Модуль №3 "UNIX-подібні операційні системи"

Лабораторна робота 3.1
Ознайомлення з настільним дистрибутивом GNU/Linux

Мета роботи: ознайомитися з програмним забезпеченням, компонентами та можливостями адміністрування настільного дистрибутиву ОС GNU/Linux.

Завдання роботи: встановити один з розповсюджених дистрибутивів GNU/Linux з графічним оточенням робочого столу, ознайомитися з його складом, можливостями настроювання оточення робочого столу та адміністрування програмного забезпечення.

.

Короткі теоретичні відомості

Лінукс (англ. Linux, повна назва – GNU/Linux) – загальна назва UNIX-подібних операційних систем з відкритим початковим кодом, що працюють на основі однойменного ядра (Linux) та програмного забезпечення, розробленого в рамках проекту GNU. Початково Linux був розроблений окремими ентузіастами для використання на своїх персональних комп'ютерах. Однак пізніше, завдяки підтримці ряду потужних компаній, Linux набув значної популярності і як серверна операційна система. Так, за оцінками, у 2017 році 30-40% серверів у всесвітній мережі Інтернет працювали під керівництвом ОС GNU/Linux.

Операційна система, файли якої записані на жорсткому диску, починає свою роботу із завантаження системного коду з диска у оперативну пам’ять комп’ютера. Цю операцію виконує спеціальна програма – завантажувач (bootloader). У сучасних дистрибутивах Linux використовуються переважно два завантажувачі – GRUB або LILO.

Грандіозний Уніфікований Завантажувач [GRand Unified Bootloader], скорочено GNU GRUB, або просто GRUB – системний завантажувач від проекту GNU, призначений для завантаження операційних систем. GRUB дозволяє користувачеві мати кілька встановлених операційних систем і при вмиканні комп'ютера обирати одну з них для завантаження.

GRUB є еталонною реалізацією завантажувача, відповідного специфікаціям Multiboot, і може завантажити будь-яку сумісну з нею операційну систему. Серед них: Linux, FreeBSD, Solaris і багато інших. Крім того, GRUB вміє по ланцюжку передавати управління іншому завантажувачу, що дозволяє йому завантажувати Windows (через завантажувач NTLDR), MS-DOS, OS/2 та інші системи.

GRUB дозволяє користувачеві при завантаженні задавати довільні параметри і передавати їх в ядро Multiboot-сумісної ОС для подальшої обробки.

У переважній більшості нових дистрибутивів GNU/Linux використовується версія GRUB 2. Ця версія надає наступні нові можливості:

- сценарії в grub.cfg (синтаксис, подібний до bash);
- підтримка сучасних таблиць розділів, таких як GPT;
- роздільне генерування grub.cfg через update-grub.

Пакети, що встановлюють доповнення до GRUB, можуть впроваджувати власні сценарії командою update-grub.

Термін «середовище робочого столу операційної системи» звичайно використовують для позначення конкретної реалізації графічного інтерфейсу користувача операційної системи (ОС), що використовується на сучасних персональних комп'ютерах. Середовище робочого столу призначене бути інтуїтивно зрозумілим способом взаємодії користувача з комп'ютером, використовуючи поняття, аналогічні до тих, що використовуються при взаємодії з фізичним світом. Цей інтерфейс допомагає користувачу легко викликати, налаштувати і модифікувати багато важливих і часто використовуваних функцій конкретної ОС. Однак, GUI зазвичай не дає доступ до всіх численних функцій, які можна знайти в ОС. Графічний інтерфейс ОС використовується замість традиційного інтерфейсу командного рядка (CLI) у випадках, коли повний контроль над ОС не потрібен, а пріоритетом є зручність користування.

Середовище робочого столу зазвичай складається з іконок (значків), вікон, панелей інструментів, папок, «шпалер» (wallpapers) та віджетів робочого столу. (Ці аспекти графічного інтерфейсу користувача іноді називають WIMP – абревіатура від слів:

Windows, Icons, Menus, Pointer – вікна, іконки, меню, вказівник). Графічні інтерфейси робочого столу також можуть забезпечити функціональність перетягування та інші додаткові можливості.

GNOME є вільним середовищем робочого столу для UNIX-подібних операційних систем. Назва "GNOME" є абревіатурою англійських слів GNU Network Object Model Environment. GNOME є частиною проекту GNU, будучи офіційним оточенням робочого столу для операційної системи GNU.

Розробники GNOME орієнтуються на створення повністю вільного середовища, доступного для всіх користувачів, незалежно від їх рівня технічних навичок, фізичних обмежень і мови, якою вони розмовляють. Результатами проекту є програмне забезпечення GNOME, що надає власне середовище для кінцевих користувачів, а також набір інструментів для створення нових додатків, що інтегруються в робоче середовище.

Розробники GNOME надають значну увагу простоті і зручності використання середовища, в тому числі для недосвідчених або фізично обмежених користувачів. Політика розробки середовища викладена в спеціальному документі – «Керівництво по створенню людського інтерфейсу GNOME» (GNOME Human Interface Guidelines, HIG). HIG – це керівництво, покликане допомогти розробникам створювати високоякісний, послідовний і зручний графічний інтерфейс. В результаті застосування HIG середовище стало простішим; багато параметрів, які раніше були доступні в GNOME, тепер стали непотрібними або незначущими для більшості користувачів, і зникли з головного вікна конфігурації.

Велика частина коду GNOME написана на C, але бібліотеки GNOME надають механізми так званого пов'язування, які дозволяють використовувати інші мови. Таким чином, багато додатків для GNOME написані на інших мовах – C++, Python, C# та інших.

На даний час є дві основні гілки розвитку GNOME. GNOME версії 3 – «прогресивний» варіант; на жаль, виявився незручним для багатьох користувачів та споживає багато системних ресурсів. Старий GNOME версії 2 в останні роки отримав нове народження у вигляді робочого столу MATE.

KDE (K Desktop Environment) є іншим популярним вільним середовищем робочого столу для GNU / Linux. У багатьох аспектах, KDE в конфігурації за замовчуванням дуже схожа на Microsoft Windows, і користувачі Windows відчувають себе «вдома» при її використанні. KDE, можливо, найпотужніша, найбільш універсальна, і найбільш візуально приємна з усіх користувацьких середовищ для настільних Linux-систем. KDE має більше варіантів налаштування в стилі «point-and-click» і більше елементів «гарного вигляду», ніж GNOME, XFCE, LXDE або будь-який інший робочий стіл Linux. За допомогою можливостей Plasma Workspaces користувачі можуть легко додавати різні віджети на робочий стіл. KDE має найбільш досконалий зовнішній вигляд у порівнянні з іншими оточеннями робочого столу Linux, але водночас це середовище може бути вельми ресурсомістким. З іншого боку, KDE вимагає менше ресурсів процесора, ніж Unity в Ubuntu, і менше оперативної пам'яті, ніж GNOME 3.x. OpenSUSE, PCLinuxOS, Mageia, Mandriva, Chakra – кілька основних дистрибутивів Linux, в яких використовується KDE. Kubuntu є KDE-версією Ubuntu.

Unity – оточення робочого столу, спочатку розроблене компанією Canonical для використання на нетбуках та планшетах. Починаючи з Ubuntu 11.04, оточення Unity замінило GNOME 2.x в якості оболонки робочого столу за замовчуванням в Ubuntu Linux. Починаючи з Ubuntu 11.10, Unity тепер працює поверх GNOME 3.x і вимагає більше системних ресурсів, ніж оболонки GNOME 3.x або KDE (не кажучи вже про інші, «полегшені» оточення робочого столу Linux). Однак Unity включає менш вимогливий до ресурсів 2D-режим, що дозволяє працювати на старому обладнанні.

