Державний торговельно-економічний університет
Кафедра інженерії програмногозабезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ФУНКЦІОНАЛЬНІ СТРУКТУРИ ДАНИХ В ЕКОСИСТЕМІ LINUX»

Студента 2м курсу, 2 групи, спеціальності 121 «Інженерія програмного забезпечення» освітньої програми «Інженерія програмного забезпечення» Москаленко Володимира Володимировича

_____
підпис студента

Науковий керівник
кандидат економічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки Тищенко Дмитро Олександрович

_____
підпис керівника

Гарант освітньої програми
кандидат педагогічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки Котенко Наталія Олексіївна

_____
підпис гаранта

КИЇВ – 2023

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Освітня програма 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«13» грудня 2022 р.

Завдання
на випускну кваліфікаційну роботу студентові

Москаленко ВолодимираВолодимировича
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи «Функціональні структури даних в екосистемі Linux»

Затверджена наказом ректора від «06» грудня 2022 р. № 3285

2. Строк здачі студентом закінченої роботи 27 листопада 2023

3. Цільова установка та вихідні дані до роботи

Мета роботи розкриття основних принципів роботи функціональних структур даних в екосистемі Linux, їх вплив на продуктивність та можливості оптимізації.

Об'єкт дослідження екосистема Linux, що включає в себе ядро операційної системи, системні утиліти, бібліотеки та інші компоненти, що забезпечують її роботу.

Предмет дослідження функціональні структури даних, які використовуються в різних складових Linux для забезпечення ефективності та надійності операцій.

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1 ТЕОРЕТИЧНІ АСПЕКТИ ФУНКЦІОНАЛЬНИХ СТРУКТУР ДАНИХ

1.1. Характеристика функціональних структур даних

1.2. Класифікація функціональних структур даних

1.3. Роль функціональних структур даних

1.4. Переваги та недоліки використання функціональних структур даних

1.5. Висновки до розділу 1

РОЗДІЛ 2 ВИКОРИСТАННЯ ФУНКЦІОНАЛЬНИХ СТРУКТУР ДАНИХ У ПРОЕКТАХ

2.1.Архітектура та компоненти Linux

2.2.Аналіз реалізації функціональних структур даних в ядрі Linux

2.3.Особливості використання функціональних структур даних у користувацьких програмах Linux

2.4. Висновки до розділу 2

РОЗДІЛ 3. ПРАКТИЧНІ АСПЕКТИ ВИКОРИСТАННЯ ФУНКЦІОНАЛЬНИХ СТРУКТУР ДАННИХ У LINUX

3.1. Реалізація функціональних структур даних у проектах

3.2. Інструменти та ресурси для розробниківу галузі функціональних структур даних

3.3. Вплив використання функціональних структур даних на продуктивність системи

3.4 Висновки до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*07.11.2022*	*07.11.2022*
2.	*Розробка та затвердження завдання на роботу магістра (стац/заоч)*	*13.12.2022*	*13.12.2022*
3.	*Вступ та перелік літературних джерел*	*24.02.2023*	*24.02.2023*
4.	*Розробка технічного завдання*	*15.03.2023*	*15.03.2023*
5.	*Розділ 1. Теоретичні аспекти функціональних структур даних*	*10.04.2023*	*10.04.2023*
6.	*Розділ 2. Використання функціональних структур даних у проектах*	*24.05.2023*	*24.05.2023*
7.	*Розділ 3. Практичні аспекти використання функціональних структур даних у Linux*	*06.09.2023*	*06.09.2023*
8.	*Розробка програми та методики тестування*	*18.10.2023*	*18.10.2023*
9.	*Написання наукової статті*	*17.05.2023*	*17.05.2023*
10.	*Керівництво користувача*	*25.10.2023*	*25.10.2023*
11.	*Висновки та пропозиції*	*01.11.2023*	*01.11.2023*
12.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*	*06.11.2023*	*06.11.2023*
13.	*Підготовка автореферату та презентації доповіді*	*06.11.2023*	*06.11.2023*
14.	*Попередній захист випускної кваліфікаційної роботи*	*20.11.2023 – 24.11.2023*	*20.11.2023 – 24.11.2023*
15.	*Здача зброшурованої випускної кваліфікаційної роботи*	*01.12.2023*	*01.12.2023*
16.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*02.12.2023*	*02.12.2023*
17.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*05.12.2023-06.12.2023*	*05.12.2023-06.12.2023*

7. Дата видачі завдання «13» грудня 2022 р.

8. Науковий керівник випускної кваліфікаційної роботи Тищенко Д.О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Котенко Н.О.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Москаленко В.В.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускної кваліфікаційної роботи

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента Москаленко В.В.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Котенко Н.О.

(прізвище, ініціали, підпис)

Завідувач кафедри Криворучко О. В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Випускна кваліфікаційна робота на тему «Функціональні структури даних в екосистемі Linux» присвячена дослідженню та аналізу функціональних структур даних, використовуваних в екосистемі операційної системи Linux. Робота вивчає ключові аспекти функціональних структур даних, такі як списки, дерева, та інші, і їх вплив на продуктивність та ефективність операційної системи.

Дослідження включає в себе аналіз внутрішньої реалізації цих структур, їх використання в ядрі Linux, а також їх взаємодію з різними підсистемами та додатками. Висвітлюються особливості використання функціональних структур даних у контексті багатозадачного та багатокористувацького середовища Linux.

Результати дослідження можуть слугувати важливим внеском у розуміння принципів функціональної організації даних у системі Linux та їх можливого впливу на розробку операційних систем.

Випускна кваліфікаційна робота на тему «Функціональні структури даних в екосистемі Linux» містить 60 сторінок, 8 рисунків. Перелік використаних джерел налічує 8 найменувань. Ключові слова

Ключові слова: LINUX, ФУНКЦІОНАЛЬНІ СТРУКТУРИ ДАНИХ, ЯДРО ОПЕРАЦІЙНОЇ СИСТЕМИ, СПИСКИ, ЧЕРГИ, ДЕРЕВА, ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ.

ABSTRACT

The diploma thesis entitled "Functional Data Structures in the Linux Ecosystem" explores and analyzes the functional data structures used in the ecosystem of the Linux operating system. The work investigates key aspects of functional data structures, such as lists, queues, trees, and others, and their impact on the performance and efficiency of the operating system.

The research includes an analysis of the internal implementation of these structures, their use in the Linux kernel, and their interaction with various subsystems and applications. The peculiarities of employing functional data structures in the context of multitasking and multi-user environments in Linux are highlighted.

The findings of this research contribute significantly to the understanding of the principles of functional data organization in the Linux system and their potential impact on operating system development.

The thesis «Functional Data Structures in the Linux Ecosystem» has 60 pages, 8 figures. The reference list contains 8 titles.

Keywords: LINUX, FUNCTIONAL DATA STRUCTURES, OPERATING SYSTEM KERNEL, LISTS, QUEUES, TREES, PERFORMANCE OPTIMIZATION.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

SQL- Structured Query Language;

APT- Advanced Package Tool;

YUM- Yellowdog Updater Modified;

KVM- Kernel-based Virtual Machine;

GCC- GNU Compiler Collection;

GDB- GNU Debugger;

LWN.net- Linux Weekly News;

JIT- Just-In-Time.

.

ЗМІСТ

					ДТЕУ 121 02-12.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		01.11.23	*Функціональні структури даних в екосистемі Linux*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Тищенко Д.О.		01.11.23		*Зміст*	*3*	*61*
Гарант		Котенко Н.О.		01.11.23		*Факультет інформаційних технологій 2м курс, 2 група*		
Розробив		Москаленко В. В.		01.11.23	*Зміст*			

ВСТУП

Сучасний інформаційний вік характеризується швидким темпом зростання обсягів даних та необхідністю ефективного їх оброблення. Одним із важливих аспектів цього процесу є використання оптимізованих структур даних, які забезпечують ефективний доступ, модифікацію та управління інформацією. У світі відкритого програмного забезпечення Linux виявляється невід'ємною складовою, забезпечуючи не лише стабільність операційних систем, але й високу швидкодіючість завдяки вдосконаленим функціональним структурам даних.

Тема дипломної роботи спрямована на дослідження та аналіз функціональних структур даних в екосистемі Linux, яка включає ядро операційної системи, різноманітні модулі та підсистеми. Висвітлення цієї теми є актуальним завданням, оскільки дозволяє розкрити внутрішню організацію системи, її оптимізацію та можливості для подальшого розвитку.

На сьогоднішній день особливо актуальною стає проблема обробки та зберігання даних в умовах постійно зростаючого обсягу інформації. Серед різноманітних операційних систем важко переоцінити внесок Linux у розвиток інформаційних технологій. Однак, не зважаючи на його широке використання, існує потреба в подальшому вдосконаленні його функціональних структур даних для забезпечення більшої ефективності та оптимізації роботи системи.

Вибір теми дослідження обумовлений актуальністю проблеми оптимізації функціональних структур даних в екосистемі Linux.

Зростання обсягу даних та потреб користувачів у високопродуктивних

рішеннях ставить перед розробниками величезне завдання забезпечення швидкодії та надійності операційних систем. Враховуючи популярність та великий обсяг використання Linux, важливо дослідити його функціональні структури даних для вдосконалення роботи системи в цілому.

Предметом дослідження даної дипломної роботи є функціональні структури даних в екосистемі Linux. Це обумовлено важливістю вивчення внутрішньої організації операційної системи для розуміння принципів її функціонування та оптимізації роботи з нею.

