МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Кафедра обчислювальної техніки

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з програмного модулю

“ СИСТЕМНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ”

Для студентів напрямку підготовки 6.050102 (123) «Комп’ютерна інженерія»,

Розробник:
Сімоненко Валерій Павлович, д.т.н., професор
Сімоненко Андрій Валерійович, старший викладач

Затверджено на засіданні кафедри
Протокол № 1 від 30 серпня 2017 р.

Завідувач кафедри ОТ
_____ Стіренко С.Г.
(підпис) (прізвище, ініціали)

Київ – 2017.

Системне програмне забезпечення. Методичні вказівки до виконання лабораторних робіт
Для студентів напрямку підготовки 6.050102 « Комп’ютерна інженерія», Професійного спрямування « Комп’ютерні системи та мережі» денної та заочної форми навчання/ укладачі: В.П. Сімоненко, А.В. Сімоненко – К.: НТУУ «КПІ» ,2017. – 83 с.

Навчальне електронне видання

“ Системне програмне забезпечення ”
Методичні вказівки
до виконання лабораторних робіт
Для студентів напрямку підготовки 6.050102 «Комп’ютерна інженерія»
Професійного спрямування « Комп’ютерні системи та мережі»

СПЗ (7 семестр). Лабораторна робота № 2-1

Найпростіший синтаксичний аналізатор (парсер)

Коротка теорія

Задача синтаксичного аналізатора (парсера) полягає у перевірці відповідності вхідного файлу допустимому формату і у виявленні з символів вхідного файлу синтаксичних конструкцій. Входом для парсера є потік символів і визначення формату синтаксису. Виходом для парсера є розпізнані синтаксичні конструкції.

Парсер можна побудувати декількома способами. Один із способів - це розбір вхідного файлу за допомогою автомата. Символ із вхідного файлу і поточний стан автомата визначають дію і наступний стан автомата. Цей процес можна відобразити наступним чином:

Автомат (символ, стан) → (дія, стан)

Перевага такого методу побудови парсера в тому, що швидкість розбору вхідного файлу не залежить від складності перевіряємого формату. Логіка роботи автомата однакова як для простих, так і для складних форматів. Застосування автоматів дозволяє створювати уніфікований інтерфейс для розбору файлів.

Автомат можна представити таблицею. Координата стовпця таблиці відповідає номером стану автомата, а координата рядка таблиці відповідає коду символу. Немає необхідності створювати рядки для кожного можливого символу, так як деякі символи вимагають однакової обробки. Для цього кожному символу призначається клас символу (деяке число). Метод перекодування символів у класи символів залежить від кількості класів.

Якщо синтаксис вхідного файлу призводить до створення великого автомата, то замість одного автомата можна побудувати дерево автоматів. Спочатку створюються і відлагоджуються невеликі автомати для розбору

деяких частин синтаксису. Потім створюється головний автомат, який зв'язує в дерево всі автомати. Використання дерева автоматів дозволяє використовувати вже готові автомати при побудові нових.

Завдання на роботу

Розробити інфраструктуру (структури даних, константи, набір функцій, методів, класів, порядок роботи і т. п.) для побудови парсера на основі роботи автоматів.

Необхідні умови:

a. Символи вхідного файлу можуть бути в будь-якому кодуванні (один символ займає не обов'язково 1 байт), вхідний файл складається з будь-якої кількості символів;

b. Один парсер може складатися з декількох автоматів, завжди є один головний автомат, з головного автомата можна переходити в інші автомати парсера

і т. д. вниз по дереву;

c. Кожен автомат управляється одним вхідним символом, поточним станом автомата і можливо поточним контекстом розбору;

d. Повинна бути функція / метод отримання чергової синтаксичної кон-струкції з вхідного файлу або отримання інформації про помилку розбору.

Використовуючи розроблену інфраструктуру створити парсери для наступних вхід-них файлів:

1. Вхідний файл складається з email адрес розділених пробілами або символами перекладу рядка, синтаксична конструкція - це ім'я і домен. Email адреса складається з імені, символу '@' і доменного імені. Ім'я складається з букв англійського алфавіту, цифр та символу підкреслення. Доменне ім'я складається з одного або декількох імен піддоменів, розділених символом '.' (Крапка). Ім'я піддомену складається з однієї або декількох букв англійського

алфавіту, цифр та символу "-" (мінус), повинно починатися і закінчуватися буквою або цифрою.

2. Вхідний файл має наступний синтаксис:

a. Вхідний файл складається з коментарів, секцій та параметрів, використовуються ASCII символи.

b. Коментар може починатися з символу '#' і закінчуватися символом нового рядка або кінцем файлу.

c. Коментар може починатися з послідовності символів / * та за-закінчуються послідовністю символів * /.

d. Коментарі можуть бути розташовані в будь-якому місці вхідного файлу.

e. Параметр починається з імені параметра, далі йде символ '=', потім значення параметра, значення параметра завершується символом ';'.

f. Секція починається з імені секції, далі йде опціональне значення, яке завершується символом "{', секція завершується символом'} '. Всередині секції можуть бути коментарі, параметри і секції.

g. Значення можуть складатися з будь-яких символів, розділених пробілами і символами нового рядка.

h. Значення можуть містити рядки, укладені в подвійні лапки. Подвійні лапки в рядку кодуються двома символами \ ", символ '\' кодується - \ \, символ нового рядка кодується - \ n і символ табуляції кодується - \ t. Будь-який інший символ після '\' вважається помилкою.

i. Праворуч і ліворуч від '=' після імені параметра, праворуч і ліворуч від ';' в кінці значення параметра, праворуч і ліворуч від "{'i'} ' в секції і між ім'ям секції і опціональним значенням допускається будь-яка кількість пробілів символів нового рядка.

j. Формат значень не обмовляється. Ім'я параметра або секції повинно починатися з літери, далі можуть йти букви, цифри і символ підкреслення.

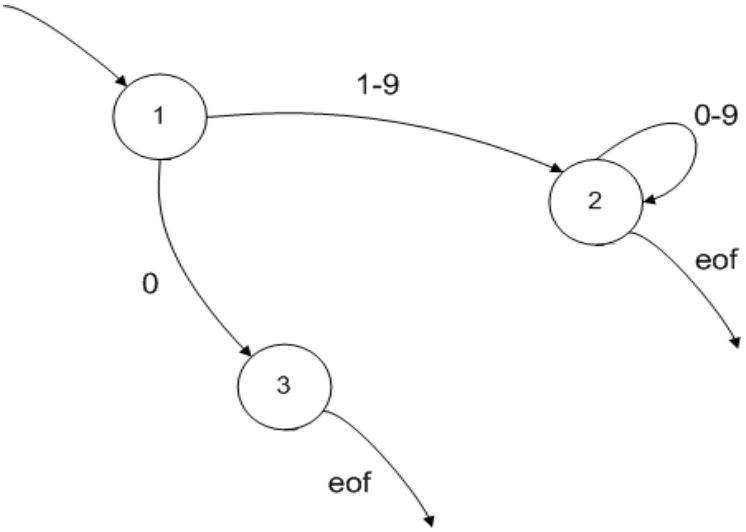
Розбір синтаксису в пунктах b, c, h необхідно реалізувати в окремих автоматах.