За зовнішнім виглядом оточення Unity схоже на робочий стіл Mac OS X. На екрані Unity присутня док-подібна програма під назвою «панель швидкого запуску», яка завжди знаходиться біля лівого краю робочого столу. Крім того, Unity має додаткову пошукову панель, яка завжди знаходиться у верхній частині робочого столу. Замість традиційного меню робочого столу, Unity пропонує доступ до усіх можливостей за допомогою «Heads-Up Display» (HUD) – текстового пошуку, подібного до пошуку Google або будь-якої іншої пошукової машини в інтернеті.

Робочий стіл Unity гарно виглядає і має відмінну сумісність з технологією сенсорного екрану. Однак Unity не дуже зручний для налаштування, і, на думку багатьох користувачів, він являє собою крок назад у порівнянні з традиційними робочими столами GNOME 2.x.

Порядок виконання роботи

1. Ознайомтеся з теоретичними відомостями.
2. Підберіть та встановіть дистрибутив GNU/Linux, сумісний з апаратним забезпеченням Вашого персонального комп'ютера.
3. Вивчіть пункти головного меню системи. Ознайомтеся з основними програмами прикладного та системного призначення, що поставляються з Вашим дистрибутивом та доступні через пункти головного меню.
4. Вивчіть структуру директорій Вашої Linux-системи. З'ясуйте призначення кожного каталогу першого рівня у дереві каталогів, відповідно до стандарту FHS.
5. Встановіть у вашій Linux-системі кілька варіантів оточень робочого столу (щонайменше GNOME і KDE). Яким чином Ви можете перемикатися між ними?
6. Ознайомтеся з основними характеристиками оточень робочого столу GNOME і KDE. (Використовуйте онлайн-документацію, що постачається з ними.)
7. Порівняйте можливості параметрів налаштування робочого столу в KDE та GNOME середовищах.
8. Порівняйте головні меню в середовищах GNOME та KDE.
9. Порівняйте можливості панелей середовищ GNOME і KDE. Вивчіть можливості створення нових панелей та керування їх вмістом.
10. Ознайомтеся з графічними середовищами керування програмним забезпеченням (встановлення, оновлення, видалення програм тощо), доступними у Вашому дистрибутиві. В разі, якщо Ваш дистрибутив заснований на Ubuntu, встановіть Synaptic (*apt-get install synaptic*) та ознайомтеся з можливостями цієї програми.
11. Ознайомтеся з можливостями адміністрування програмного забезпечення засобами командного рядка. Вивчіть ключі утиліт dpkg, apt-get, rpm, що дозволяють виконувати основні дії з адміністрування пакетів програмного забезпечення.

12. Складіть порівняльну таблицю можливостей менеджерів пакетів dpkg, apt-get, aptitude, rpm, dnf, yum.

13. Ознайомтеся з можливостями керування користувачами та персоналізації вигляду та властивостей системи для кожного окремого користувача.

14. Ознайомтеся з основами системи безпеки Linux – права на читання, запис, виконання файлів (r,w,x), та засобами для їх налаштування (chown, chgrp, chmod, відповідні засоби графічного інтерфейсу робочого столу).

15. Ознайомтеся з конфігураційним файлом завантажувача операційної системи.

16. Підготуйте звіт про виконану роботу.

Контрольні питання

1. Що називають оточенням робочого столу?

2. Які оточення робочого столу Linux ви знаєте? Охарактеризуйте їх.

3. Що називається панеллю GNOME? Як її налаштувати?

4. Чим розрізняються головні меню в середовищах GNOME та KDE?

5. Порівняйте характеристики панелей в середовищах GNOME та KDE.

6. Охарактеризуйте призначення найважливіших каталогів (директорій), створених під час встановлення Вашої Linux-системи.

7. Охарактеризуйте основні групи програмного забезпечення, доступні за замовчуванням у Вашому дистрибутиві.

8. Охарактеризуйте графічні засоби адміністрування програмного забезпечення, доступні у Вашому дистрибутиві.

9. Охарактеризуйте консольні утиліти адміністрування програмного забезпечення, доступні у Вашому дистрибутиві.

10. Поясніть поняття «залежності пакетів».

11. Які дії програми dpkg ви знаєте?

12. Наведіть приклади можливого використання програми dpkg.

13. Охарактеризуйте програму apt-get.

14. В чому полягає головна різниця між apt-get та dpkg?

15. Охарактеризуйте призначення та можливості програми Synaptic.

16. Розгляньте внутрішню структуру пакета програмного забезпечення в системі GNU/Linux.

17. Розгляньте систему дозволів на операції з файлами. Поясніть сенс аналогічних дозволів для директорій.

18. Які можливості конфігурації має завантажувач Вашої ОС GNU/Linux?

Література: [2], [7].

Лабораторна робота 3.2

Основи роботи в командному рядку Linux. Написання shell-скриптів.

Мета роботи: ознайомитись з роботою в командному рядку, з оболонкою (bash) і командами.

Завдання роботи: вивчити основні прийоми роботи у командному рядку, ознайомитись з базовими командами Linux, навчитися писати прості скрипти для командного інтерпретатора bash.

Теоретичні відомості

1. Загальні відомості про оболонку.

Командний рядок – простий спосіб для відправки команд вашій системі. У час появи перших версій Unix командний рядок був єдиним засобом спілкування між людиною і комп'ютером. Функції спілкування з боку комп'ютера виконував командний інтерпретатор, який також називають оболонкою (shell).

Оболонка – це програма, яка виконує діалог операційної системи з користувачем. Оболонка запускається кожен раз, коли наступний користувач входить в систему в текстовому режимі. Крім того, оболонка запускається при відкритті емулятора терміналу (в цьому випадку її запускає утиліта Getty). Оболонка отримує всі команди, які вводяться користувачем з клавіатури, і організовує виконання цих команд. У більшості сучасних дистрибутивів GNU/Linux за замовчанням використовується оболонка bash, але досвідчений користувач може встановити собі і іншу – доступний вибір складає кілька десятків варіантів, з яких близько 10 користуються популярністю у спеціалістів.

Щоб з'ясувати, де знаходиться ваш інтерпретатор bash, введіть в командному рядку таку команду:

```
$ which bash
```

Ви отримаєте відповідь наступного вигляду:

```
/bin/bash
```

Список вбудованих команд оболонки можна отримати за допомогою команди help. Детальна інформація про конкретну

вбудовану команду видається за тією ж командою, наприклад: help cd.

2. Способи запуску терміналу

У графічному середовищі (GNOME, KDE) для виклику вікна терміналу передбачено спеціальний пункт у головному меню: System | Terminals, або Applications | Accessories | Terminal , або Applications | System Tools | Terminal. Цей пункт меню відкриває для Вас графічні вікна, зазвичай емулятор терміналу, з рядком тексту команди всередині. Крім цього, у складі дистрибутива GNU/Linux можуть бути присутні (або можна встановити) додаткові програми – емулятори терміналу, які надають ту ж функціональність командного рядка, алу в іншому вікні.

Також, у багатьох дистрибутивах Linux є додаткові текстові консолі, до яких можна перейти, натиснувши Alt+F1 (чи Ctrl+Alt+F1), Alt+F2 і т.д. Кількість таких консолей може коливатися від 2 до 6, залежно від конкретного дистрибутива. Також можуть бути доступні спеціалізовані консолі (Alt+F10, Alt+F12), в яких не працює оболонка, зате можна побачити службову інформацію ядра системи. Для повернення у графічний режим натисніть Alt+F7.

