 Пояснити відміну в роботі функцій GetLogicalDrives і GetLogicalDrivesString.
4.5.3 Яку інформацію можна отримати за допомогою функції GetVolumeInformation?
4.5.4 Пояснити структуру фізичного диску, розділеного на томи.
4.5.5 Пояснити призначення та зміст FAT.
4.5.6 Пояснити призначення, структуру та зміст Master Boot.
4.5.7 Пояснити призначення, структуру та зміст Boot.
4.5.8 Пояснити призначення, структуру та зміст Root.
4.5.9 Принципи побудови графічного інтерфейсу користувача в середовищі Windows.

5 ЛАБОРАТОРНА РОБОТА №5

Базові операції Linux-системи. Основні команди ОС

Мета роботи: Придбати основні навички роботи в командному рядку ОС Linux. Освоїти основні команди керування файлами, каталогами та потоками даних в ОС Linux.

5.1 Стислі теоретичні відомості

5.1.1 Командний рядок

Після завантаження операційної системи Linux необхідно переувімкнути режим роботи із системою з графічного на консольний за допомогою комбінації клавіш **Ctrl+Alt+F2**. Надалі обрати один з 5 терміналів для входу (комбінації клавіш: з **Alt+F2** до **Alt+F6**) та увійти до сеансу або увійти до сеансів декількох терміналів одночасно, наприклад до одного терміналу як користувач **stud** та до іншого як адміністратор **root** (обов'язково при виконанні роботи №7 «Системне адміністрування Linux»).

Після входу до системи під зазначеним акаунтом користувач знаходиться у інтерфейсі командного рядка. З цього моменту починається робота з командним інтерпретатором, який очікує від користувача команду (можливо з опціями та аргументами). Інтерпретатор команд (***shell***, оболонка) – це спеціальна програма, яка здійснює функції інтерфейсу між користувачем та операційною системою. Команди, які вводяться користувачем з командного рядка, інтерпретуються оболонкою та посилаються як інструкції низького рівня у операційну систему. Початок командного рядка відзначається запрошенням інтерпретатора – на екрані буде відображена «строка запрошення» з ім'ям користувача та робочим каталогом. Строка закінчується символом **$**, що є очікуванням для введення команд *bash*-середовища:

```
$
```

Поява запрошення означає, що ви увійшли в систему; тепер можна вводити команди. Наприклад, команда date виводить на екран поточну дату і час:

```
$ date
Sun July 6 10:30:21 PST  2014
$
```

Інтерпретація командного рядка відбувається у відповідності з особливим синтаксисом. Першим словом, яке вводиться в командному рядку, повинно бути ім'я команди. Наступні слова автоматично сприймаються як опції та аргументи команди.

$ <команда> <опції та аргументи>

Кожне слово в командному рядку повинно бути відділено від інших слів одним або кількома пробілами або знаками табуляції. Опція – це однобуквений код, що починається дефісом, який модифікує тип дії, яка виконується командою.

При ознайомленні з командами Linux необхідно організувати перехід від однієї до іншої команди таким чином, щоб створювалась логічна послідовність. Наприклад, якщо середовище незнайоме користувачу, то має сенс виконати команду **help**. Результатом команди буде список декількох допустимих команд середовища.

Вивчення середовища починаємо з огляду робочого каталогу командою:

$ ls

Зрозуміло, що команда створена від англійського слова ***listen***. Якщо результат виконання команди незрозумілий, то необхідно оглянути параметри команди, використовуючи вбудовану до середовища інформаційно-допоміжну систему *Manual*. Наприклад:

$ man ls

Результатом буде інформація для роботи з командою **ls**, а точніше у розділі SYNOPSIS буде відображено формат запуску команди, де вираз у дужках **[ ]** є необов'язковим параметром, а аргументи команди розділяються символом, який отримується натисканням ***«space»***. Кожна команда середовища має додаткові параметри з назвою *ключі або опції*. Ключі починаються зі знаку "**–**"та найчастіше знаходяться між командою та аргументами.

$ ls –l

Команда **ls** видає ім'я всіх файлів і каталогів в поточному, або заданому каталозі (аналог команди **dir -w** у DOS). З опцією **-l** команда **ls** видає для кожного файлу рядок з більш детальною інформацією: розмір файлу, дату і час останньої зміни, та ін. Опція -a дозволяє вивести на екран всі файли поточного каталогу, включаючи сховані файли (як правило файли конфігурації), а також посилання на поточний і батьківський каталоги (**.** та **..**).

Багато команд задаються разом з їх аргументами. Аргумент - це слово, яке вводиться у командному рядку після опцій:

```
$ ls –l –a –r –R file1
```

або скорочено:

```
$ ls –larR file1
```

У даному прикладі використовувалась група опцій. Слід відзначити, що в таких випадках дефіс ставиться один.

Результатом виконання команди на екрані терміналу буде виведення списку файлів у поточному каталозі з наступними опціями: **–l** довгий формат; **–a** усі файли (включно з системними, які починаються з символу "**.**"); **–r** сортування у зворотному напрямку за алфавітом («реверсно»); **–R** відобразити також зміст підкаталогів («рекурсивно»).

Якщо аргумент команди **file1** не вказано, то буде відображатися не тільки файл з ім'ям **file1** але й усі інші файли, які підходять під параметри команди.

Слід відзначити, що командний рядок – це буфер тексту, який редагується. До натискання клавіші Enter, над введеним текстом можна виконувати операції редагування:

Ctrl+u видаляється цілий рядок;

Ctrl+f переміщення на знак вперед;

Ctrl+b переміщення на знак назад;

Ctrl+d видаляється символ, на якому встановлено курсор;

Ctrl+h, Del або Backspace видаляється знак перед курсором;

Ctrl+cперериваєтьсявиконання команди і знімається задача;

Ctrl+zпризупиняється виконання команди і задача переводиться в розряд зупинених.

Слід зазначити, що деякі клавіші співпадають з клавішами, зарезервованими ОС, під якою виконується програма TELNET (в Windows, як відомо, Ctrl+c копіює вміст виділення в буфер). У командному інтерпретаторі передбачено велику кількість ефективних засобів: спеціальні символи, оператори переадресації потоків (файлів) вводу-виводу, канали, сценарії і засоби керування задачами. Зокрема, в shell застосовуються універсальні символи підстановки. Символ "*****" позначає яку завгодно послідовність символів, включаючи порожню; "**?**" - який завгодно один символ.

У командному рядку можуть одночасно бути присутніми кілька спеціальних символів. Наприклад, «зірочкою» можна користуватися для звернення до файлів, імена яких містять визначену комбінацію символів:

```
$ ls
doc1 doc2 document docs mydoc monday tuesday
$ ls doc*
doc1 doc2 document docs
$ ls *day
monday tuesday
$ ls doc?
doc1 doc2 docs
```

Ці ж універсальні символи допускаються при використанні команди **rm** для одночасного видалення декількох файлів.

5.1.2 Стандартний ввід, стандартний вивід і переадресація потоків

Всі звичайні файли в ОС Linux (а також вхідна і вихідна інформація команд) мають однакову структуру - байтовий потік. Вхідні дані для команди направляються в потік даних, що називається стандартним вводом **stdin**, а вся вихідна інформація направляється в потік даних, що називається стандартним виводом **stdout**.

Існує ще один стандартний потік – потік повідомлень про помилки **stderr** (по суті – другий вихідний потік). Оскільки стандартний ввід і стандартний вивід мають таку ж структуру, як і файли, вони вільно сполучаються з останніми. За допомогою операторів переадресації, стандартний ввід і стандартний вивід можна перенаправляти з файлу і в файл. За допомогою оператора переадресації "**>**" стандартний вивід можна переадресувати з команди в файл. За допомогою оператора переадресації "**<**" можна переадресувати стандартний ввід так, щоб дані вводились з існуючого файлу. Оператор "**>>**" служить для добавлення вмісту стандартного виводу до існуючого файлу.

Наприклад, вам необхідно сформувати файл, що містить лістинг поточного каталогу. Це можна зробити, направивши стандартний вивід команди **ls** в файл, а не на екран:

```
$ ls –larR > list2
```

Результатом виконання даної команди буде виведення списку файлів у поточному каталозі із зазначеними опціями але не на екран терміналу, а у файл **list2** . Якщо цей файл вже існує, то він буде переписаний наново. У разі необхідності дозаписати файл треба виконати команду:

```
$ ls –larR >> list2
```

За допомогою оператора переадресації створюється файл **list2**, він заповнюється даними з команди **ls –larR**. Однак, сама переадресація організується до того, як почнуть надходити дані зі стандартного виводу. Якщо файл вже існує, він буде знищений і замінений новим файлом під тим же ім'ям. Команда, що генерує вихідні дані, виконується тільки після створення файлу переадресації.

Якщо користувач спробує використати один і той самий файл в якості вхідного і вихідного потока, виникне помилка. Оскільки переадресація виконується перед виконанням команди, вхідний файл буде знищено і замінено порожнім файлом з тим же ім'ям, тобто виникне втрата даних. Коли команда почне виконуватися, вона знайде порожній вхідний файл, і, як наслідок, користувач не отримає бажаного результату. Прикладом такої помилки може бути наступний рядок:

```
$ cat myletter > myletter
```

Ви можете додати стандартний вивід команди до вже існуючого файлу, додавши дані потоку в кінець файлу. В наступному прикладі файли myletter і oldletter по черзі дописуються в кінець файлу all_let:

```
$ cat myletter >> all_let
$ cat oldletter >> all_let
```

Поєднавши команду **cat** з оператором переадресації, ви отримаєте легкий спосіб збереження інформації, що вводиться, в файл:

```
$ cat > mydat
This  is  a new line
for the cat command
^D
$
```

Виконання команди **cat** здійснюється до тих пір, доки користувач не натисне комбінацію клавіш Ctrl+D. В ОС Linux цей символ позначає кінець файлу (**EOF**).

Аналогічною операцією у DOS можна вважати наступну команду:

```
C:\copy con mydat
```

(символом кінця файлу у DOS є Ctrl+Z)

Стандартний ввід можна переадресувати так само, як і стандартний вивід. Стандартний ввід може прийматися не з клавіатури, а з файлу. Так, взявши в якості вхідного потоку тільки що створений файл, отримаємо:

```
$ cat < mydat
This  is  a new line
for the cat command
$
```

Операції стандартного вводу і стандартного виводу можна об'єднувати:

```
$ cat < myletter > newletter
```

5.1.3 Програмні канали

Оскільки формат вхідної і вихідної інформації команд однаковий, вихідні дані однієї команди можна використати як вхідні для іншої. Для цього служать канали, які дозволяють стандартний вихідний потік однієї команди посилати в якості стандартного вхідного потоку для іншої команди. В одному командному рядку можна зв'язувати кілька команд, формуючи тим самим конвеєр, в якому кожна команда буде приймати на вхід вихідні дані попередньої команди.