Синтаксична конструкція - це тип конструкції (початок секції, кінець секції, параметр) і, залежно від типу конструкції, ім'я секції плюс опціональне значення, нічого, ім'я параметра плюс значення.

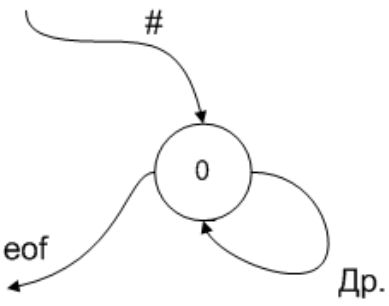
Якщо в значенні присутні кілька пробілів, символів табуляції або символів перекладу рядка, тоді їх необхідно замінити одиночним пропуском.

Приклади:

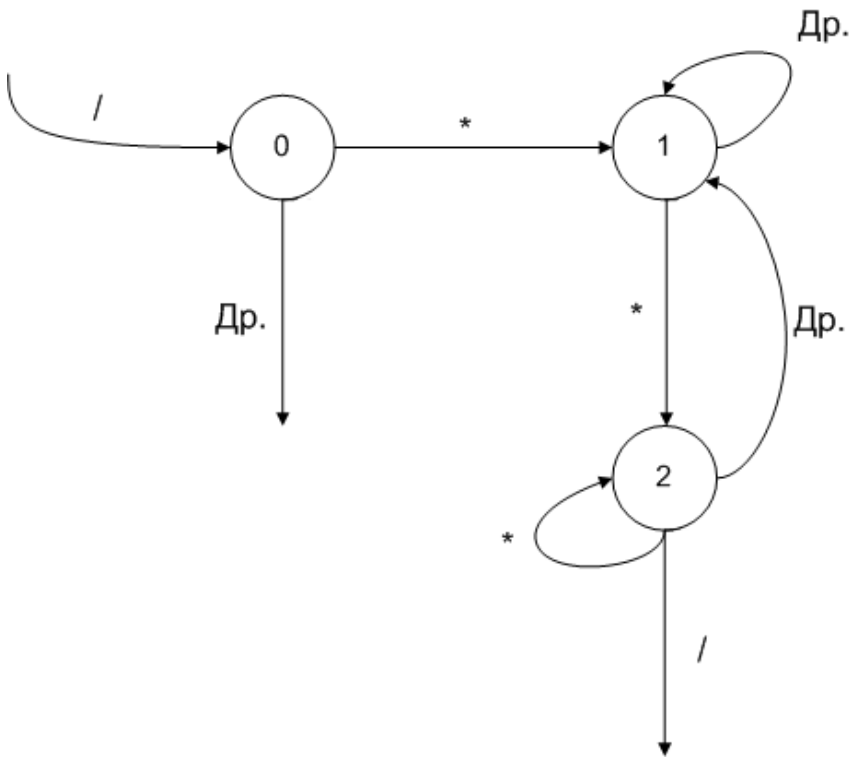
1) Граф автомата для одного цілого десяткового числа:



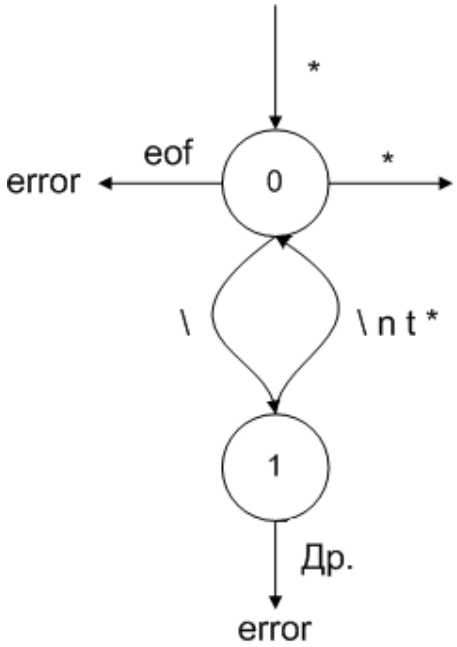
2) Граф автомата для однорядкового коментаря:



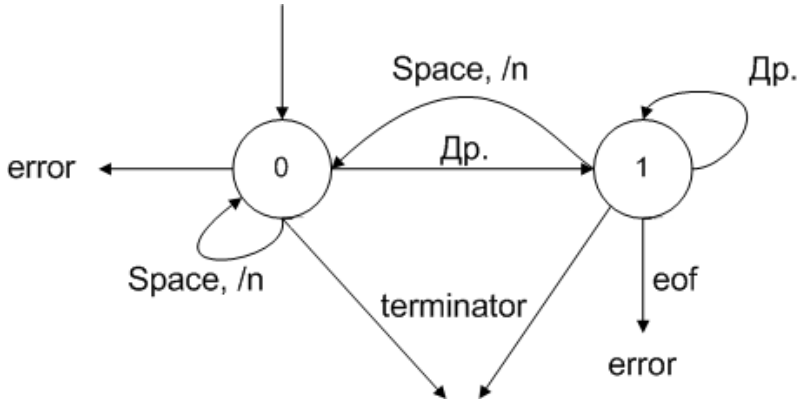
3) Граф автомата для багаторядкового коментаря:



4) Граф автомата для рядка:



5) Граф автомата для значення (як узагальнення замість “;” чи “{“ в якості завершувача значення використовується “terminator”):



Обмеження при реалізації

При виконанні цієї роботи не можна використовувати генератори лексичних і синтаксичних аналізаторів, регулярні вирази і т. п.

Звіт

Звіт повинен містити:

1. Опис ідеї розробленої інфраструктури.
2. Графічне представлення розробленої інфраструктури.
3. Графічне подання автоматів для кожного синтаксису.
4. Лістинг розробленої інфраструктури.
5. Лістинг двох парсерів.
6. Приклад роботи парсера для кожного синтаксису.

