НАЦІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ
Кафедра інформаційних технологій та математики

ОПЕРАЦІЙНІ СИСТЕМИ

Методичні вказівки
щодо виконання практичних занять
з дисципліни "Операційні системи"

Частина 1

КИЇВ - 2009

ББК 32.973-018.2
УДК 681.3.066

Операційні системи. Процеси. Методичні вказівки
щодо виконання практичних занять з дисципліни "Операційні системи". Частина 1 /Упор. доц. Баклан І.В. - К.: НАУ, 2009. - 48 с.: ил.

Рецензент М.К.Печурін, проф., докт. техн. Наук
Відповідальний за випуск зав. кафедрою інформаційних технологій та математики О.К.Лопатін.

ЗМІСТ

Заняття №1.

Введення до курсу практичних занять. Знайомство з операційною системою UNIX

Введення до курсу практичних занять. Коротка історія операційної системи UNIX, її структура. Системні виклики і бібліотека libc. Поняття login і password. Спрощене поняття про структуру файлової системи в UNIX. Повні імена файлів. Поняття про поточну директорію. Команда pwd. Відносні імена файлів. Домашня директорія користувача та її визначення. Команда man – універсальний довідник. Команди cd – зміни поточної директорії та ls – перегляду змісту директорії. Команда cat і створення файлу. Перенаправлення введення та виведення. Найпростіші команди для роботи з файлами – cp, rm, mkdir, mv. Історія редагування файлів – ed, vi. Система Midnight Commander – mc. Вбудований mc редактор і редактор joe. Користувач і група. Команди chown і chgrp. Права доступу до файлу. Команда ls з опціями -al. Використання команд chmod і umask. Системні виклики getuid і getgid. Компіляція програм мовою C у UNIX і запуск їх на опрацювання.

Введення до курсу практичних занять

Дійсний курс практичних занять є спробою систематично проілюструвати лекційний курс «Операційні системи» на прикладі конкретної операційної системи, а саме – операційної системи UNIX.

Необхідність зв'язування систематичного викладу матеріалу семінарських і практичних занять з матеріалом лекцій була позичена з досвіду МФТІ. Важливою відміною від фізтехівського підходу є посилення засвоєння матеріалу у курсовому проекті, який присвячений основам побудови мікроядерних операційних систем. Ретельно вивірена версія цих методичних указівок, розширена, доповнена і модифікована за результатами численних обговорень, пропонується зараз вашій увазі.

Семестровий курс «Операційні системи» є третім по рахунку курсом циклу, якому передують курси «Архітектура ЕОМ” і „Системне програмування”. Передбачається, що до початку практичних занять студенти вміють програмувати мовою С (з використанням функцій стандартної бібліотеки для роботи з файлами та рядками) і мають уявлення про внутрішнє облаштування ЕОМ.

Перехід від навчання студентів інформатиці з використанням мейнфреймів до навчання з використанням мережевих класів персональних комп'ютерів, який здійснився за останні двадцять років, неминуче наклав свій відбиток на форму проведення практичних занять. Замість роздільного проведення семінарів і практикуму (лабораторних робіт) з'явилося щось змішане – семінарський практикум або практичний семінар, коли новий матеріал отримує реалізацію у практичних програмних роботах. Саме у вигляді таких семінарів-практикумів і

побудований наш курс. Через досить високу складність використовуваних програмних конструкцій ми вирішили приводити готові приклади програм для ілюстрації розглянутих понять з наступною їх модифікацією студентами. Це дозволило збільшити насиченість занять і за семестровий курс охопити більшу кількість матеріалу. Для ілюстрації лекцій була обрана операційна система UNIX (на практиці один з її клонів – ASPLinux), як найбільш відкритого, витонченого та простого для розуміння.

У цілому практичний курс містить у собі 16 занять, одне з яких наприкінці семестру присвячене проведенню контрольної роботи з матеріалів лекцій. Деяким темам виділено по два заняття, і відповідні семінари мають здвоєні номери. Природньо, розбивка на заняття є досить умовним. Головне, що вони безпосередньо випливають з матеріалу лекцій, на яких ґрунтуються.

Далі ми переходимо до викладу матеріалу практичного курсу.

По своєму змісті матеріал поточних семінарів 1–2 є найбільш критичним стосовно використовуваного вигляду операційної системи та політики адміністрування. Тому багато питань будуть містити посилання **«довідайтеся у свого системного адміністратора»**. Перш ніж приступати до занять, необхідно забезпечити наявність користувацьких account'ов для навчання. Так що **«Довідайтеся у свого системного адміністратора»**, як це зробити.

У тексті семінарів програмні конструкції, включаючи імена `системних викликів`, стандартних функцій і команди оболонки операційної системи, `виділені іншим шрифтом`. У UNIX `системні виклики` та команди оболонки ініціюють складні послідовності дій, залучаючи різні аспекти функціонування операційної системи. Як правило, у рамках одного семінару повне пояснення всіх нюансів їхнього поводження є неможливим. Тому докладні описи більшості використовуваних `системних викликів`, системних функцій та деяких команд оболонки операційної системи при першій зустрічі з ними винесені з основного тексту на сіре тло й обведені рамочкою, а в основному тексті розглядаються тільки ті деталі їхнього опису, для розуміння яких вистачає накопичених знань.

Якщо будь-який параметр у команди оболонки є необов'язковим, він буде вказуватися в квадратних дужках, наприклад, `[who]`. У випадку, коли можливий вибір тільки одного з декількох можливих варіантів параметрів, варіанти будуть перелічуватися у фігурних дужках і розділятися вертикальною рисою, наприклад, `{+ | - | =}`.

Коротка історія операційної системи UNIX, її структура

На першій лекції ми розібрали зміст поняття «операційна система», обговорили функції операційних систем і способи їхньої побудови. Усі матеріали першої і наступної лекцій ми будемо ілюструвати практичними прикладами, пов'язаними з використанням однієї з різновидів операційної системи UNIX – операційної системи Linux, хоча постараємося не пов'язувати свою розповідь саме з її особливостями. Ядро операційної системи Linux являє собою монолітну систему. При компіляції ядра Linux можна дозволити динамічне завантаження та вивантаження дуже багатьох компонентів ядра – так званих модулів. У момент завантаження модуля його код завантажується для виконання в привілейованому режимі та пов'язується з іншою частиною ядра.

Усередині модуля можуть використовуватися будь-які експортовані ядром функції.

Свій нинішній вигляд ця операційна система знайшла в результаті тривалої еволюції UNIX-подібних операційних систем. Історія розвитку UNIX докладно освітлена практично у всій літературі, присвяченій обчислювальній техніці. Як правило, це багато в чому той самий текст, який з невеликими змінами переписується з одного видання в інше, і певним чином не хотілося в цьому курсі б повторюватися. Ми просто зробимо посилання на досить докладний виклад у книзі [23] та на оригінальну роботу одного з родоначальників UNIX [7].

Системні виклики і бібліотека libc

Основною постійно функціонуючою частиною операційної системи UNIX є її ядро. Інші програми (системні або користувацькі) можуть спілкуватися з ядром за допомогою системних викликів, які за змістом є прямими пунктами входу програм у ядро. При виконанні системного виклику програма користувача тимчасово переходить у привілейований режим, отримуючи доступ до даних або пристроїв, які недоступні при роботі в режимі користувача.

Реальні машинні команди, необхідні для активізації системних викликів, природно, відрізняються від машини до машини, поряд із способом передачі параметрів і результатів між програмою визовом і ядром. Однак з погляду програміста використання системних викликів мовою С нічим зовні не відрізняється від використання інших функцій стандартної ANSI бібліотеки мови С, таких як функції роботи з рядками `strlen()`, `strcpy()` і т. ін. Стандартна бібліотека UNIX – `libc` – забезпечує С-інтерфейс до кожного системного виклику. Це призводить до того, що системний виклик виглядає для програміста як функція мови С. Більш того, відомі вам стандартні функції, наприклад функції для роботи з файлами: `fopen()`, `fread()`, `fwrite()` при реалізації в операційній системі UNIX будуть застосовувати різні системні виклики. Протягом курсу нам доведеться ознайомитися з великою кількістю різноманітних системних викликів і їхніх С-інтерфейсів.

Більшість системних викликів, які повертають ціле значення, використовує значення `-1` для оповіщення про виникнення помилки, та значення більше або дорівнює нулю – при нормальному завершенні. Системні виклики, які повертають покажчики, звичайно для ідентифікації помилкової ситуації користуються значенням `NULL`. Для точного визначення причини помилки С-інтерфейс надає глобальну перемінну `errno`, описану у файлі `<errno.h>` разом з її можливими значеннями та їх короткими визначеннями. Відмітимо, що аналізувати значення перемінної `errno` необхідно відразу після виникнення помилкової ситуації, оскільки ті системні виклики, що успішно завершилися, не змінюють її значення. Для отримання символьної інформації про помилку на стандартному виведенні програми для помилок (за замовчуванням екран термінала) може застосовуватися стандартна UNIX-функція `perror()`.

Функція perror()
Прототип функції

```
#include <stdio.h>
void perror(char *str);
```

Опис функції

Функція `perror()` призначена для виведення повідомлення про помилку, яка відповідає значенню системної змінної `errno` на стандартний потік виведення помилок. Функція друкує вміст рядка `str` (якщо параметр `str` не дорівнює `NULL`), двокрапка, пропуск і текст повідомлення, яке відповідає виниклій помилці, з наступним символом перекладу рядка `('\n')`.