Припустимо, необхідно послати список імен файлів поточного каталогу на принтер. Для цього необхідно як мінімум дві команди: **`ls`** и **`lpr`**. Тобто, необхідно направити вивід команди **`ls`** у якості вводу для команди **`lpr`**. Для утворення такого з'єднання в ОС Linux використовується оператор "канал" (**`|`** - вертикальна риска), поміщений між двома командами. За його допомогою стандартні потоки двох команд зв'язуються:

```
$ ls | lpr
```

Пересилатися по каналу з однієї команди в іншу може вміст цілого файлу:

```
$ cat mydata | lpr
```

Можна вивести на друк дані, що вводяться з клавіатури:

```
$ cat | lpr
This text will
be printed
^D
```

Наприклад, щодо виведення результатів на екран терміналу, строк може бути значно більше, ніж на екрані. У цьому випадку краще використати сполучення за допомогою каналів з командами **more** та **less** . Наприклад:

```
$ ls –larR | more
```

У процесі виводу на екран після заповнення усіх строк буде очікуватись натискання трьох клавіш – або *Enter* (до наступної строки), або *Space* (до наступного заповнення екрану), або *Q* (вихід з даного режиму перегляду).

```
$ ls –larR | less
```

Перегляд строк (вперед/назад/ліворуч/праворуч) можна виконувати за допомогою клавіш $\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$ та перегортати сторінки вперед/назад клавішами *PgUp, PgDn*. Вихід з даного режиму перегляду - *Q* . За допомогою клавіші *Q* також можна виходити з режиму перегляду *Manual.*

5.1.4 Задачі: перевід в фоновий режим, відміна і переривання

При виконанні якої-небудь команди ОС Linux розглядає її як належну до виконання задачу (завдання). Ви можете дати системі вказівку виконувати завдання у фоновому режимі, щоб мати можливість продовжувати виконувати інші команди. Оператор "**&**" (амперсанд) в кінці командного рядка сповіщає систему про те, що дана команда повинна виконуватися в фоновому режимі:

```
$ lpr mydata &
[1]   1925
```

Друк файлу **mydata** здійснюється в фоновому режимі, користувач не чекає виконання цієї задачі, а продовжує працювати в нормальному режимі.

Наприклад, необхідно створити нескінченний процес виводу символу на консоль (фоновий режим) для демонстрації списку завдань і процесів, та виконати перенаправлення потоку у пристрій з ім’ям “*null*”:

```
$ yes > /dev/null &
[2]   1926
$ yes > /dev/null &
[3]   1927
$ yes > /dev/null &
[4]   1928
```

За допомогою команди jobs можна отримати список задач, що переведені у фоновий режим, а також номера і статус цих задач:

```
$ jobs
[1] + Running lpr intro
[2] - Running yes
[3]   Running yes
[4]   Running yes
```

Знак "**+**" позначає завдання, що обробляється в даний момент. Знак "**–**" позначає наступне завдання, що підлягає виконанню.

Поточну задачу можна перервати комбінацією клавіш Ctrl+z. При цьому вона приймає статус зупиненої. Запустити її на фоновому режимі можна командою **bg**; на передній фон (у пріоритетний режим) поточна задача відновлюється командою **fg**. Фонові і зупинені завдання знімаються (відмінюються) командою **kill**. Команди керування задачами реалізовано у командному інтерпретаторі:

```
$ bg %3
$ fg %4
$ kill %2
```

5.1.5 Операції керування файлами

Файли у ОС Linux організовано в каталоги. Всі каталоги з'єднуються один з одним, утворюючи ієрархічну деревовидну структуру. При звертанні до файлів необхідно вказувати не тільки ім'я, але й місце, яке він займає в цій файловій структурі. Ім'я файлу може містити які завгодно букви, знаки підкреслювання і цифри. Можна включати в імена крапки, коми, спеціальні і керуючі символи. Максимальна довжина імені файлу - 255 символів. Ім'я файлу може включати в себе розширення, причому не одне. Для відокремлення розширення від імені служить крапка. Існують спеціальні файли ініціалізації, які використовуються для зберігання параметрів конфігурації різноманітних програм, в тому числі й самого

командного інтерпретатора. Як правило, це сховані файли, імена яких починаються з крапки.

Як вже говорилось, всі файли в ОС Linux мають один і той же формат - байтовий потік. Байтовий потік представляє собою просто послідовність байтів. Це дозволяє системі Linux розповсюджувати файлову концепцію на всі компоненти даних и навіть на апаратні засоби. Розглядаючи всі ці об'єкти (файли, каталоги, пристрої та ін.) як файли, Linux дозволяє спростити організацію даних і обмін ними. Кількість типів файлів залежить від конкретної реалізації ОС, однак існує чотири стандартних типи файлів: звичайні, файли-каталоги, байт-орієнтовані файли пристроїв і блок-орієнтовані файли пристроїв. Існують файли-символічні посилання, іменовані канали та ін.

Як відзначалось вище, файли мають один формат (байтовий потік), але можуть використовуватися по-різному. Найбільш суттєва різниця має місце між двійковими і текстовими файлами. Приклад двійкового файлу - програмний код після його компіляції. Команда file допомагає визначити, чим є той чи інший файл:

```
$ file monday report hello.c
monday:  ASCII text
report:  directory
hello.c: C source file
```

В багатьох випадках виникає необхідність продивлятися вміст файлу. Команди **cat** і **more** виводять вміст файлу на екран (more здійснює посторінковий вивід):

```
$ more mydata
Linux, DOS, Win95 - good, bad and ugly:-)
```

Команда **touch** змінює час останньої модифікації файлу на поточний. Якщо до цього файлу не існувало, то буде створено порожній файл.

5.1.6 Керування каталогами

Користувач може створювати і видаляти власні каталоги, а також змінювати свій робочий каталог. Каталоги створюються і видаляються відповідно командою **mkdir** і командою **rmdir**:

```
$ mkdir reports
$ mkdir /home/chris/letters
$ rmdir reports
$ rmdir /home/chris/letters
```

З опцією **-r** команда rm рекурсивно видаляє вкладені каталоги.

Для того щоб розрізняти файли і каталоги щодо отримання лістингу директорії (результат роботи команди **ls**), використовується опція **-F**:

```
$ ls
weather reports letters
$ ls -F
weather reports/ letters/
```

Для того щоб визначити поточний каталог, необхідно дати команду **pwd**, результат якої - абсолютне (с повним шляхом) ім'я поточного робочого каталогу:

```
$ pwd
/home/dylan/bin
```

Перехід з одного каталогу в інший здійснюється по команді **cd**. В якості аргументу команда **cd** використовує ім'я каталогу, в який ви хочете перейти:

```
$ cd /home/chris/letters
$ cd props
$ cd ..
```

cd без параметрів повертає користувача в домашній каталог. Дві крапки (**..**) є посиланням на батьківський каталог. Одна крапка (**.**) позначає поточний каталог. Таким чином, обидва імені **myfile** і **./myfile** вказують на один і той же файл в поточному каталозі.

5.1.7 Операції з файлами і каталогами

По мірі створення файлів, виникає необхідність зміни їх імен, видалення деяких з них, присвоювання їм додаткових імен та ін. Розглянемо основні операції, що це забезпечують.

Команда **find** здійснює пошук файлів за іменем, типом, володарем, і навіть за часом останньої зміни.

```
$ find <список_каталогів> -<опції> <критерії>
```

При опції **-name** в якості критеріїв задається зразок імені файлу.

Для того щоб команда **find** повідомила імена файлів, які знайшла, необхідно в командному рядку вказати опцію **-print**.

```
$ find reports -name mon* -print
report/monday
```

В даному випадку команда **find** виводить на екран всі файли каталогу **reports**, що відповідають масці **mon***.

Для того щоб створити копію файлу, треба вказати команді **cp** два імені файлу: вихідний файл і новий файл:

```
$ cp <вихідний_файл> <вхідний_файл>
$ cp proposal oldprop
$ ls proposal oldprop
```

Для того щоб скопіювати файл з робочого каталогу в інший каталог, потрібно вказати ім'я цього каталогу команді **cp** в якості другого аргументу:

```
$ cp preface doc1 props
```

Файли **preface** и **doc1** копіюються в каталог **props**.

```
$ cp props/*.* oldprop
```

Копіюються всі файли каталоге **props** в каталог **oldprop**.

За допомогою команди **mv** можна або змінити ім'я файлу, або перемістити файл з одного каталогу в інший.

```
$ mv propossal version1
```

Ім'я файлу **proposal** змінюється на **version1**.

Файл можна перенести з одного каталогу в інший:

```
$ mv newprop props
```

Файл **newprop** переміщується з поточного каталогу в каталог **props**.

Якщо при переміщенні файлу ви хочете перейменувати його, вкажіть нове ім'я файлу після імені каталогу:

```
$ mv newprop props/version1
```

Припустимо зараз, що ви зробили робочим каталогом дочірній і хочете перемістити файл з дочірнього каталогу в батьківський:

```
$ mv version1 ..
```

Система Linux дозволяє копіювати і переміщувати цілі каталоги. Для копіювання каталогу команду **cp** необхідно використати з опцією **-r**:

```
$ cp -r letters/thankyou oldletters
```

Каталог **thankyou** копіюється в каталог **oldletters**. Після завершення цієї операції починають рівноправно співіснувати два підкаталогу **thankyou**: один в каталозі **letters**, інший в **oldletters**.

За допомогою команди **ln** файлам можна присвоювати додаткові імена. Це необхідно для того, щоб мати можливість звертатися до одного й того ж файлу по різним іменам з різних каталогів. Додаткові імена часто називають посиланнями або зв'язками. Формат команди:

```
$ ln <вихідне_ім'я_файла> <додаткове_ім'я_файла>
$ ln today weather
$ ls
today weather
$ ls -l today weather
-rw-rw-r-- 2 chris group 563  Feb 14 10:30 today
-rw-rw-r-- 2 chris group 563  Feb 14 10:30 weather
```

Слід звернути увагу: кількість посилань (друга колонка лістинга) у обох файлів дорівнює двом. Більш того, співпадають їх розмір і дата створення. Це вказує на те, що ці файли – просто різні імена одного і того ж файлу.

Кожен файл в Linux-системі має свого володаря, групу користувачів, яка може бути наділена особливими привілеями. Хазяїн файлу визначає права доступу до свого файлу. Існує три категорії користувачів, яким можуть бути надані ті або інші привілеї (права доступу) до файлу (каталогу, пристрою та ін.), а саме:

- хазяїн (**user**);
- група користувачів (**group**);
- всі інші користувачі (**other**).

Зміна володаря файла здійснюється за допомогою команди **chown**; зміна групи користувачів, що асоціюється з даним файлом, здійснюється по команді **chgrp**. Командою зміни прав доступу до файлу є **chmod**:

```
$ chmod <права_доступа> <файл(и)>
```

Параметр <права_доступа> є тризначним представленням нових прав доступу до файлу. Перша цифра відповідає за привілеї хазяїна, друга – за привілеї групи користувачів, третя визначає права доступу для всіх інших. Візьмемо, наприклад файл **weather**:

```
-rw-rw-r-- 2 chris group 563 Feb 14 10:30 weather
```

Перша колонка – це права доступу до файлу. Якщо всі букви в правах доступу замінити на "**1**", а всі прочерки на "**0**", то буде **110110100** (перший прочерк говорить про те, що даний файл не є каталогом). Перетворивши кожну трійку двійкових цифр в вісімковий код, отримаємо **664**. Зробимо цей файл доступним для виконання (для хазяїна). Для цього потрібно включити третій біт в правах доступу, тобто провести наступні перетворення:

```
110 110 100  ->  111 110 100
664      ->  764
```

Цьому відповідає команда:

```
$ chmod 764 weather
$ ls -l weather
-rwxrw-r-- 2 chris group 563 Feb 14 10:30 weather
```

Окрім цього способу, права доступу можна включати/виключати за допомогою ключів **+/-r, +/-w, +/-x** (**r** - читання, **w** - зміна, **x** - виконання).