Об'єктом дослідження виступає сама екосистема Linux, що включає в себе ядро операційної системи, системні утиліти, бібліотеки та інші компоненти, що забезпечують її роботу.

Предметною областю є функціональні структури даних, які використовуються в різних складових Linux для забезпечення ефективності та надійності операцій.

Метою дослідження є розкриття основних принципів роботи функціональних структур даних в екосистемі Linux, їх вплив на продуктивність та можливості оптимізації.

З метою досягнення поставленої мети визначені ***наступні завдання***:

- Провести огляд існуючих функціональних структур даних в екосистемі Linux.
- Виявити проблеми та обмеження, які виникають при використанні цих структур даних.
- Розробити методологію для оцінки ефективності функціональних структур даних.
- Запропонувати можливі шляхи оптимізації виявлених проблем.

Практичне завдання дослідження полягає в можливості покращення ефективності використання операційної системи Linux шляхом оптимізації функціональних структур даних, що використовуються в її складі.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ ФУНКЦІОНАЛЬНИХ СТРУКТУР ДАНИХ

1.1. Характеристика функціональних структур даних

Функціональні структури даних в сучасній інформатиці є ключовим елементом розробки ефективних програмних систем. Вони визначають, як дані організовані та доступні для операцій, що визначають їх поведінку. Огляд функціональних структур даних дозволяє краще розуміти їх призначення, переваги та недоліки.

Масиви є однією з базових функціональних структур даних, що забезпечують доступ до елементів за допомогою індексів. Вони ефективні при роботі з великими обсягами даних, але можуть бути обмежені в розмірах та неефективні при вставці та видаленні елементів.

Списки представляють собою динамічні структури даних, де кожен елемент містить посилання на наступний елемент. Це дозволяє ефективно вставляти та видаляти елементи, але може призвести до надмірного використання пам'яті через вказівники.

Стеки та черги - це структури даних, що регулюють порядок доступу до елементів. Стек використовує принцип "останній прийшов - перший вийшов" (Last In, First Out), тоді як черга використовує "перший прийшов - перший вийшов" (First In, First Out). Вони знаходять широке застосування в алгоритмах та системах обробки даних[1].

Дерева та графи є більш складними функціональними структурами даних, які дозволяють представляти відносини між елементами. Дерева використовуються для ієрархічного представлення даних, тоді як графи можуть включати будь-яке кількість зв'язків між елементами.

Хеш-таблиці дозволяють швидкий доступ до даних за допомогою хеш-функцій. Вони ефективно вирішують проблему пошуку, але можуть стикатися з колізіями, що вимагають високої якості хеш-функцій.

Функціональні структури даних відіграють важливу роль в розробці програмного забезпечення, надаючи зручні та ефективні засоби для роботи з різними видами інформації. Вибір відповідної структури даних залежить від конкретних завдань та вимог програмного проекту. Огляд різноманітних функціональних структур даних допомагає розуміти їхню природу та визначати найбільш підходящі рішення для конкретного випадку.

Для ефективної роботи з текстовою інформацією використовуються спеціалізовані структури даних, такі як рядки та дерева суфіксів. Рядки надають зручний інтерфейс для представлення та операцій над текстом, тоді як дерева суфіксів дозволяють швидко виконувати операції пошуку та аналізу підстрок у тексті.

Динамічні масиви, також відомі як списки зі змінною довжиною, комбінують переваги масивів та списків. Вони забезпечують ефективний доступ до елементів та можливість динамічної зміни розміру. Однак вони можуть стикається з проблемами фрагментації пам'яті та витратами на зберігання додаткових метаданих.

У випадках, коли програма пов'язана з обробкою географічних даних, використовуються структури даних, такі як географічні карті та графи. Ці структури надають ефективний спосіб збереження та взаємодії з просторовою інформацією, що є важливим у галузях, таких як геоінформаційні системи та картографія.

У галузі комп'ютерної графіки та обробки зображень використовуються спеціалізовані структури даних. Наприклад, графічні буфери, дерева прискорення та меш-структури дозволяють ефективно обробляти та відображати графічну інформацію. Ці структури важливі для швидкодії та реалізації складних графічних алгоритмів.

Реляційні бази даних використовують табличні структури для зберігання та управління даними. SQL (Structured Query Language) використовується для роботи з такими базами даних, надаючи мову запитів для отримання, модифікації та аналізу інформації. Ці структури є важливим елементом багатьох інформаційних систем та підтримують ефективну організацію даних у великих обсягах.

Сучасні тенденції включають в себе використання NoSQL баз даних, які дозволяють зберігати та обробляти дані без строго табличного формату. Графові бази даних, також стали популярними у сферах, де важлива взаємодія та аналіз відносин між елементами[3].

У галузях великих даних та машинного навчання важливо вибирати та оптимізовувати структури даних для швидкого та ефективного аналізу великих обсягів інформації. Спеціалізовані структури, такі як розподілені бази даних та тензори, дозволяють ефективно обробляти дані у великих масштабах.

Однією з ключових задач при використанні функціональних структур даних є забезпечення ефективності. Оптимізація роботи з даними включає в себе вибір оптимальних структур для конкретних операцій, уникання зайвого використання ресурсів та забезпечення швидкодії.

В сучасному світі безпека даних стала найважливішим питанням. Використання відповідних функціональних структур даних, таких як хеш-таблиці та шифрування, грає важливу роль у збереженні конфіденційності та недопущенні несанкціонованого доступу до інформації.

З огляду на постійний розвиток технологій та зростання обсягів даних, нові тенденції у розробці програмного забезпечення включають в себе використання більш розподілених та паралельних структур даних, які дозволяють ефективно використовувати ресурси сучасних апаратних засобів.

Не зважаючи на значний прогрес у галузі функціональних структур даних, існують виклики, які варто враховувати. Зокрема, зростання обсягів даних та потреба у їхньому швидкому обробленні вимагають розробки нових алгоритмів та структур, які відповідали б цим вимогам.

Забезпечення гнучкості та розширюваності функціональних структур даних є важливою задачею. Програмісти та інженери повинні мати можливість легко адаптувати структури даних до змін у вимогах та розмірах даних без значних змін у програмному коді.

Функціональні структури даних відіграють ключову роль у розвитку інтелектуальних систем та штучного інтелекту. Використання спеціалізованих структур для представлення та аналізу даних є важливим елементом у вдосконаленні алгоритмів машинного навчання та обробки природних мов.

З переходом до квантових обчислень виникає необхідність адаптації функціональних структур даних до нового типу обчислювальної архітектури. Використання квантових структур даних може вирішити питання ефективного оброблення складних завдань у світі квантового обчислення.

1.2. Класифікація функціональних структур даних

В сучасному інформаційному суспільстві обробка та зберігання даних стає все більше актуальною задачею. Одним із ключових аспектів цього процесу є вибір та оптимізація структур даних, що відіграють важливу роль у розробці програмних систем. Класифікація функціональних структур даних є

важливим етапом в цьому процесі та сприяє ефективній організації та використанню інформації.

Лінійні структури даних. Лінійні структури даних є основним елементом при розв'язанні задач обробки та управління інформацією. Вони дозволяють організовувати дані в послідовній формі, надаючи можливість ефективної адресації та модифікації. Лінійні структури даних включають в себе такі типи, як масиви, списки та стеки.

Масиви представляють собою найпростіші лінійні структури даних, які забезпечують константний доступ до елементів. Однак їхній розмір фіксований, що може призводити до проблем при динамічному управлінні пам'яттю. Масиви дозволяють ефективно зберігати дані одного типу у послідовному порядку, що робить їх ідеальними для операцій індексації.

Списки дозволяють динамічно змінювати розмір структури, вставляти та видаляти елементи. Виділяють однозв'язні, двозв'язні та циклічні списки. Списки дозволяють зберігати дані різних типів у послідовності, а стеки реалізують принцип "Last In, First Out" (LIFO), що важливо для деяких задач, наприклад, управління викликами функцій.

Стеки та черги — це лінійні структури даних, які дозволяють управляти доступом до елементів за принципом Last-In-First-Out (LIFO) для стеків та First-In-First-Out (FIFO) для черг[2].

Деревоподібні структури даних. Деревоподібні структури даних включають в себе велику кількість алгоритмів та структур, що базуються на ієрархічних відносинах між елементами. Ці структури грають важливу роль у забезпеченні ефективного доступу до даних, сортуванні, пошуку та інших операціях обробки інформації. Деревоподібні структури, такі як бінарні дерева та бінарні дерева пошуку, забезпечують ефективний пошук та сортування даних.

Бінарні дерева є однією з основних деревоподібних структур, де кожен вузол може мати не більше двох нащадків. Ці структури застосовуються для реалізації багатьох алгоритмів, таких як обходи дерев, сортування, пошук тощо. Бінарне дерево розгалуження дозволяє швидше здійснювати операції порівняння, а бінарне дерево пошуку дозволяє швидко знаходити та вставляти дані в відсортованому порядку.

Бінарні дерева пошуку вдосконалюють ідею бінарних дерев, додаючи умову сортування: всі ліві нащадки менше, а всі праві більше за батьківський вузол. Це дозволяє здійснювати ефективний пошук за ключами.

AVL-дерева є самобалансованими бінарними деревами пошуку, де висота кожного піддерева обмежена. Це забезпечує оптимальну швидкодію при операціях вставки та видалення елементів[4].

Червоно-чорні дерева — це інша форма самобалансованих бінарних дерев, де введені правила для забезпечення рівноваги дерева. Вони використовуються для уникнення надмірного збільшення висоти дерева та забезпечення ефективних операцій[7].