Поняття login і password

Операційна система UNIX є багатокористувацькою операційною системою. Для забезпечення безпечної роботи користувачів і цілісності системи доступ до неї повинен бути санкціонований. Для кожного користувача, якому дозволений вхід до системи, заводиться спеціальне реєстраційне ім'я – username (або login) та зберігається спеціальний пароль – password, який однозначно відповідає цьому імені. Як правило, при реєстрації нового користувача початкове значення пароля для нього задає системний адміністратор. Після першого входу до системи користувач повинен змінити початкове значення пароля за допомогою спеціальної команди. Надалі він може в будь-який момент змінити пароль за своїм бажанням.

«Довідайтеся у свого системного адміністратора» реєстраційні імена та паролі, установлені для вас особисто.

Вхід до системи і зміна пароля

Настав час у перший раз ввійти до системи. Якщо в системі встановлена графічна оболонка поряд із звичайними алфавітно-цифровими терміналами, найкраще це зробити з алфавітно-цифрового термінала або його емулятора. На екрані з'являється напис, який пропонує ввести реєстраційне ім'я, як правило, це «`login:`». Набравши своє реєстраційне ім'я, натисніть клавішу <Enter>. Система запросить у вас пароль, що відповідає введеному імені, видавши спеціальне запрошення – за звичаєм «`Password:`». Уважно наберіть пароль, встановлений для вас системним адміністратором, і натисніть клавішу <Enter>. **Пароль, що вводиться, на екрані не відображається, тому набирайте його акуратно!** Якщо все було зроблено правильно, у вас на екрані з'явиться запрошення до введення команд операційної системи.

Пароль, встановлений системним адміністратором, необхідно змінити. **«Довідайтеся у свого системного адміністратора»**, яка команда для цього використовується на вашій обчислювальній системі (найчастіше це команда `passwd` або `yppasswd`). У більшості UNIX-подібних систем потрібно, щоб новий пароль мав не менше шести символів і мав у складі, принаймні, дві не букви та дві не цифри. **«Довідайтеся у свого системного адміністратора»**, які обмеження на новий пароль існують у вашій операційній системі.

Придумайте новий пароль і гарненько його запам'ятайте, а ще краще запишіть. Паролі в операційній системі зберігаються в закодованому вигляді, і якщо ви його забули, ніхто не зможе допомогти вам його згадати. Єдине, що може зробити системний адміністратор, так це установити вам новий пароль. **«Довідайтеся у свого системного адміністратора»**, що потрібно робити, якщо

ви забули пароль.

Введіть команду для зміни пароля. Звичайно система просить спочатку набрати старий пароль, потім ввести новий та підтвердити правильність його набору повторним введенням. Після зміни пароля вже ніхто сторонній не зможе ввійти в систему під вашим реєстраційним ім'ям.

Congratulations!!! Тепер Ви повноцінний користувач операційної системи UNIX.

Спрощене поняття про побудову файлової системи в UNIX. Повні та відносні імена файлів

В операційній системі UNIX існує три базових поняття: **«процес»**, **«файл»** і **«користувач»**. З поняттям «користувач» ми тільки що вже ознайомилися та будемо торкатися надалі при вивченні роботи операційної системи UNIX. Поняття «процес» характеризує динамічну сторону того, що відбувається в обчислювальній системі. Більш докладно про це буде в лекції 2 і в описі наступних практичних занять. Поняття «файл» характеризує статичну сторону обчислювальної системи.

З попереднього досвіду роботи з обчислювальною технікою ви вже маєте деяке уявлення про файл, як про іменований набір даних, який зберігається будь-де на зовнішніх пристроях пам'яті (магнітні диски, оптичні диски, інші пристрої). Для нашого сьогоднішнього обговорення нам досить такого розуміння, щоб розібратися в тому, як організована робота із файлами в операційній системі UNIX. Більш докладний розгляд поняття «файл» та організації файлових систем для операційних систем у цілому буде наведені в лекціях 11 і 12, а також на практичних роботах 11 і 12, присвячених організації файлових систем у UNIX.

Усі файли, доступні в операційній системі UNIX, як й у вже відомих вам операційних системах, поєднуються в деревоподібну логічну структуру. Файли можуть поєднуватися в **каталоги** або **директорії**. Не існує файлів, які би не входили до складу будь-якої директорії. Директорії у свою чергу можуть входити до складу інших директорій. Допускається існування порожніх директорій, до яких не входить жоден файл, та жодна інша директорія (див. мал. 1–2.1). Серед усіх директорій існує тільки одна директорія, яка не входить до складу інших директорій – її прийнято називати **кореневою або коренем (root)**. На чинному рівні нашого незнання UNIX ми можемо зупинитися на тому, що у файловій системі UNIX є присутнім, принаймні, два типи файлів: звичайні файли, які можуть містити тексти програм, виконуваний код, дані та т.ін. – їх прийнято називати **регулярними файлами**, і директорії.

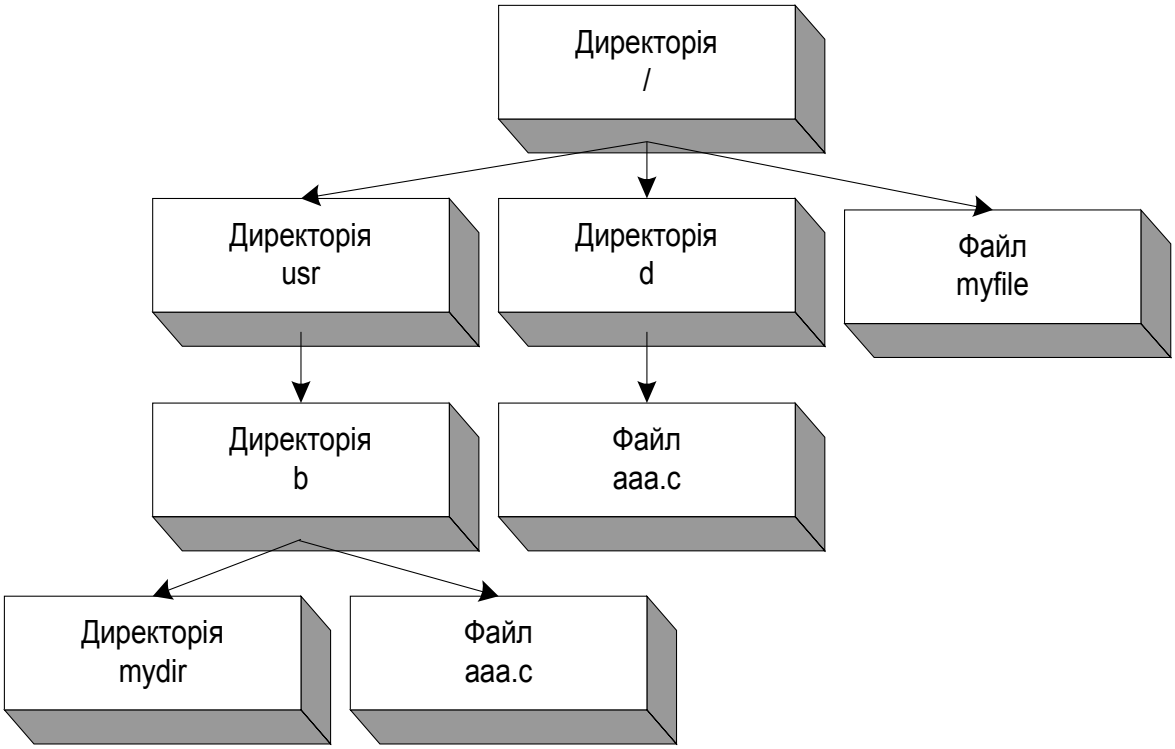
Кожному файлу (регулярному або директорії) повинне бути надане ім'я. У різних версіях операційної системи UNIX існують ті або інші обмеження на побудову імені файлу.

У стандарті POSIX на інтерфейс системних викликів для операційної системи UNIX утримується лише три явних обмеження:

- Не можна створювати імена більшої довжини, ніж це передбачено операційною системою (для Linux – 255 символів).
- Не можна використовувати символ `NUL` (не плутати з покажчиком `NULL`!) – він же символ з нульовим кодом, бо він є ознакою кінця

рядка в мові С.

- Не можна використовувати символ `'/'`.



Мал. 1-2.1. Приклад структури файлової системи

Від себе додатково відзначимо, що також небажано застосовувати символи «зірочка» – «`*`», «знак питання» – «`?`», «лапки» – «`\"`», «апостроф» – «`\'`», «пробіл» – «` `» і «зворотний слеш» – «`\\`»(символи записані в нотації символьних констант мови С).

Єдиним виключенням є коренева директорія, яка **завжди** має ім'я «`/`». Ця ж директорія за цілком зрозумілою причиною являє собою єдиний файл, що повинний мати унікальне ім'я у всій файловій системі. Для всіх інших файлів імена повинні бути унікальними тільки в рамках тієї директорії, у яку вони безпосередньо входять. Яким же чином відрізнити два файли з іменами «`aaa.c`», що входять у директорії «`b`» і «`c`» на малюнку 1–2.1, щоб було зрозуміло про який з них йде мова? Тут на допомогу приходить **поняття** `повного імені файлу`.

Подумки побудуємо шлях від кореневої верхівки дерева файлів до цікавлячого нас файла та випишемо усі імена файлів (тобто вузлів дерева), які зустрічаються на нашому шляху, наприклад, «`/ usr b aaa.c`». У цій послідовності першим буде завжди стояти ім'я кореневої директорії, а останнім – ім'я цікавлячого нас файлу. Відокремимо імена вузлів друг від друга в цьому записі не пробілами, а символами «`/`», за винятком імені кореневої директорії та наступного за ним імені («`/usr/b/aaa.c`»). Отриманий запис однозначно ідентифікує файл у всій логічній конструкції файлової системи. Такий запис й отримав назву `повного`

імені файлу.