Для ведення протоколу роботи у системі використовуються команди **script** та **history**. Вони записують усі дії користувача та стан екрану у файл до того моменту, як буде введена команда exit.

5.2 Завдання до роботи

5.2.1 Ознайомитися з теоретичними відомостями про базові операції і можливості ОС Linux:

– командний рядок;

– стандартний ввід, вивід і переадресація потоків;

– програмні канали;

– файлова система ОС Linux.

5.2.2 Використовуючи Додаток Б і методичні вказівки до роботи, виконати наступні дії:

а) у домашньому каталозі створити текстовий файл;

б) створити підкаталог і перемістити цей файл в нього;

в) обмежити доступ до нового каталогу;

д) створити підкаталоги з двома-трьома рівнями вкладеності та придбати навички переходу із одного каталогу до іншого;

ж) створити нові файли у деякому або декількох підкаталогах різноманітними вищезазначеними методами або іншими;

к) змінити атрибути створених файлів (ім'я, дату, привілеї) різними методами;

л) використовуючи програмні канали, відсортувати рядки файлу (**sort**), результат помістити у новий файл;

м) об'єднати два файли в один і створити посилання на цей файл у цьому ж каталозі;

н) вивчити команди пошуку даних **find**, **grep** та інші, а також продемонструвати роботу команд із різними опціями на створених файлах;

п) виконати команди одиночного та групового копіювання, зміни ім'я та переміщення.

5.2.3 Скласти звіт про зроблену роботу (при формуванні звіту можна скористатися командами **script** та **history**). Звіт повинен містити:

а) мета роботи;

б) роздруківка станів екрану (протокол роботи).

5.3 Домашнє завдання

5.3.1 Освоїти інтерфейс командного рядка.

5.3.2 На практиці навчитися застосовувати спеціальні символи з елементарними командами (див. Додаток Б).

5.3.3 Освоїти оператори переадресації стандартного вводу і виводу.

5.3.4 Придбати навички керування задачами (призупинка, відміна, запуск на фоні, повернення в пріоритетний режим).

5.3.5 Освоїти основні команди керування каталогами і файлами:

а) створення;

б) видалення;

в) переміщення;

д) зміна прав доступу.

5.4 Контрольні питання

5.4.1 Формат даних, що вводяться з командного рядка.

5.4.2 Універсальні символи.

5.4.3 Потоки даних. Стандартні потоки даних. Переадресація потоків.

5.4.4 Канали і конвеєри.

5.4.5 Задачі в ОС Linux. Фоновий і пріоритетний режими. Команди керування задачами.

5.4.6 Процеси та керування ними.

5.4.7 Файлова концепція Linux-системи. В чому її особливість, переваги і недоліки?

5.4.8 Скільки імен може бути у файлу? Файлові посилання.

5.4.9 Права доступу до файлів. Хто і як їх може змінити?

6 ЛАБОРАТОРНА РОБОТА №6

Командний інтерпретатор bash. Основи написання сценаріїв (скриптів)

Мета роботи: Освоїти командний інтерпретатор bash. Набути початкових навичок написання командних файлів (скриптів).

6.1 Стислі теоретичні відомості

Командний інтерпретатор (оболонка, shell) є інтерфейсом користувача в UNIX-системі. Командний інтерпретатор - це просто програма, яка дозволяє системі розуміти команди користувача (звідси назва), і дає йому можливість створювати зручне для себе середовище роботи в UNIX. Як правило, дії інтерпретатора не помітні користувачеві, вони відбуваються як наче за кулісами.

Командний інтерпретатор можна розглядати як захисну оболонку ядра системи. Як відомо, при запуску системи ядро завантажується в пам'ять і виконує багато низькорівневих системних функцій. Ядро регулює роботу процесора, здійснює і регулює протікання процесів, відповідає за ввід/вивід даних. Можливе існування тільки одного ядра. Інструкції ядра складні, громіздкі, і сильно прив'язані до апаратних засобів. Працювати на мові такого рівня дуже важко, тому і виникло багато командних інтерпретаторів (оболонок). Вони захищають користувача від складності ядра, а ядро - від некомпетентності користувача. Користувач дає команди інтерпретатору, той в свою чергу перекладає їх на системну мову і передає ядру.

Функціями якого завгодно інтерпретатора є:

- інтерпретація командного рядка;
- ініціалізація програм;
- перенаправлення потоків вводу/виводу;
- організація конвеєрного виконання програм (каналів);
- підстановка імен файлів;
- робота зі змінними;
- контроль за середовищем;
- надання засобів керування задачами;
- програмування.

В оболонці визначаються змінні, які керують поведінкою Вашої сесії роботи з UNIX. Вони повідомляють системі, який каталог вважати Вашим робочим каталогом, в якому файлі зберігати вхідну електронну пошту та ін. Деякі змінні попередньо установлюються операційною системою, інші можна визначити самому в файлах початкового завантаження.

В командних інтерпретаторах передбачені спеціальні вбудовані команди, які можуть використовуватися для побудови командних файлів. Командний файл (скрипт, сценарій) - це текстовий файл, що містить UNIX-команди (аналог bat-файлів в DOS).

Звичайно з тою, або іншою ОС поставляються кілька оболонок. Як правило, програмісти працюють в одній оболонці, більш гнучкою і зручною для користування (наприклад csh або bash), а командні файли пишуть для іншої, більш простої (такої як Bourne). Оболонка, яка буде використовуватися по замовченню при реєстрації, та інша особиста інформація визначається в файлі /etc/passwd для кожного користувача окремо. Користувач може в який завгодно момент змінити вибір оболонки, що використовується при реєстрації (команда chsh). Основними оболонками різних версій UNIX є:

Bourne Shell	sh
Korn Shell	ksh
C Shell	csh
Bourne Again Shell	bash
Public Domain Korn Shell	pdksh
A Shell	ash
Tcl Shell	tclsh
X-Windows Shell	wish
Remote Shell	rsh

Деякі оболонки можуть бути присутніми в одній версії ОС і не бути представлені в іншій. Іноді оболонки сполучаються. Так, в деяких версіях Linux sh і bash - це одна і та ж програма (sh і bash є посиланнями на один і той ж файл). Незалежно від того, який командний інтерпретатор використовується, його основною задачею є надання інтерфейсу для користувача.

6.1.1 Командний інтерпретатор bash (bash)

Bash shell - один з найбільш популярних командних інтерпретаторів в системі Linux. Ця оболонка була розроблена на основі оболонок Bourne Shell і C Shell з додаванням функцій, що помітно полегшують роботу з Linux. Виклик оболонки відбувається по команді bash (файл /bin/bash або /usr/bin/bash). Показником того, що ви знаходитесь в bash, є системний запит "$".

6.1.2 Стандартні командні файли

Кожен раз при реєстрації користувача в системі виконується командний файл .bash_profile (аналог autoexec.bat і config.sys в DOS), який знаходиться в його домашньому каталозі. Файл .bash_profile являє собою файл ініціалізації командного інтерпретатора bash. Він виконується автоматично при кожному завантаженні оболонки. Даний файл містить команди, які визначають спеціальні змінні середовища, як системні, так і користувачеві. Розглянемо приклад стандартного файлу .bash_profile:

```
# .bash_profile                                    (1)
# Get the aliases and functions                    (2)
if [ -f ~/.bashrc ]; then                          (3)
        . ~/.bashrc                                (4)
fi                                                 (5)
# User specific environment and startup programs   (6)
PATH=$PATH:$HOME/bin                               (7)
ENV=$HOME/.bashrc                                  (8)
USERNAME=""                                        (9)
export USERNAME ENV PATH                           (10)
```

Рядки, що починаються зі знака # (1, 2, 6), трактуються як коментарі. Їх вміст, як правило, ігнорується.

В рядку 3 перевіряється, чи існує в домашньому каталозі користувача файл .bashrc. Якщо такий присутній, то виконується гілка умови then (рядок 4) - файл .bashrc запускається на виконання. Рядок 5 - кінець конструкції if-then.

Рядки 7-9 визначають змінні середовища. Зверніть увагу на те, що змінні PATH і HOME - це системні змінні, які система попередньо визначила сама. В даному файлі значення змінної PATH модифікується (розширюється). Спеціальні змінні та режими оболонки bash наведені в Додатку В.

Спеціальні змінні, крім усього іншого, потрібно експортувати з допомогою команди export. Це робиться для того, щоб вони стали доступними для всіх можливих вторинних оболонок. Однією командою export можна експортувати кілька змінних, перерахувавши їх в командному рядку як аргументи (рядок 10).

Ще одним файлом ініціалізації оболонки bash є .bashrc. Він виконується кожен раз, коли користувач входить в оболонку. Цей файл також запускається кожен раз при запуску якого завгодно командного файлу (скрипта). В .bashrc звичайно містяться визначення псевдонімів і змінних, які служать для включення тих чи інших функцій командного інтерпретатора. Приклад файлу .bashrc наведено нижче:

```
# .bashrc                                          (1)
# User specific aliases and functions              (2)
# Source global definitions                        (3)
if [ -f /etc/bashrc ]; then                        (4)
     . /etc/bashrc                                 (5)
fi                                                 (6)
set -o noclobber                                   (7)
alias l='/bin/ls -al'                              (8)
```

Як і в попередньому прикладі, в файлі використовується конструкція if-then (рядка 4-5). В ній наведене посилання на стандартний файл завантаження, єдиний для всіх користувачів системи - /etc/bashrc.

В рядку 7 установлюється режим noclobber. Він охороняє існуючі файли від запису поверх них переадресованої вхідної інформації. Можливі ситуації, коли в якості імені файлу, в який переадресується вивід, користувач може випадково вказати ім'я існуючого файлу. Якщо режим noclobber установлено, то при переадресації вхідної інформації в уже існуючий файл цей файл не буде замінено стандартнім вихідним потоком. Файл-оригінал збережеться.

Іноді при роботі доводиться часто використовувати одну і ту ж команду або послідовність команд. Цю проблему можна вирішити, створивши командний файл і вказавши каталог, в якому він знаходиться, в змінній оточення PATH. Однак не завжди раціонально зберігати окремий скрипт для кожної команди, що часто застосовується (надто багато файлів малої довжини). Для цього існують псевдоніми (aliases). Наприклад, однією з найчастіше

використовуваних команд є ls, але формат вихідних даних незручний (якщо команда запущена без опцій). Набагато зручніше і наочніше виглядає результат роботи команди /bin/ls -al. Але це неможлива розкіш - вводити такий довгий рядок кожен раз при необхідності довгого лістингу каталогу. В рядку 8 ми створюємо псевдонім. Псевдонім працює як макрос, який перетворює його в команду.