3. Використання облікового запису адміністратора

Вхід в Unix/Linux систему під обліковим записом root не рекомендується, оскільки будь-яка необережна дія супер-користувача може призвести до небажаних наслідків, аж до руйнування операційної системи. У випадку входу в систему під ім'ям звичайного користувача у вас обмежені права, ви не можете видалити або змінити (зіпсувати) системні файли з необережності. Можна відкрити два термінали (консолі), і в перший віртуальний термінал увійти в систему як root, а в іншому терміналі – під ім'ям звичайного користувача. Ви можете виконувати основну роботу в якості звичайного користувача, а коли це буде необхідно для виконання адміністративних функцій, натиснути Ctrl + Alt + <F1> (перехід до першої консолі, де ви увійшли в систему з правами адміністратора) і працювати у ній від імені root. Після завершення операції, яку може виконувати тільки адміністратор, вам слід негайно повернутися в число регулярних користувачів (<Ctrl> +

<Alt> + <F2>). Тепер Ви не ризикуєте зламати щось у системі через помилку.

4. Використання клавіші <Tab>

Оболонка bash має вбудовану підпрограму, призначену для спрощення введення команд у командному рядку. Ця підпрограма викликається командою <Tab> після того, як ви вже ввели певну кількість символів. Якщо ці символи є початком назви однієї зі стандартних команд, відомих оболонці, то можливі два варіанти реакції оболонки на <Tab>. Якщо з введених символів команда визначається однозначно, то оболонка просто доповнює команду в командному рядку. Якщо введених символів недостатньо, щоб однозначно відновити ім'я команди, оболонка показує список можливих варіантів продовження, так що користувач може звернутися до нього і ввести ще кілька символів, які дозволяють однозначно визначити команду, а потім знову натиснути клавішу <Tab> .

5. Shell-скрипти (скрипти оболонки)

Оболонка – це не тільки командний процесор, але і інтерпретатор, для якого існує потужна мова програмування. За традицією, програми для оболонок Unix-подібних систем називаються «скрипти».

Скрипт оболонки – це текстовий файл, що містить послідовність команд оболонки. Скрипти можна виконувати так же, як і звичайні команди. Синтаксис доступу до аргументів залишається таким же. Важливо, щоб файл скрипта був виконуваним (мав дозвіл x), в противному випадку команди скрипта виконуватись не будуть.

Для прикладу, наведемо код скрипта "Hello World". Відкрийте будь-який текстовий редактор і створіть файл, який називається hello_world.sh. Впишіть в файл наступні рядки:

```
#!/bin/bash
# declare STRING variable
STRING="Hello World"
# print variable on a screen
echo $STRING
```

Для виконання скрипта, необхідно перш за все перейти у терміналі в каталог, де знаходиться файл hello_world.sh, і виконати команду:

$ chmod +x hello_world.sh

Призначення цієї команди – надати файлу право на виконання; також можна встановити це право і в графічній оболонці.

Тепер можна запустити скрипт на виконання:

./hello_world.sh

Якщо ви все зробили правильно, то в терміналі з’являться слова "Hello World".

Ось ще один невеликий скрипт bash, що демонструє один з варіантів реалізації інтерактивної взаємодії з користувачем:

```
#!/bin/bash
echo "Якій ОС ви надаєте перевагу?"
select var in "Linux" "Gnu Hurd" "Free BSD" "Other"; do
break
done
echo "Ви б обрали $ var"
```

Якщо зберегти цей текст у файлі, зробити файл виконуваним і запустити, на екран буде виведений наступний запит:

```
Якій ОС ви надаєте перевагу?
1) Linux
2) Gnu Hurd
3) Free BSD
4) Other
#?
```

Тепер введіть будь-яку з чотирьох запропонованих цифр (1,2,3,4). Якщо ви, наприклад, введете 1, то побачите повідомлення:

"Ви б обрали Linux".

У загальному випадку при запуску скрипта запускається новий процес. Для того, щоб виконати скрипт всередині поточної сесії

bash, необхідно використовувати команду source або її синонім – крапку ".". В такому разі скрипт оболонки вказується як аргумент цієї команди. Її формат:

source filename [arguments]

або

. filename [arguments]

Ця команда читає і виконує команди з файлу з ім'ям filename в поточному оточенні і повертає статус, який визначається останньою командою з файлу filename.

Порядок виконання роботи

1. Відкрийте термінал.
2. Введіть команди: whoami, who, w, last. Поясніть отримані результати.
3. Ознайомтеся з вбудованою системою допомоги по командах командного рядка (команди man, xman, info, help та інші засоби.)
4. Випробуйте дію 50-60 найбільш вживаних команд командного рядка GNU/Linux.
5. Ознайомтеся з основними утилітами для пошуку об’єктів у GNU/Linux: find, apropos, whatis, whereis, which, locate, updatedb та інші.
6. Використайте клавішу <Tab> під час роботи в командному рядку. Поясніть отримані результати.
7. Ознайомтеся з командами, що використовуються для визначення пріоритетів процесів (nice, renice, kill, top, gtop, htop, qtop та ін.)
8. Ознайомтеся з утилітами для архівації даних та настроювання мережі.
9. Реалізуйте приклади скриптів з теоретичної частини роботи.
10. Використовуючи літературу, ознайомтеся з основними елементами мови програмування командної оболонки bash. Придумайте прості приклади, які б дозволили практично випробувати використання цих елементів.
11. Знайдіть у каталогах встановленої системи GNU/Linux або в інтернеті bash-скрипти, які, на Ваш погляд, є цікавими для вивчення. У звіті розберіть їх код.
12. Підготуйте звіт про виконану роботу.

Контрольні питання

1. Що називають командним рядком?
2. Що називають командною оболонкою? Для чого її використовують?
3. Що у Linux називають скриптом оболонки?
4. Які ви знаєте способи запуску терміналу?
5. Яку функцію у системах GNU/Linux виконує файл під назвою bash? Як з'ясувати місце його розміщення в системі?
6. Назвіть по пам'яті 50 часто вживаних команд командного рядка GNU/Linux. Поясніть їх призначення.
7. Охарактеризуйте систему допомоги man.
8. Знайдіть у своїй системі файли, що є джерелом інформації для утиліт man та xman.
9. Охарактеризуйте засоби пошуку файлів та інформації у системі GNU/Linux.
10. Як з командного рядка довідатися номер версії ядра Linux, яке використовується у Вашій системі?
11. Які значення можуть отримувати пріоритети процесів у системах GNU/Linux?
12. Охарактеризуйте основні елементи мови програмування командної оболонки bash, які Ви вивчили.
13. Що означає рядок #!/bin/bash на початку скрипта?

Література: [2], [7].

Лабораторна робота 3.3
Ознайомлення з серверним дистрибутивом GNU/Linux

Мета роботи: ознайомитися з програмним забезпеченням, компонентами та можливостями адміністрування серверного дистрибутиву ОС GNU/Linux.

Завдання роботи: встановити один з розповсюджених серверних дистрибутивів GNU/Linux, ознайомитися з його складом, програмним забезпеченням, конфігураційними файлами, можливостями адміністрування та застосування.