Поняття про поточну директорію. Команда pwd. Відносні імена файлів

Повні імена файлів можуть містити в собі досить багато імен директорій та бути занадто довгими, з ними не завжди зручно працювати. У той же час, існують такі поняття як поточна або робоча директорія та відносне ім'я файлу.

Для кожної працюючої програми в операційній системі, в тому числі командний інтерпретатор (shell), який обробляє введені команди та висвітлює запрошення до їх введення, одна з директорій у логічній структурі файлової системи призначається поточною або робочою для даної програми. Довідатися про те, яка директорія є поточною для вашого командного інтерпретатора, можна за допомогою команди операційної системи pwd.

Команда pwd

Синтаксис команди

pwd

Опис команди

Команда pwd виводить повне ім'я поточної директорії для працюючого командного інтерпретатора.

Знаючи поточну директорію, ми можемо прокласти шлях за графом файлів від поточної директорії до цікавлячого нас файла. Запишемо послідовність вузлів, які зустрінуться на цьому шляху, у такий спосіб. Вузол, який відповідає поточної директорії, у запис не включаємо. При русі за напрямком до кореневого каталогу кожний вузол будемо позначати двома символами «крапка» – «..», а при русі за напрямком від кореневого каталогу будемо записувати ім'я вузла, який зустрівся. Розділимо позначення, які відносяться до різних вузлів у цьому записі, символами «/». Отриманий рядок прийнято називати відносним ім'ям файлу. Відносні імена файлів змінюються при зміні робочого каталогу. Так, у нашому прикладі, якщо робочий каталог – це директорія «/d» , те для файлу «/usr/b/aaa.c» відносним ім'ям буде «../usr/b/aaa.c», а якщо робочий каталог – це директорія «/usr/b», то його відносне ім'я – «aaa.c».

Для повноти картини ім'я чинного каталогу можна вставляти у відносне ім'я файлу, позначаючи поточний каталог одиночним символом «крапка» – «.». Тоді наші відносні імена будуть виглядати як «./../usr/b/aaa.c» і «./aaa.c» відповідно.

Програми, запущені на виконання за допомогою командного інтерпретатора, будуть мати у якості робочої директорію його робочу директорію, якщо усередині цих програм не змінити її розташування за допомогою спеціального системного виклику.

Домашня директорія користувача та її визначення

Для кожного нового користувача в системі заводиться спеціальна директорія,

яка стає поточної відразу після його входу до системи. Ця директорія отримала назву домашньої директорії користувача. Скористайтеся командою pwd для визначення своєї домашньої директорії.

Команда man – універсальний довідник

По ходу вивчення операційної системи UNIX вам часто буде вимагатися інформація про те, що робить та або інша команда або системний виклик, які в них параметри й опції, для чого призначені деякі системні файли, який їх формат і т.ін. Ми постаралися, у міру можливості, включити опису більшості використовуваних у курсі команд і системних викликів у наш текст. Однак іноді для отримання більш повної інформації ми відсилаємо читачів до UNIX Manual – посібника з операційної системи UNIX. На щастя, велика частина інформації в UNIX Manual доступна в інтерактивному режимі за допомогою утиліти man.

Користуватися утилітою man досить просто – наберіть команду

```
man ім'я
```

де ім'я – це ім'я цікавлячої вас команди, утиліти, системного виклику, бібліотечної функції або файлу. Спробуйте з її допомогою подивитися інформацію про команду pwd.

Щоб перегорнуть сторінку отриманого опису, якщо воно не розмістилося на екрані цілком, варто натиснути клавішу <пробіл>. Для прокручування одного рядка скористайтеся клавішею <Enter>. Повернутися на сторінку назад дозволить одночасне натискання клавіш <Ctrl> і <b>. Вийти з режиму перегляду інформації можна за допомогою клавіші <q>.

Іноді імена команд інтерпретатора та системних викликів або які-небудь ще імена збігаються. Тоді щоб знайти цікавлячу вас інформацію, необхідно задати утиліті man категорію, до якої відноситься ця інформація (номер розділу). Розподіл інформації за категоріями може злегка відрізнятися у однієї версії UNIX від іншої. У Linux, наприклад, має місце наступний поділ:

1. Файли, що виконуються, або команди інтерпретатора.
2. Системні виклики.
3. Бібліотечні функції.

Спеціальні файли (звичайно файли пристроїв) – що це таке, ви довідаєтеся на 13 та 14 практичних заняттях.

Формат системних файлів і прийняті угоди.

Ігри (за звичаєм відсутні).

4. Макропакети й утиліти – такі самі як man.
5. Команди системного адміністратора.

Підпрограми ядра (нестандартний розділ).

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

```
man номер_розділу ім'я
```

В інших операційних системах цей виклик може виглядати інакше. Для

отримання точної інформації про розбивку на розділи, формі подання номера розділу та додаткових можливостей утиліти man наберіть команду

```
man man
```

Команди cd – для зміни поточної директорії та ls – для відображення на моніторі складу директорії

Для зміни поточної директорії командного інтерпретатора можна скористатися командою cd (change directory). Для цього необхідно набрати команду у наступному вигляді

```
cd ім'я_директорії
```

де ім'я_директорії – повне або відносне ім'я директорії, яку ви хочете зробити поточною. Команда cd без параметрів зробить поточною директорією вашу домашню директорію.

Переглянути зміст поточної або будь-якої іншої директорії можна, скориставшись командою ls (від англійського list). Якщо ввести її без параметрів, ця команда роздрукує вам перелік файлів, які розташовані в поточній директорії. Якщо ж у якості параметра задати повне або відносне ім'я директорії:

```
ls ім'я_директорії
```

то вона роздрукує перелік файлів у зазначеній директорії. Треба зауважити, що в отриманий перелік не увійдуть файли, імена яких починаються із символу «крапка» – «.». Такі файли звичайно створюються різними системними програмами для своїх цілей (наприклад, для настроювання). Подивитися повний перелік файлів можна, додатково вказавши команді ls опцію -a, тобто набравши її у вигляді

```
ls -a або
ls -a ім'я_директорії
```

У команди ls існує багато інших опцій, частина з яких ми ще розглянемо на семінарах. Для отримання повної інформації про команду ls скористайтеся утилітою man.

Подорож за файловою структурою

Користуючись командами cd, ls і pwd, помандруйте за структурою файлової системи та перегляньте її зміст. Можливо, зайти в деякі директорії або подивитися їх наповнення вам не удасться. Це пов'язано з роботою механізму захисту файлів і директорій, про який ми поговоримо пізніше. Не забудьте наприкінці подорожі повернутися у свою домашню директорію.

Команда cat і створення файлу. Спрямування вводу та виведення

Ми вміємо переміщатися за логічною структурою файлової системи та розглядати її наповнення. Хотілося б ще вміти і переглядати вміст файлів, і створювати їх. Для перегляду вмісту невеликого текстового файлу на екрані

можна скористатися командою `cat`. Якщо набрати неї у вигляді

```
cat ім'я_файлу
```

то на екран виплеснеться весь його вміст.

Увага! Не намагайтеся розглядати на екрані вміст директорій – усе рівно не вийде! Не намагайтеся переглядати вміст невідомих файлів, особливо якщо ви не знаєте, текстовий він або бінарний. Виведення на екран бінарного файлу може привести до непередбаченої поведінки вашого термінала.

Якщо навіть ваш файл і текстовий, але великий, то все рівно ви зможете побачите тільки його останню сторінку. Великий текстовий файл зручніше розглядати за допомогою утиліти `more` (опис її використання ви знайдете в UNIX Manual). Команда `cat` буде нам цікава з іншого погляду.

Якщо ми у якості параметрів для команди `cat` задамо не одне ім'я, а імена декількох файлів

```
cat файл1 файл2 ... файл
```

то система видасть на екран їх вміст у визначеному порядку. Виведення команди `cat` можна перенаправляти з екрана терміналу до будь-якого файл, скориставшись символом перенаправлення вихідного потоку даних – знаком «більше» – «>». Команда

```
cat файл1 файл2 ... файл > файл_результату
```

зіллє вміст усіх файлів, чиї імена розташовані перед знаком «>», воєдино у `файл_результату` – поєднає їх (від слова concatenate походить назва команди). Практика перенаправлення вихідних даних зі стандартного потоку виведення (екрана) у файл є стандартним для всіх команд, виконуваних командним інтерпретатором. Ви можете отримати файл, який містить перелік усіх файлів `поточної директорії`, якщо виконаєте команду `ls -a` з перенаправленням вихідних даних

```
ls -a > новий_файл
```

Якщо імена вхідних файлів для команди `cat` не задані, то вона буде використовувати в якості вхідних даних інформацію, яка вводиться з клавіатури, доти, поки ви не наберете ознаку закінчення введення – комбінацію клавіш <CTRL> і <d>.
Таким чином, команда

```
cat > новий_файл
```

дозволить вам створити новий текстовий файл з ім'ям `новий_файл` і вмістом, що користувач введе з клавіатури. У команди `cat` існує безліч різних опцій. Подивитися її повний опис можна в UNIX Manual.
Зауважимо, що поряд з перенаправленням вихідних даних існує спосіб перенаправляти вхідні дані. Якщо під час виконання деякої команди потрібно ввести дані з клавіатури, можна заздалегідь покласти їх у файл, а потім перенаправляти стандартне введення цієї команди за допомогою знака «менше» – «<» і наступного за ним імені файлу з вхідними даними. Інші варіанти

перенаправлення потоків даних можна подивитися в UNIX Manual для командного інтерпретатора.

Створення файлу за допомогою команди cat

Переконайтеся, що ви у своїй `домашній директорії`, і створіть за допомогою команди `cat` новий текстовий файл. Перегляньте його вміст.