Графові структури даних. У сучасному світі, де обробка великих обсягів даних стала нормою, графові структури даних набувають особливої важливості. Вони забезпечують ефективне подання та аналіз складних систем та взаємодій, що забезпечує нові можливості для розвитку інноваційних інформаційних технологій. Графові структури даних, такі як орієнтовані та неорієнтовані графи, використовуються для моделювання складних зв'язків між об'єктами. Вони дозволяють представляти взаємодії між об'єктами та використовуються у великій кількості сфер, включаючи мережі, транспортні системи та аналіз соціальних мереж. Графові структури даних складаються з вузлів (вершин) та ребер (зв'язки), що визначають взаємозв'язки між вузлами. Глибокий розуміння цих структур є ключовим для ефективного моделювання та аналізу складних систем. У соціальних мережах графи використовуються

для моделювання дружби, взаємодій та впливу між користувачами. Це дозволяє покращити алгоритми рекомендацій та аналізу трендів. В транспортних системах графи використовуються для моделювання маршрутів, з'єднань та оптимізації трафіку. Це сприяє покращенню ефективності та безпеки транспортних систем.

Хеш-таблиці. Хеш-таблиці є однією з ключових структур даних у сучасному програмуванні, використовуючи хеш-функції для ефективного зберігання та отримання даних. Ці структури дозволяють здійснювати операції вставки, вилучення та пошуку за час O(1), що робить їх надзвичайно ефективними для розв'язання різноманітних завдань у сферах програмування, баз даних та інших областях. У хеш-таблиці дані зберігаються у вигляді пар ключ-значення. Ключі перетворюються за допомогою хеш-функцій у числа, які визначають індекси у внутрішньому масиві таблиці. Цей підхід дозволяє швидко знаходити необхідний елемент за його ключем, оптимізуючи час доступу до даних[4].

Хоча хеш-таблиці надзвичайно ефективні, вони можуть стикатися з проблемою колізій, коли різні ключі мають однаковий хеш-код та спрямовуються до одного й того ж індексу. Існує кілька методів вирішення цієї проблеми, таких як відкрита адресація та метод ланцюгків.

Відкрита адресація включає в себе пошук іншого вільного слоту в таблиці для зберігання колізійного елементу. З іншого боку, метод ланцюгків передбачає створення списку (ланцюга) для кожного індексу таблиці, в якому зберігаються всі елементи з однаковим хешем.

Існують також різноманітні види хеш-функцій, що використовуються для максимізації рівномірності розподілу хеш-кодів та зменшення ймовірності колізій. Успішне використання хеш-таблиць вимагає ретельного вибору хеш-функцій та ефективного управління колізіями.

Загалом, хеш-таблиці є потужним інструментом для швидкого доступу до даних, і їх використання розповсюджується в різних галузях програмування, де швидкість та ефективність обробки даних важливі.

Файлові структури даних. Файлові структури, такі як таблиці баз даних, використовуються для організації великих обсягів даних у вигляді таблиць та записів. Вони забезпечують зручний та швидкий доступ до інформації, а також можливість виконання складних запитів. Файлові структури даних грають важливу роль в організації та збереженні інформації в інформаційних системах. Вони дозволяють ефективно управляти даними та надають можливість швидкого доступу до інформації.

Послідовні файли представляють собою структуру, в якій дані записуються або читаються послідовно. Цей тип структури ефективний для збереження великих обсягів даних, але може виникати проблема доступу до конкретного запису.

Індексовані файли дозволяють швидкий доступ до даних за допомогою індексів. Ці індекси значно полегшують пошук та зменшують час доступу до конкретної інформації, але вимагають додаткового простору для зберігання індексів.

Файлові структури даних використовуються в різних сферах, таких як бази даних, операційні системи, веб-розробка та інші. Класифікація функціональних структур даних є необхідним етапом у розробці програмних систем. Вибір відповідної структури даних залежить від конкретних вимог та характеристик задачі. Інженери та програмісти повинні уважно аналізувати особливості кожного типу структури для досягнення оптимальної продуктивності та ефективності програмного забезпечення. Деревоподібні структури даних представляють собою важливий компонент сучасних інформаційних технологій. Вивчення їх властивостей та застосувань дозволяє розробникам та інженерам створювати більш ефективні та оптимізовані

програмні рішення. Деревоподібні структури даних вже знаходять застосування в інших галузях, таких як штучний інтелект, обробка природних мов, графічні системи та інше. Це свідчить про універсальність та важливість цих структур в різних сферах науки та техніки. Деревоподібні структури даних визначають нові горизонти в області інформаційних технологій. Їхнє дослідження та вдосконалення є ключовими завданнями для розробників та науковців. Вони стають основою для створення більш продуктивних, швидких та надійних програмних рішень, що відповідають сучасним вимогам сфери інформаційних технологій[7].

Графові структури даних мають широкий спектр застосувань у різних галузях. Розвиток нових методів аналізу графів та їхнє поєднання з іншими сучасними технологіями, такими як штучний інтелект та машинне навчання, відкриває нові перспективи для вирішення складних завдань у сучасному інформаційному середовищі. Хеш-таблиці використовують хеш-функції для ефективного зберігання та пошуку даних. Вони забезпечують швидкий доступ до даних, особливо при великому об'ємі інформації, проте вимагають правильного вибору хеш-функцій та вирішення колізій. Інтеграція різних типів файлових структур дозволяє оптимізувати роботу інформаційних систем та підвищує ефективність обробки даних.

1.3 Роль функціональних структур даних в програмуванні

В сучасному програмуванні велику увагу приділяють вибору та оптимізації структур даних для ефективної обробки і зберігання інформації. Вони відіграють важливу роль у забезпеченні ефективності та гнучкості програмного забезпечення.

Функціональні структури даних сприяють оптимізації роботи програм, оскільки вони дозволяють ефективно виконувати операції, що

стосуються зберігання та обробки інформації. Наприклад, використання оптимальних структур для пошуку, сортування чи фільтрації даних може значно покращити час виконання програми.

Також функціональні структури даних надають програмістам можливість легко змінювати та розширювати функціональність програмного забезпечення. Завдяки правильному вибору структур даних, можна легко адаптувати програму до нових вимог без значного впливу на її продуктивність. Функціональні структури даних грають важливу роль у забезпеченні безпеки та надійності програм. Використання правильних механізмів зберігання та обробки даних може допомогти у запобіганні можливих помилок та забезпеченні цілісності інформації.

Крім того, функціональні структури даних сприяють сумісності та інтеграції з іншими програмними модулями. Правильний вибір структур даних дозволяє легко обмінюватися інформацією між різними частинами програми та забезпечує їх взаємодію без зайвих труднощів. Функціональні структури даних є важливим елементом для ефективної реалізації алгоритмів. Вони дозволяють використовувати різноманітні алгоритмічні підходи та полегшують реалізацію різних операцій над даними.

Функціональні структури даних грають ключову роль у підтримці масштабованості програмних рішень. Здатність швидко та ефективно обробляти великі обсяги даних визначає продуктивність програми при роботі з великими системами чи великою кількістю користувачів.

Вибір правильних функціональних структур даних може сприяти економії ресурсів комп'ютерної системи, таких як пам'ять та процесорний час. Ефективне використання ресурсів є ключовим фактором для оптимізації роботи програм та підвищення ефективності комп'ютерних систем.

У деяких випадках, особливо в області вбудованих систем та систем реального часу, вибір відповідних функціональних структур даних є критичним для забезпечення обробки даних в обмежених часових рамках.

Функціональні структури даних дозволяють легко адаптувати програмне забезпечення до змін у вимогах або середовищі. Заміна або модифікація структур даних може бути виконана без значних змін в інших частинах програми, що сприяє гнучкості та підтримці продукту в актуальному стані.

Використання відповідних функціональних структур даних сприяє полегшенню читабельності та розуміння коду. Коректний та логічний вибір структур даних робить код більш зрозумілим для інших розробників та сприяє підтримці програми протягом часу.

Функціональні структури даних дозволяють ефективно використовувати пам'ять комп'ютерної системи. Оптимізація розміру та структури даних допомагає зменшити використання пам'яті, що особливо важливо в обмежених середовищах, таких як мобільні пристрої чи вбудовані системи.

У світі сучасних обчислювальних систем, функціональні структури даних використовуються для підтримки паралельного та розподіленого програмування. Вони дозволяють ефективно обробляти дані в різних потоках виконання чи на різних вузлах мережі, що сприяє підвищенню продуктивності.

Функціональні структури даних часто служать основою для нових інноваційних алгоритмів та підходів у програмуванні. Вони стимулюють розробників до пошуку ефективних та експериментальних методів обробки даних.

Застосування функціональних структур даних може покращити стійкість програм до помилок. Механізми обробки помилок та відновлення

даних, реалізовані в структурах даних, сприяють підтримці стабільної роботи програм навіть у випадку виникнення непередбачуваних ситуацій.

Використання функціональних структур даних допомагає відділити логіку програми від її даних. Це сприяє створенню більш модульних та легко змінюваних систем, де зміни в одній частині не суттєво впливають на інші.

Функціональні структури даних допомагають управляти складністю програмних проектів. Розподілення даних за логічними частинами полегшує розуміння та підтримку коду, забезпечуючи більш ефективне управління проектом.

Використання правильних функціональних структур даних сприяє відповідності програмних проектів стандартам та рекомендаціям. Це полегшує роботу команди розробників, сприяє обміну знаннями та допомагає в утриманні високого рівня якості коду.