Короткі теоретичні відомості

Конфігураційні файли Linux

Конфігураційний файл – це локальний файл, який використовується для керування роботою програми; зазвичай це текстовий файл, який можна читати та редагувати (за наявності відповідних прав) будь-яким текстовим редактором. Виконувані бінарні файли та скрипти не відносять до конфігураційних файлів.

Директорія /etc – це центр конфігурації вашої системи Linux, в його підкаталогах знаходяться всі пов'язані з системою файли конфігурації. (З цієї причини системним адміністраторам Unix-подібних систем рекомендується регулярно створювати резервні копії цього каталогу.) Як правило, в директорії /etc не розміщують двійкових файлів.

Каталог /etc/X11/ містить дерево каталогів, у якому знаходяться файли конфігурації для графічної системи X Window. Багато файлів, розташованих у цьому каталозі, насправді є символічними посиланнями на дерево каталогів /usr/X11R6. В серверних дистрибутивах, призначених переважно для консольного адміністрування, такий каталог може бути відсутнім.

Каталог /etc/X11/gdm/ містить конфігураційну інформацію GNOME Display Manager (gdm). gdm надає еквівалент запиту "login:" для графічного дисплею, тобто створює вікно входу в систему та запускає графічну X-сесію.

Каталоги /etc/cron.d, /etc/cron.daily, /etc/cron.weekly, /etc/cron.monthly містять скрипти, які регулярно виконуються демоном cron. /etc/crontab – файл конфігурації cron. Цей файл

містить таблицю, відповідно до якої cron автоматично виконує задані системні процедури. Також можна додатково встановити таблиці cron для окремих користувачів.

Каталог /etc/cups містить файли конфігурації для загальної системи друку UNIX (CUPS). Ці конфігураційні файли використовуються для визначення параметрів клієнта, таких як стандартний сервер або параметри шифрування за умовчанням.

/etc/fstab – конфігураційний файл для утиліт mount та supermount. В ньому вказані файлові системи, які монтуються автоматично при запуску системи, коли з файла /etc/rc або еквівалентного йому запускається команда mount -a. В системах Linux файл fstab також може містити інформацію про розділи підкачки, які автоматично монтуються при виконанні команди swapon -a.

Конфігураційні файли /etc/passwd, /etc/group містять інформацію про користувачів та групи користувачів системи.

/etc/hosts – аналог файлу hosts у Windows, використовується для визначення відповідностей між назвами системи та доменів і IP-адресами у мережі.

/etc/httpd – конфігураційні файли веб-сервера Apache.

Каталог /etc/init.d та файл /etc/inittab містять опис дій, що мають виконуватися під час ініціалізації системи.

/etc/inetd.conf, /etc/xinet.conf, /etc/xinetd.d/ – конфігураційна інформація служб, які запускаються мережевим суперсервером (inetd, xinetd). В найновіших системах Linux демон inetd не підтримується; натомість використовується новіший, покращений демон xinetd. Суперсервер мережевих служб запускається під час завантаження системи за допомогою відповідних команд у /etc/init.d/inetd (або /etc/rc.local в деяких системах). Будучи запущений, він прослуховує підключення деяких інтернет-сокетів. Коли на одному з сокетів виявляється спроба підключення, він вирішує, яка служба відповідає цьому сокету, і викликає відповідну програму для обслуговування запиту. Сервіси, що керуються через xinetd, також розміщують тут свої файли конфігурації.

Порядок виконання роботи

1. Ознайомтеся з теоретичними відомостями.

2. Підберіть та встановіть серверний дистрибутив GNU/Linux, сумісний з апаратним забезпеченням Вашої комп'ютерної системи.

3. Ознайомтеся з основними конфігураційними файлами дистрибутива. Вивчіть інформацію, розміщену у цих файлах, та формат її подання. (Рекомендується використовувати відомості, подані у відповідних ман-сторінках.)

4. Ознайомтеся з програмним забезпеченням, придатним для створення та відновлення резервних копій інформації сервера.

5. Ознайомтеся з серверним програмним забезпеченням, наявним у складі дистрибутива.

6. Ознайомтеся з особливостями налаштування програмного забезпечення дистрибутива для наступних застосувань:

6.1. веб-сервер (зокрема LAMP);

6.2. файловий сервер (FTP-сервер, Samba File Server);

6.3. email сервер;

6.4. сервер друку;

6.5. сервер баз даних;

6.6. серверна платформа для розробників;

6.7. розгортання віртуальних контейнерів;

6.8. використання хмарних сервісів.

7. Підготуйте звіт про виконану роботу.

Контрольні питання

1. Охарактеризуйте особливості серверного дистрибутива операційної системи.

2. Охарактеризуйте програмне забезпечення, наявне у вивченому дистрибутиві для розв'язання серверних завдань.

3. В разі доступності кількох різних програм одного призначення (наприклад, веб-сервери, сервери електронної пошти тощо), на підставі яких міркувань Ви оберете одну з них?

4. Охарактеризуйте конфігураційні файли Вашого дистрибутива.

5. Які задачі, на Вашу думку, доцільно задавати системі для автоматичного виконання за розкладом у конфігураційних файлах демона cron?

6. Охарактеризуйте структуру рівнів виконання (runlevels) Вашого дистрибутива та відповідні конфігураційні файли.

Література: [2], [3], [7].

Модуль №4 "Архітектура операційних систем"

Лабораторна робота 4.1
Розробка багатопоточних додатків

Мета роботи: ознайомитись з технологією розробки багатопоточних програм та синхронізації потоків.

Завдання роботи: вивчити основи побудови багатопоточних програм; створити приклади багатопоточних програм; застосувати у них засоби узгодження роботи потоків..

Короткі теоретичні відомості

Процеси

Процес включає в себе завантажену в оперативну пам'ять виконувану програму, віртуальний адресний простір, системні і призначені для користувача ресурси, виділені програмі, і один або декілька потоків (ниток) виконання.

Процес в Linux або Windows складається з наступних елементів:

Адресний простір – як правило, захищений та віртуальний (відображається в пам'яті), *код* працюючої програми, *дані* працюючої програми, *стек виконання* і *вказівник стека* (ВС), *програмний лічильник* (ПЛ), набір *регістрів* процесора (загального призначення і статусу), набір системних *ресурсів*.

Для користувачів, і інших процесів, процес ідентифікується за числовим ідентифікатором – *ID процесу* (PID).

В операційній системі, процеси представлені записами в таблиці процесів (ТП; англ. – Process Table, PT): PID – ін.декс (або вказівник на) запис таблиці процесів. Запис таблиці процесів носить назву Process Control Block (PCB).

PCB – це велика структура даних, яка містить всю інформацію про процес (або посилання на таку інформацію).

GNU/Linux використовує для цієї мети структуру task_struct, яка включає в себе більше 70 полів; Windows (лінійки NT) – структуру EPROCESS – близько 60 полів.

Процеси створюються: при завантаженні системи, в результаті дій іншого процесу (див. нижче), в результаті дій користувача, під дією менеджера процесів.

Процес може бути нормально завершений (вихід), може припинити своє виконання через помилку, а також може бути завершений (вбитий) в результаті дій користувача або іншого процесу.

Процес може знаходитися у системі в одному з кількох станів:

новий (new) – додавання нового процесу в список виконання,

готовий(ready) – чекає на те, щоб бути відправленими до процесору;

працює (running) – виконується (використовує процесор);

очікування (waiting) – чекає на яку-небудь подію, наприклад, введення/виведення;

припиняється (terminated) – ліквідація процесу зі списку виконання.

Потоки

*Потік (*Thread, нитка виконання*)* – це виконувана суть процесу.