Найпростіші команди роботи з файлами – cp, rm, mkdir, mv

Для нормальної роботи з файлами необхідно не тільки вміти створювати файли, переглядати їх вміст і переміщуватися за логічним деревом файлової системи. Потрібно ще вміти створювати власні піддиректорії, копіювати та видаляти файли, перейменовувати їх. Це той мінімальний набір операцій, не володіючи яким, не можна почувати себе впевнено при роботі з комп'ютером.

Для створення нової піддиректорії використовується команда `mkdir` (скорочення від make directory). У найпростішому вигляді команда виглядає в такий спосіб:

```
mkdir ім'я_директорії
```

де `ім'я_директорії` – повне або відносне ім'я створюваної директорії. У команди `mkdir` мається набір опцій, опис яких можна переглянути за допомогою утиліти `man`.

Команда cp

Синтаксис команди

```
cp файл_джерело файл_призначення
cp файл1 файл2 ... файл дир_призначення
cp -r дир_джерело дир_призначення
cp -r дир1 дир2 ... дир дир_призначення
```

Опис команди

Чинний опис є не повним описом команди `cp`, а лише коротким введенням до її використання. Для отримання повного опису команди зверніться до UNIX Manual.

Команда `cp` у формі

```
cp файл_джерело файл_призначення
```

служить для копіювання одного файлу з ім'ям `файл_джерело` до файлу з ім'ям `файл_призначення`.

Команда `cp` у формі

```
cp файл1 файл2 ... файл дир_призначення
```

служить для копіювання файлу або файлів з іменами `файл1, файл2, ... файл` до вже існуючої директорії з ім'ям `дир_призначення` під своїми іменами. Замість імен скопійованих файлів можуть використовуватися файлові шаблони.

Команда `cp` у формі

```
cp -r дир_джерело дир_призначення
```

служить для рекурсивного копіювання однієї директорії з `ім'ям дир_джерело` до нової директорії з ім'ям `дир_призначення`. Якщо директорія

дир_призначення вже існує, то ми отримуємо команду cp у наступній формі

```
cp -r дир1 дир2 ... дир дир_призначення
```

Така команда служить для рекурсивного копіювання директорії або директорій з іменами дир1, дир2, ... дир до вже існуючої директорії з ім'ям дир_призначення під своїми власними іменами. Замість імен скопійованих директорій можуть використовуватися файлові шаблони.

Для копіювання файлів може використовуватися команда cp (скорочення від copy). Команда cp уміє копіювати не тільки окремий файл, але й набір файлів, та навіть директорію цілком разом із усіма вхідними до неї піддиректоріями (рекурсивне копіювання). Для завдання набору файлів можуть використовуватися шаблони імен файлів. Точно так само шаблон імені може бути використаний і в командах перейменування файлів й їхнього видалення, яке ми розглянемо нижче.

Шаблони імен файлів

Шаблони імен файлів можуть застосовуватися у якості параметрів для завдання набору імен файлів у багатьох командах операційної системи. При використанні шаблона проглядається вся сукупність імен файлів, яка знаходяться у файловій системі, і включаються до набору ті імена, які задовольняють шаблону. У загальному випадку шаблони можуть задаватися з використанням наступних метасимволів:

* – відповідає всім ланцюжкам літер, включаючи порожню;

? – відповідає всім одиночним літерам;

[...] – відповідає будь-якій літері, вкладеної в дужки. Пара літер, розділених мінусом, задає діапазон літер.

Так, наприклад, шаблонові *.c задовольняють усі файли поточної директорії, чиї імена закінчуються на .c. Шаблонові [a-d]* задовольняють усі файли поточної директорії, чиї імена починаються з букв a, b, c, d. Існує одне обмеження на використання метасимволу * на початку імені файлу, наприклад, у випадку шаблона *c. Для таких шаблонів імена файлів, що починаються із символу крапка, вважається не задовільнюють шаблонові.

Для видалення файлів або директорій застосовується команда rm (скорочення від remove). Якщо ви хочете видалити один або кілька регулярних файлів, то найпростіший вигляд команди rm буде таким:

```
rm файл1 файл2 ... файл
```

де файл1, файл2, ... файл – повні або відносні імена регулярних файлів, що потрібно видалити. Замість імен файлів можуть використовуватися файлові шаблони. Якщо ви хочете видалити одну або кілька директорій разом з їх вмістом (рекурсивне видалення), то до команди додається опція -r:

```
rm -r дир1 дир2 ... дир
```

де дир1, дир2, ... дир – повні або відносні імена директорій, які потрібно видалити. Замість безпосередньо імен директорій також можуть використовуватися файлові шаблони. У команди rm є ще набір корисних опцій, які ретельно прописані в UNIX Manual. Насправді процес видалення файлів не

так простий, який здається на перший погляд. Він буде розглянутий більш докладніше на практичних заняттях 11 та 12, коли ми будемо обговорювати операції над файлами в операційній системі UNIX.

Команда mv

Синтаксис команди

```
mv ім'я_джерела ім'я_призначення
mv ім'я1 ім'я2 ... ім'я дир_призначення
```

Опис команди

Чинний опис не є повним описом команди `mv`, а є коротким введенням до її використання. Для отримання повного опису команди звертайтеся до UNIX Manual.

Команда `mv` у формі

```
mv ім'я_джерела ім'я_призначення
```

служить для перейменування або переміщення одного файлу (не є важливим, регулярного або директорії) з ім'ям `ім'я_джерела` до файла з ім'ям `ім'я_призначення`. При цьому перед виконанням команди файлу з ім'ям `ім'я_призначення` не повинно існувати.

Команда `mv` у формі

```
mv ім'я1 ім'я2 ... имя дир_призначення
```

служить для переміщення файлу або файлів (неважливо, регулярних файлів або директорій) з іменами `ім'я1, ім'я2, ... имя` до вже існуючої директорії з ім'ям `дир_призначення` під власними іменами. Замість імен переміщуваних файлів можуть використовуватися файлові шаблони.

Командою видалення файлів і директорій варто користуватися з обережністю. Вилучену інформацію відновити неможливо. Якщо ви системний адміністратор і ваша `поточна директорія` **– це коренева директорія, будь ласка, не виконуйте команду** `rm -r *`**!**

Для перейменування файлу або його переміщення до іншого каталога застосовується команда `mv` (скорочення від move). Для завдання імен переміщуваних файлів за допомогою цієї команди можна використовувати файлові шаблони.

Історія редагування файлів – ed, vi

Отримані знання вже дозволяють нам досить вільно оперувати файлами. Але що нам робити, якщо буде потрібно змінити вміст файлу, відредагувати його?

Коли з'явилися перші варіанти операційної системи UNIX, пристрої введення та відображення інформації істотно відрізнялися від існуючих сьогодні. На клавіатурах були присутні тільки алфавітно-цифрові клавіші (не було навіть клавіш курсорів), а дисплеї не допускали екранного редагування. Тому перший редактор операційної системи UNIX – редактор `ed` – вимагав від користувача строгої вказівки того, що та як буде редагуватися за допомогою спеціальних команд. Так, наприклад, для заміни першого сполучення символів «`ra`» на «`ru`» в одинадцятому рядку файлу, що редагується, треба було б ввести команду

```
11 s/ra/ru
```

Редактор ed,власне кажучи, був порядковим редактором. Згодом з'явився екранний редактор – vi,однак і він вимагав суворихої вказівок того, що і як у поточній позиції на екрані ми повинні зробити, або яким образом змінити поточну позицію, за допомогою спеціальних команд, що відповідають алфавітно-цифровим клавішам. Ці редактори можуть показатися нам зараз анахронізмами, але вони дотепер входять до складу усіх варіантів UNIX і іноді (наприклад, при роботі з віддаленою машиною за повільним каналом зв'язку) є єдиним засобом, який дозволяє редагувати файл на віддаленні.

Система Midnight Commander – mc. Вбудований mc редактор і редактор joe

Напевно, ви вже переконалися в тому, що робота в UNIX винятково на рівні командного інтерпретатора й убудованих редакторів далека від вже звичних для нас зручностей. Але не все так погано в юніксовому королівстві. Існують різноманітні пакети, які полегшують задачу користувача в UNIX. До таких пакетів варто віднести Midnight Commander – аналог програм Norton Commander для DOS і FAR для Windows 9x і NT – із своїм вбудованим редактором, який запускається командою `mc`, і екранний редактор `joe`. Інформацію про них можна знайти в UNIX Manual. Більшими можливостями володіють багатофункціональні текстові редактори, наприклад, emacs.

Ввійдіть у `mc` і спробуйте переміщатися по директоріях, створювати та редагувати файли.

Користувач і група. Команди chown і chgrp. Права доступу до файлу

Як уже говорилося, для входу в операційну систему UNIX кожен користувач повинний бути зареєстрований у ній під визначеним ім'ям. Обчислювальні системи не вміють оперувати іменами, тому кожному імені користувача в системі відповідає деяке числове значення – його ідентифікатор – UID (user identificator).

Усі користувачі в системі поділяються на групи. Наприклад, студенти однієї навчальної групи можуть складати окрему групу користувачів. Групи користувачів також отримують свої імена та відповідні ідентифікаційні номери – `GID` (`group identificator`). В одних версіях UNIX кожний користувач може входити тільки в одну групу, в інші – у кілька груп.

Команда chown

Синтаксис команди

```
chown owner файл1 файл2 ... файл
```

Опис команди

Команда `chown` призначена для зміни власника (хазяїна) файлів. Чинний опис не є повним описом команди, а адаптовано стосовно до даного курсу. Для отримання повного опису звертайтеся до UNIX Manual. Нового власника файлу можуть призначити тільки попередній власник файлу або системний адміністратор.