Крім файлів .bash_profile і .bashrc в домашньому каталозі користувача, як правило, ведеться протокол команд, введених користувачем раніше (.bash_history), і скрипт виходу з системи (.bash_logout).

6.1.3 Робота командного інтерпретатора в інтерактивному режимі

Коли користувач вводить команду на місці запиту ($), він передає її на обробку командному інтерпретатору. Інтерпретатор сприймає рядок команди як послідовність символів, в кінці якої знаходиться "повернення каретки" (Enter). Оболонка сприймає кілька типів команд: команди Linux системи, вбудовані команди інтерпретатора, команди, визначені користувачем, і команди-псевдоніми.

На свій розсуд, користувач може вводити команди по черзі, за принципом "один рядок - одна команда". Однак оболонка не накладає в цьому плані жодних обмежень. Дозволяється вводити по кілька команд в одному рядку, розділяючи їх крапкою з комою. Можливий випадок, коли команда не вміщується на один рядок - тоді можна сховати "повернення каретки" від оболонки, поставивши перед ним зворотну риску "\", і продовжувати ввід команди на наступному рядку. Таким чином, всі нижче приведені команди приведуть до однакових результатів:

```
1)
$ who; ps; echo JUNK MESSAGE
 ...                    (результат роботи who)
 ...                    (результат роботи ps)
JUNK MESSAGE            (результат роботи echo)
2)
$ who
...                     (результат роботи who)
$ ps
...                     (результат роботи ps)
$ echo JUNK MESSAGE
JUNK MESSAGE            (результат роботи echo)
```

```
3)
$ who; ps; echo JUNK \
>MESSAGE
...                          (результат роботи who)
...                          (результат роботи ps)
JUNK MESSAGE                 (результат роботи echo)
```

Команда (або група команд), поміщена в дужки, виконується у вторинній оболонці (підоболонці). Підоболонка - це нова оболонка, що викликається поверх старої (первинної) оболонки. При цьому виконавчі команди вносять зміни тільки в цю підоболонку, не змінюючи параметрів первинного командного інтерпретатора (змінних, поточного каталогу і та ін.). Порівняйте:

```
[stud@localhost stud]$ cd /home
[stud@localhost home]$
                  ^^^^
i
[stud@localhost stud]$ (cd /home)
[stud@localhost stud]$
                  ^^^^
```

В другому випадку команда зміни поточного каталогу (cd /home) виконувалась в підоболонці, тобто поточний каталог змінився тільки для цієї підоболонки. Після виконання команди існування підоболонки закінчилось, і керування перейшло назад в первинну оболонку. Поточний каталог первинної оболонки залишився старим.

Іноді необхідно, щоб вихідні дані однієї команди слугували параметром (але не вхідним потоком!) для іншої. Для цього команду поміщають в зворотні лапки і ставлять на місці параметрів для зовнішньої команди. Наприклад:

```
$ elm `whoami`
```

Команда whoami повертає ім'я, під яким користувач зареєструвався в системі. Це ім'я підставляється в командний рядок в якості параметра для команди elm (посилка поштового повідомлення). Таким чином користувач посилає самому собі e-mail.

Оболонка bash веде історію введених з консолі команд. Проглянути її можна по команді history. Крім того, введені команди можна використовувати повторно. Найпростішими прикладами використання введених раніше команд є !! та !n.

!! остання введена з консолі команда (рядок)

!n n-а команда історії

!-n n-а команда історії, взятої в зворотному порядку (!-1 еквівалентно !!)

!str найостанніша команда з історії, що починається рядком "str"

Вихід з оболонки здійснюється по команді exit [expr]. Ця команда забезпечує вихід з поточної оболонки (командного інтерпретатора) з кодом expr. Вихід з оболонки також здійснюється при досягненні символу "кінець файлу" (Ctrl-D).

6.1.4 Командний інтерпретатор як процес

Спробуємо розглянути командний інтерпретатор більш формально. Будучи звичайною виконуваною програмою, оболонка є процесом. Як і кожен процес в системі UNIX, командний інтерпретатор має унікальний номер процесу, свої вхідні і вихідні потоки даних і всі інші атрибути процесу. Коли інтерпретатор виконується в інтерактивному режимі, вхідний і вихідний потоки асоційовані з терміналом - користувач вводить команди з клавіатури, результат виводиться на екран. Наприклад, в деякому каталозі знаходиться файл script з таким вмістом.

```
ps
echo end of script
```

Нагадаємо, що команда echo виводить повідомлення в стандартний вихідний файл (потік). Команда ps друкує в вихідний файл інформацію про процеси, які запустив користувач. Запустимо скрипт на виконання: по команді . (крапка) виконується командний файл, що передається як параметр.

```
[stud@localhost stud]$ . script
  PID TTY STAT TIME COMMAND
  269   1 S    0:00 /bin/login -- stud
  270   1 S    0:00 -bash
  360   1 R    0:00 ps
end of script
[stud@localhost stud]$
```

Як бачимо з результатів, на момент виконання команди ps з командного файлу script, в системі виконувалось одночасно три процеси, якими володіє користувач stud. Перший (під номером 269) - це процес підключення до системи (реєстрація). Другий (270) - це командний інтерпретатор. Третій (360) - це безпосередньо команда ps.

Скористуємося механізмом перенаправлення потоків:

```
[stud@localhost stud]$ bash < script > outfile
[stud@localhost stud]$ cat outfile
  PID TTY STAT TIME COMMAND
  269   1 S    0:00 /bin/login -- stud
  270   1 S    0:00 -bash
  361   1 S    0:00 bash
  362   1 R    0:00 ps
end of script
[stud@localhost stud]$
```

Опишемо ситуацію, що відбулася. Користувач stud, знаходячись в оболонці bash (процес 270), запускає нову оболонку bash (процес 361), вторинну по відношенню до першої (первинна помічена дефісом). Вхідним потоком нової оболонки є не термінал, а файл. Результат роботи перенаправлюється в інший файл. Команди файлу script, зокрема, команда ps (процес 362), виконуються в новій оболонці. Результат був би аналогічний, якщо б користувач просто викликав підоболонку, а потім послідовно ввів з клавіатури команди ps, echo і символ кінця файлу (Ctrl-D).

Вхідний потік для командного інтерпретатора являє собою послідовність лексем. Основними синтаксичними елементами (лексемами) вважаються.

1. Коментарі. Коментар починається з символу # і продовжується до кінця рядка. Для того, щоб запобігти інтерпретації знака # як початку коментарю, необхідно помістити його в лапки, або поставити перед ним зворотну похилу риску "\".

2. Пропускові символи. Під пропусками розуміють символи "пропуск" (#20h), "tab", "повернення каретки". Пропуски використовуються для відокремлення окремих слів в рядку.

3. Відокремлювачі між висловлюваннями. До таких відокремлювачів відносяться крапка з комою (;) і повернення каретки. Кілька команд можуть бути введені з одного рядка, відокремлені крапками с комою - це еквівалентно вводу кожної команди з окремого рядка. Деякі команди вимагають кілька рядків вводу (if або while).

4. Оператори. Оператор – це спеціальний символ або послідовність символів, за якою оболонка закріплює окремий синтаксичний зміст. Знаки пунктуації, що мають значення для оболонки, повинні бути сховані від неї в лапках, щоб не привести до їх невірного тлумачення.

5. Слова. Словом будемо називати яку завгодно послідовність символів, заключених між пропусковими символами, відокремлювачами і операторами. Словом може бути група послідовних символів, рядок в лапках, посилання на змінну, маска файлу, заміщена команда та ін. Словом може бути комбінація всього вищезазначеного. Кінцеве значення слова - це результат виконання всіх підстановок і замін, який разом зі звичайними символами формує рядок. Цей рядок інтерпретується оболонкою як команда і список параметрів, що передаються.

6.1.5 Шаблони і підстановки

В кожному командному інтерпретаторі реалізовано механізм підстановки імен файлів. При цьому використовуються наступні конструкції:

* яка завгодно послідовність символів, включаючи порожню;

? який завгодно символ. Кілька знаків питання означають яку завгодно послідовність символів заданої довжини;

[] який завгодно символ з списку;

[^] все що завгодно, крім символів списку;

{} кожен елемент списку. При цьому не відбувається перевірка існування файлу (каталогу), а виконується безумовна підстановка, причому стільки разів, скільки елементів в списку;

~ домашній каталог.

Наприклад, нехай деякий каталог містить такі файли:

aaa, bbb, abc, cba, cccc

Наведемо приклади підстановки імен файлів в командний рядок:

Мета-послідовність	Результат підстановки
*	aaa abc bbb cba cccc
??a	aaa cba
*[b,c]	abc bbb cccc
[a-z]?a	aaa cba
*[^b,c]	aaa cba
{a,b,c}bb	abb bbb cbb

(при цьому не перевіряється, чи існують ці файли в дійсності)

6.1.6 Спеціальні символи (метасимволи)

Багато знаків пунктуації інтерпретуються оболонкою як службові. До них відносяться:

~ ` ! @ # $ % ^ & * ( ) \ | { } [ ] ; ' " < > ?

Для того, щоб не дати інтерпретатору обробляти метасимволи по-своєму, необхідно перед ними ставити зворотну косу риску "\", або поміщати необхідну лексему в прямі одинарні або подвійні лапки. Зворотна коса риска "ховає" від оболонки значущу характеристику наступного символу і заставляє обробляти його як простий символ ASCII. Дія подвійних і одинарних лапок практично однакова, але подвійні лапки допускають дію деяких спеціальних символів. Наприклад:

```
$ touch a\ strange\ file
```

В результаті цієї команді в поточному каталозі буде створено файл, в імені якого будуть присутніми два пропуски - "a strange file".

Необхідно розрізняти метасимволи в шаблонах команд (що передаються як аргументи) і метасимволи підстановки імен файлів. Вводячи команду з термінала, не забувайте, що спочатку в неї "загляне" командний інтерпретатор, а лиш потім - програма. Тому потрібно стежити, щоб оболонка не перехопила спеціальні символи, їй не назначені, і не проводила підстановку імен файлів. Яскравим прикладом може служити програма grep - пошук в файлах по зразку.

Якщо з термінала ввести

```
$ grep [A-Z]* chap[12]
```

то інтерпретатор підставить в командний рядок всі імена файлів, що відповідають шаблону. В результаті підстановки може статися так:

```
$ grep Array.c Bug.c Comp.c README chap1 chap2
```

Таким чином, утиліта grep буде виконувати пошук рядка "Array.c" в файлах Bug.c, Comp.c, README, chap1, chap2. Для того щоб передати команді grep метасимволи, застосовуються лапки:

```
$ grep "[A-Z]*" chap[12]
```

При цьому буде виконуватися пошук послідовності з нуля і більш великих латинських букв в файлах chap1 chap2.

6.1.7 Програмування в bash

Як уже зазначалось, командні інтерпретатори володіють деякими властивостями мов програмування, які дозволяють створювати досить складні програми. Така програма об'єднує

звичайно ряд команд UNIX, направлених на виконання конкретної задачі. Інструкції, з яких складається програма, вводяться в командний файл (скрипт, сценарій), який підлягає виконанню. Створити такий файл можна з допомогою звичайного текстового редактора.