Електронна версія(мережа КПІ)

ftp://vt513.comsys.ntu-kpi.kiev.ua/pub/edu/spo2/lab2-1.doc

Література

1. Ахо, Сети, Ульман. *Компиляторы: принципы, технологии и инструменты*.

Лабораторна робота №2-2

АЛГОРИТМИ ЗАМІЩЕННЯ СТОРІНОК ВІРТУАЛЬНОЇ ПАМ'ЯТІ

Короткі теоретичні відомості

Однією із задач віртуальної пам'яті є надання програмам адресних просторів пам'яті, що перевищують обсяг фізичної пам'яті. Така організація роботи з пам’яттю досягається шляхом створення карт відображення віртуальних сторінок у фізичні сторінки для кожного процесу. Одна сторінка віртуальної відображається в одну або декілька фізичних сторінок. Карта відображення віртуальних сторінок у фізичні сторінки складається із записів. Кожен запис має містити, як мінімум, адресу фізичної сторінки, права доступу та біт присутності. Зазвичай кожен запис також містить біт модифікації і біт звернення.

Так як кількість сторінок фізичної пам'яті менше ніж кількість сторінок віртуальної пам'яті, виникає необхідність заміщення відображених сторінок сторінками необхідними для продовження роботи процесів. Складність прийняття рішення, яку сторінку можна перемістити в зовнішній файл (своп), викликана тим, що інформація про подальші звернеенях до пам'яті процесами відсутня. Однак можна побудувати ефективні алгоритми заміщення сторінок, оскільки робота процесів характеризується принципом локальності: в деякий проміжок часу будь-який процес звертається до обмеженого набору сторінок, які складають робочий набір у цей проміжок часу.

Алгоритм заміщення сторінок накопичує статистику звернення до пам'яті (досліджуються біт обігу та/або біт модифікації кожної сторінки) і при необхідності вибирає "саму непотрібну сторінку", переміщує її вміст у зовнішній файл (своп) і віддає звільнену фізичну сторінку для нового відображення.

Щоб дати можливість операційній системі збирати корисні статистичні дані про те, які сторінки використовуються, а які - ні, більшість комп'ютерів з віртуальною пам'яттю підтримують два статусних бита, пов'язаних з кожною сторінкою. Біт R (від Referenced - звернення) встановлюється кожен раз, коли відбувається звернення до сторінки (читання або запис). Біт M (від Modified - зміна) встановлюється, коли сторінка записується (тобто змінюється).

Алгоритм NRU

Алгоритм NRU (Not Recently Used) видаляє сторінку за допомогою випадкового пошуку в не порожньому класі з найменшим номером. Мається на увазі, що краще вивантажити змінену сторінку, до якої не було звернень, хоча б протягом одного такту системних годин (зазвичай 20 мс), ніж стерти часто використовувану.

Коли процес запускається, обидва сторінкових біта (R і M) для всіх його сторінок операційною системою скинуті в 0. Періодично (наприклад, при кожному перериванні по таймеру) біт R очищається з метою відрізнити сторінки, до яких давно не було звернення, від тих, на які посилання були.

При порушенні сторінкового переривання операційна система перевіряє всі сторінки і ділить їх на чотири категорії на підставі поточних значень бітів R і M:

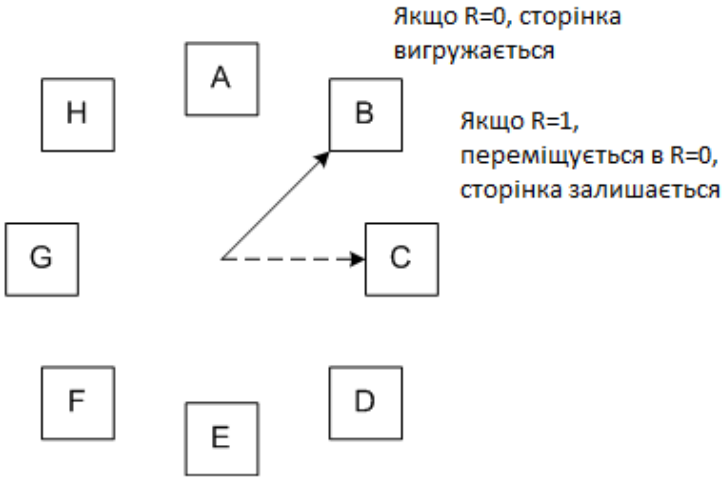
- клас 0: не було звернень і змін (R = 0, M = 0);
- клас 1: не було звернень, сторінка змінена (R = 0, M = 1);
- клас 2: було звернення, сторінка не змінена (R = 1, M = 0);
- клас 3: сталося і звернення, і зміна (R = 1, M = 1).

Хоча клас 1 на перший погляд здається неможливим, таке трапляється, коли у сторінки з класу 3 біт R скидається під час переривання по таймеру. Переривання по таймер не затирають біт M, оскільки ця інформація необхідна для того, щоб знати, чи потрібно переписувати сторінку на диску чи ні. Тому якщо біт R встановлюється на нуль, а M залишається недоторканим, сторінка потрапляє в клас 1.

Алгоритм «годинник»

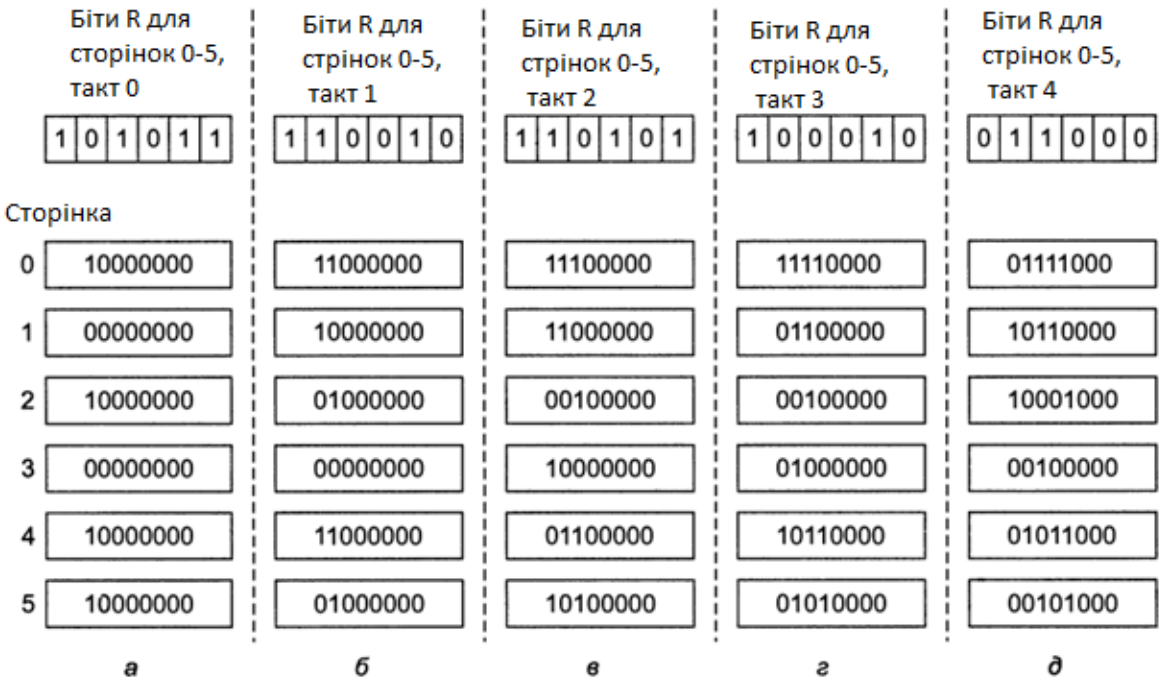
Щоб уникнути переміщення сторінок по списку, можна використовувати покажчик, який переміщається по списку.