Функціональні структури даних дозволяють проводити аналіз продуктивності програм та прогнозувати можливі вузькі місця чи проблеми в роботі системи. Це сприяє забезпеченню оптимального функціонування програми та уникненню непередбачених труднощів. Функціональні структури даних є ключовим елементом в автоматизації обробки великих обсягів даних. Вони забезпечують зручний та ефективний доступ до інформації, а також дозволяють швидко виконувати операції над великими масивами даних[5].

Вибір оптимальних функціональних структур даних сприяє ефективному використанню обчислювальних ресурсів. Врахування особливостей обчислювального середовища дозволяє максимізувати використання ресурсів та забезпечує оптимальну продуктивність програми.

Ці додаткові аспекти підкреслюють комплексний характер впливу функціональних структур даних на програмування. Вони визначають якість,

ефективність та масштабованість програм, роблячи вибір та оптимізація цих структур ключовим етапом у процесі розробки програмного забезпечення.

Функціональні структури даних сприяють розвитку алгоритмічного мислення у розробників. Обробка та маніпулювання даними за допомогою різних структур вимагає глибокого розуміння алгоритмів та їх впливу на ефективність програми. Функціональні структури даних можуть бути використані для підтримки об'єктно-орієнтованого програмування. Це сприяє створенню об'єктів та класів, що мають внутрішні структури для зберігання та обробки даних.

Функціональні структури даних можуть бути оптимально адаптовані для взаємодії із зовнішніми джерелами даних, такими як бази даних, файлові системи, API. Це забезпечує зручний обмін інформацією між програмою та зовнішніми ресурсами.

Використання функціональних структур даних є важливою частиною впровадження принципів доброго проектування програмного забезпечення. Зокрема, принципи, такі як SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion), можуть бути ефективно реалізовані за допомогою відповідно підібраних структур даних.

Функціональні структури даних відіграють важливу роль у підтримці сучасних технологій, таких як штучний інтелект, машинне навчання та обробка великих даних. Вони дозволяють ефективно обробляти та аналізувати великі об'єми інформації, що є важливим для розвитку нових технологій.

Ці аспекти вказують на постійний розвиток та розширення ролі функціональних структур даних у світі програмування. Їхнє впровадження та оптимізація залишається важливим завданням для розробників, спрямованим на створення ефективних, надійних та інноваційних програмних продуктів.

Функціональні структури даних грають важливу роль у забезпеченні конфіденційності та безпеки даних. Вірний вибір та ефективне використання таких структур дозволяють розробникам реалізувати механізми шифрування, контролю доступу та інших методів захисту інформації. Функціональні структури даних є важливим елементом для забезпечення сумісності та інтеграції програм з іншими системами. Вони дозволяють ефективно обмінюватися даними між різними системами, що стає важливим в умовах великої кількості різнорідних програм.

Крім того, функціональні структури даних можуть бути оптимально використані для аналізу та візуалізації даних. Вони дозволяють швидко та ефективно виконувати операції агрегації, групування та фільтрації, що важливо для розуміння великих масивів інформації. Функціональні структури даних є ключовим елементом у розробці інтерактивних та веб-додатків. Вони дозволяють ефективно взаємодіяти з клієнтами, забезпечуючи швидкий доступ до даних та оптимальну реакцію на користувацькі запити.

У світі сучасних обчислювальних систем, де важливо використовувати паралельні та розподілені обчислення, функціональні структури даних надають засоби для ефективного управління даними в різних обчислювальних середовищах.

Ці аспекти підкреслюють динаміку та розширення ролі функціональних структур даних у високотехнологічному світі. Їх використання відображається в різноманітних сценаріях розробки програм, де вони забезпечують не лише ефективність, але й високий рівень безпеки та зручності взаємодії з користувачами та іншими системами.

Використання функціональних структур даних в програмуванні визначає успішність розробки програмного забезпечення. Вони забезпечують ефективність, гнучкість, надійність та легкість інтеграції, роблячи їх невід'ємною частиною сучасних програмних рішень. Функціональні

структури даних відіграють критичну роль у розробці програмного забезпечення, забезпечуючи ефективність, гнучкість та інші ключові характеристики програм. Вибір та оптимізація таких структур є необхідним етапом у процесі створення програмних продуктів різної складності. Усі зазначені аспекти підкреслюють важливість функціональних структур даних у світі програмування. Невірний вибір або неоптимальна реалізація може призвести до погіршення продуктивності, незручностей у роботі з програмою та інших негативних наслідків.

1.4 Переваги та недоліки використання функціональних структур даних

Функціональні структури даних відіграють важливу роль у сучасних програмних системах, надаючи ефективні та оптимізовані механізми для організації та доступу до інформації. Проте, як і будь-яка технологія, вони мають свої переваги та недоліки, які потрібно враховувати при розробці програмного забезпечення.

Функціональні структури даних є важливим елементом сучасних програмних систем і відіграють ключову роль у забезпеченні ефективності та оптимізації програмного коду. Нижче розглянуто основні переваги використання функціональних структур даних, які допомагають програмістам досягати більшої продуктивності та якості програмного забезпечення.

Основні переваги функціональних структур даних:

- Ефективність в операціях читання та запису. Функціональні структури даних, такі як списки, дерева та графи, можуть забезпечити швидкий доступ до даних при виконанні операцій читання та запису. Це особливо важливо в задачах, де необхідно ефективно взаємодіяти з великим обсягом інформації.

- Легка розширюваність. Функціональні структури даних можуть легко адаптуватися до змін у розмірі даних або вимогах системи. Їх гнучкість дозволяє швидко змінювати та доповнювати структури без значних зусиль.
- Зручність у використанні функціональних мов програмування. Функціональні структури даних ефективно взаємодіють з функціональними мовами програмування, такими як Haskell або Lisp. Це робить їх привабливими для розробників, які віддають перевагу функціональному підходу у програмуванні. Функціональні структури даних забезпечують зручний та ефективний доступ до елементів. Наприклад, використання хеш-таблиць дозволяє швидко здійснювати пошук, вставку та видалення елементів, забезпечуючи стабільну часову складність операцій.
- Модульність та перевикористовуваність коду. Функціональні структури даних сприяють розділенню коду на модулі, що полегшує його розробку і обслуговування. Модульний підхід сприяє перевикористовуванню коду та розвитку програм з використанням готових компонентів.
- Безпека та стійкість. Функціональні структури даних сприяють безпеці програм, оскільки вони можуть допомагати у уникненні багатьох типових помилок, таких як витоки пам'яті та інші види помилок, пов'язаних із звертанням до даних. Наприклад, коректно реалізовані черги та стеки допомагають уникнути витоку пам'яті та інших проблем, пов'язаних із керуванням пам'яттю[9].

Функціональні структури даних, які визначаються специфічними операціями та взаємодією функцій, мають свої переваги, проте їх використання пов'язане із рядом недоліків. Нижче представлено аналіз основних недоліків функціональних структур даних.

Недоліки функціональних структур даних:

- Підвищена потреба в пам'яті. Використання функціональних структур даних часто вимагає додаткового використання пам'яті. Коли

потрібно створити новий об'єкт для кожної зміни, це може призводити до значного збільшення обсягу використаної пам'яті, особливо при роботі з великими наборами даних.

- Складність для реалізації деяких операцій. Деякі операції, які в інших структурах даних можуть бути реалізовані просто, виявляються складними при використанні функціональних підходів. Наприклад, вставка елемента в середину структури даних може вимагати складних алгоритмів та математичних обчислень.
- Обмежена підтримка імперативних операцій. Функціональні структури даних не завжди ефективно підтримують імперативні операції, такі як зміна стану чи ітерація. Це може ускладнювати використання таких структур в деяких областях програмування, де імперативний стиль важливий.
- Вартість копіювання даних. При зміні даних часто виникає необхідність копіювати значення, замість простої модифікації. Це може впливати на продуктивність та ефективність програми, особливо при роботі з великими об'ємами даних.
- Важкість відлагодження та аналізу коду. Функціональний підхід може ускладнювати відлагодження коду через високий рівень абстракції та функціональну композицію. Це може затруднити знаходження помилок та розуміння логіки програми[10].

Хоча функціональні структури даних мають свої переваги, розуміння їхніх недоліків дозволяє розуміти ситуації, в яких інші підходи можуть бути більш ефективними.

Функціональні структури даних є потужним інструментом у розробці програмного забезпечення, але їх використання повинно бути обдуманим і зорієнтованим на конкретні завдання. Розробники повинні уважно враховувати переваги та недоліки цих структур, щоб забезпечити оптимальну продуктивність та ефективність програмної системи.

Можливі шляхи вдосконалення:

- Оптимізація використання пам'яті. Для подолання проблеми витрат пам'яті можна розглядати методи оптимізації, такі як стиснення даних чи використання більш ефективних алгоритмів зберігання.
- Асинхронні операції. Врахування асинхронних операцій може полегшити взаємодію з функціональними структурами даних, зменшуючи час виконання та ресурсозатрати при модифікації даних.
- Оптимізація алгоритмів. Для зменшення складності реалізації та оптимізації ефективності можна використовувати оптимальні алгоритми для конкретних операцій, що здійснюються над даними.
- Контроль за витратами пам'яті. Реалізація механізмів, що контролюють та мінімізують витрати пам'яті при операціях зберігання та обробки даних, може сприяти покращенню продуктивності.
- Навчання та документація: Забезпечення належної освіти для розробників стосовно ефективного використання функціональних структур даних та надання якісної документації може зменшити складність їх імплементації та використання.