У кожному процесі завжди неявно (непомітно для програміста і незалежно від його волі) присутній один основний потік додатку, решту потоків при необхідності програміст створює сам. Потоки виконуються паралельно у віртуальному адресному просторі процесу, який їх породив, і розділяють ресурси цього процесу. Одиницею багатозадачності в Windows є потік, а не процес. Потік можна розглядати як автономно працюючу і керовану частину додатку, а багатопоточність додатку – як багатозадачність усередині програми.

Серед причин, що мотивують розробників створювати багатопоточні додатки, перш за все слід виділити наступні.

• *Підвищення надійності програми.* Зациклення основного потоку додатка повністю блокує його роботу, при цьому додаток може бути завершений лише за допомогою диспетчера задач (Task Manager), що, як правило, супроводжується втратою незбережених даних. Тому «неблагонадійні» обчислювальні фрагменти рекомендується переносити з основного потоку в окремі додаткові потоки, передбачивши можливість їх дострокового завершення.

• *Підвищення швидкодії, економія ресурсів і розширення функціональних можливостей програми.* Багатопоточність дозволяє паралельно виконувати окремі ділянки програми на ЕОМ з кількома процесорами, або виконувати їх на одному процесорі «псевдопаралельно», використовуючи механізм витісняючої багатозадачності Windows. Наприклад, різні потоки в Microsoft Word одночасно приймають введення користувача, перевіряють орфографію у фоновому режимі і друкують документ. Microsoft Excel будує діаграми і виконує математичні розрахунки у фоновому режимі. Сервер баз даних для відповіді на кожен запит клієнта запускає окремий потік, інакше довелося б або запускати окрему копію сервера, марно витрачаючи ресурси, або надзвичайно ускладнювати логіку його роботи. Анімація, відтворення звуку і інші подібні можливості, що прикрашають інтерфейси деяких прикладних програм, виконуються окремими потоками.

В умовах витісняючої багатозадачності потоки виконуються попеременно, час процесора виділяється потокам *квантами* (slices). ОС витісняє потік, коли закінчиться його квант, або коли з черги надходить потік з більшим пріоритетом. Пріоритети постійно перераховуються, щоб уникнути монополізації процесора одним потоком. З кожним потоком пов'язаний *контекст* – структура, що містить інформацію про стан потоку, зокрема вміст регістрів процесора. Витіснення потоку супроводжується збереженням вмісту регістрів в контексті, а отримання потоком кванта часу – відновленням регістрів з контексту.

Інтерфейси програмування додатків операційних систем, що підтримують багатопоточність, надають відповідні засоби для створення потоків, керування ними, та, зрештою, їх знищення.

В свою чергу, розвинуті середовища програмування пропонують відповідні засоби та компоненти, які стають «обгортками» для цих функцій (wrappers), спрощуючи для розробника задачі керування потоками. Такі засоби є у мовах C/C++, Java, C# та багатьох інших

Створення потоку у Windows

Створення потоку у Windows виконується за допомогою виклику API функції:

```
HANDLE CreateThread(
PSECURITY_ATTRIBUTES psa,
DWORD cbStack,
PTHREAD_START_ROUTINE pfnStartAddr,
PVOID pvParam,
DWORD tdwCreate,
PDWORD pdwThreadID
);
```

Виклик цієї функції створює об'єкт ядра «потік» і повертає його дескриптор. Система виділяє пам'ять під стек нового потоку з адресного простору процесу, ініціалізує структури даних потоку і передає управління потоковій функції. Новий потік виконується в контексті того ж процесу, що і батьківський потік. Тому він має доступ до всіх дескрипторів процесу, адресного простору. Тому всі потоки можуть легко взаємодіяти один з одним

Параметри функції CreateThread мають наступний зміст:

• psa – вказівник на структуру SECURITY_ATTRIBUTES. Якщо ви хочете, щоб потоку були привласнені параметри захисту за умовчанням – передайте сюди NULL.

• cbStack – розмір стека потоку. Якщо параметр дорівнює нулю - використовується розмір за умовчанням. Якщо ви передаєте не нульове значення, система вибере більше між настройками поточного виконуваного файлу і вашим значенням. Цей параметр резервує тільки адресний простір, а фізична пам'ять виділяється в міру необхідності.

• pfnStartAddr –вказівник на потокову функцію.

• pParam – довільне значення. Цей параметр ідентичний параметру потокової функції. CreateThread передасть цей параметр в потокову функцію. Це може бути число, або покажчик на структуру даних. Можна створювати декілька потоків з однією і тією ж потоковою функцією. Кожному потоку можна передавати своє значення. Однак сюди не можна передавати вказівник на локальні змінні! Оскільки батьківський потік працює одночасно з новим - локальні змінні можуть вийти з області видимості і

зруйнуватися компілятором, в той час, як новий потік намагатиметься дістати до них доступ.

• tdwCreate – додаткові параметри створення потоку. Може набувати значення 0, якщо треба почати виконання потоку негайно, або CREATE_SlJSPENDED. У останньому випадку система виконує всю ініціалізацію, але не запускає виконання потоку. Потік можна запустити у будь-який момент, викликавши WinAPI функцію ResumeThread.

• pdwThreadID – Вказівник на змінну, яка на виході міститиме ідентифікатор потоку. Windows (починаючи з Windows 2000) дозволяє передавати сюди NULL, якщо Вам не потрібне це значення. Проте рекомендується завжди передавати адресу змінної для сумісності з більш ранніми ОС.

Завершення потоку у Windows

Потік може завершитися в наступних випадках:

• потік самознищується за допомогою виклику ExitThread (не рекомендується);

• функція потоку повертає управління (рекомендований спосіб);

• один з потоків даного або стороннього процесу викликає функцію TerminateThread (небажаний спосіб);

• завершується процес, що містить даний потік (теж небажано).

Функцію потоку слід проектувати так, щоб потік завершувався тільки після того, як вона повертає управління. Це єдиний спосіб, що гарантує коректне очищення всіх ресурсів, що належали Вашому потоку. При цьому:

• будь-які C++- об'єкти, створені даним потоком, знищуються відповідними деструкторами;

• система коректно звільняє пам'ять, яку займав стек потоку;

• система встановлює код завершення даного потоку (підтримуваний об'єктом ядра "потік") – його і повертає Ваша функція потоку;

• лічильник користувачів даного об'єкту ядра "потік" зменшується на 1.

Критичні секції

Критичні секції – це об'єкти, використовувані для блокування одночасного доступу всіх ниток (threads) додатку, окрім однієї, до деяких важливих даних. Наприклад, є змінна m_pObject і декілька ниток, що викликають методи об'єкту, на який посилається m_pObject, причому ця змінна може змінювати своє значення час від часу. Іноді там навіть виявляється нуль. Припустимо, є ось такий код:

```
// Нитка №1
void Proc1()
{
    if (m_pObject)
        m_pObject->SomeMethod();
}

// Нитка №2
void Proc2(IObject *pNewObject)
{
    if (m_pObject)
        delete m_pObject;
    m_pObject = pNewobject;
}
```

Тут ми маємо потенційну небезпеку виклику m_pObject -> SomeMethod() після того, як об'єкт був знищений при допомозі delete m_pObject. Річ у тому, що в системах з витісняючою багатозадачністю виконання будь-якої нитки процесу може урватися в самий невідповідний для неї момент часу, і почне виконуватися абсолютно інша нитка. У даному прикладі невідповідним моментом буде той, в якому нитка №1 вже перевірила m_pObject, але ще не встигла викликати SomeMethod(). Припустимо, що в цей момент виконання нитки №1 урвалося, і почала виконуватися нитка №2. Причому нитка №2 встигла викликати деструктор об'єкту. Що ж відбудеться, якщо нитка №1 отримає трохи процесорного часу і викличе-таки SomeMethod() у вже неіснуючого об'єкту? Очевидно, при цьому виникне загроза того, що вміст пам’яті постраждає від некоректної операції.