Коли відбувається сторінкове переривання, перевіряється та сторінка, на яку спрямована стрілка. Якщо її біт R дорівнює 0, сторінка вивантажується, на її місце в часовому колі стає нова сторінка, а стрілка зсувається вперед на одну позицію. Якщо біт R дорівнює 1, він скидається, стрілка переміщається до наступної сторінки. Цей процес повторюється до тих пір, поки не знаходиться та сторінка, у якій біт R = 0.



Алгоритм старіння "aging", модифікований NFU.

Алгоритм старіння є модифікацією алгоритму NFU, який, у свою чергу, є різновидом алгоритму LRU. Змінення зводяться до двох модифікацій. По-перше, кожен лічильник зсувається вправо на один розряд перед додаванням біта R. По-друге, біт R вдвигается в крайній зліва, а не в крайній праворуч розряд лічильника.



На рисунку продемонстровано, як працює алгоритм «старіння». Припустимо, що після першого тіку годин біти R для сторінок від 0 до 5 мають значення 1, 0, 1,0, 1, 1 відповідно (у сторінки 0 біт R дорівнює 1, у сторінки 1 - R = 0, у сторінки 2 - R = 1 і т. д.). Іншими словами, між тіком 0 і тіком 1 відбулося звернення до сторінок 0, 2, 4 і 5, їх біти R взяли значення 1, решта зберегли значення 0. Після того як шість відповідних лічильників зрушилися на розряд і біт R зайняв крайню зліва позицію, лічильники отримали значення, показані на рис. а. Інші чотири колонки рисунка зображують шість лічильників після наступних чотирьох тіків годин.

Коли відбувається сторінкове переривання, віддаляється та сторінка, лічильник якої має найменшу величину. Ясно, що лічильник сторінки, до якої не було звернень, скажімо, за чотири тіка, буде починатися з чотирьох нулів і, таким чином, мати більш низьке значення, ніж лічильник сторінки, на яку не посилалися протягом лише трьох тіків годин.

У цьому алгоритмі лічильник має кінцеву кількість розрядів.

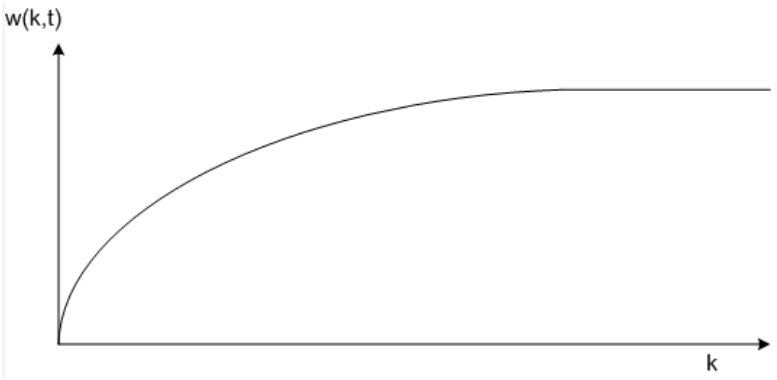
Алгоритм "робочий набір"

Заміщення сторінок за запитом - коли сторінки завантажуються на вимогу, а не заздалегідь, тобто процес переривається і чекає завантаження сторінки.

Буксування - коли кожну наступну сторінку доводиться процесу завантажувати в пам'ять.

Щоб не відбувалося частих переривань, бажано щоб часто запитувані сторінки завантажувалися заздалегідь, а решта довантажувати за потребою.

Множина сторінок (k), яку процес використовує в даний момент (t), називається *робочим набором*. Тобто можна записати функцію w (k, t).



Тобто робочий набір виходить в насичення, значення w (k, t) в режимі насичення може служити для робочого набору, який необхідно завантажувати до запуску процесу.

Алгоритм полягає в тому, щоб визначити робочий набір, знайти і вивантажити сторінку, яка не входить в робочий набір.

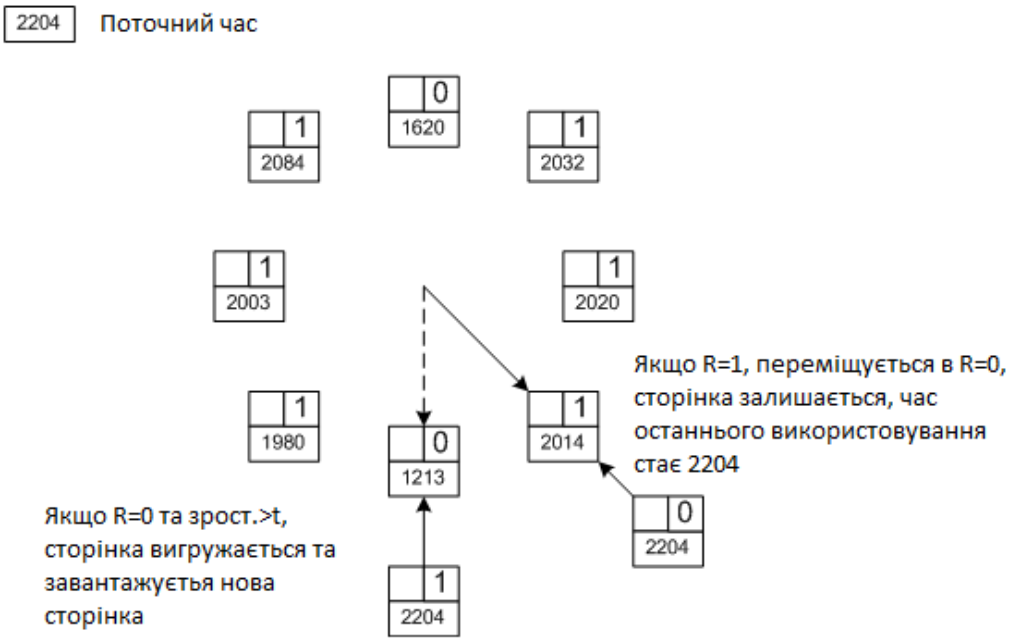
Для реалізації моделі робочого набору необхідно, щоб операційна система відстежувала, які сторінки в ньому знаходяться. Один зі способів отримати таку інформацію - використовувати описаний вище алгоритм старіння. Нехай встановлений старший біт лічильника сторінки означає, що вона входить в робочий набір. Якщо протягом n послідовних тактів годин до такої сторінці не було зроблено звернень, вона викидається з робочого набору. Параметр n доведеться визначити експериментальним шляхом, але продуктивність системи зазвичай не надто чутлива до його точного значення.