Параметр `owner` задає нового власника файлу в символьному вигляді, як його `username`,або в числовому вигляді, як його `UID`.

Параметри `файл1, файл2, ... файл` – це імена файлів, для яких робиться зміна власника. Замість імен можуть використовуватися файлові шаблони.

Для кожного файлу, створеного у файловій системі, запам'ятовуються імена його хазяїна та групи хазяїв. Зауважимо, що група хазяїв не обов'язково повинна бути групою, у яку входить хазяїн. Спрощено можна вважати, що в операційній системі Linux при створенні файлу його хазяїном стає користувач, який створив файл, а його групою хазяїв – група, до якої цей користувач належить. Згодом хазяїн файлу або системний адміністратор можуть передати його у власність іншому користувачеві або змінити його групу хазяїв за допомогою команд `chown` і `chgrp`, опис яких можна знайти в UNIX Manual.

Команда chgrp

Синтаксис команди

```
chgrp group файл1 файл2 ... файл
```

Опис команди

Команда `chgrp` призначена для зміни групи власників (хазяїв) файлів. Чинний опис не є повним описом команди, а адаптованим у відповідності до даного курсу. Для отримання повного опису звертайтеся до UNIX Manual. Нову групу власників файлу можуть призначити лише власник файлу або системний адміністратор.

Параметр `group` задає нову групу власників файлу в символьному вигляді, як ім'я групи, або в чисельному вигляді, як її `GID`.

Параметри `файл1, файл2, ... файл` – це імена файлів, для яких виконується зміна групи власників. Замість імен можуть використовуватися файлові шаблони.

Як ми бачимо, для кожного файлу виділяється три категорії користувачів:

- Користувач, який є хазяїном файлу;

Користувачі, що входять до групи хазяїв файлу;

Всі інші користувачі.

Для кожної з цих категорій хазяїн файлу може визначити різні права доступу до файлу. Розрізняють три типи прав доступу: право на читання файлу – `r` (від слова read), право на модифікацію файлу – `w` (від слова write) і право на виконання файлу — `x` (від слова execute). Для регулярних файлів зміст цих прав збігається з наведеним вище. Для директорій він трохи інший. Право читання для каталогів дозволяє читати імена файлів, які знаходяться в цьому каталозі (і тільки імена). Оскільки «виконувати» директорію безглуздо (як, утім, і невиконуваний регулярний файл), право доступу на виконання для директорій змінює зміст: наявність цього права дозволяє отримати додаткову інформацію про файли, що входять у каталог (їх розмір, хто їх хазяїн, дата створення та т.ін.). Без цього права ви не зможете ні читати вміст файлів, що лежать у директорії, ні модифікувати їх, ні виконувати ці файли. Право на виконання також потрібно для директорії, щоб зробити її `поточною`, а також для всіх директорій на шляху до неї. Право запису для директорії дозволяє змінювати її вміст: створювати та видаляти в ній файли, перейменовувати їх. Відзначимо, що для видалення файлу досить мати права запису та виконання для директорії, до якої входить даний файл, незалежно від прав доступу до самого файлу.

Команда ls з опціями -al. Використання команд chmod і umask

Отримати докладну інформацію про файли в деякій директорії, в тому числі імена хазяїна, групи хазяїв і права доступу, можна за допомогою вже відомої нам команди `ls` з опціями `-al`. У виданій цією командою інформації третій стовпчик ліворуч містить імена користувачів хазяїв файлів, а четвертий стовпчик ліворуч – імена груп хазяїв файлу. Самий лівий стовпчик містить типи файлів і права доступу до них. Тип файлу визначає перший символ у наборі символів. Якщо це символ '`d`', то тип файлу – директорія, якщо там коштує символ '-', те це – регулярний файл. Наступні три символи визначають права доступу для хазяїна файлу, що випливають три – для користувачів, що входять у групу хазяїв файлу, і останні три – для всіх інших користувачів. Наявність символу (`r`, `w` або `x`) для деякої категорії користувачів означає, що дана категорія користувачів має це право.

Викличте команду `ls-al` для своєї `домашньої директорії` та проаналізуйте її інформацію, яку видає ця команда..

Команда chmod

Синтаксис команди

```
chmod [who] { + | - | = } [perm]
    файл1 файл2 ... файл
```

Опис команди

Команда `chmod` призначена для зміни прав доступу до одного або декількох файлів. Чинний опис не є повним описом команди, а адаптованим у відповідності до даного курсу. Для отримання повного опису звертайтеся до UNIX Manual. Права доступу до файлу можуть змінювати тільки власник (хазяїн) файлу або системний адміністратор.

Параметр `who` визначає, для яких категорій користувачів надаються права доступу. Він може являти собою один або кілька символів:

`a` – надання прав доступу для всіх категорій користувачів. Якщо параметр `who` не заданий, то за замовчуванням приймається значення `a`. При визначенні прав доступу з цим значенням задані права встановлюються з урахуванням значення маски створення файлів;

`u` – надання прав доступу для власника файлу;

`g` –надання прав доступу для користувачів, які входять до групи власників файлу;

`o` – надання прав доступу для всіх інших користувачів.

Операція, яка виконується над правами доступу для заданої категорії користувачів, визначається одним з наступних символів:

`+` – додавання прав доступу;
`-` – скасування прав доступу;

=заміна прав доступу, тобто скасування все існуючих і додавання перелічених.

Якщо параметр `perm` не визначений, то всі існуючі права доступу скасовуються.

Параметр `perm` визначає права доступу, які будуть додані, скасовані або

встановлені замість існуючих відповідною командою. Він є комбінацією наступних символів або одного з них:

`r` – право на читання;
`w` – право на модифікацію;

`x` – право на виконання.

Параметри `файл1, файл2, ... файл` – це імена файлів, для яких виконується зміна прав доступу. Замість імен можуть використовуватися файлові шаблони.

Господар файлу може змінювати права доступу до нього, користуючись командою `chmod`.

Створіть новий файл і подивіться на права доступу до нього, які встановлені системою при його створенні. Чим керується операційна система при призначенні цих прав? Вона використовує для цього `маску створення файлів` для програми, яка цей файл створює. Є прийнятним, для кожної програми-оболонки маска має деяке значення за замовчуванням.

Маска створення файлів чинного процесу

Маска створення файлів чинного процесу (`umask`) використовується системними викликами `open()` і `mknod()` при наданні початкових прав доступу для знову створюваних файлів або FIFO. Молодші 9 біт маски створення файлів відповідають правам доступу користувача, який створює файл, групи, до якої він належить, і всіх інших користувачів таким чином, як записано нижче із застосуванням вісімкових значень:

`0400` – право читання для користувача, який створив файл;

`0200` – право запису для користувача, який створив файл;

`0100` – право виконання для користувача, який створив файл;

`0040` – право читання для групи користувача, який створив файл;

`0020` – право запису для групи користувача, який створив файл;

`0010` – право виконання для групи користувача, який створив файл;

`0004` – право читання для всіх інших користувачів;

`0002` – право запису для всіх інших користувачів;

`0001` – право виконання для всіх інших користувачів.

Зміна значення будь-якого біта на `1` забороняє ініціалізацію відповідного права доступу для знову створюваного файлу. Значення маски створення файлів може змінюватися за допомогою системного виклику `umask()` або команди `umask`.

Маска створення файлів успадковується процесом-дитиною при породженні нового процесу системним викликом `fork()` і входить до складу незмінної частини системного контексту процесу при системному виклику `exec()`. У результаті цього спадкування зміна маски за допомогою команди `umask` вплине на атрибути доступу до знову створюваних файлів для всіх процесів, породжених надалі командною оболонкою.

Змінити поточне значення маски для програми-оболонки або подивитися його можна за допомогою команди `umask`.

Команда umask

Синтаксис команди

```
umask [value]
```

Опис команди

Команда `umask` призначена для зміни маски створення файлів командної оболонки або перегляду її чинного значення. При відсутності параметра команда видає значення встановленої маски створення файлів у вісімковому вигляді. Для встановлення нового значення воно задається як параметр `value` у вісімковому вигляді.

Якщо ви хочете змінити його для Midnight Commander, необхідно вийти з `mc`, виконати команду `umask` і запустити `mc` знову. `Маска створення файлів` не зберігається між сеансами роботи в системі. При новому вході в систему значення маски знову буде встановлено за замовчуванням.

Системні виклики getuid і getgid

Розпізнати `ідентифікатор користувача` (що запустив програму на виконання, – `UID`) і `ідентифікатор групи` (до якої він відноситься, – `GID`) можна за допомогою системних викликів `getuid()` і `getgid()`, застосувавши їх усередині цієї програми.

Системні виклики getuid() і getgid()

Прототипи системних викликів

```
#include <sys/types.h>
#include <unistd.h>
uid_t getuid(void);
gid_t getgid(void);
```

Опис системних викликів

`Системний виклик getuid повертає ідентифікатор користувача для чинного процесу.`

`Системний виклик getgid повертає ідентифікатор групи користувача для чинного процесу.`

Типи даних `uid_t` і `gid_t` є синонімами для одного з цілих типів мови С.

Компіляція програм мовою С у UNIX і запуск їх на опрацювання

Тепер ми готові до того, щоб написати першу програму в нашому курсі. Залишилося тільки навчитися компілювати програми мовою С і запускати їх на опрацювання. Для компіляції програм у Linux ми будемо застосовувати компілятор `gcc`.

Для того щоб він нормально працював, необхідно, щоб вихідні файли, які містять текст програми, мали імена, що закінчуються на `.c`.