Використання функціональних структур даних може стати ключовим чинником у побудові ефективних та гнучких програмних рішень. Необхідно враховувати як їхні переваги, так і недоліки, та вживати заходів для максимальної оптимізації їхнього використання в конкретних умовах розробки. Розуміння і балансування цих аспектів дозволить побудувати надійне та продуктивне програмне забезпечення.

1.5. Висновки до розділу 1

Огляд функціональних структур даних свідчить про їхню важливість та розмаїття застосувань у сучасному інформаційному суспільстві. На шляху до подальшого розвитку та вдосконалення програмного забезпечення

важливо продовжувати дослідження нових структур даних та методів їхнього використання. Подолання викликів та впровадження інновацій дозволять досягти вищого рівня ефективності та надійності у програмній інженерії.

Дослідження функціональних структур даних підкреслює їхню різноманітність та важливість у сфері розробки програмного забезпечення.

Вибір відповідної структури даних визначається конкретними вимогами завдань та характеристиками даних. Інженери та програмісти повинні мати глибоке розуміння цих структур для ефективного та оптимального проектування програмних систем. Дослідження та розвиток нових функціональних структур даних продовжуються, щоб задовольняти зростаючі вимоги сучасних програмних застосунків.

В сучасному програмуванні та розробці програмного забезпечення важливо мати глибоке розуміння різних типів структур даних та вміти вибирати найбільш ефективні для конкретного завдання. Від традиційних масивів до сучасних графових баз даних, розуміння та використання цих структур допомагає побудувати ефективні та потужні програмні рішення.

Широкий спектр застосувань та постійне розширення галузей, де вони використовуються, свідчать про необхідність глибокого розуміння їхніх особливостей та вміння вибирати найбільш підходящі для конкретних завдань рішення. Розуміння та застосування цих структур допомагає створювати програмне забезпечення, яке ефективно взаємодіє з даними, забезпечуючи швидкість, безпеку та надійність.

РОЗДІЛ 2
ВИКОРИСТАННЯ ФУНКЦІОНАЛЬНИХ СТРУКТУР ДАНИХ У ПРОЕКТАХ

2.1. Архітектура та компоненти Linux

Лінукс, операційна система з відкритим вихідним кодом, служить основою для безлічі серверів, вбудованих систем, суперкомп'ютерів та персональних комп'ютерів. Його успіх ґрунтується на ефективній та стабільній архітектурі, яка включає в себе різноманітні компоненти для забезпечення високої продуктивності та надійності.

Лінукс базується на монолітній архітектурі ядра, що дозволяє обробляти всі системні виклики в межах одного адресного простору ядра. Ядро Лінукса взаємодіє з апаратною частиною комп'ютера та керує ресурсами системи. Ця архітектура дозволяє високий рівень продуктивності та швидкість виконання операцій.

Компоненти Лінукс:

Ядро (Kernel). Ядро Лінукса відповідає за керування ресурсами, такими як процесор, пам'ять, введення/виведення та мережа. Воно забезпечує базові сервіси для роботи із апаратною частиною та іншими системними ресурсами.

Системні бібліотеки (System Libraries). Лінукс використовує набір системних бібліотек, які надають інтерфейс для програмного забезпечення.

Вони містять функції, які можуть бути використані різними програмами для взаємодії із системою. Найпростіший модуль ядра Linux виглядає наступним чином (Рис. 2.1).

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Mark Poliakov");
MODULE_DESCRIPTION("Hello world Linux module");
MODULE_VERSION("1.0.0");
static int __init mod_init(void){
    printk(KERN_INFO "Hello, World\n");
    return 0;
}
static void __exit mod_exit(void){
    printk(KERN_INFO "The module stopped\n");