Алгоритм WSClock

Алгоритм заснований на алгоритмі "годинник", але використовує робочий набір. Зазвичай, коли «стрілка годинника» показує на сторінку, у якій біт R дорівнює нулю, ця сторінка видаляється. Щоб поліпшити

алгоритм, можна додатково перевіряти, чи входить сторінка в робочий набір поточного процесу, і якщо так, залишати її.

В алгоритмі використовуються біти R та M, а також час останнього використання.



Вихідні дані

В системі є N сторінок фізичної пам'яті. Кожній віртуальній сторінці відповідає рівно одна фізична сторінка.

В системі є декілька процесів. Кожен процес має власний віртуальний адресний простір, число сторінок в якому може бути як менше, так і більше ніж N. Будь-яка фізична сторінка може бути відображена у віртуальному адресному просторі лише одного процесу.

Кожен процес періодично звертається до якоїсь своєї сторінки. Це може бути як нова сторінка, так і та до якої було звернення в минулий раз, але всі процеси характеризуються локальністю звернень (90/10). Звернення може бути читанням або записом (50/50).

В системі використовується один з алгоритмів заміщення сторінок. Сторінки заміщуються за вимогою (demand paging). У системі є таймер і за необхідності можуть бути запущені додаткові системні процеси, наприклад, фоновий процес для різних перевірок стану пам'яті.

Для кожної віртуальної сторінки в карті відображення є адреса відповідної фізичної сторінки, біт присутності, біт модифікації та біт звернення. Біти прав доступу не використовуються.

У системі присутній своп, який реалізовувати не потрібно, але необхідно розрізняти сторінки, переміщені в своп від сторінок, до яких ще не було звернень.

Завдання

Розробити модель системи із сторінковою організацією пам'яті, що задовольняє наведені вище вихідні дані, з реалізацією одного з алгоритмів заміщення сторінок.

Моделююча програма повинна в стандартний потік виводу і в лог-файл виводити звіт про характеристики віртуальної пам'яті процесів, сторінкових перериваннях, звіт про рішення алгоритму заміщення сторінок і т. п.

Алгоритм заміщення сторінок вибирається за варіантом (визначається за номером залікової книжки, останні_дві_ціфри mod 5 + 1):

1. NRU, сторінка, що останнім часом не використовувалась.
2. Алгоритм "годинник", модифікований алгоритм "друга спроба".
3. Алгоритм старіння "aging", модифікований NFU.
4. Алгоритм "робочий набір".
5. Алгоритм WSClock.

Політику розподілу пам'яті (локальна або глобальна) вибрати самостійно.

Звіт

1. Опис алгоритму заміщення сторінок.
2. Опис структур даних, що представляють віртуальну пам'ять, із зазначенням залежності по даним.
3. Лістинг основної частини програми.
4. Висновки: опис переваг і недоліків реалізованого алгоритму заміщення сторінок.

Електронна версія (мережа КПІ)

ftp://vt513.comsys.ntu-kpi.kiev.ua/pub/edu/spo2/lab2-2.doc

СПЗ (7 семестр). Лабораторні роботи №3, 4

Файлова система для флеш-пам’яті типу NAND

Теорія

Флеш-пам'ять типу NAND (далі «флеш-пам'ять») має властивості, які відрізняють цей тип пам'яті від оперативної пам'яті, магнітних дисків і CD-ROM.

Флеш-пам’ять складається з блоків, кожен блок складається із сторінок (зазвичай розміром 512 байт), кожна сторінка може мати до 16 байт додаткової пам’яті, яку можна використовувати для зберігання контрольної суми.

Читання даних з флеш-памяті відбувається посторінково, обмежень на кількість зчитувань даних з однієї сторінки немає. Зміна одного біта зі значення 1 в значення 0 називається програмуванням (programming). Програмування флеш-пам’яті відбувається посторінково. Деякі моделі флеш-пам’яті дозволяють програмувати дані в одній сторінці до 10 разів, після чого вміст сторінки стане невизначеним. Змінити один біт із значення 0 в значення 1 неможливо, але можна очистити (erase) увесь блок, тим самим встановивши значення усіх бітів блоку в 1. Кількість циклів запису-очищення обмежена і дорівнює 100.000 для більшості моделей флеш-пам’яті.

Файлові системи, розроблені для магнітних дисків, не можна застосовувати для флеш-пам’яті, оскільки є обмеження на число циклів запису-очищення одного блоку. З цієї причини для флеш-пам’яті необхідно застосовувати спеціально розроблені файлові системи, які дозволяють рівномірно розподілити цикли запису-очищення по усіх блоках флеш-

пам’яті, а так само мінімізувати кількість циклів запису-очищення блоків. Альтернативою є апаратне рішення, в якому підтримується таблиця відповідності між номером логічного блоку і номером фізичного блоку, при записі даних відповідність між блоками міняється, щоб рівномірно розподілити цикли запису-очищення на усі наявні фізичні блоки.

При проектуванні файлових систем для флеш-пам’яті використовують метод журналювання або ведення історії або логу змін у файловій системі. Записи змін у файловій системі оформляються в один блок, блок очищається і потім записується цілком, оскільки можна провести очищення тільки цілого блоку і різні моделі флеш-пам’яті гарантують різне число гарантованих циклів програмування однієї сторінки. Якщо запис змін у файловій системі перевищує розмір одного блоку, то цей запис розділяється і записується в декілька блоків.