6.1.8 Оператор "документ тут"

Оператор "документ тут" (<<) дозволяє використовувати рядки командного файлу в якості вхідних даних (вхідного потоку) для якої-небудь команди. Цим усувається необхідність читати вхідні дані з зовнішнього джерела, наприклад з файлу або каналу. Після оператора << на тому ж рядку слід помістити обмежувач потоку, а з наступного рядка - самі дані для вхідного потоку. Наприклад:

```
$ cat << zzz
>   This line will be printed
>zzz
   This line will be printed
$
```

6.1.9 Виконання наступної команди за умовою

Іноді в процесі роботи необхідно виконувати умовне розгалуження програми, або послідовності дій. В оболонці Сі подвійний амперсанд "&&" еквівалентний логічному "І". Команда, що стоїть після "&&", буде виконуватися тільки в тому випадку, якщо попередня команда завершилась успішно (код виходу 0). Наприклад:

```
% cp hello.c hello.bak && rm hello.c
```

Файл hello.c не буде видалено, якщо сталася помилка при копіюванні, тобто якщо команда cp завершилась невдало.

"||" - еквівалент логічного "АБО". Команда, що стоїть після "||", буде виконуватися тільки в тому випадку, якщо попередня команда завершилась невдало (код виходу відрізняється від 0). Наприклад:

```
% cp hello.c hello.bak || echo Copy file error
```

Якщо пройшла помилка при копіюванні, то буде надруковано повідомлення Copy file error.

6.1.10 Заміна оболонки новою програмою - команда exec

Команда exec замінює поточний процес (командний файл, що виконується, або оболонку) новою задачею, ім'я якої передається в якості параметра. Команда часто використовується при написанні скриптів для організації процесів, що повторюються.

6.1.11 Визначення і розрахунок змінних

Командний інтерпретатор bash володіє можливостями роботи зі змінними. Імена змінних не обмежені по довжині і можуть містити великі і малі букви латинського алфавіту, цифри і знак підкреслювання. Ім'я змінної не може починатися с цифри.

Значення змінної присвоюється за допомогою оператора присвоювання (=). Оператор присвоювання пропусками не відділяється. Змінній може бути присвоєна яка завгодно сукупність символів. Наприклад:

```
$ greeting="How do you do?"
```

Значення змінних часто використовуються як аргументи команд. Розрахунок (підстановка) значення змінної відбувається за допомогою оператора $. Результатом розрахунку є набір символів. Цей набір замінює ім'я змінної в командному рядку. Наприклад:

```
$ echo $greeting
How do you do?
```

Список всіх визначених змінних можна отримати по команді set. Якщо яка-небудь змінна більше не потрібна, її можна видалити командою unset.

6.1.12 Ввід і вивід даних в сценаріях

Для виводу даних в сценарії можна використовувати команду echo, а для зчитування вхідної інформації в змінні - команду read. Фактично команда read читає рядок зі стандартного вводу. Все, що посилається на стандартний ввід з клавіатури (файлу або каналу при переадресації) - аж до символу нового рядка - зчитується і присвоюється в якості значення змінної. Наприклад:

```
$ read greeting
How do you do
$ echo $greeting
How do you do
```

Крім того, за допомогою конструкції "документ тут" в сценарій можна вводити дані і переадресовувати їх в команду.

6.1.13 Аргументи командного рядка

Аналогічно тому, як це робиться в командах UNIX, в скриптах можна використовувати аргументи. Аргументи вводяться при виклику командного файла після його імені і нумеруються, починаючи з 1. Перший аргумент програми, що оброблюється, позначається $1, другий - $2, і т.ін. Аргумент $0 - це ім'я програми, що виконується на даний момент (назва оболонки, або командного файлу) - фактично перше слово в командному рядку. Аргументи можуть розглядатися як локальні змінні процесу. Існують і інші спеціальні змінні, пов'язані з характеристиками командного рядка і процесу (див. Додаток Г).

6.1.14 Арифметичні операції

В оболонці bash присутня команда let, яка дозволяє виконувати операції з арифметичними величинами. За допомогою цієї команди можна порівнювати числа і виконувати з ними такі операції, як додавання або множення. Команда let може замінюватися подвійними круглими дужками. Оператори let, що співпадають зі спеціальними символами оболонки, брати в лапки не потрібно. Якщо операнди арифметичного виразу розділені пропусками, цей вираз слід взяти в лапки. Наприклад:

```
$ let "res = 2 * 7"
$ echo $res
14
$
```

6.1.15 Команда порівняння test

Часто буває необхідно виконати перевірку, в ході якої порівнюються дві величини або перевіряється наявність того або іншого файлу. За допомогою команди test можна порівнювати цілі числа, рядки, і виконувати логічні операції. Результат перевірки - це код завершення операції test, який, як було сказано раніше, зберігається в спеціальній змінній $?. Замість ключового слова test можна використовувати квадратні дужки. Наприклад:

```
$ greeting="hallo"
$ num=5
```

```
$ test $num -eq 5 ; echo $?
0
$ [ $greeting = "hallo" ] ; echo $?
0
$ test -f main.c ; echo $?
1
```

В даному прикладі ми ініціалізуємо дві змінні. Потім виконуємо перевірки на рівність їх тому або іншому значенню (зверніть увагу, що рядки порівнюються за допомогою оператора =, а численні значення - за допомогою опції -eq). Остання перевірка - це перевірка наявності файлу main.c в поточному каталозі. Результат – неправда (файл не знайдено). Операції команди test наведені у Додатку Д.

6.1.16 Умови

В оболонці bash є набір умовних керуючих структур, які забезпечують умовне розгалуження програм. Багато з цих структур аналогічні умовним керуючим структурам мов програмування, але є деяка різниця.

Конструкція if ставить умову для виконання команди. Цією умовою є код завершення якоїсь конкретної команди. Якщо команда виконана успішно (код завершення дорівнює нулю), то команди всередині структури if виконуються. В іншому випадку виконується гілка else (якщо вона присутня) або керування передається наступному за if-конструкцією оператору. Керуюча структура if повинна закриватися ключовим словом fi (if навпаки). Синтаксис if-конструкції такий:

```
if <команда-умова>
     then
          <команди>
     else
          <команди>
fi
```

В якості команди-умови як правило використовується команда test або її альтернативна форма - квадратні дужки [].

Вкладення умов if здійснюється за допомогою структури elif. Вкладеність elif не обмежується. Остання гілка каскаду elif повинна починатися зі слова else:

```
if <команда-умова>
     then
          <команди>
     elif
```

```
            <команди>
      else
            <команди>
fi
```

Керуюча структура case забезпечує вибір одного з кількох можливих варіантів. Вибір здійснюється шляхом порівняння заданого в структурі значення з кількома можливими зразками. Кожне можливе значення змінної, що перевіряється (зразок) пов'язується з сукупністю операцій. Кожен зразок являє собою регулярний вираз, що завершується круглою дужкою. Список команд, що виконуються завершується двома крапками з комами, що стоять на окремому рядку. Вся конструкція завершується ключовим словом esac (case навпаки). Синтаксис структури:

```
case <рядок> in
      зразок)
            команди
            ;;
      зразок)
            команди
            ;;
      *)
            команди по замовчуванню
            ;;
esac
```

Зразок може містити спеціальні символи: *, [], ?, |. Варіант по замовчуванню включати до структури не обов'язково.

6.1.17 Цикли

Командний інтерпретатор дозволяє створювати гнучкі циклічні структури. Конструкція while забезпечує виконання окремої команди, доки виконується умова (код виконання команди-умови дорівнює нулю). Замикає конструкцію ключове слово done:

```
while <команда-умова>
      do
            команди
      done
```

Як і в if-структурі, команда-умова частіше всього представляється перевіркою test (або квадратними дужками).

Цикл until аналогічний циклу while. Він відрізняється тим, що команди тіла циклу виконуються до того часу, доки умова залишається НЕ виконаною:

```
until <команда-умова>
     do
          команди
     done
```

Структура for-in призначена для послідовного звернення до значень, перерахованих у списку. В ній два операнди - змінна і список значень. Кожне зі значень по черзі присвоюється змінній структури. Цикл завершується, коли всі значення зі списку будуть вичерпані. Тіло циклу поміщується в операторні дужки do-done, синтаксис конструкції for-in:

```
for <змінна> in <список_значень>
     do
          команди
     done
```

Структура for без явно заданого списку значень використовує в якості такого аргументи командного рядка. При першому проході змінній присвоюється значення першого аргументу командного рядка, при другому - значення другого, та ін.

6.1.18 Приклади скриптів

Розглянемо простий інтерактивний скрипт whoareyou. Нижче наводиться його текст:

```
echo What is your name?                                  (1)
read reply                                               (2)
case $reply in                                           (3)
  root)                                                  (4)
    echo You are system administrator of $HOSTNAME       (5)
    ;;                                                   (6)
  stud)                                                  (7)
    echo You are a student                               (8)
    ;;                                                   (9)
  *)                                                     (10)
    echo You are a mortal user on $HOSTNAME              (11)
    ;;                                                   (12)
esac                                                     (13)
```

Скрипт пропонує користувачеві відрекомендуватися (рядок 1). Користувач вводить своє ім'я, яке заноситься в змінну reply. Далі відбувається розгалуження в залежності від значення reply (іншими словами, в залежності від відповіді користувача). Скрипт друкує одне з трьох повідомлень. При друку використовується значення іншої змінної - HOSTNAME. Вона визначається безпосередньо системою і містить мережеве ім'я UNIX-машини, на котрій виконується скрипт.

Розглянемо інший приклад. Іноді, при великій кількості файлів, буває важко за ними услідити. Наведена нижче програма chkown дозволяє знаходити файли, які не належать користувачеві.

```
for filename in `ls`                                    (1)
do                                                      (2)
   if [ ! -O $filename ]; then                          (3)
      echo $filename does not belong to `whoami`        (4)
   fi                                                   (5)

   if [ -d $filename -a -x $filename ]; then             (6)
      echo Entering $filename ...                       (7)
      cd $filename                                      (8)
      $0                                                (9)
      Echo Leaving $filename ...                        (10)
      cd ..                                             (11)
   fi                                                   (12)
done                                                    (13)
```

Розглянемо дію скрипта порядково.

В першому рядку ініціалізується for-цикл. Змінною циклу є filename. Список значень визначається результатом роботи команди ls (вона поміщена в зворотні лапки). Іншими словами, змінна filename по черзі приймає значення імен файлів і каталогів поточної директорії.

В рядку 3 відбувається перевірка, чи належить файл або каталог користувачеві, який запустив скрипт (операція -O команди test). Зауважимо, що якщо ключове слово then стоїть на тому ж рядку, що і умова, воно повинно відокремлюватися крапкою з комою (;). Якщо користувач не є хазяїном файлу (знак оклику - логічне заперечення), то друкується повідомлення (рядок 4). При виконанні рядка 4 замість `whoami` в команду echo підставляється ім'я користувача (зворотні лапки).