}
module_init(mod_init);
module_exit(mod_exit);
```

Рис. 2.1. Приклад найпростішого модуля ядра Linux

Системні інструменти (System Utilities). Ці інструменти включають команди і програми, які дозволяють користувачам керувати та адмініструвати систему. Наприклад, командна оболонка (shell) надає інтерфейс для введення команд та взаємодії з операційною системою.

Системний рівень (System Level). Системні рівні включають різноманітні демони, служби та фонові процеси, які працюють на системі для виконання різних завдань, таких як мережеве управління, логування тощо.

Графічне середовище (Graphical Environment).Для користувачів, які віддають перевагу графічному інтерфейсу, Лінукс надає графічне середовище, таке як X Window System або Wayland, яке дозволяє запускати графічні програми та взаємодіяти з їхнім інтерфейсом.

Лінукс забезпечує потужну та гнучку архітектуру, яка враховує потреби різних видів користувачів та систем. Кожен компонент взаємодіє із іншими, створюючи стійку та ефективну операційну систему, яка продовжує збирати популярність серед спільноти та користувачів у всьому світі.

Розвиток Лінукса нещадно ведеться активною спільнотою розробників та користувачів. Вона вносить удосконалення, оновлення та нововведення, розширюючи можливості операційної системи. Комунікація між розробниками та користувачами здійснюється через системи контролю версій, форуми, електронні листи та інші онлайн-ресурси.

Ще однією ключовою особливістю Лінукса є підтримка великої кількості архітектур та апаратних платформ. Це дозволяє використовувати Лінукс на різних пристроях, від вбудованих систем до суперкомп'ютерів.

У майбутньому можна очікувати подальший розвиток та вдосконалення архітектури Лінукса в напрямку підвищення продуктивності, оптимізації споживання ресурсів та підтримки новітніх технологій.

Ще однією перспективою є розширення використання Лінукса в інтернеті речей (IoT), хмарних технологіях та сферах штучного інтелекту. Це відкриває нові можливості для інтеграції та розвитку систем, що використовують Лінукс.

Загалом, архітектура та компоненти Лінукса є основою для стабільної та високопродуктивної операційної системи, яка продовжує зберігати свою

популярність серед різних категорій користувачів та вирізняється своєю відкритістю та гнучкістю.

Незважаючи на успіх та популярність, Лінукс також стикається з викликами, які вимагають уваги та вирішення. Один із таких викликів - це підтримка нового апаратного забезпечення та технологій. Забезпечення сумісності та інтеграція з новими пристроями та стандартами є важливою задачею для розвитку операційної системи.

Другим аспектом є питання безпеки. З урахуванням постійного зростання загроз кібербезпеки, розробники Лінукса мають постійно вдосконалювати систему та вживати заходи для захисту від потенційних атак.

Окрім того, управління ресурсами та енергозбереженням стає все важливішим у світі мобільних та вбудованих пристроїв. Лінукс розвивається в напрямку оптимізації споживання енергії та ресурсів, щоб відповідати сучасним екологічним та технічним вимогам.

У сучасному інформаційному суспільстві, Лінукс відіграє важливу роль у різноманітних сферах, починаючи від великих дата-центрів та хмарних обчислень, і закінчуючи вбудованими системами, такими як маршрутизатори та смарт-пристрої.

Особливості застосування Лінукса в сучасному інформаційному суспільстві:

1. Серверне застосування. Лінукс визнаний своєю стабільністю та високою продуктивністю, що робить його популярним в сфері серверів. Багато великих компаній використовують сервери, що працюють під управлінням Лінукса для забезпечення надійності та ефективності своїх послуг.

2. Вбудовані системи. Лінукс широко застосовується у вбудованих системах, таких як маршрутизатори, смарт-пристрої, автомобільні системи,

телевізори та інші пристрої Інтернету речей (IoT). Його гнучкість дозволяє виробникам адаптувати операційну систему до конкретних потреб вбудованого пристрою.

3. Розробка та програмування. Лінукс є популярною платформою для розробників та програмістів. Заснована на відкритих стандартах, система надає величезний набір інструментів для розробки програмного забезпечення. Різноманітні дистрибутиви та пакетні менеджери роблять процес розробки зручним та ефективним.

4. Лінукс на персональних комп'ютерах. Декілька дистрибутивів Лінукса (наприклад, Ubuntu, Fedora, Debian) надають користувачам альтернативу традиційним операційним системам. Вони дозволяють використовувати потужність та гнучкість Лінукса навіть на персональних комп'ютерах.

5. Відкритий Вихідний Код та Спільнота. Важливою особливістю Лінукса є його відкритий вихідний код. Це сприяє створенню великої та активної спільноти розробників та користувачів, яка спільно працює над вдосконаленням системи та розв'язанням проблем[6].

У майбутньому можна очікувати подальше розширення використання Лінукса в нових технологіях, таких як штучний інтелект, блокчейн, квантові обчислення та інші. Розвиток додаткових інструментів, оптимізація продуктивності та забезпечення безпеки залишаються ключовими завданнями для подальшого успіху системи.

Лінукс продовжує бути важливою складовою інформаційного ландшафту, надаючи високий рівень ефективності та стабільності у різних галузях та сприяючи розвитку та інноваціям у світі ІТ.

Лінукс продовжує залишатися однією з найбільш важливих операційних систем у світі, завдяки своїй стабільності, відкритості та гнучкості. Постійний розвиток архітектури та компонентів Лінукса

відбувається завдяки активній спільноті та розробникам, які забезпечують адаптацію до змін у технологічному середовищі та вирішують виклики майбутнього. Ця операційна система залишається відкритою платформою для інновацій та розвитку в різних галузях комп'ютерної техніки.

2.2. Аналіз реалізації функціональних структур даних в ядрі Linux

Ядро Linux є однією з найпоширеніших та найбільш використовуваних операційних систем у світі, завдяки своїй відкритій архітектурі та високій продуктивності. Реалізація функціональних структур даних в ядрі грає важливу роль у забезпеченні ефективної роботи операційної системи. У цьому науковому тексті розглядається аналіз реалізації функціональних структур даних в ядрі Linux та їхній вплив на продуктивність та функціональність системи.

Функціональні структури даних в ядрі Linux включають різноманітні типи, такі як списки, черги, дерева та багато інших. Ці структури дозволяють ефективно вирішувати задачі, пов'язані з керуванням пам'яттю, плануванням завдань, роботою з файловою системою та іншими аспектами операційної системи.

Однією з основних функціональних структур даних є двоспрямовані списки, які використовуються для зберігання та обробки великої кількості даних. Вони дозволяють ефективно вставляти, видаляти та обходити елементи. Операційна система Linux використовує багатопроцесорну модель для обробки багатьох задач паралельно. Ця модель базується на списках та чергах для ефективного розподілу та виконання завдань. Проте, існує постійна потреба в оптимізації цих механізмів для забезпечення оптимальної продуктивності.

У ядрі Linux використовуються різні типи списків та черг, такі як однонаправлені, двунаправлені, FIFO (першим прийшов — першим вийшов)

тощо. Аналіз роботи кожної моделі є ключовим для визначення потенційних точок оптимізації. Черги в ядрі Linux використовуються для реалізації планування завдань та обробки подій у системі.

Однією з ключових структур даних у операційних системах, включаючи Linux, є пов'язані списки. Вони грають важливу роль у забезпеченні ефективної організації та доступу до даних у різних операційних системах. Одним з основних відмінностей в їх реалізації є використання однонаправленого або двунаправленого підходу.

Однонаправлений список є структурою даних, де кожен вузол має посилання лише на наступний вузол у послідовності. Це спрощує процес вставки та видалення елементів, оскільки досить змінити посилання у попередньому вузлі (Рис.2.2)

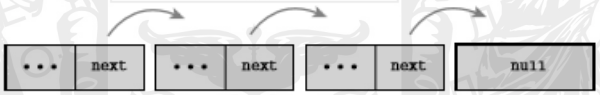

Рис. 2.2. Однонаправлений пов'язаний список

У Linux однонаправлені списки широко використовуються у різних підсистемах ядра, таких як списки завдань, списки буферів, тощо. Це забезпечує швидкий доступ та ефективне управління ресурсами.

Насупний рівень складності управління даними може бути досягнутий за допомогою двунаправленого пов'язаного списку. У цьому випадку, кожен вузол має посилання як на наступний, так і на попередній вузол. Це сприяє більш гнучкому доступу до даних в обох напрямках (Рис.2.3)

Рис .2.3. Двунаправлений пов'язаний список

В ядрі Linux двунаправлені списки застосовуються в ситуаціях, де необхідно здійснювати швидкий доступ до сусідніх елементів у обох напрямках. Наприклад, в списку завдань, де важливо ефективно переміщатися як вперед, так і назад між завданнями.

Циклічні однонаправлені та двунаправлені пов'язані списки відіграють важливу роль у структурах даних ядра операційної системи Linux. Вони забезпечують ефективну організацію та доступ до даних, що використовуються для різних операцій, таких як планування задач, керування пам'яттю та обробка подій.

Циклічний однонаправлений список - це структура даних, де останній елемент посилається на перший, утворюючи замкнений цикл. У ядрі Linux цей тип списку може використовуватися для реалізації черг або кільцевих буферів(Рис.2.4). Перевагою цього підходу є швидкий доступ до першого елементу, але недолік - обмежена можливість просування лише в одному напрямку[2].

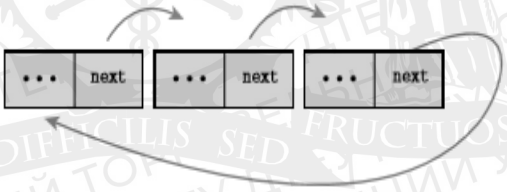

Рис. 2.4. Циклічний днонаправлений пов'язаний список

Циклічний двунаправлений список розширює можливості циклічного однонаправленого списку, дозволяючи просування в обох напрямках. У ядрі Linux використовується, наприклад, для подій, які можуть виникнути в будь-якому порядку. Кожен елемент має посилання як на попередній, так і на наступний, що спрощує операції вставки та видалення, але при цьому вимагає більше пам'яті(Рис.2.5)

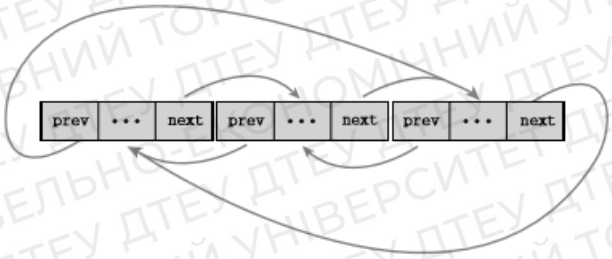

Рис. 2.5. Циклічний двунаправлений пов'язаний список

Однією з основних проблем є можливість виникнення конфліктів доступу до списків та черг при паралельному виконанні задач. Це може викликати блокування та зниження продуктивності системи. Іншою проблемою є неефективне використання ресурсів при видаленні та вставці елементів в черги.

Для розв'язання вищезазначених проблем важливо розглядати нові алгоритми та підходи до управління списками та чергами. Адаптивні механізми блокування, розподілена блокування та інші техніки можуть бути використані для зменшення конфліктів та підвищення швидкодії.

Ядро Linux — це основна частина операційної системи, яка взаємодіє з апаратним забезпеченням та забезпечує роботу високорівневих програм. Одним із ключових елементів ядра є дерево процесів, яке відіграє важливу роль у керуванні процесами та ресурсами системи[4].

Дерева є ще однією важливою функціональною структурою даних в ядрі. Зокрема, дерева використовуються для представлення структур файлової системи та ієрархії завдань. Вони дозволяють швидкий доступ до конкретних елементів та виконання операцій пошуку та сортування.

Дерево процесів у ядрі Linux організоване у вигляді ієрархічної структури, де кожен процес є нащадком певного батьківського процесу, крім кореневого процесу PID 1. Це структуроване дерево дозволяє ефективно керувати та відстежувати процеси в системі.