Історія змін у файловій системі повинна дозволити реконструювати стан файлової системи при підключенні флеш-пам’яті. При підключенні (монтуванні) флеш-пам’яті драйвер файлової системи зчитує вміст заголовків усіх записів змін і складає представлення про файли в оперативній пам'яті. При зміні файлової системи запису зі змінами зберігаються у флеш-пам’яті і так само відбиваються в представленні файлової системи в оперативній пам'яті.

Кожен запис про зміну у файловій системі складається з метаданих і відповідних даних. Метадані описують зміну (напр. ідентифікатор файлу, код зміни, зміщення від початку файлу і розмір даних), дані містять самі зміни. Щоб відновити образ файлової системи по історії змін кожен запис для кожного файлу повинен мати номер версії. Неважливо в якому порядку будуть прочитані заголовки змін при монтуванні файлової системи, оскільки по номеру версії завжди можна визначити послідовність їх застосування.

Кожен блок флеш-пам’яті знаходиться в одному з трьох станів. Блок називається «вільним» якщо в ньому не зберігається жоден запис історії змін (такий блок готовий до очищення або вже очищений). Блок називається

«чистим» якщо усі записи історії змін записані в ньому містять дійсні дані (розмір даних в такому блоці не можна зменшити). Блок називається «брудним» якщо в ньому є хоча б один запис історії змін, яка повністю або частково містить недійсні дані (така ситуація виникає якщо подальші зміни для цього ж файлу оновлюють зміни описані в цьому записі). Розмір даних у брудному блоці можна зменшити, оскільки частина даних вже недійсна.

При створенні нової файлової системи уся флеш-пам’ять складається з вільних блоків, які драйвер файлової системи використовує послідовно для збереження історії змін файлів. У якийсь момент часу кількість вільних блоків скоротиться і виникне необхідність створення вільних блоків. Вільні блоки можна отримати за допомогою зчитування декількох брудних блоків, знищення недійсних даних в них, об'єднання записів (сумарний об'єм яких буде меншим ніж початковий об'єм), що залишилися, збереження отриманих записів змін у вільних блоках, при цьому раніше брудні блоки стають вільними.

Процес запису історії змін можна представити таким чином. Один потік записує історію змін послідовно на флеш-пам’ять. Інший потік стежить за кількістю вільних блоків, що залишилися, коли кількість вільних блоків стає дорівнює деякому граничному значенню запускається процес чистки. Процес чистки об'єднує записи у брудних блоках і переміщує їх, а також переміщує чисті блоки. Переміщення чистих блоків потрібне для рівномірного розподілу кількості циклів запису-очищення по усіх блоках флеш-пам’яті. Очевидно, що завжди необхідно мати деяку кількість вільних блоків, тому драйвер файлової системи не повинен дозволити повністю заповнити флеш-пам’ять даними.

Wear-Leveling

Wear-Leveling (урівномірювання зносу) – це технологія продовження терміну служби флеш-пам’яті.

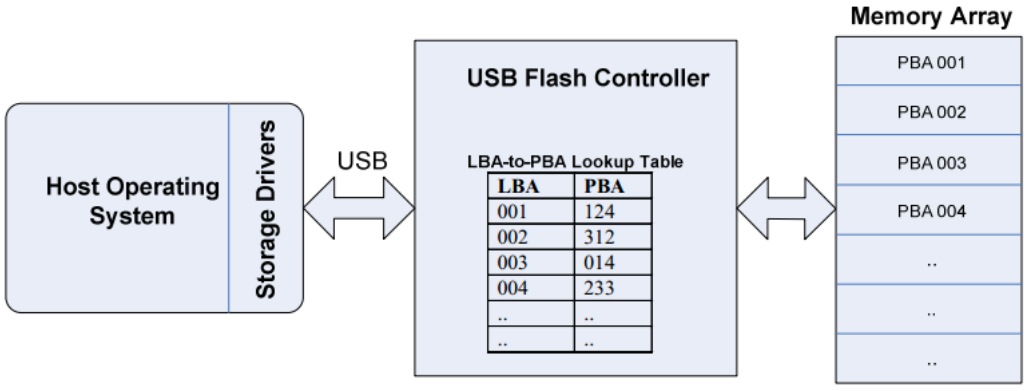
Дані можуть записуватись на флеш-пам’ять кінцеву кількість разів. Не зважаючи на те, що допустима кількість перезаписувань є великою (зазвичай 10 000 або 100 000), якщо запис даних здійснюється на одному і тому ж місці знову і знову, логічно припустити, що пам’ять зношуватиметься в цьому місці. Для уникнення цього здійснюється «урівномірювання зносу» (Wear-Leveling), тобто дані рівномірно розміщуються по всім блокам флеш-пам’яті. Цей процес зменшує сумарний знос пам’яті, тим самим збільшуючи термін її служби.

Щоб зрозуміти як здійснюється Wear-Leveling, слід знати наступні терміни:

LBA (Logical Block Address) – адреса, яка використовується операційною системою для читання або запису блоку даних на флеш-пам’ять.

PBA (Physical Block Address) – фіксована фізична адреса блоку даних на флеш-пам’яті.

Контролер флеш-пам’яті – мікросхема, яка знаходиться разом із флеш-пам’яттю, і предоставляє таблицю відповідностей PBA та LBA (тобто за якою PBA знаходяться дані, що мають відповідний LBA)



Таким чином, таблиця відповідностей PBA та LBA – це щось схоже на зміст книги. Дані можуть бути фізично переміщені, і все, що потрібно буде

зробити - це змінити значення таблиці відповідностей, після чого дані все ще можна буде з легкістю знайти.

Так можна переміщувати дані по всій пам’яті практично без втрат, контролер постійно цим і займається. Обновлені або нові дані записуються у перший-ліпший вільний блок із найменшою кількістю операцій запису. Блок, що містить старі дані стирається у фоновому режимі і позначається як вільний блок. Такий «блокооборот» забезпечує рівномірне зношення блоків пам’яті. Wear-Leveling – це прозорий процес для операційної системи.

Нижче наводиться дуже спрощений приклад.

На початку, у нас є дані за адресою PBA з #1 по #4.

PBA #5 та #6 є "вільними", або порожніми.