В рядку 6 відбувається подвійна перевірка: чи є значення змінної filename каталогом (операція -d), і чи є значення змінної filename файлом, виконавчим для користувача (-x). Обидва логічних результата перемножуються (операція -a відповідає логічному "І") і, згідно кінцевому результату, гілка then виконується або не виконується. Як відомо, права на виконання каталогу визначають можливість користувача увійти в нього (cd).

Якщо змінна filename дійсно містить ім'я доступного каталогу, скрипт входить в нього (рядок 8) і рекурсивно запускає самого себе уже в цьому каталозі (в рядку 9 змінна $0 відповідає імені виконавчої

програми, тобто імені скрипта). Після завершення роботи цього другого скрипта, керування повертається в первинний процес, відбувається перехід в початковий каталог (рядок 11), і на цьому гілка then завершується.

Необхідно відзначити, що якщо шлях до даного скрипта не буде визначено в змінній PATH, то його виконання буде перерване вже на другому рівні, оскільки програма вже покине початковий каталог (каталог першого рівня), в якому знаходиться файл chkown. Команда в рядку 9 не містить абсолютного імені файлу, тому скрипт звернеться до змінної PATH і почне його пошук у всіх визначених там каталогах.

6.2 Завдання до роботи

6.2.1 Ознайомитися с можливостями і принципами роботи командного інтерпретатора bash shell.

6.2.2 Оволодіти початковими навичками написання командних файлів (скриптів).

6.2.3 Отримавши номер варіанту у викладача, написати і відлагодити командний файл згідно завдання:

Варіант 1

Написати скрипт, що посилає всім користувачам, що знаходяться в даний момент в системі, яке-небудь повідомлення (електронною поштою або безпосередньо на екран). Прикладом повідомлення може бути поточна дата і час.

Варіант 2

Написати і відлагодити скрипт, який в домашньому каталозі користувача і в нижчеслідуючих підкаталогах знаходить найдовший файл, а потім визначає його тип.

Варіант 3

Написати скрипт, який в домашньому каталозі і підкаталогах користувача підраховує кількість файлів, що містять тексти вихідних програм на Сі.

Варіант 4

Написати скрипт, який в домашньому каталозі і підкаталогах знаходить вихідні тексти програм на Сі і виводить на екран імена всіх файлів-заголовків (stdio.h, stdlib.h, iostream.h, і т.ін.) що згадуються в них.

Варіант 5

Написати скрипт, який розраховує максимальну глибину дерева каталогів файлової системі.

6.2.4 Скласти звіт про проробленую роботу. Звіт повинен містити тему і мету роботи, тексти вихідних командних файлів, роздруківку повідомлень програми, висновки.

6.3 Домашнє завдання

6.3.1 Використовуючи методичні вказівки і конспект лекцій, ознайомитися з теоретичними відомостями про командні інтерпретатори.

6.3.2 Вивчити призначення, формат і дію команд, аргументів, операторів, спеціальних символів, змінних і конструкцій командного інтерпретатора bash shell.

6.4 Контрольні питання

6.4.1 Командний інтерпретатор, його основні функції.

6.4.2 Стандартні командні файли, їх вміст і послідовність виконання.

6.4.3 Локальні і глобальні змінні. Схожість і різниці змінних і псевдонімів.

6.4.4 Робота інтерпретатора в інтерактивному режимі. Історія (протокол) введених з консолі команд.

6.4.5 Лексеми оболонки.

6.4.6 Шаблони і підстановки. Пріоритети виконання (підстановки) спеціальних символів.

6.4.7 Умовні конструкції bash.

6.4.8 Команда exec. Приклад використання.

6.4.9 Робота з аргументами командного рядка.

6.4.10 Команда test. Її оператори і операнди.

6.4.11 Циклічні конструкції в командних файлах.

7 ЛАБОРАТОРНА РОБОТА №7

Системне адміністрування Linux

Мета роботи: освоєння програмного забезпечення, призначеного для заведення і видалення користувача і групи користувачів, зміни пароля користувача, зміни облікових записів про користувача і групу. Пакет sudo. Монтування файлових систем.

7.1 Стислі теоретичні відомості

Основними задачами системного адміністрування є:
- підключення і видалення користувачів;
- підключення і видалення апаратних засобів;
- резервне копіювання;
- установка нового програмного забезпечення;
- моніторинг системи;
- пошук несправностей;
- ведення локальної документації;
- контроль захисту;
- надання допомоги користувачам.

В даній лабораторній роботі будуть частково розглянуті пункти 1, 2 і 8 даного списку.

7.1.1 Заведення і видалення користувачів

Інформація про всіх користувачів системи Unix зберігається в файлі /etc/passwd. Детально структура цього файлу описана в секції 5 розділу passwd "оперативної інструкції користувача". Заведення нового користувача зводиться до внесення нового запису в цей файл. Однак, ідея самостійного внесення реєстраційного запису в цей файл за допомогою якого-небудь текстового редактора, не дивлячись на досить прозору структуру цього файлу, не є плідною. Не будемо зупинятися на можливості внесення простих синтаксичних помилок (людині властиво помилятися), через які даний обліковий запис буде просто ігноруватися. Також можливо вас не зупинить і те, що ви не бажаючи того порушите логічну цілісність даного файлу, що призведе до дірок в захисті вашої системи. Просто подумайте над тим, що станеться, якщо два адміністратори одночасно почнуть редагувати цей

файл, внесуть зміни і доповнення, але ваш колега збережеться на пару секунд пізніше.

Для заведення нового користувача в Linux призначені наступні утиліти:

- useradd (пакетна утиліта);

- adduser (інтерактивна утиліта).Програма призначена для роботи на алфавітно-цифрових терміналах. В режимі діалогу запитується вся необхідна інформація, після чого викликається утиліта useradd;

- glint (графічна утиліта), аналог User Manager Windows NT. В кінцевому підсумку також звертається до useradd.

Для видалення користувача призначена утиліта userdel. Для заведення і видалення груп користувачів призначені утиліти groupadd і groupdel. Змінити обліковий запис користувача можна утилітою usermod, для групи користувача існує утиліта groupmod. Всі ці утиліти, а також деякі інші, входять в пакет shadow.

7.1.2 Утіліта useradd

Розглянемо детальніше утіліту useradd:

USERADD

Section: Maintenance Commands (8)

NAME

useradd - створення нового користувача або зміна інформації для заведення нового користувача

useradd
[**-c** comment] [**-d** home_dir]
[**-e** expire_date] [**-f** inactive_time]
[**-g** initial_group] [**-G** group[,...]]
[**-m** [**-k** skeleton_dir]] [**-s** shell]
[**-u** uid [**-o**]] login

useradd
-D [**-g** default_group] [**-b** default_home]
[**-f** default_inactive] [**-e** default_expire_date]
[**-s** default_shell]

7.1.3 Заведення нових користувачів

При виклику без ключа -D команда useradd створює обліковий запис нового користувача, використовуючи значення, що визначені в командному рядку і значення по замовчуванню з системи. В залежності від ключів командного рядка при необхідності буде внесено обліковий запис в облікові файли, створено домашній каталог, а також скопійовані ініціалізаційні файли. Ключі, які можуть бути передані команді useradd:

-c comment

Вміст поля коментарю файлу паролів для користувача, що створюється.

-d home_dir

Новий користувач буде створений з використанням home_dir в якості значення домашнього каталогу. По замовчуванню реєстраційне ім'я login додається до default_home і отримане значення використовується як ім'я домашнього каталога.

-e expire_date

Дата блокування користувача. Дата задається в форматі MM/DD/YY.

-f inactive_days

Число днів після спливання строку дії пароля до блокування користувача. 0 блокує користувача зразу ж після спливання строку дії пароля, -1 відключає дану можливість. По замовчуванню використовується значення -1.

-g initial_group

Ім'я або номер початкової групи користувача. Група повинна існувати. Номер групи повинен посилатися на вже існуючу групу. Номер групи по замовчуванню 1.

-G group,[...]

Список додаткових груп, членом яких, також, є користувач. Групи відокремлюються комами, без пропускових символів. На групи накладаються ті ж обмеження, що і на групу, задану ключем -g. По замовчуванню користувач належить тільки до початкової групи.

-m

Створити домашній каталог користувача, якщо він не існує. При заданні ключа -k файли, що знаходяться в каталозі

skeleton_dir, будуть скопійовані в домашній каталог, інакше будуть використані файли з каталогу /etc/skel. Також всі каталоги, що містяться в skeleton_dir або /etc/skel, будуть створені в домашньому каталозі користувача. Ключ -k припустимий лише сумісно з ключем -m. По замовчуванню домашній каталог не створюється і ніякі файли не копіюються.

-s shell

Найменування реєстраційного командного інтерпретатора користувача. По замовчуванню це поле залишається порожнім, що примушує систему вибрати реєстраційний командний інтерпретатор по замовчуванню.

-u uid

Числове значення ідентифікатора користувача. Значення повинно бути унікальним, в випадку якщо не задано ключ -o. Значення повинно бути невід'ємним. По замовчуванню використовується найменший ідентифікатор, більший 99 і більший ніж ідентифікатор якого завгодно іншого користувача. Величини між 0 і 99 звичайно зарезервовані для системних облікових записів.

7.1.4 Зміна значень по замовчуванню

При виклику з ключем -D useradd покаже поточні значення по замовчуванню, або замінить значення по замовчуванню відповідними значеннями з командному рядка. Дозволеними ключами є:

-b default_home

Початкова частина для домашнього каталогу користувача. При створенні нового облікового запису для отримання імені домашнього каталогу користувача ім'я користувача додається в кінець default_home, за виключенням випадку, коли каталог користувача задано ключем -d.

-e default_expire_date

Дата блокування користувача.

-f default_inactive

Число днів після спливання строку дії пароля до блокування користувача.

-g default_group

Ім'я або номер початкової групи користувача. Група повинна існувати. Номер групи повинен посилатися на вже існуючу групу.

-s default_shell

Найменування реєстраційного комп'ютера і адміністратор відповідальний за розміщення користувальницьких файлів по замовчуванню в каталозі /etc/skel. Першими кандидатами для розміщення там є файл .inputrc і каталог .mc з настройками для роботи з кирилицею.

7.1.5 Неприємності

Ви не можете додати користувача в групу NIS. Ця операція повинна проводитись на сервері NIS.

7.1.6 Файли

/etc/passwd - файл облікових записів.
/etc/shadow - файл тіньових паролів.
/etc/group - інформація о групах користувачів.
/etc/default/useradd - інформація по замовчуванню.
/etc/skel - каталог, що містить файли по замовчуванню.

7.1.7 Монтування файлових систем

Файлове дерево формується з окремих частин, званих файловими системами. Для того, щоб зробити файлову систему доступну для процесів Unix, її треба змонтувати. Точкою монтування файлової системи може служити який завгодно каталог. Його файли і каталоги будуть недосяжні, доки файлова система змонтована поверх них. Файлові системи прикріплюються до файлового дерева з допомогою команди mount. Ця команда бере з існуючого файлового дерева каталог, званий точкою монтування, і робить його кореневим каталогом що приєднується до файлової системи. Наприклад команда

$ mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom

монтує пристрій CD-ROM в каталог /mnt/cdrom в режимі "тільки читання".