Дерево процесів визначає взаємозв'язки між процесами, об'єднує їх у групи та визначає правила спадкування ресурсів. Кожен процес може мати багато дочірніх процесів, які спадковують його властивості та можливості. Це дозволяє оптимізувати використання ресурсів та забезпечує стабільність системи.

Структура дерева процесів важливо впливає на продуктивність операційної системи. Добре побудоване дерево дозволяє ефективно розподіляти ресурси та зменшує конфлікти в управлінні процесами. Оптимізована структура дерева сприяє стійкості та працездатності системи в умовах великої навантаженості.

Двійкові дерева пошуку є структурами даних, які дозволяють ефективно організовувати та шукати дані, забезпечуючи швидкий доступ і вставку. У Linux, де продуктивність та оптимізація грають критичну роль, використання BST може бути ключовим фактором у вирішенні різноманітних завдань, починаючи від керування пам'яттю до оптимізації пошуку файлів(Рис.2.6 і Рис.2.7)

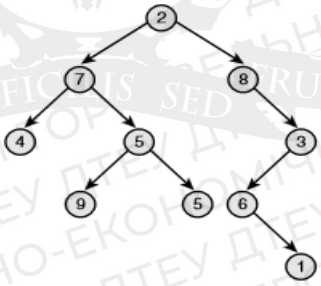

Рис. 2.6. Двійкове дерево

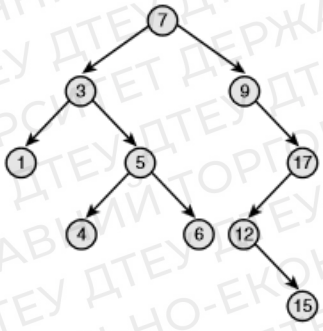

Рис. 2.7. Двійкове дерево пошуку

Збалансоване двійкове дерево пошуку (Balanced Binary Search Tree або AVL Tree) є структурою даних, яка забезпечує швидкий пошук, вставку і видалення елементів. Його особливість полягає в тому, що висота лівого і правого піддерева кожного вузла відрізняється не більше, ніж на одиницю.(Рис.2.8)

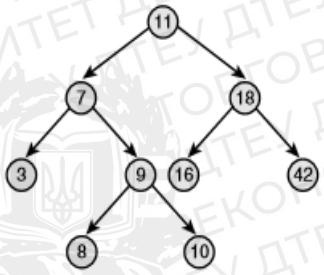

Рис. 2.8. Збалансоване двійкове дерево пошуку

Дерево процесів у ядрі Linux відіграє важливу роль у керуванні та організації процесів в операційній системі. Його структура та функції суттєво впливають на продуктивність системи. Розуміння цих аспектів допомагає розробникам оптимізувати ядро Linux для покращення продуктивності та стабільності операційної системи в цілому.

Ефективна реалізація функціональних структур даних в ядрі Linux сприяє високій продуктивності системи. Швидкий доступ до даних, ефективне використання ресурсів та оптимізована обробка операцій дозволяють оптимізувати використання апаратних ресурсів та підвищувати швидкодію системи в цілому.

Реалізація функціональних структур даних в ядрі Linux є ключовим аспектом для забезпечення стабільної та ефективної роботи операційної системи. Використання різноманітних структур, таких як списки, черги та дерева, дозволяє оптимізувати роботу з пам'яттю, виконувати операції швидкого доступу до даних та забезпечувати високу продуктивність системи

в цілому. Подальший розвиток та оптимізація цих структур в ядрі Linux визначатиме ефективність операційної системи у майбутньому[9].

Зростання складності операційних систем та збільшення їхніх можливостей вимагає постійного розширення функціональних структур даних в ядрі Linux. Наприклад, введення багаторівневих структур для оптимізації доступу до об'єктів або впровадження алгоритмів балансування для деревовидних структур може покращити продуктивність та ефективність операцій.

Функціональні структури даних в ядрі Linux також використовуються для реалізації систем кешування, що сприяє швидкому доступу до часто використовуваних даних. Оптимізація алгоритмів кешування та їхнє вдосконалення грають ключову роль у підвищенні ефективності операційних систем.

Забезпечення підтримки реал-тайм операцій в ядрі Linux вимагає спеціальних функціональних структур для керування та планування завдань у реальному часі. Ефективна реалізація цих структур є важливим етапом у розширенні можливостей операційних систем для вимог реального часу.

Функціональні структури даних грають важливу роль у забезпеченні безпеки та стійкості операційних систем. Розробка ефективних механізмів контролю доступу, аудиту та виявлення вразливостей базується на правильній реалізації структур даних для зберігання та обробки конфіденційної інформації.

Аналіз реалізації функціональних структур даних в ядрі Linux свідчить про їхню ключову роль у забезпеченні продуктивності, ефективності та стабільності операційних систем. Постійний розвиток та оптимізація цих структур є важливим напрямком для покращення функціональності та відповіді операційних систем на зростаючі вимоги користувачів.

Розширення функціональних структур даних також пов'язане з здатністю операційної системи масштабуватися. Зокрема, ефективна обробка великої кількості об'єктів та подій вимагає оптимізованого використання структур даних та алгоритмів, що дозволяють системі масштабуватися як на рівні апаратного, так і на програмного забезпечення.

Реалізація функціональних структур даних також безпосередньо впливає на взаємодію операційної системи з апаратним забезпеченням. Інтелігентне керування ресурсами та використання структур, що оптимізовані під конкретне обладнання, може значно підвищити продуктивність та забезпечити ефективне використання апаратних ресурсів.

Однією з ключових особливостей реалізації функціональних структур даних в ядрі Linux є велика активна спільнота розробників. Відкритий характер операційної системи сприяє постійній зміні та вдосконаленню коду, що реалізує ці структури, завдяки внесенню виправлень, оптимізацій та нововведень з боку спільноти.

Аналіз реалізації функціональних структур даних в ядрі Linux демонструє їхню важливість для роботи операційної системи. Надалі розвиток цих структур повинен бути спрямований на підтримку новітніх технологій, підвищення продуктивності та забезпечення стабільної та безпечної роботи системи в умовах постійних змін у сфері обчислювальної техніки.

2.3.Особливості використання функціональних структур даних у користувацьких програмах Linux

Операційна система Linux відзначається високою гнучкістю та можливістю налаштування, що робить її привабливою для розробників. Одним із ключових аспектів, які визначають продуктивність та ефективність

користувацьких програм у Linux, є використання функціональних структур даних.

Функціональні структури даних відіграють важливу роль у взаємодії користувацьких програм з ядром операційної системи. Зокрема, вони використовуються для зберігання та обробки інформації, що стосується процесів, файлової системи, мережевої активності та інших системних аспектів.

Операційні системи Linux, які базуються на ядрі Linux, володіють високою ефективністю та надійністю, частково завдяки використанню оптимізованих структур даних. Функціональні структури даних відіграють ключову роль у взаємодії різних компонентів операційної системи, сприяючи швидкому та ефективному виконанню операцій.

Однією з основних функціональних структур даних є списки. Вони використовуються для організації та керування даними, такими, як процеси, файли, та ресурси. В операційних системах Linux списки є невід'ємною частиною багатьох механізмів ядра, таких як списки процесів, списки вільних блоків пам'яті, тощо. Вони забезпечують ефективний доступ до даних та швидке їхнє оновлення.

Деревовидні структури даних використовуються для представлення ієрархії процесів в операційній системі Linux. Це особливо важливо для керування взаємозв'язком між процесами та їхнім життєвим циклом. Забезпечуючи шляхи спадкоємства та батьківщини, ці структури дозволяють оптимально керувати процесами та їх ресурсами.

Хеш-таблиці використовуються для ефективного пошуку даних в системі Linux. Вони дозволяють швидко знаходити необхідні елементи за допомогою хеш-функцій. Це особливо корисно при роботі зі списками файлів, таблицями процесів та іншими ресурсами, що потребують швидкого доступу[8].

Функціональні структури даних, такі як буфери та кеші, грають ключову роль в оптимізації роботи з пам'яттю. Вони забезпечують швидкий доступ до часто використовуваних даних та зменшують час доступу до дисків чи інших повільних ресурсів. Це дозволяє підвищити продуктивність операційної системи.

Для забезпечення взаємовиключення та синхронізації між процесами в операційних системах Linux використовуються семафори та м'ютекси. Ці функціональні структури даних дозволяють ефективно керувати доступом до критичних ресурсів, запобігаючи конфліктам та забезпечуючи коректну роботу програм.

Стеки використовуються для зберігання локальних змінних, адрес повернення та інших даних, пов'язаних з викликами функцій та процедур. Це важливо для правильної обробки викликів та рекурсії, а також для забезпечення ефективної роботи операційної системи при виконанні різних завдань.

Функціональні структури даних, такі як менеджери пам'яті, використовуються для ефективного розподілу та вивільнення пам'яті в операційних системах Linux. Вони грають ключову роль у забезпеченні оптимального використання ресурсів та попередженні витоків пам'яті, що можуть призвести до нестабільної роботи системи.

Операційні системи Linux використовують функціональні структури даних для організації файлової системи. Деревовидні структури, хеш-таблиці та інші механізми допомагають в ефективному управлінні файлами та каталогами, а також в забезпеченні швидкого доступу до даних на диску.

У Linux, кожен процес в системі має свою функціональну структуру даних, яка містить інформацію про стан процесу, його пам'ять, відкриті файли та інші параметри. Використання таких структур дозволяє ефективно керувати процесами, забезпечуючи їхню ізоляцію та безпеку.

Linux, як відкрита операційна система, забезпечує широкий функціонал для керування процесами та завданнями. Ця стаття розглядає специфікації цих процесів та завдань, які є ключовими компонентами операційної системи.

Процес у Linux є виконавчою одиницею, яка виконує конкретну задачу в системі. Кожен процес має унікальний ідентифікатор, стек, контекст процесу та інші атрибути. Система специфікує структуру процесу та механізми його створення, завершення та взаємодії з іншими процесами.

Завдання в Linux представляють собою виконавчі одиниці, які можуть бути запущені в межах процесу. Завдання включають в себе виконувані файли, які розширюють функціональність системи. Специфікації завдань визначають правила їхнього виконання, взаємодії та можливості обміну даними між ними.

Система Linux визначає механізми розподілу ресурсів між процесами та завданнями. Це включає в себе управління пам'яттю, процесами введення/виведення та іншими системними ресурсами. Специфікації розкладування задач регулюють ефективність використання ресурсів та забезпечують стабільність системи[2].

Лінукс використовує системні виклики для взаємодії між ядром та процесами. Специфікації цих викликів визначають можливості процесів отримувати доступ до системних ресурсів та взаємодіяти з ядром. Міжпроцесорна взаємодія включає механізми спільного доступу до ресурсів та комунікації між процесами.

Специфікації процесів та завдань у Linux грають важливу роль у забезпеченні ефективності та стабільності операційної системи. Розробка та дотримання цих специфікацій визначають функціональність системи та її здатність взаємодіяти з різноманітними завданнями та програмами. Для розробників та системних адміністраторів розуміння цих специфікацій є

ключовим для оптимального використання можливостей операційної системи Linux.

Функціональні структури даних взаємодіють із файловою системою, забезпечуючи доступ до файлів та директорій. Особливо важливим є використання таких структур для взаємодії із системним викликом, що дозволяє користувачеві та програмам працювати з файловою системою.

Однією з ключових складових операційної системи Linux є її файлова система, яка відповідає за організацію та управління файлами та каталогами. Розуміння принципів роботи цієї системи має важливе значення для розробників, системних адміністраторів та користувачів Linux.