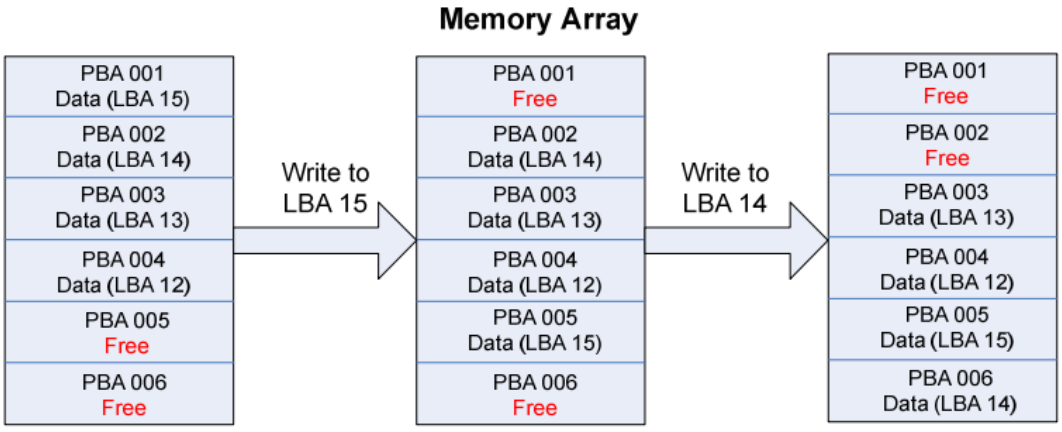


Figure 2

Коли комп'ютер записує нові дані до LBA15, контролер записує їх у PBA #5, а не в PBA #1, якому раніше відповідав LBA15. PBA #1 тепер став порожнім.

Після цього ми будемо записувати LBA14. Як і у випадку із LBA15, дані записуються в новому PBA. Таким чином PBA #2 став вільним.

Допустим, ми записуємо LBA14 ще раз. Знову ж таки, LBA14 тепер матиме нову фізичну адресу.

Статичний та динамічний Wear-Leveling

Як правило, на флеш-пам’яті зберігаються як статичні (записуються лише раз), так і динамічні дані (постійно змінюються і перезаписуються, наприклад, LOG-файли, бази даних і т.д)

Перепризначення статичних даних є більш складною задачею, ніж динамічних даних, оскільки перше вимагає кілька операцій для безпечного руху статичних даних у флеш-пам’яті. Це може сильно вплинути на загальну продуктивність запису.

Wear-Leveling динамічних даних здійснюється за принципом циклічного обороту даних у множині вільних блоків. Динамічний Wear-Leveling сильно зменшує життєвий цикл флеш пам’яті через те, що для обороту даних можуть використовуватись лише вільні динамічні блоки пам’яті.

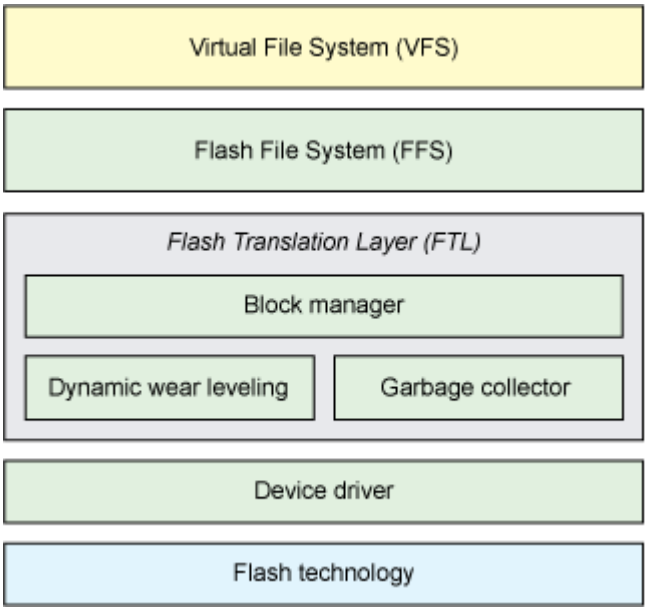
Порівняльна таблиця:

	Статичний	**Динамічний**
Термін служби пам’яті	Дуже довго	Довго
Продуктивність	Повільніше	Швидше
Складність реалізації	Більш складно	Менш складно

Як правило виробники флеш-пам’яті використовують динамічний Wear-Leveling.

Причиною цьому є те, що динамічний Wear-Leveling менш складний у реалізації та забезпечує надійність збереження данних - цього більш ніж достатньо для хорошого попиту на флеш-пам’ять на ринку.

Загальна архітектура файлових систем флеш-носіїв



Як видно з малюнку, нагорі знаходиться шар віртуальної файлової системи (VFS, virtual file system), що надає високорівневий інтерфейс для програм. Нижче йде рівень файлових систем для флеш. Далі розташовується FTL-рівень (Flash Translation Layer, рівень флеш-перетворення), який займається управлінням флеш-носієм, включаючи виділення блоків під дані, перетворення адрес, динамічну оптимізацію зносу і збірку сміття. У деяких пристроях частина функцій FTL-рівня може бути реалізована апаратно.

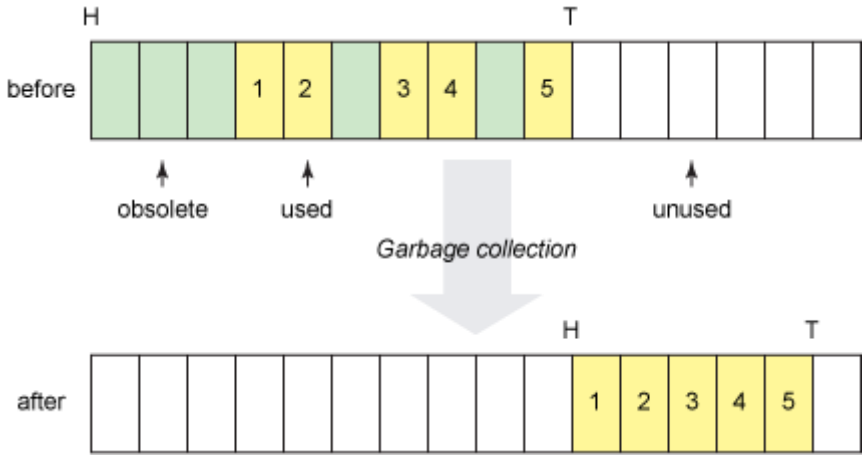
Для взаємодії з флеш-пристроями зазвичай застосовується *інтерфейс пристроїв на основі технологій пам'яті* (Memory Technology Device, MTD). MTD-інтерфейс автоматично розпізнає розрядність шини флеш-пристроїв і визначає необхідну кількість пристроїв для отримання шини необхідної розрядності.

JFFS

Журнальована файлова система для флеш-носіїв (Journaling Flash File System, JFFS) є однією з перших систем для флеш.