У найпростішому випадку відкомпілювати програму можна, запускаючи компілятор командою

```
gcc ім'я_вихідного_файлу
```

Якщо програма була написана без помилок, то компілятор створить файл, що виконується, з `ім'ям a.out`. Змінити ім'я створюваного файлу, що виконується,

можна, задавши його за допомогою опції -o:

```
gcc ім'я_вихідного_файлу -o
    ім'я_файлу_що_виконується
```

Компілятор gcc має кілька сотень можливих опцій. Отримати інформацію про них ви можете в UNIX Manual.

«Довідайтеся у свого системного адміністратора», як називається компілятор з мови С для вашої операційної системи й які опції він має. Звичайно у всіх версіях UNIX мається компілятор з ім'ям cc, що підтримує опцію -o.

Запустити програму на виконання можна, набравши ім'я файлу, zrbq виконується, і натиснувши клавішу <Enter>.

Написання, компіляція і запуск програми з використанням системних викликів getuid() і getgid()

Напишіть, відкомпілюйте та запустіть програму, яка друкувала б ідентифікатор користувача, що запустив програму, і ідентифікатор його групи.

Заняття №2-3 (4 аудиторних години)

Процеси в операційній системі UNIX

Поняття процесу в UNIX, його контекст. Ідентифікація процесу. Стани процесу. Коротка діаграма станів. Ієрархія процесів. Системні виклики getpid(), getppid(). Створення процесу в UNIX. Системний виклик fork(). Завершення процесу. Функція exit(). Параметри функції main() у мові С. Змінні середовища й аргументи командного рядка. Зміна користувацького контексту процесу. Сімейство функцій для системного виклику exec().

Поняття процесу в UNIX. Його контекст

Уся побудова операційної системи UNIX заснована на використанні концепції процесів, що обговорювалася на лекції. Контекст процесу складається з користувацького контексту і контексту ядра, як зображено на малюнку 2-3.1.

Під користувацьким контекстом процесу розуміють код і дані, розташовані в адресному просторі процесу. Усі дані підрозділяються на:

- ініційовані незмінні дані (наприклад, константи);
- ініційовані змінювані дані (усі змінні, початкові значення яких привласнюються на етапі компіляції);
- неініційовані змінювані дані (усі статичні змінні, котрим не привласнені початкові значення на етапі компіляції);
- стек користувача;

дані, розташовані в динамічно виділюваній пам'яті (наприклад, за допомогою стандартних бібліотечних С-функцій malloc(), calloc(), realloc()).

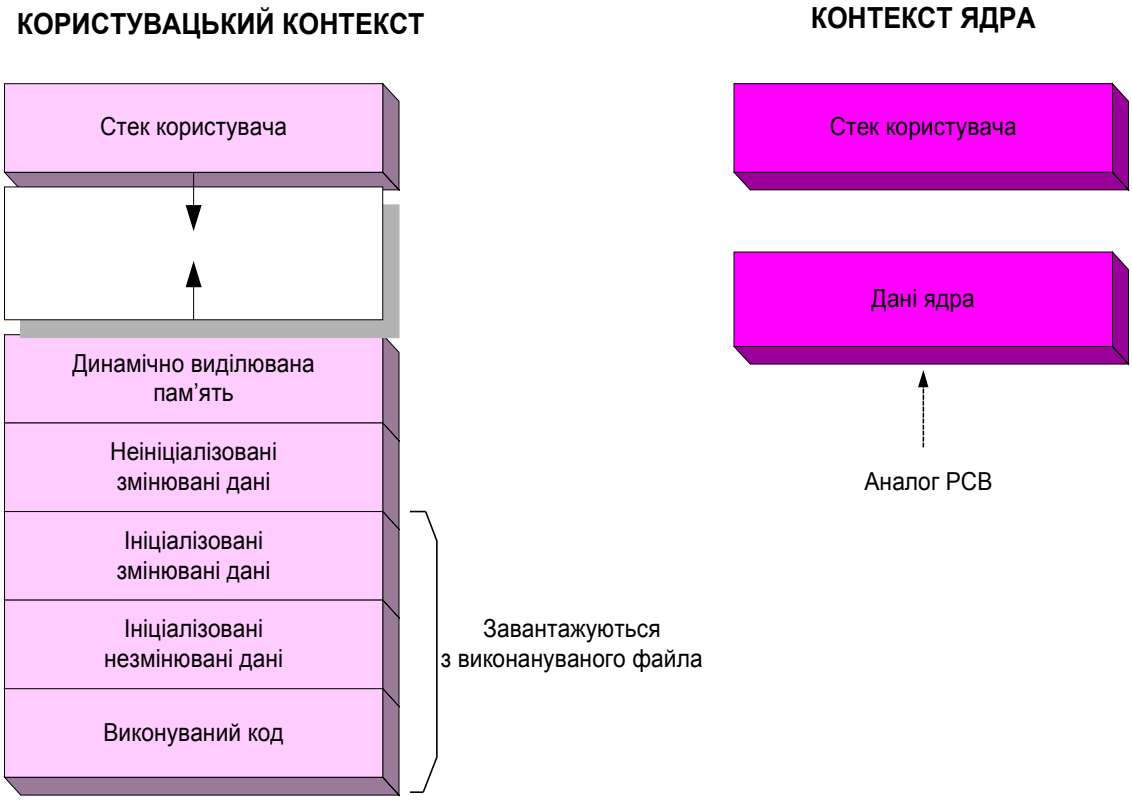
Код, який виконується, й ініційовані дані складають вміст файлу програми, яка виповнюється в контексті процесу. Користувацький стек застосовується при роботі процесу в користувацькому режимі (user-mode).

Під поняттям «контекст ядра» поєднуються системний контекст і реєстровий контекст, розглянуті на лекції. Ми будемо виділяти в контексті ядра стек ядра, який використовується при роботі процесу в режимі ядра (kernel mode), і дані ядра, що зберігаються в структурах, що є аналогом блоку керування процесом — PCB. Склад даних ядра будуть поступово уточнюватися на наступних практичних заняттях. На цьому занятті нам досить знати, що в дані ядра входять:

- ідентифікатор користувача — UID,
- груповий ідентифікатор користувача — GID,

ідентифікатор процесу — PID,

- ідентифікатор батьківського процесу — PPID.



Мал. 2-3.1. Контекст процесу в UNIX

Ідентифікація процесу

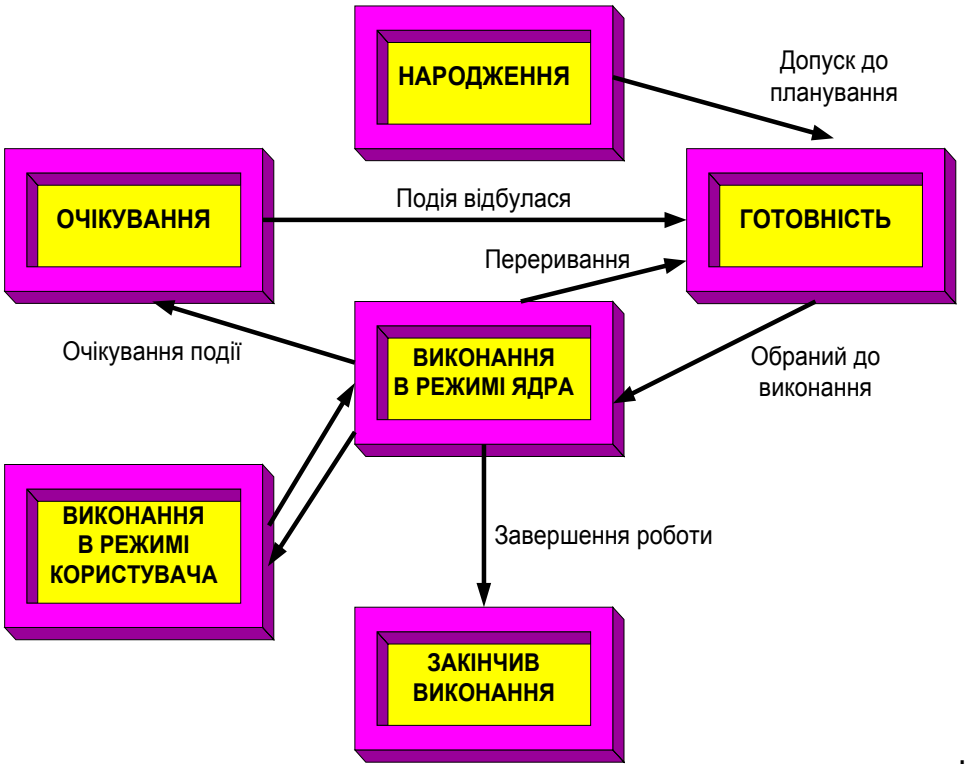
Кожний процес в операційній системі отримує унікальний ідентифікаційний номер – PID (process identificator). При створенні нового процесу операційна система намагається привласнити йому вільний номер більший, ніж у процесу, створеного перед ним. Якщо таких вільних номерів не виявляється (наприклад, ми досягли максимально можливого номера для процесу), то операційна система вибирає мінімальний номер із усіх вільних номерів. В операційній системі Linux присвоєння ідентифікаційних номерів процесів починається з номера 0, який отримує процес `kernel` при старті операційної системи. Цей номер згодом не може бути привласнений ніякому іншому процесу. Максимально можливе значення для номера процесу в Linux на базі 32-розрядних процесорів Intel складає 2^{31}-1.