Саме тут приходять на допомогу критичні секції. Перепишемо наш приклад.

```
// Нитка №1
void Proc1()
{
    ::EnterCriticalSection(&m_lockObject);
    if (m_pObject)
        m_pObject->SomeMethod();
    ::LeaveCriticalSection(&m_lockObject);
}

// Нитка №2
void Proc2(IObject *pNewObject)
{
    ::EnterCriticalSection(&m_lockObject);
    if (m_pObject)
        delete m_pObject;
    m_pObject = pNewobject;
    ::LeaveCriticalSection(&m_lockObject);
}
```

Код, розміщений між ::EnterCriticalSection() і ::LeaveCriticalSection(), якщо його викликано більше ніж один раз з однією і тією ж критичною секцією як параметром, не виконуватиметься паралельно. Якщо нитка №1 встигла "захопити" критичну секцію m_lockObject, то при спробі нитки №2 отримати цю ж критичну секцію в своє одноосібне користування її виконання буде припинено до тих пір, поки нитка №1 не "відпустить" m_lockObject за допомогою виклику ::LeaveCriticalSection(). І навпаки, якщо нитка №2 встигла раніше нитки №1, то нитка №1 "почекає", перш ніж почне роботу з m_pObject.

Сигнали

Сигнал – це асинхронне повідомлення процесу про яку-небудь подію. Сигнали в Unix-подібних та інших POSIX-сумісних операційних системах є одним із способів взаємодії між процесами (англ. – IPC, inter-process communication). Коли надходить сигнал, надісланий процесу, операційна система перериває виконання процесу. Якщо процес встановив власний обробник сигналу,

операційна система запускає цей обробник, передавши йому інформацію про сигнал. Якщо процес не встановив обробник, то виконується обробник за умовчанням.

Назви сигналів «SIG.» є числовими константами (макровизначеннями мови Сі). Їх значення встановлені в заголовочному файлі signal.h. Числові значення сигналів можуть бути різними у різних системах, хоча основна їх частина має в різних системах одні і ті ж значення. Утиліта kill дозволяє задавати сигнал як числом, так і символьним позначенням.

Сигнали можуть бути надіслані кількома способами:

з терміналу, натисненням спеціальних клавіш або комбінацій (наприклад, натиснення CTRL-C генерує SIGINT, а CTRL-Z SIGTSTP);

ядром системи:

при виникненні апаратних виключень (неприпустима інструкція, порушення при зверненні в пам'ять, системний збій і т. п.);

при помилкових системних викликах;

для інформування про події введення-виведення;

одним процесом іншому (або самому собі), за допомогою системного виклику kill();

з оболонки (shell) утилітою /bin/kill з відповідними параметрами.

Порядок виконання роботи

1. Ознайомтеся з наведеними теоретичними відомостями.
2. Використовуючи літературу та джерела мережі Інтернет, самостійно ознайомтеся з засобами створення потоків та керування ними в середовищі програмування gcc (для Unix-подібних систем), в мовах C/C++, Java, C#.
3. Спроектуйте свою майбутню багатопоточну програму. (Запропонуйте власний алгоритм взаємодії потоків у Вашій програмі.) Обговоріть свої пропозиції з викладачем.

3. Напишіть код своєї багатопоточної програми в трьох варіантах: для Windows, для Linux та для одного з високорівневих середовищ програмування. Відладьте програму.

4. Протестуйте вашу програму при різних відносних швидкостях виконання потоків.

5. Додайте до програми засоби синхронізації потоків та випробуйте їх працездатність.

6. Ознайомтеся з сигналами, які реалізовані у Вашому дистрибутиві Linux. (Первинну інформацію про сигнали можна отримати за допомогою man kill.)

7. Реалізуйте можливість програмної передачі сигналів іншому процесу.

8. Підготуйте звіт про виконану роботу. Наведіть у ньому результати експериментів та зробіть з них короткий висновок.

Контрольні питання

1. Що називається процесом, потоком?

2. В чому полягає різниця між процесом та потоком виконання програми?

3. Назвіть та охарактеризуйте стани процесу в комп'ютерній системі.

4. Що мається на увазі при вживанні терміну «багато-поточність»?

5. Охарактеризуйте переваги використання багатопоточних програм.

6. Охарактеризуйте засоби, за допомогою яких створюється та завершується потік.

7. Охарактеризуйте відомі Вам програмні засоби синхронізації роботи потоків. Які з них Ви обрали для реалізації у своїй програмі? Чим був продиктований Ваш вибір?

8. Що називається критичною секцією?

9. Які сигнали реалізовані у Вашому дистрибутиві GNU/Linux? Для чого можуть бути використані сигнали, що передаються процесам?

10. Охарактеризуйте програмні засоби передачі та обробки сигналів.

Література: [1], [3], [4], [5].

Лабораторна робота 4.2
Прогнозування тупикових ситуацій в операційній системі

Мета роботи: ознайомитись з поняттям тупикової ситуації.
Завдання роботи: вивчити алгоритм дій, спрямованих на запобігання утворенню тупикових ситуацій.

Короткі теоретичні відомості

Тупикова ситуація, тупик (deadlock) – це ситуація, в якій деякі сутності чекають на деяку подію, яка насправді ніколи не відбувається.

Частим випадком тупикової ситуації є група з двох або більшої кількості конкуруючих процесів, кожен з яких очікує на завершення іншого процесу з тієї ж групи.

В операційній системі така тупикова ситуація виникає, коли процес або потік входить в стан очікування, тому що ресурс, якого він потребує, зайнятий іншим процесом, який, у свою чергу, чекає ще один ресурс, і т.д. Якщо процес не може змінити свій стан очікування через те, що ресурс, який йому потрібен, використовується іншим процесом, який також знаходиться в стані очікування, то кажуть, що система знаходиться в тупиковій ситуації.

Тупик є поширеною проблемою в багатопроцесних системах, системах паралельних обчислень, розподілених системах, де для обробки загальних ресурсів і здійснення синхронізації процесів використовуються програмні і апаратні «замки».

Є чотири умови, які мають бути виконані для того, щоб стан тупику став можливим:

1. *Взаємне виключення*: хоча б один із ресурсів повинен працювати не в режимі спільного доступу; Будь-який інший процес, який запитує цей ресурс, має чекати, поки ресурс буде звільнений.

Вільне, незалежне використання спільних ресурсів (наприклад, файлів тільки для читання) не призводить до тупикової ситуації. На жаль, деякі ресурси, такі як принтери і стрічкові пристрої, вимагають монопольного доступу єдиним процесом.

2. *Утримування і очікування*: процес повинен одночасно володіти не менше ніж одним ресурсом і очікувати принаймні на один ресурс, який в цей час утримується яким-небудь іншим процесом.

3. *Відсутність переривань*: якщо процес утримує ресурс (тобто він отримав ресурс після того, як його запит на ресурс був виконаний), цей ресурс не може бути забраний від цього процесу, поки процес добровільно не звільнить його.