JFFS, в основі структури якої лежить журнал, призначалася для NOR-пристроїв. Система на момент створення була унікальною і вирішувала безліч проблем флеш-носіїв, але мала один суттєвий недолік.

JFFS оперує флеш-пам'яттю на основі циклічного журналу блоків. Дані на запис пишуться в блоки з хвоста журналу, а блоки, що знаходяться в голові, готуються для повторного використання. Діапазон між хвостом і головою журналу означає вільне місце.Коли його стає занадто мало, в справу вступає збирач сміття, який знаходить серед застарілих і дефектних блоків блоки з актуальними даними, переміщує їх в хвіст, а потім стирає місце, що звільнилося



В результаті такого підходу відбувається як статична, так і динамічна оптимізація зносу. Проблема даного методу полягає в тому, що відсутня ефективна стратегія стирання - флеш-пам'ять дуже часто перезаписується, в результаті чого знос пристрою відбувається занадто швидко.

Під час монтування структура і розташування блоків зчитуються в пам'ять, тому монтування JFFS-розділу відбувається повільно і використовується досить багато оперативної пам'яті.

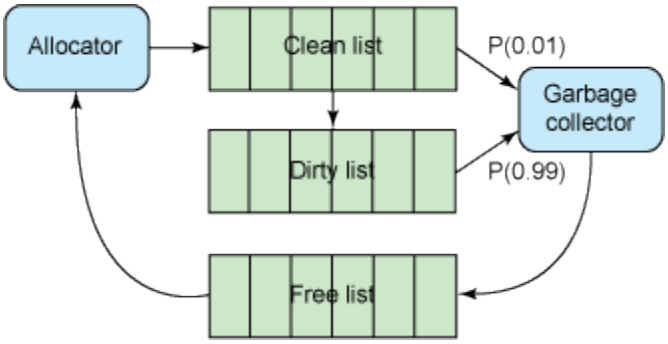
JFFS2

По при всі недоліки, які знижували термін служби FLASH-носіїв відповідно до свого алгоритму оптимізації зносу, система JFFS досить широко застосовувалася. В результаті було вирішено переробити її алгоритм і відмовитися від циклічного журналу. Так з'явилася JFFS2, орієнтована на NAND-пам'ять. Ця система демонструє більш високу швидкість роботи і підтримує функцію стиснення.

JFFS2 розглядає кожен блок флеш-пам'яті незалежно від інших. Для ефективної оптимізації зносу ведуться спеціальні списки. Список чистих блоків відповідає блокам, які містять тільки пакети з актуальними

даними. Список «брудних» блоків містить блоки, кожен з яких має хоча б один пакет із застарілими даними. Список вільних блоків зберігає відомості про блоки, які були стерті і вже готові до використання.

В цих умовах алгоритм збірки сміття може в залежності від ситуації ефективно приймати рішення про те, які блоки готувати до повторного використання. У поточних реалізаціях алгоритм обирає їх зі списків або чистих, або брудних блоків на основі заданої ймовірності. У 99% випадків



використовується список брудних блоків (при цьому актуальні дані переносяться в інший блок). В 1% випадків береться список чистих блоків (при цьому весь вміст просто переноситься в новий блок). У кожному разі цільовий блок стирається і заноситься в список вільних блоків.

Все це дозволяє збирачеві сміття повторно використовувати блоки з неактуальними даними (або частково неактуальними), в той час як дані все одно "проходять" всю пам'ять, що й дає статичну оптимізацію зносу.

Початкові дані

В якості флеш-пам’яті при розробці необхідно використати звичайний файл. Розмір блоку і сторінки у флеш-пам’яті задаються параметрами. Флеш-пам'ять підтримує одноразове програмування сторінки.

Завдання на лабораторну роботу № 3

Розробити драйвер файлової системи для флеш-пам’яті використовуючи метод, описаний в теоретичній частині. У файловій системі є одна

директорія, кількість файлів не обмежена, довжина імені файлу не перевищує 255 символів. Драйвер повинен дозволяти зберігати зміни у файловій системі до тих пір, поки на флеш-пам’яті є вільне місце (тобто не вимагається обробляти брудні блоки).

Розробити консольну програму, що підтримує наступні команди :

- *mount - підключити файлову систему, збережену у файлі;*
- *unmount* - відключити файлову систему, при цьому в пам'яті драйвера не повинно залишитися яких-небудь даних про файлову систему, усі дані мають бути збережені у файлі;
- *blockstat [номерблока]* - вивести інформацію в читабельному вигляді про стан блоку флеш-пам’яті;
- *create ім'я* - створити файл із заданим ім'ям;
- *list* - вивести імена усіх файлів;
- *open ім'я* - відкрити файл із заданим ім'ям, ця команда повинна вивести номер файлового дескриптора *fd*, який можна використати в інших командах;
- *close fd - закрити файл із заданим дескриптором;*
- *read fd зміщення розмір* - прочитати з файлу із заданим дескриптором по цьому зміщенню дані вказаного розміру;
- *write fd зміщення дані - записати у файл із заданим дескриптором по цьому зміщенню дані, зміщення не виходить за межі файлу;*

Звіт

Звіт повинен містити:

1. Опис ідеї реалізації файлової системи.
2. Опис форматів структур файлової системи.
3. Лістинг основної частини розробленої програми.

Завдання на лабораторну роботу № 4

Продовжити роботу над драйвером файлової системи для флеш-пам’яті і додати можливість чистки брудних блоків (тобто реалізувати звільнення

місця на флеш-пам’яті за допомогою знищення застарілої інформації у блоках). Також додати можливість створення нових імен для існуючих файлів.

Продовжити роботу над консольною програмою і додати підтримку наступних команд :

unlink ім'я - знищити це посилання на файл (якщо кількість посилань на файл стало рано нулю, то необхідно знищити вміст файлу);

- *link ім'я нове ім’я* - створити нове ім'я для існуючого файлу з вказаним ім'ям.
- *truncate ім'я розмір* - змінити розмір файлу, якщо новий розмір файлу більше ніж старий розмір, то нові дані дорівнюють 0.

Звіт

Звіт повинен містити:

1. Опис алгоритму чистки брудних блоків.
2. Опис форматів структур образу файлової системи в оперативній пам'яті.
3. Лістинг основної частини змін в розробленій програмі.

Електронна версія (мережа КПІ)

ftp://vt513.comsys.ntu-kpi.kiev.ua/pub/edu/spo2/spo2-3,4.doc

Література

1. David Woodhouse JFFS : The Journalling Flash File System.

Uresh Vahalia Unix Interna