Стани процесу. Коротка діаграма станів

Модель станів процесів в операційній системі UNIX являє собою деталізацію моделі станів, прийнятої в лекційному курсі. Коротка діаграма станів процесів в операційній системі UNIX зображена на малюнку 2-3.2

Як ми бачимо, стан процесу **виконання** розщеплений на два стани: виконання в режимі ядра та виконання в режимі користувача. до стануі виконання в режимі користувача процес виконує прикладні

інструкції користувача. до стануі виконання в режимі ядра виконуються інструкції ядра операційної системи в контексті поточного процесу (наприклад, при обробці системного виклику або переривання). Зі стану `виконання в режимі користувача` процес не може безпосередньо перейти до стани **очікування**, **готовність** і **закінчив виконання**. Такі переходи можливі тільки через проміжний стан «`виконується в режимі ядра`» . Також заборонений прямий перехід зі стану **готовність** до стану `виконання в режимі користувача`.



Мал. 2-3.2. Скорочена діаграма станів процесу в UNIX

Наведена вище діаграма станів процесів у UNIX не є повною. Вона показує тільки стани, для розуміння яких досить вже отриманих знань. Мабуть, найбільш повну діаграму станів процесів в операційній системі UNIX можна знайти в книзі - Бах М. „Архитектура операционной системы Unix" - http://rsusu1.rnd.runnet.ru/unix/bach/chap06.html (малюнок 6.1.) .

Ієрархія процесів

В операційній системі UNIX усі процеси, крім одного, який створюється при старті операційної системи, можуть бути породжені тільки будь-якими іншими процесами. Як прабатька всіх інших процесів у подібних UNIX системах можуть виступати процеси з номерами `1` або `0`. В операційній системі Linux таким родоначальником, який існує тільки при завантаженні системи, є процес `kernel` з ідентифікатором `0`.

Таким чином, усі процеси в UNIX пов'язані відносинами процес-батько – процес-дитина й утворять генеалогічне дерево процесів. Для збереження цілісності генеалогічного дерева в ситуаціях, коли процес-батько завершує свою роботу до завершення виконання процесу-дитини, ідентифікатор батьківського процесу в дані ядра процесу-дитини (PPID – parent process identificator) змінює своє значення на значення 1, що відповідає ідентифікатору процесу init, час життя якого визначає час функціонування операційної системи. Тим самим процес init як би усиновляє осиротілі процеси. Напевно, логічніше було б заміняти PPID не на значення 1, а на значення ідентифікатора найближчого існуючого процесу-прабатька померлого процесу-батька, але в UNIX чомусь така схема реалізована не була.

Системні виклики getppid() і getpid()

Дані ядра, які знаходяться в контексті ядра процесу, не можуть бути прочитані процесом безпосередньо. Для отримання інформації про них процес повинний зробити відповідний системний виклик. Значення ідентифікатора поточного процесу може бути отримане за допомогою системного виклику getpid(), а значення ідентифікатора батьківського процесу для поточного процесу – за допомогою системного виклику getppid(). Прототипи цих системних викликів і відповідні типи даних описані в системних файлах <sys/types.h> і <unistd.h>. Системні виклики не мають параметрів і повертають ідентифікатор поточного процесу й ідентифікатор батьківського процесу відповідно.

Системні виклики getpid() і getppid()

Прототипи системних викликів

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid(void);
pid_t getppid(void);
```

Опис системних викликів

Системний виклик getpid повертає ідентифікатор поточного процесу.

Системний виклик getppid повертає ідентифікатор процесу-батька для поточного процесу.

Тип даних pid_t є синонімом для одного з цілих типів мови С.

Написання програми з використанням getpid() і getppid()

Як приклад використання системних викликів getpid() і getppid() самостійно напишіть програму, що друкує значення PID і PPID для поточного процесу. Запустите її кілька разів підряд. Подивитеся, як міняється ідентифікатор поточного процесу. Поясните зміни, що спостерігаються.

Створення процесу в UNIX. Системний виклик fork()

В операційній системі UNIX новий процес може бути породжений єдиним способом – за допомогою системного виклику fork(). При цьому знову створений процес буде практично повною копією батьківського процесу. У породженого процесу в порівнянні з батьківським процесом (на рівні вже

отриманих знань) змінюються значення наступних параметрів:

- ідентифікатор процесу – `PID`;
- ідентифікатор батьківського процесу – `PPID`.

Додатково може змінитися поводження породженого процесу стосовно деяких сигналів, про які докладніше ознайомимося на практичних 13 та 14, коли ми будемо говорити про сигнали в операційній системі UNIX.

Системний виклик для породження нового процесу

Прототип системного виклику

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

Опис системного виклику

Системний виклик `fork` служить для створення нового процесу в операційній системі UNIX. Процес, який ініціював системний виклик `fork`, прийнято називати батьківським процесом (`parent process`). Знову породжений процес прийнято називати процесом-дитиною (`child process`). Процес-дитина є майже повною копією батьківського процесу. У породженого процесу в порівнянні з батьківським змінюються значення наступних параметрів:

6. ідентифікатор процесу;

ідентифікатор батьківського процесу;

час, який залишився до отримання сигналу `SIGALRM`;

сигнали, які очікували доставки батьківському процесові, не будуть доставлятися породженому процесові.

При однократному системному виклику повернення з нього може відбутися двічі: один раз у батьківському процесі, а другий раз у породженому процесі. Якщо створення нового процесу відбулося успішно, то в породженому процесі системний виклик поверне значення `0`, а в батьківському процесі – позитивне значення, яке дорівнює ідентифікатору процесу-дитини. Якщо створити новий процес не вдалося, то системний виклик поверне до процесу, який його ініціював, негативне значення.

Системний виклик `fork` є єдиним способом породити новий процес після ініціалізації операційної системи UNIX.

У процесі виконання системного виклику `fork()` породжується копія батьківського процесу та повернення із системного виклику буде відбуватися вже як у батьківському, так і в породженому процесах. Цей системний виклик є єдиним, котрий викликається один раз, а при успішній роботі повертається два рази (один раз у процесі-батьку й один раз у процесі-дитині)! Після виходу із системного виклику обидва процеси продовжують виконання регулярного користувацького коду, що випливає за системним викликом.

Приклад програми з fork() з однаковою роботою батька та дитини

Для ілюстрації вищенаведеного розглянемо наступну програму:

```
/* Програма 03-1.c – приклад створення нового
   процесу з однаковою роботою процесів
   дитину і батька */

#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main()
{
    pid_t pid, ppid;
    int a = 0;
    (void)fork();
    /* При успішному створенні нового процесу
       с цього місця псевдопаралельно
       починають працювати два процеси: старий
       і новий */
    /* Перед виконанням наступного вираження
       значення змінної a в обох процесах
       дорівнює 0 */
    a = a+1;
    /* Довідаємося про ідентифікатори поточного та
       батьківського процесу (у кожному з
       процесів !!!) */
    pid = getpid();
    ppid = getppid();
    /* Друкуємо значення PID, PPID і обчислене
       значення змінної a (у кожному з
       процесів !!!) */
    printf("My pid = %d, my ppid = %d,
      result = %d\n", (int)pid, (int)ppid, a);
    return 0;
}
```

ЗАВДАННЯ 1: Наберіть цю програму, відкомпілюйте її та запустіть на виконання (найкраще це робити не з оболонки mc, тому що вона не дуже коректно звільнює буфери введення-виведення). Проаналізуйте отриманий результат.

Системний виклик fork() (продовження розгляду)

Для того щоб після повернення із системного виклику `fork()` процеси могли визначити, хто з них є дитиною, а хто батьком, і, відповідно, по-різному організувати своє поводження, системний виклик повертає до них різні значення. При успішному створенні нового процесу до процесу-батька повертається позитивне значення, яке дорівнює ідентифікатору процесу-дитини. До процесу-дитини ж повертається значення `0`. Якщо з будь-якої причини створити новий процес не вдалося, то системний виклик поверне до процесу, який його ініціював, значення -1. Таким чином, загальна схема організації різної роботи процесу-дитини і процесу-батька виглядає таким чином:

```
pid = fork();
if(pid == -1){
    ...
    /* помилка */
    ...
} else if (pid == 0){
    ...
    /* дитина */
    ...
} else {
    ...
    /* батько */
    ...
}
```

Написання, компіляція та запуск програми з використанням виклику fork() з різною поведінкою процесів дитини та батька

ЗАВДАННЯ 2: Змініть попередню програму з `fork()` таким чином, щоб батько та дитина виконували різні дії (які – не важливо).

Завершення процесу. Функція exit()

Існує два способи коректного завершення процесу в програмах, написаних мовою С. Перший спосіб ми використовували дотепер: процес коректно завершувався по досягненню кінця функції `main()` або при виконанні оператора `return` у функції `main()`. Другий спосіб застосовується при необхідності завершити процес у деякому іншому місці програми. Для цього використовується функція `exit()` із стандартної бібліотеки функцій для мови С. При виконанні цієї функції відбувається скидання всіх частково заповнених буферів введення-виведення із закриттям відповідних потоків, після чого ініціюється системний виклик припинення роботи процесу та перекладу його до стану **закінчив виконання**.

Повернення з функції до поточного процесу не відбувається та функція нічого не повертає.

Значення параметра функції `exit()` – коду завершення процесу – передається ядру операційної системи та може бути потім отримано процесом, який породив завершений процес. Насправді при досягненні кінця функції `main()` також неявно викликається ця функція із значенням параметра `0`.

Функція для нормального завершення процесу

Прототип функції