Демонтуються файлові системи з допомогою команди umount. Демонтувати файлову систему можна лише тоді, коли в ній нема

відкритих файлів і процесів, що використовують цю файлову систему (тобто файлова система не повинна бути зайнятою). Якщо файлова система, що демонтується, містить виконавчі програми, то вони не повинні бути запущені. Для визначення процесів, які використовують файлову систему, під Linux існує програма fuser.

7.1.8 Пакет sudo

На практиці часто виникає ситуація, коли для виконання своїх обов'язків деяким користувачам необхідно надати привілеї, які доступні звичайно тільки root'у. Існує кілька способів доступу до бюджету привілейованого користувача. Найпростіший з них - зареєструватися під іменем root. Але на жаль, вихід з власного бюджету і реєстрація в якості привілейованого користувача часто дуже незручні. Краще використовувати команду su. Будучи викликаною без аргументів, ця команда запросить вас ввести пароль привілейованого користувача, а потім запустить shell з відповідними правами. Привілеї цього інтерпретатора команд залишаються в силі до завершення його роботи.

З привілейованим користувальницьким доступом сполучені три проблеми: безмежні повноваження, відсутність обліку операцій, що виконуються, імовірність того, що під іменем root може працювати група користувачів. Оскільки повноваження привілейованого користувача розподілити не можна, то важко надати комусь можливість зняття резервних копій (що повинно робитися під ім’ям root), не даючи можливості вільної роботи в системі. Якщо ж бюджет root доступний групі користувачів, то ви і поняття не будете мати про того, хто їми користується і що робить.

Щоб вирішити ці проблеми, використовується програма sudo. Ця програма в якості аргументу приймає командний рядок, який підлягає виконанню з правами root. Команда sudo звертається до файлу /etc/sudoers, що містить список користувачів, що мають повноваження на її виконання, і перелік команд, які вони мають право виконувати на конкретній машині. Якщо команда, що пропонується, дозволена, sudo пропонує користувачеві ввести його власний пароль і виконує команду як root.

До спливання п'ятихвилинного періоду бездіяльності sudo можна виконувати інші sudo-команди, не вводячи пароля. Така міра -

захист від тих користувачів з sudo-привілеями, які кидають свої термінали без нагляду.

Файл /etc/sudoers виглядає приблизно так:

```
# Host alias specification Host_Alias
HUB=houdini.rootgroup.com:\
REMOTE=merlin,kodiakthorn,spirit
Host_Alias MACHINES=kalkan,alpo,milkbones
Host_Alias SERVERS=houdini,merlin,kodiakthorn,spirit
# Command alias specification
Cmnd_Alias LPCS=/usr/etc/lpc,/usr/ucb/lprm
Cmnd_Alias SHELLS=/bin/sh,/bin/csh,/bin/tcsh
Cmnd_Alias SHUTDOWN=/etc/halt,/etc/shutdown
# User specification
Britt       REMOTE=SHUTDOWN:ALL=LPCS
Robh        ALL=ALL,!SHELLS
nieusma     SERVERS=SHUTDOWN,/etc/reboot:\,HUB=ALL,!SHELLS
jill        houdini.rootgroup.com=/etc/shutdown,MISC
markm       HUB=ALL,!MISC,!/etc/shutdown,!/etc/halt
billp       ALL=/usr/local/bin/top:MACHINES=SHELLS
davehieb    merlin=ALL:SERVERS=/etc/halt:\
kodiakthorn=ALL
```

В цьому прикладі користувачеві britt дозволено виконувати програми /etc/halt, /etc/shutdown на машинах merlin, kodiakthorn і spirit. robh може виконувати які завгодно програми, крім перерахованих в макрозмінній SHELLS, на всіх машинах.

Що можуть робити користувачі jill, markm, billp і davehieb і на яких машинах, ви розкажете самі, вивчивши документацію по пакету sudo. Також в документації перераховані деякі додаткові можливості, які з'явились в цій програмі, наприклад, можливість виконання програми, що вказана, не від root'a, а від іншого користувача.

Для модифікації файлу /etc/sudoers використовується програма visudo, яка дозволяє редагувати цей файл, перевіряючи синтаксис заданих змін. Зверніть увагу, що всі команди в цьому файлі задаються з абсолютними шляхами, щоб попередити можливість виконання користувальницьких програм з тими ж іменами з правами root.

Крім виконання вказаних команд, sudo веде файл реєстрації виконаних команд, осіб, що їх викликали, каталогів, з яких викликались програми і час їх виклику.

7.1.9 Моніторинг процесів

Іноді виникає ситуація, коли необхідно з під акаунта **root** керувати процесами, запуск яких був зроблен іншими користувачами. Ці процеси навантажують центральний процесор та виникає ситуація припинення даних процесів. Існує кілька способів перегляду інформації щодо процесів. Це наступні команди:

top – запуск інтерактивної програми моніторингу активності процесів системи (аналог диспетчеру задач у Windows);

ps – утілита виводу процесів у системі з фільтрацією за користувачем, терміналом або ін. Відмінність від **top**, це – відображення процесів, які знаходяться у призупиненому стані.

Утілита керуванням процесами є **kill**. Параметрами цієї утілити є тип сигналу та **PID** процесу. Тип сигналу задається або як константа або числовим значенням, яке відповідає відповідній константі. Для виводу списку усіх констант з їх значеннями необхідно виконати команду **kill** з ключем **-l**. Наприклад, для безапеляційної швидкої ліквідації процесу з атрибутом **PID=2344** необхідно виконати команду:

```
kill -9 2344
```

Ключ **9** відповідає константі **SIGKILL**, що означає абсолютне «вбивство» процеса. Існують сигнали для ліквідації процесів з деякими попереджувальними умовами. Комбінація горячих клавіш **Ctrl+C** генерує сигнал з кодом 15, що означає програмне завершення процесу та вигрузку його із пам'яті, а **Ctrl+Z** – генерує сигнал з кодом 20, що означає тільки призупинення дії процесу але залишає його у пам'яті (процеси призупинені даним способом не відображаються у програмі **top**, але видимі у програмі **ps**).

7.2 Завдання до роботи

7.2.1. Вивчити документацію по пакетам shadow, sudo.
7.2.2. Завести групу користувачів stud.
7.2.3. Завести користувача stud, первинною групою якого є stud.
7.2.4. Встановити пароль користувачеві stud.
7.2.5. Додати користувача stud в групу floppy.

7.2.6 Реалізувати на shell сценарій монтування ГМД з файловою системою vfat. За допомогою програми visudo дозволити користувачеві stud виконувати цей сценарій з привілеями користувача root.

7.2.7 Реалізувати на shell сценарій видалення із каталогу /usr/tmp усіх файлів до яких не звертались більше 10 діб. За допомогою програми visudo дозволити користувачеві stud виконувати цей сценарій з привілеями користувача root. Рекурсивно видалити користувача stud (разом з його домашнім каталогом).

7.2.8. Видалити групу stud.

7.2.9. Під одним терміналом зайти як користувач stud та запустити декілька нескінченних процесів типу:

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

Під іншим терміналом зайти під акаунтом адміністратора root, та відстежити запущені процеси. Під stud призупинити деякі з процесів, та відфільтрувати їх під root. Призупинити, програмно завершити та ліквідувати процеси за допомогою різних сигналів.

7.3 Контрольні питання

7.3.1. Для чого необхідно включати користувачів в групу floppy?

7.3.2. Кому будуть належати файли користувача stud після видалення відповідного облікового запису?

7.3.3. Яким чином можна дозволити користувачам монтувати деякі файлові системи без використання пакета sudo?

7.3.4. Які типи файлових систем підтримує система Linux?

7.3.5. Створіть файл a, після чого файли b і z, що є жорсткими посиланнями на файл a. Чому при зміні режиму доступу файлу a змінились режими доступу файлів b і z?

7.3.6 Монтування різних файлових систем та пристроїв.

7.3.7. Яким чином користувач може змінити початковий командний інтерпретатор і поле коментарю свого облікового запису з допомогою системного адміністратора?

7.3.8. Які сигнали можут бути використані щодо припинення процесів командою **`kill`** (призупинення, завершення, ліквідація)?

ПЕРЕЛІК ПОСИЛАНЬ

1. Задерейко О. В. Операційні системи: навчальний посібник / О. В. Задерейко, С. Л. Зіноватна, А. А. Толокнов. – Одеса : Фенікс, 2022. – 140 с.

2. Nemeth, E. UNIX and Linux System Administration Handbook, 5th Edition / Evi Nemeth, Garth Snyder, Trent Hein, Ben Whaley, Dan Mackin. – Addison-Wesley Professional, 2017. – 1232 p.

3. Шеховцов В. А. Операційні системи. – К.: BHV, 2005. – 576с.

4. Stallings W. Operating systems: internals and design principles.-8-th ed.-Upeer Saddle River, New Jersey.: Prentice- Hall, 2015.-800 p.

5. Використання Turbo Assembler при розробці програм. – Київ: «Діалектика», 1994. – 288 с.

Додаток А

Список основних функцій BIOS та DOS

Функція	Призначення
00H	ділення на 0
01H	покрокове виконання (при TF=1)
02H	немаскуюче переривання (вивід NMI процесору)
03H	команда INT без числового параметру
04H	INTO- переривання по переповненню (ініціїрується апаратно при наявності в программі команди INTO)
05H	переривання при натинанні клавіши Print Screen
08H	таймер (апаратне)
09H	клавіатура (апаратне)
0AH	нестандартні прилади (апаратне, зарезервовано)
0BH	другий послідовний порт COM2 (апаратне)
0CH	перший послідовний порт COM1 (апаратне)
0DH	жорсткий диск (для PC,XT); другий паралельний порт LTP2 для AT (апаратне)
0EH	гнучкий диск (апаратне)
0FH	паралельний порт LPT 1(апаратне)
10H	відеоадаптер BIOS
13H	драйвер BIOS диску
14H	драйвер послідовного порту
16H	драйвер BIOS клавіатури
17H	драйвер BIOS принтеру
19H	початковий завантажник BIOS
1AH	календар - годинник BIOS
1BH	обробщик по Ctrl+Break
1CH	програма - заглушка BIOS для обробки переривань від системного таймеру (18,2 переривань за секунду)
1DH	адреса таблиці відеопараметрів, BIOS
1EH	адреса таблиці параметрів дискети, BIOS
1FH	адреса другої половини таблиці шрифтів графічних режимів 4...6, BIOS
21H	диспетчер функцій DOS

22H	адреса переходу при завершенні процесу, DOS
23H	обробщик Ctrl+C
24H	обробщик переривання по критичний помилці
25H	абсолютне читання диску
26H	абсолютне записування на диск
27H	завершення процесу, але програма залишається резидентною
28H	програма - заглушка DOS для активізації резидентних програм командами з клавіатури
2FH	мультіплексне переривання DOS
33H	драйвер мишки фірми Microsoft
43H	адрес таблиці шрифтів графічних режимів, BIOS
60H..66H	переривання користувача
67H	драйвер додатковой пам’яті LIM EMS
68H..6FH	вільні вектори
70H	КМОП - годинник дійсного часу (AT, апаратне)
71H	програма BIOS, збуджуюча переривання INT 0Ah для сумісності XT та AT в частині обслуговування нестандартних зовнішніх приладів (AT, апаратне)
72H..73H	зарезервовано (AT, апаратне)
74H	мишка (PS/2, апаратне)
76H	жорсткий диск (AT, апаратне)
77H	зарезервовано (AT, апаратне)
78H..7FH	вільні вектори
0F1H... 0FFH	не використовуються