4. *Циклічне очікування*: має існувати така сукупність процесів {P0, P1, P2,. . ., PN} , що кожний процес P[i] чекає на P [(i + 1)% (N + 1)]. (Слід зазначити, що ця умова означає також наявність стану утримання та очікування (умова № 2), але простіше розглядати ці умови окремо).

Граф розподілу ресурсів

У деяких випадках тупики можна зрозуміти більш чітко за допомогою графу розподілу ресурсів, які включають в себе такі аспекти:

1. Набір категорій ресурсів, {R1,R2,R3,...,RN}, які виглядають як прямокутники-вузли на графі. Точки всередині вузлів ресурсу вказують на конкретні екземпляри ресурсів. (Наприклад, дві точки можуть представляти два принтери).

2. Набір процесів, {P1, P2, P3,. . ., PN}

3. Ребра запитів (Request Edges) – набір орієнтованих дуг. Дуга, проведена з Pi в Rj , вказує, що процес Pi звернувся до ресурсу Rj, і в даний час чекає моменту, коли ресурс стане доступним.

4. Ребра Призначення (Assignment Edges) – набір орієнтованих дуг. Дуга, проведена з Rj в Pi, вказує, що ресурс Rj було виділено процесу Pi , тобто, що процес Pi в даний час утримує ресурс Rj.

Ребро запиту може бути перетворене в ребро призначення шляхом зміни напрямку дуги, коли запит буде виконано. (Проте відзначимо також, що ребро запиту вказує на вузол (вікно категорії ресурсів) в цілому, в той час як ребро призначення виходить з певної точки – екземпляра ресурсу у цьому вікні. Цим відображається той факт, що запит надходить на будь-який ресурс даного типу, а призначається завжди конкретний ресурс.)

Приклад графа розподілу ресурсів наведено на рис. 4.4.1.

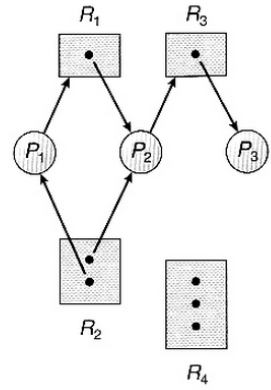

Рис. 4.4.1. Приклад графу розподілу ресурсів.

Стан розподілу ресурсів визначається кількістю доступних і розподілених ресурсів, та максимальними вимогами всіх процесів в системі.

Стан є безпечним, якщо система може виділити всі ресурси, запитані всіма процесами, без входу в тупиковий стан.

Більш формально, стан є безпечним, якщо існує безпечна послідовність задоволення заявок процесів {P0, P1, P2, ..., PN} така, що всі заявки на виділення ресурсів для процесу Pi можуть бути надані з використанням ресурсів, що виділені в даний час для процесу Pi і всіх процесів Pj , де j<i. (Тобто, якщо всі процеси до Pi завершені і звільнили свої ресурси, то Pi зможе закінчитись також, використовуючи ресурси, які вони звільнили.)

Якщо безпечної послідовності не існує, то система знаходиться в небезпечному стані, що може призвести до тупикової ситуації (рис. 4.4.2). Однак ця можливість може і не реалізуватися. Тобто, небезпечний стан системи може привести до тупику, але не обов’язково. Безпечні стани гарантовано захищені від тупикової ситуації,

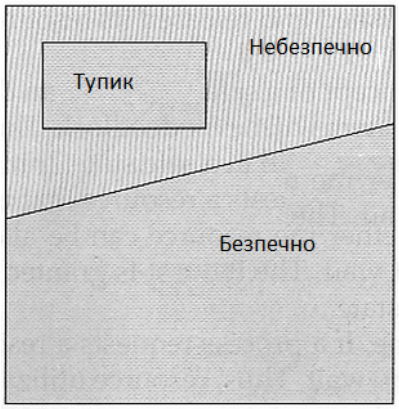

Рис.4.4.2 Співвідношення між безпечним станом, небезпечним станом і тупиковою ситуацією.

Методи боротьби з тупиками

Існують три основних шляхи розв’язання проблеми тупиків:

1. *Профілактика або ухилення від тупику* – дії, які спрямовані на те, щоб не допустити потрапляння системи в тупиковий стан.

2. *Виявлення тупиків; відновлення після тупиків* – дії, які дозволяють виявити тупиковий стан, що вже виник, і вийти з нього.

Реалізувати виявлення тупиків досить просто. Але для відновлення тупиків доводиться перервати один з процесів, що увійшли в тупикове очікування, або відібрати деякі ресурси у одного або більшої кількості процесів. В обох випадках частина виконаної процесами роботи буде, імовірно, втрачена.

3. *Ігнорування проблеми.* Якщо тупикові ситуації виникають рідко, наприклад, тільки один-два рази на рік, то, можливо, краще просто дозволити їм траплятися і при необхідності зняти програму, що зависла, або перезавантажити систему, ніж нести постійні накладні витрати і втрати продуктивності системи, пов'язані із запобіганням або виявленням тупиків. Саме такий підхід використовується в настільних системах Windows і GNU/Linux.

Якщо тупики ні перешкоджено, ні виявлено, то при виникненні тупика система буде поступово сповільнюватися, оскільки все більше і більше процесів застрягатимуть в очікуванні ресурсів, що

в даний час утримуються тупиком і іншими процесами які очікують.

Попередження тупиків

Тупики можуть бути попереджені шляхом порушення принаймні однієї з чотирьох необхідних умов їх існування, перерахованих вище. Наприклад, щоб запобігти стану *утримання та очікування*, слід позбавити процеси від можливості утримання одного або декількох ресурсів одночасно, чекаючи на один або декілька інших. Є кілька можливостей досягти цього.

Уникнення тупиків

Загальна ідея уникнення тупиків полягає в тому, щоб запобігти можливості виникнення тупика, порушуючи хоча б одну із умов його існування. Для цього потрібно мати більше інформації про кожний процес, і це може призводити до низької завантаженості пристрою. (Тобто, це, загалом, консервативний підхід).

Є цілий ряд алгоритмів, призначених для досягнення цієї мети. Найвідомішим серед них є так званий алгоритм банкіра, запропонований Дейкстрою. У деяких алгоритмах планувальник повинен знати тільки максимальну кількість кожного ресурсу, що процес потенційно може використовувати. У більш складних алгоритмах планувальник може також скористатися даними про те, які саме ресурси можуть знадобитися в якому порядку. Коли планувальник виявляє, що початок виконання процесу або виконання запиту процесу на ресурси можуть в майбутньому призвести до тупиків, то такий процес просто не запускається або запит не виконується.

Порядок виконання роботи

1. Вивчіть наведену теоретичну інформацію.
2. Складіть опис алгоритму передбачення та запобігання тупиків. Обговоріть його з викладачем.
3. Реалізуйте алгоритм передбачення та запобігання тупиків у вигляді програми.
4. Запустіть програму та випробуйте її роботу на простих контрольних прикладах.
5. Підготуйте короткий звіт про роботу вашої програми.

Контрольні питання

1. Поясніть поняття «тупикова ситуація».
2. Які методи обробки тупикових ситуацій ви знаєте?
3. Як можна запобігти тупиковій ситуації?
4. Як можна виявити тупикову ситуацію?
5. Розгляньте умови виникнення тупикової ситуації. Для кожної з умов поясніть, чому її порушення робить неможливим існування тупика.

Література: [1], [3], [4].

Лабораторна робота 4.3

Робота з носіями інформації та файловими системами в операційних системах Windows та Linux

Мета роботи: ознайомитись з засобами роботи з носіями інформації та файловими системами.