```
#include <stdlib.h>
void exit(int status);
```

Опис функції

Функція `exit` служить для нормального завершення процесу. При виконанні

цієї функції відбувається скидання всіх частково заповнених буферів введення-виведення із закриттям відповідних потоків (файлів, ріре, FIFO, сокетів), після чого ініціюється системний виклик припинення роботи процесу та переведення його до стану **закінчив виконання**.

Повернення з функції до чинного процесу не відбувається, і функція нічого не повертає.

Значення параметра `status` – коду завершення процесу – передається ядру операційної системи та може бути потім отримане процесом, який породив завершений процес. При цьому використовуються тільки молодші 8 біт параметра, так що для коду завершення припустимі значення від `0` до `255`. За згодою, код завершення `0` означає безпомилкове завершення процесу.

Якщо процес завершує свою роботу раніш, ніж його батько, і батько явно не вказав, що він не хоче отримувати інформацію про статус `завершення породженого процесу` (про це буде розказано докладніше на практичних заняттях 13 та 14 при вивченні сигналів), то завершений процес, не зникає із системи остаточно, а переходить до стану **закінчив виконання** або до завершення процесу-батька, або до того моменту, коли батько отримає цю інформацію. Процеси, які знаходяться у стані **закінчив виконання**, в операційній системі UNIX прийнято називати зомбі (`zombie`, `defunct`).

Параметри функції main() у мові С. Змінні середовища й аргументи командного рядка

У функції `main()` у мові програмування С існує три `параметри`, що можуть бути передані їй операційною системою. Повний прототип функції `main()` виглядає в такий спосіб:

```
int main(int argc, char *argv[],
         char *envp[]);
```

Перші два `параметри` при запуску програми на виконання командним рядком дозволяють довідатися повний зміст командного рядка. Весь командний рядок розглядається як набір слів, розділених пропусками. Через параметр `argc` передається кількість слів у командному рядку, яким була запущена програма. Параметр `argv` є масивом покажчиків на окремі слова. Так, наприклад, якщо програма була запущена командою

```
a.out 12 abcd
```

то значення параметра `argc` буде дорівнює `3`, `argv[0]` буде вказувати на ім'я програми — перше слово — «`a.out`», `argv[1]` — на слово «`12`», `argv[2]` — на слово «`abcd`». Тому що ім'я програми завжди присутнє на першому місці в командному рядку, то `argc` завжди більше `0`, а `argv[0]` завжди вказує на ім'я запущеної програми.

Аналізуючи в програмі `вміст командного рядка`, ми можемо передбачити її різну поведінку в залежності від слів, які слідкують за ім'ям програми. Таким чином, не вносячи змін у текст програми, ми можемо змусити її працювати в залежності від запуску до запуску. Наприклад, компілятор `gcc`, викликаний командою `gcc 1.c` буде генерувати файл, який виконується, з `ім'ям a.out`, а при виклику командою:

```
gcc 1.c –o 1.exe
```

файл з ім'ям `1.exe`.

Третій параметр – `envp` – є масивом покажчиків на параметри навколишнього середовища процесу. Початкові параметри навколишнього середовища процесу задаються в спеціальних конфігураційних файлах для кожного користувача і встановлюються при вході користувача до системи. Надалі вони можуть бути змінені за допомогою спеціальних команд операційної системи UNIX. Кожний параметр має вигляд: `змінна=рядок`. Такі змінні використовуються для зміни довгострокового поводження процесів, на відміну від `аргументів командного рядка`. Наприклад, призначення параметра `TERM=vt100` може говорити процесам, які здійснюють виведення на екран дисплея, що працювати їм доведеться з терміналом `vt100`. Змінюючи значення `змінного середовища TERM`, наприклад на `TERM=console`, ми повідомляємо таким процесам, що вони повинні змінити свою поведінку та здійснювати виведення для системної консолі.

Розмір масиву `аргументів командного рядка` у функції `main()` ми отримували в якості її параметра. Осільки для масиву посилань на параметри навколишнього середовища такого параметра немає, то його розмір визначається в інший спосіб. Останній елемент цього масиву містить покажчик `NULL`.

Написання, компіляція та запуск програми, яка роздруковує аргументи командного рядка і параметри середовища

ЗАВДАННЯ 3: Самостійно напишіть програму, яка роздруковує значення аргументів командного рядка та параметрів навколишнього середовища для чинного процесу.

Зміна користувацького контексту процесу. Сімейство функцій для системного виклику exec()

Для зміни `користувацького контексту процесу` використовується системний виклик `exec()`, що користувач не може викликати безпосередньо. Виклик `exec()` замінює `користувацький контекст поточного процесу` на вміст деякого виконуваного файлу, і встановлює початкові значення регістрів процесора (у тому числі встановлює програмний лічильник на початок завантаженої програми). Цей виклик вимагає для своєї роботи завдання імені виконуваного файлу, аргументів командного рядка та параметрів навколишнього середовища. Для здійснення виклику програміст може скористатися однієї із шести функцій: `execlp()`, `execvp()`, `execl()` і, `execv()`, `execle()`, `execve()`, які відрізнюються одна від одною представленням параметрів, необхідних для роботи системного виклику `exec()`. Взаємозв'язок зазначених вище функцій зображений на малюнку 2–3.3.

Мал. 2-3.3. Взаємозв'язок різних функцій для виконання системного виклику

Функції зміни користувацького контексту процесу

Прототипи функцій

```
#include <unistd.h>
int execlp(const char *file,
    const char *arg0,
... const char *argN,(char *)NULL)
int execvp(const char *file, char *argv[])
int execl(const char *path,
   const char *arg0,
... const char *argN,(char *)NULL)
int execv(const char *path, char *argv[])
int execle(const char *path,
    const char *arg0,
... const char *argN,(char *)NULL,
    char * envp[])
int execve(const char *path, char *argv[],
    char *envp[])
```

Опис функцій

Для завантаження нової програми до системного контексту поточного процесу використовується родина взаємозалежних функцій, які відрізняються одна від одної формою представлення параметрів.

Аргумент `file` є покажчиком на ім'я файлу, який повинний бути завантажений. Аргумент `path` – це покажчик на повний шлях до файлу, який повинний бути завантажений.

Аргументи `arg0, ..., arg` є покажчиками на аргументи командного рядка. Зауважимо, що аргумент `arg0` повинний показувати на ім'я файлу, який завантажується. Аргумент `argv` являє собою масив з покажчиків на аргументи командного рядка. Початковий елемент масиву повинний показувати на ім'я програми, яка завантажується, а закінчуватися масив повинен елементом, що містить `покажчик` NULL.

Аргумент `envp` є масивом покажчиків на параметри навколишнього середовища, задані у виді рядків «`змінна=рядок`». Останній елемент цього масиву повинний містити покажчик `NULL`.

Оскільки виклик функції не змінює системний контекст поточного процесу, завантажена програма успадкує від завантажуючого процесу наступні атрибути:

- ідентифікатор процесу;

ідентифікатор батьківського процесу;

груповий ідентифікатор процесу;

ідентифікатор сеансу;

час, який залишився до виникнення сигналу `SIGALRM`;

поточну робочу директорію;

маску створення файлів;

ідентифікатор користувача;

груповий ідентифікатор користувача;

явне ігнорування сигналів;

таблицю відкритих файлів (якщо для файлового дескриптора не встановлювалася ознака «закрити файл при виконанні `exec()`»).

У випадку успішного виконання повернення з функцій у програму, що здійснила виклик, не відбувається, а керування передається завантаженій програмі. У випадку невдалого виконання в програму, яка ініціювала виклик, повертається негативне значення.

Оскільки системний контекст процесу при виклику `exec()` залишається практично незмінним, більшість атрибутів процесу, доступних користувачу через системні виклики (`PID`, `UID`, `GID`, `PPID` та інші, зміст яких стане зрозумілий з поглибленням наших знань на подальших заняттях), після запуску нової програми також не змінюється.

Важливо розуміти різницю між системними викликами `fork()` і `exec()`. Системний виклик `fork()` створює новий процес, у якого користувацький контекст збігається з користувацьким контекстом процесу-батька. Системний виклик `exec()` змінює користувацький контекст поточного процесу, не створюючи нового процесу.

Приклад програми з використанням системного виклику exec()

Для ілюстрації використання системного виклику `exec()` розглянемо наступну програму

```
/* Програма 03-2.c, що змінює користуваць-
кий контекст процесу (який запускає
   іншу програму) */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[],
         char *envp[]){
/* Ми будемо запускати команду cat с аргументом
   командного рядка 03-2.c без зміни
   параметрів середовища, тобто фактично виконуючи
   команду "cat 03-2.c", яка повинна видати
   уміст даного файлу на екран. Для
   функції execle у якості імені програми
   ми вказуємо її повне ім'я із шляхом від
   кореневої директорії —/bin/cat.

   Перше слово в командному рядку в нас
   повинно збігатися з ім'ям запущеної
   програми. Друге слово в командному рядку
```

```
   – це ім'я файлу, вміст якого ми
   хочемо роздрукувати. */

(void) execle("/bin/cat", "/bin/cat",
              "03-2.c", 0, envp);

/* Сюди попадаємо тільки при
   виникненні помилки */
printf("Error on program start\n");
exit(-1);
return 0;      /* Ніколи не виконується, потрібний
                  для того, щоб компілятор не
                  видавав warning */
}
```

<u>ЗАВДАННЯ 4</u>: Відкомпілюйте наведену вище програму та запустіть на виконання. Оскільки при нормальній роботі буде роздруковуватися вміст файлу з ім'ям 03-2.c, такий файл при запуску повинен бути присутнім у поточній директорії (найпростіше записати вихідний текст програми під цим ім'ям). Проаналізуйте результат.

Написання, компіляція та запуск програми для зміни користувацького контексту в породженому процесі

<u>ЗАВДАННЯ 5</u>: Для закріплення отриманих знань модифікуйте програму, створену при виконанні завдання роздягнуло *«Написання, компіляція та запуск програми з використанням виклику fork() з різним поводженням процесів дитини та батька»* таким чином, щоб породженим процесом була запущена на виконання нова (будь-яка) програма.