Додаток Б

Основні команди UNIX

basemane повертає ім’я файла без шляху
bc калькулятор
cal календар
cat читає один або кілька файлів і послідовно друкує іх на стандартний потік stdout
cd змінює поточний каталог
chgrp **chgrp newgroup files** змінює групу користувачів для файлів files на newgroup (необхідно бути членом цієї групи)
chmod змінює привілеї (права доступу) до файлів
chown **chown newowner files** призначає нового хазяїна newowner файлам files
clear очистка екрана
cmp порівняння двох файлів
cp копіювання файлу(ів)
csh командний інтерпретатор C-shell
cut робить виборку визначених полів або колонок в файлі(ах)
-c виборка за колонками (-c7; -c12-15; -c40)
-f виборка за полями
-d визначення відокремлювача полів
date друк поточної дати
diff порівняння двох файлів
dircmp порівняння вмісту двох каталогів
dirname друкує шлях до файлу, що передається як параметр, але опускає безпосереднє ім’я файлу
echo виводить повідомлення на стандартний вихідний потік
elm обробка електронної пошти
env друкує список змінних оточення і їх значення
expr оцінює значення виразу, що передається в якості параметра
file класифікує файли згідно інформації, що міститься в них
find пошук файлів за багатьма параметрами
finger видає інформацію про користувачів системи
grep пошук заданого виразу (рядка) в одному або кількох файлах
head друкує кілька перших рядків файла (по замовчуванню 10)
kill призупиняє задачу і знімає її з виконання

joe	текстовий редактор
ln	створює файлам додаткові імена
lpr	друк файлу(ів) на принтері
ls	друк вмісту каталога
mail	обробка електронної пошти
man	справка по елементам (командам, змінним і т.ін.) ОС Linux
mc	Midnight Commander (аналог Norton Commander для Linux)
mesg	дозвіл/заборона приймати на екран повідомлення, послані за допомогою команди write
mkdir	створення нового каталога
more	поекранний друк файлів
mv	переміщення (переіменування) файлів
passwd	зміна пароля
pico	текстовий редактор
pine	обробка електронної пошти
ps	відображує процеси, що протікають в системі на даний момент
pwd	друкує повне ім’я поточного каталога
rm	видалення файлів
rmdir	видалення каталогів
script	ведення протокола роботи в системі
sleep	чекає визначену кількість секунд перед тим як виконати чергову команду
sort	сортування файлів
tail	друк кількох останніх рядків файла
tee	дублювання стандартного потоку
touch	зміна дати і часу останньої модифікації файла на поточні
tty	друкує ім’я пристрою вашого термінала
vi	текстовий редактор
wait	чекає, поки не будуть завершені процеси, що виконуються на фоні
who	відображає інформацію про поточний стан системи
write	посилає повідомлення на екран іншого користувача, що знаходиться в системі

Додаток В

Спеціальні змінні і режими *bash shell*

В.1 Системні змінні

HOME	Шляхове ім'я початкового каталога користувача
LOGNAME	Реєстраційне ім'я
USER	Реєстраційне ім'я
TZ	Годинниковий пояс, що використовується системою

В.2 Перевизначені змінні

SHELL	Шляхове ім'я програми командного інтерпретатора
PATH	Список шляхових імен каталогів, в яких слід шукати виконавчі каталоги
PS1	Основне запрошення (запит) оболонки
PS2	Додаткове запрошення оболонки
IFS	Символ-розділювач полів
MAIL	Ім'я файла поштової скриньки, в якому утиліта електронної пошти шукає вхідні повідомлення
MAILCHECK	Період між перевірками поштової скриньки

В.3 Змінні користувачів

MAILPATH	Список файлів поштових скриньок, в яких утиліта електронної пошти шукає вхідні повідомлення
TERM	Тип термінала
CDPATH	Шляхові імена каталогів, в яких інтерпретатор шукає виконавчі файли
EXINIT	Команди установки режимів для текстових редакторів ex і vi

В.4 Спеціальні режими

ignoreeof	Блокування можливості виходу з оболонки за допомогою символа кінця файла (Ctrl-D)
noclobber	Запобігання запису файлів поверх існуючих при переадресації
noglob	Блокування спеціальних символів, що використовуються для формування списку імен файлів: *, ?, ~ і []

Додаток Д

Вбудовані команди

Деякі функції вбудовані у оболонку або за необхідностю або за ефективністю. Ці команди виконуються у рамках цього ж процесу, що й оболонка. Переадресування введення-виведення для них не допустимо за винятком спеціальних випадків:

:

пуста команда. Ця команда ні до чого не призводить та повертає нульовий код відповіді;

. файл

зчитати та виконати команди із вказаного файлу, після чого повернутися назад (за команду **. файл**). Для пошуку зміста, який містить файл, застосовують шлях пошуку **$PATH**;

break [n]

вихід з циклу **for** або **while**, якщо такий існує. Якщо задано **n**, то виконується вихід з **n** вкладених циклів;

continue [n]

почати наступну ітерацію циклу **for** або **while**. Якщо задано **n**, то відновлюється виконання **n**-го циклу;

cd [парам]

зробити поточним зміст **парам**. Значенням **парам** є **$HOME**. Для пошуку змісту **парам** використовується також змінна оболонки **CDPATH**. Синонімом команди **cd** є команда **chdir**;

eval [парам ...]

задані параметри посилаються оболонці у якості вхідних даних та отримана у результаті команда(команди) виконуються;

exec [парам ...]

замість оболонки виконується команда, яка задається параметрами **exec**. Новий процес не створюється. У команді можуть бути присутні специфікації введення-виведення. Якщо окрім них у команді **exec** немає інших параметрів, виконується переназначення введення-виведення оболонки;

exit [n]

вихід з оболонки з кодом відповіді **n**. Якщо параметр **n** відсутній, то кодом відповіді буде код відповіді останньої виконаної команди. (Ознака кінця файлу також призведе до виходу з оболонки.);

export [имя-переменной ...]

вказані змінні помічаються як експортовані та будуть автоматично передаватися у середовище виконуємих наступних команд. Якщо параметри не були задані, то видається список з ім'ям експортованих змінних;

login [парам ...]

еквивалентна команді **exec login парам ...** ;

newgrp [парам ...]

еквивалентна команді **exec newgrp парам ...** ;

команди **login** та **newgrp** є вбудованими тільки у деякіх версіях оболонки;

read имя-переменной ...

зі стандартного файлу введення зчитується одна строка. Слова із цієї строки послідовно присвоюються наданим у команді змінним; усі залишені слова присвоюються останній змінній. Ненульовий код відповіді повертається тільки при досягненні кінця файлу;

readonly [имя-переменной ...]

список змінних з доступом тількі для зчитування. Значення вказаних змінних у наступних командах присвоєння змінити буде неможливо. Якщо параметри не вказані, то видається список усіх змінних, які мають доступ тількі для зчитування;

set [--ekntuvx [парам ...]]

-e у неінтерактивному режимі роботи препинити виконання командного файлу щодо невдалому завершенні будь-якої команди;

-k помістити у середовище виконуємої команди усі ключові параметри, які задані у списку параметрів виклику, а не тільки ті, що ідуть поперед ім'ям команди;

-n зчитувати команди але не виконувати їх;

-t припинити роботу після зчитування та виконання однієї команди;

-u вважати помилкою підстановку невстановлених змінних;

-v виводити строки командного файлу за мірою їх зчитування;

-x виводити команди та їх параметри за мірою їх виконання;

- відмінити опції **--x** та **--v** Форма запису цієї опції не однакова у різних версіях системи;

Ці прапорці можуть задаватися також у момент виклика оболонки. Поточний стан прапорців можна знайти у **$--**. Інші параметри команди **set** присвоюються по черзі змінним **$1**, **$2** Якщо не задано ні одного параметру, то буде виведено значення усіх змінних;

shift

позиційні параметри **$2 ...** перейменовуються у **$1, ...** ;

times

отримати сумарні часи – час користувача та системний час,– використані на виконання процесів, та які були виконані з даної оболонки;

trap [парам] [n] ...

параметр **парам** є командою, яку слід зчитати та виконати щодо отримання сигналу(сигналів) **n**. **Парам** обчислюється два рази – спочатку при встановленні реакції на сигнал, а потім у процесі його обробки. Одночасно отриманні сигнали оброблюються по черзі їх номерів. Якщо **парам** відсутній, то щодо сигналу(сигналів) **n** відновлюється вихідна реакція, що малась на увазі. Якщо **парам** є пустою послідовністю, то вказаний сигнал буде ігноруватися оболонкою та викликаємими із неї командами. Якщо **n** дорівнює **0**, то команда **парам** виконується у момент виходу з оболонки. Якщо ж **n** відмінно від **0**, то **парам** виконується по отриманню сигналу з номером **n**. Команда **trap** без параметрів видає список реакций на усі сигнали;

umask [ddd]

масці користувача, яка використовується щодо обмеження повноважень при створенні файлів, присвоюється восьмерічне значення **ddd** Якщо **ddd** відсутнє, то видається поточне значення маски;

wait [n]

ця команда очікує завершення роботи даного процесу та повідомляє його статус завершення. Якщо **n** не задано, то **wait** очікує завершення усіх породжених процесів, активних у даний момент. Кодом відповіді цієї команди є код відповіді очікуваного процесу.

Додаток Ж

Операції команди порівняння test

Ж.1 Порівняння цілих

-gt	Більше ніж
-lt	Менше ніж
-ge	Більше або дорівнює
-le	Менше або дорівнює
-eq	Дорівнює
-ne	Не дорівнює

Ж.2 Порівняння рядків

-z	Перевірка на рядок нульової довжини
-n	Перевірка на рядкове значення
=	Перевірка на рівність рядків
!=	Перевірка на нерівність рядків
str	Перевірка на рядок ненульової довжини

Ж.3 Логічні операції

-a	Логічне І
-o	Логічне АБО
!	Логічне НЕ

Ж.4 Перевірка файлів

-f	Файл існує і є звичайним
-s	Файл не порожній
-r	Файл читаємий
-w	В файл можливий запис
-x	Файл виконавчий
-d	Ім'я файла - це ім'я каталога
-h	Ім'я файла - це символічне посилання
-O	Файл належить користувачеві

Додаток К

Аргументи bash shell

$0	Ім'я UNIX-команди, що виконується
$n	n-й аргумент командного рядка, починаючи з першого. Для зміни значень цих аргументів можна користуватися командою set
$*	Всі аргументи командного рядка, починаючи з першого
$@	Всі аргументи командного рядка, взяті окремо в лапки
$#	Кількість аргументів командного рядка
$$	Ідентифікаційний номер поточного процесу
$!	Ідентифікаційний номер останнього фонового завдання
$?	Код завершення останньої з виконаних команд