Завдання роботи: навчитися працювати з різними носіями інформації та файловими системами на платформах Windows та Linux.

Короткі теоретичні відомості

Різні пристрої для запису (зберігання) інформації (даних), як правило, носять загальну назву *пристрої зберігання даних*. Пристрій, який містить лише інформацію, є *носієм даних*. Є також деякі пристрої, які відносять до категорії пристроїв зберігання даних через їх здатність обробляти інформацію. (Деякі з них можуть бути призначені тільки для задач обробки, без зберігання інформації в собі.) Пристрої, які обробляють інформацію (обладнання для збереження даних) можуть або отримувати доступ до окремих портативних (зйомних) носіїв запису, або бути постійними компонентами деяких носіїв даних.

Файл – набір елементів даних, що зберігаються на носії (наприклад, на диску). Файл є формою за умовчанням для всього, що ви зберігаєте в сучасному комп'ютері: числові дані, музичні записи, фотографії, фільми, книги і т.д. Файли завжди пов'язані з пристроями зберігання даних, такими як жорсткий диск, флопі-диск і т.д.

Файли в комп'ютерній системі створюють *дерево файлової системи*. Для групування файлів з метою їх більш ефективної організації використовуються контейнери – *каталоги*. Unix-подібні системи мають кореневий каталог, який позначається / (коса риска, «слеш»). Кореневий каталог є коренем всієї файлової системи, він не може бути перейменований або видалений. Всі каталоги в кореневому каталозі (/) називають підкаталогами. Користувач може створювати, перейменовувати ці каталоги.

Типи файлових систем, що підтримує GNU/Linux

Ext2fs, ext3fs, ext4fs – рідні (англ. – native) для Linux файлові системи. Вони використовують поняття блоку, дескриптору і каталогу. Ці файлові системи підтримують традиційну систему прав доступу, загальну для всіх Unix-подібних операційних систем. POSIX ACL (Access Control Lists), також підтримуються. Ext3 насправді є модифікацією файлової системи ext2 з підвищеними можливостями журналювання. Журналювання дозволяє виконувати швидке відновлення файлової системи.

Isofs (ISO9660) та UFS – файлові системи, що використовуються на CD і DVD дисках.

Sysfs: Це файлова система, яка створюється на основі RAM і використовується для експорту об'єктів ядра, так що кінцевий користувач може легко його використовувати.. Вона заснована на іншій файловій системі ramfs.

Procfs: файлова система proc діє як інтерфейс до внутрішніх структур даних в ядрі. Вона може бути використана для отримання інформації про систему і зміни деяких параметрів ядра під час виконання команди sysctl. Наприклад, ви можете знайти вичерпну інформацію про можливості процесора, використовуючи наступну команду Linux:

#cat /proc/cpuinfo

Ще один приклад – ви можете включити або відключити маршрутизацію/пересилку IP-пакетів між інтерфейсами за допомогою наступної команди:

cat /proc/sys/net/ipv4/ip_forward

echo "1" > /proc/sys/net/ipv4/ip_forward

echo "0" > /proc/sys/net/ipv4/ip_forward

NFS (Network File System) – мережева файлова система, дозволяє багатьом користувачам або системам проводити обмін файлами по мережі, використовуючи з’єднання клієнт/сервер.

Linux також підтримує Microsoft NTFS, vfat (FAT12, FAT16, FAT32), і інші файлові системи. Список всіх підтримуваних файлових систем можна знайти в документації коду ядра Linux (каталог /filesystem дерева початкового коду ядра), або на довідковій сторінці для команди монтування (man mount).

Типи файлових систем, що підтримує Microsoft Windows

ОС Windows використовує файлові системи FAT, NTFS, ExFAT та ReFS (остання підтримується тільки у випусках Windows Server 2012 та Windows 8/8.1, причому Windows не може завантажитися з неї).

ОС Windows використовує абстракцію буквенного позначення дисків, яке дозволяє користувачу відрізнити один диск або розділ від іншого. Так, наприклад, C:\WINDOWS представляє собою каталог Windows, у розділі, що позначається буквою C. До розповсюдження жорстких дисків багато комп'ютерів мали два дисководи гнучких дисків, що позначалися літерами A: і B: , тому традиційно склалося, що для позначення головного розділу жорсткого диска, на який зазвичай встановлюється ОС Windows і з якого вона завантажується, найбільш часто використовується позначення C: .

FAT

Файлова система FAT має кілька версій. Файлові системи FAT12 і FAT16 рідко використовуються на даний момент, через обмеження на кількість записів в кореневий каталог файлової системи і обмежень на максимальний розмір FAT-відформатованих дисків або розділів. FAT32 дозволяє усунути обмеження FAT12 і FAT16, за винятком обмеження близько 4 ГБ на максимальний розмір файлу.

Сімейство файлових систем FAT підтримують практично всі операційні системи для персональних комп'ютерів, включаючи всі версії Windows, MS-DOS, PC DOS і DR-DOS. Тому файлові системи FAT можуть бути використані в якості універсального формату обміну даними між комп'ютерами і пристроями будь-якого типу і віку. Файлова система FAT часто використовується в USB дисках і носіях інформації для таких пристроїв, як портативні музичні плеєри та фотоапарати.

NTFS

Файлова система NTFS, вперше представлена з операційною системою Windows NT, дозволила використовувати управління дозволами на основі ACL. Інші функції, підтримувані NTFS, включають: жорсткі посилання, наявність кількох потоків файлу, індексація атрибута, стеження за квотою, розрідження файлів,

шифрування, стиснення і точки повторної обробки (каталоги можуть працювати як точки монтування для інших файлових систем, символьних зв'язків, переходів, зв'язків віддаленого зберігання). Не всі ці особливості добре документовані розробниками.

Порядок виконання роботи

1. Вивчіть наведені теоретичні відомості.
2. Відкрийте термінал в Linux. Розгляньте ман-сторінки команд *mkfs, fsck, mount*. Використовуючи подану у них інформацію, відформатуйте розділ зовнішнього диску у різних файлові системи, що підтримує Linux.
3. Відкрийте вікно *Мій комп'ютер* в Windows. Клацніть правою кнопкою по значку розділу зовнішнього диску. Виберіть в контекстному меню *Відформатувати*. Відформатуйте зовнішній диск з файлової системи FAT32 до NTFS. (В системах Windows присутня також утиліта, що виконує перетворення файлової системи з FAT до NTFS із збереженням даних.)
4. Відкрийте *Панель управління*. Виберіть *Адміністрування* - *Управління дисками*. Ознайомтесь з можливостями вікна Disk Management.
5. Дізнайтеся, як перевірити поточну файлову систему за допомогою Linux-команди fsck і за допомогою програми Windows Filesystem check.
6. Підготуйте звіт з коротким описом та скриншотами виконаних дій.

Контрольні питання

1. Що називається файловою системою?
2. Дайте характеристики файлових систем, використовуваних в ОС Windows і Linux.
3. Що треба зробити в ОС Windows та в ОС GNU/Linux, щоб створити певну файлову систему на даному носієві (жорсткому диску, флеш-диску, SSD тощо)?
4. Що треба зробити в ОС Windows та в ОС GNU/Linux, щоб перевірити файлову систему на даному носієві?

5. Поясніть причини, через які існує велика кількість різних файлових систем. В чому полягають відмінності між ними?

6. Які файлові системи Ви б обрали для розв’язання тих чи інших задач і чому? (Наведіть приклади.)

Література: [1–4], [7].