Основні концепції файлової системи Linux Файлова система Linux ґрунтується на принципах ієрархічності та структурованості. Основні об'єкти - це файли та каталоги, які організовані у деревоподібній структурі. Кожен файл має унікальний ідентифікатор та атрибути, що визначають його властивості.

Функціональні особливості Файлова система Linux підтримує широкий спектр функцій, включаючи контроль доступу до файлів, відновлення випадково видалених файлів, шифрування та стискання даних. Інтеграція з різноманітними типами файлових систем, такими як ext4, btrfs, та іншими, розширює можливості операційної системи.

Інтеграція та Взаємодія Файлова система Linux є важливою складовою для взаємодії з іншими підсистемами операційної системи, такими як процеси, мережа, та введення/виведення. Це сприяє ефективному використанню ресурсів та підвищує продуктивність операційної системи в цілому.

Перспективи розвитку З урахуванням сучасних технологічних тенденцій, таких як збільшення обсягів даних та використання технологій хмарного зберігання, файловій системі Linux необхідно постійно

вдосконалюватися. Розробка нових методів оптимізації, підтримка нових типів сховищ та удосконалення інтерфейсу користувача - це лише кілька напрямків для майбутнього розвитку.

Робота з файловою системою Linux є ключовим аспектом ефективного функціонування операційної системи. Розуміння її основних концепцій та функціональних особливостей, а також постійне вдосконалення на основі сучасних вимог, визначають успіх та стабільність операційної системи Linux в сучасному інформаційному середовищі.

Функціональні структури даних в Linux використовуються для управління мережевою активністю, зокрема для забезпечення обміну даними між процесами через мережу. Вони дозволяють налагоджувати мережеві з'єднання та забезпечують високий рівень надійності мережевих операцій.

Операційна система Linux визначається своєю високою стабільністю, ефективністю та відкритістю вихідних кодів. Одним із ключових аспектів, який визначає її роботу, є мережева активність та спілкування процесів.

Мережева активність в Linux є ключовим елементом, особливо в сучасному великому масштабі використання мережі. Ядро Linux підтримує широкий спектр мережевих протоколів, включаючи TCP/IP, UDP та інші. Реалізація мережевої активності базується на системі сокетів, яка дозволяє програмам взаємодіяти з мережею через стандартний інтерфейс.

Процеси в Linux можуть спілкуватися між собою через різні механізми, такі як спільна пам'ять, черги повідомлень та семафори. Ці механізми грають важливу роль у взаємодії між процесами на одній системі або навіть на різних системах, що дозволяє створювати розподілені системи та великі мережеві додатки.

Мережева активність та спілкування процесів можуть суттєво впливати на ефективність операційної системи. Ефективне управління ресурсами, оптимізація мережевого взаємодії та ефективне використання

спільної пам'яті є ключовими в задачах розробки великих мережевих додатків.

Однією з ключових переваг використання функціональних структур даних у Linux є їхня взаємодія та інтеграція. Вони дозволяють розробникам створювати комплексні, але добре організовані системи, які легко масштабуються та підтримуються.

За останні десятиліття розвитку високотехнологічних рішень в області інформаційних технологій, велика увага приділяється оптимізації та інтеграції операційних систем у складних інформаційних середовищах. Linux, відома своєю надійністю, високою продуктивністю та відкритим вихідним кодом, займає важливе місце в цьому контексті.

Однією з ключових особливостей системної інтеграції Linux є його ядро, що визначає основні принципи функціонування операційної системи. Ядро Linux підтримує множину пристроїв, файлових систем і мережевих протоколів, надаючи основу для побудови комплексних інформаційних систем.

Linux відрізняється модульною архітектурою, що дозволяє інтегрувати новий функціонал безперервно, не порушуючи стабільність системи. Модульність сприяє швидкій адаптації до змін у вимогах користувачів і виробників програмного забезпечення, забезпечуючи гнучкість та розширюваність.

Загальна системна інтеграція Linux використовує системи керування пакетами, такі як APT (Advanced Package Tool) або YUM (Yellowdog Updater Modified), для автоматизації встановлення, оновлення та видалення програм. Це сприяє створенню єдиної конфігурації системи, забезпечуючи стабільність та узгодженість середовища.

Ще однією важливою складовою інтеграції Linux є технології віртуалізації та контейнеризації, такі як KVM (Kernel-based Virtual Machine)

та Docker. Вони дозволяють ефективно використовувати ресурси серверів, розділяючи їх між різними додатками та операційними середовищами.

Загальна системна інтеграція Linux визначається надійністю, гнучкістю та відкритістю, що робить її важливим компонентом сучасних інформаційних систем. Розвиток та оптимізація цієї інтеграції допоможе вирішити виклики, пов'язані зі зростанням обсягів даних та розширенням функціональності інформаційних систем у майбутньому.

2.4. Висновки до розділу 2

Використання функціональних структур даних у користувацьких програмах Linux грає визначальну роль у забезпеченні високої продуктивності та надійності операційних систем. Їхні особливості дозволяють ефективно керувати процесами, взаємодіяти з файловою системою та підтримувати мережеві операції, створюючи стійкі та високопродуктивні програмні рішення.

Функціональні структури даних в операційних системах Linux грають важливу роль у забезпеченні ефективності, швидкодії та стабільності операційної системи. Використання оптимізованих структур даних дозволяє забезпечити оптимальне використання ресурсів та підтримує високу продуктивність у різноманітних сценаріях використання.

Функціональні структури даних в операційних системах Linux виконують ключову роль у забезпеченні високої продуктивності, стабільності та ефективності. Їх правильний вибір та оптимізоване використання дозволяють операційній системі ефективно керувати ресурсами, забезпечуючи плавну та надійну роботу в різноманітних умовах використання.

РОЗДІЛ 3
ПРАКТИЧНІ АСПЕКТИ ВИКОРИСТАННЯ ФУНКЦІОНАЛЬНИХ СТРУКТУР ДАНИХ У LINUX

3.1. Реалізація функціональних структур даних у проектах

Функціональні структури даних визначають ефективний спосіб організації та управління інформацією у програмному проекті. Їхній вибір та правильна реалізація може суттєво впливати на продуктивність, швидкість та загальну якість програмного забезпечення. Функціональні структури даних включають в себе різноманітні концепції, такі як списки, стеки, черги, дерева та графи. Кожна з них має свої унікальні властивості та використання в залежності від завдань проекту. Важливо враховувати аспекти ефективності та споживання ресурсів при виборі конкретної структури даних для реалізації.

Перед тим як обирати конкретні функціональні структури даних, важливо розробити чітку специфікацію власного проекту. Це включає в себе визначення завдань, які необхідно вирішити, та обрання відповідних структур для їх виконання.

Функціональні структури даних характеризуються тим, що вони визначаються функціональними операціями, які можна виконати над ними. Реалізація таких структур дозволяє покращити читабельність коду, полегшити супровід та оптимізувати використання ресурсів[12].

Перед початком реалізації функціональних структур даних слід ретельно аналізувати вимоги проекту. Визначення типів операцій, які часто виконуються над даними, та розуміння шаблонів доступу до даних є ключовими етапами. Наприклад, якщо необхідно виконувати часті операції вставки та вилучення, може бути доцільно використовувати структури даних, які оптимізовані для цих операцій, наприклад, динамічні масиви чи списки.

В процесі розробки функціональних структур даних для власного проекту важливо враховувати особливості операційної системи Linux. Це може включати в себе використання специфічних системних викликів, оптимізацію роботи з файловою системою та ефективне використання пам'яті.

Реалізація функціональних структур даних в проекті вимагає уважної розробки та впровадження. Основні принципи, такі як ефективність пам'яті, швидкодія та читабельність коду, повинні бути враховані. Документування коду, включаючи чітке описання вибору структур даних та їх властивостей, є важливим елементом для полегшення розуміння та співпраці в команді розробників.

Процес розробки рідко коли залишається незмінним протягом усього життєвого циклу проекту. Важливо мати можливість адаптувати реалізацію функціональних структур даних до змін у вимогах та внутрішніх потреб проекту. Гнучкість коду та здатність до модифікацій визначають його довговічність та можливість швидкого реагування на зміни.

Під час реалізації функціональних структур даних важливо передбачити можливі ситуації виняткового виконання програми та встановити ефективний механізм обробки помилок. Неправильна обробка помилок може призвести до витоку ресурсів, некоректної роботи програми або втрати даних. Забезпечення надійної системи управління винятками є важливим аспектом реалізації функціональних структур даних.

Після впровадження функціональних структур даних в проект, важливо забезпечити їхню подальшу підтримку та розвиток. Це може включати в себе виправлення помилок, вдосконалення ефективності, адаптацію до нових технологій та вимог користувачів. Активна комунікація в команді розробників та документація проекту грають ключову роль у забезпеченні успішної експлуатації функціональних структур даних.

Реалізація функціональних структур даних у власних проектах є складним завданням, яке вимагає ретельного аналізу та вибору. Відповідне використання таких структур може значно покращити продуктивність та якість програмного забезпечення. Важливо пам'ятати про постійний моніторинг та оптимізацію для досягнення оптимальних результатів у процесі розробки.

Від вибору підходящих структур до їхньої ефективної реалізації та подальшої підтримки, кожен етап є важливим для досягнення успіху у розробці програмного забезпечення. Наявність глибокого розуміння принципів функціональних структур даних та їхнє власне застосування в проекті може значно полегшити завдання розробників та покращити якість результату.

Розробка та реалізація функціональних структур даних у власних проектах для Linux є ключовим етапом в розширенні можливостей операційної системи. Ефективне використання таких структур може поліпшити продуктивність, надійність та зручність використання програмного забезпечення в цьому середовищі.

3.2. Інструменти та ресурси для розробників у галузі функціональних структур даних

Функціональні структури даних в сучасному програмуванні на Linux набувають важливості завдяки їх ефективності та здатності спрощувати

складні завдання обробки інформації. Розробники, які працюють у цій галузі, користуються різноманітними інструментами та ресурсами для реалізації та оптимізації функціональних структур даних.

Основні інструменти у галузі функціональних структур даних:

- GNU Compiler Collection (GCC): Одним із ключових інструментів для розробки функціональних структур даних у Linux є GCC. Цей компілятор дозволяє розробникам генерувати ефективний машинний код і оптимізувати програми для платформи Linux, що дозволяє ефективно використовувати функціональні можливості.
- Valgrind: Для виявлення помилок у роботі з пам'яттю та оптимізації використання ресурсів, розробники можуть скористатися Valgrind. Цей інструмент дозволяє виявляти утилізацію пам'яті та допомагає в усуненні проблем під час розробки функціональних структур даних.
- GDB (GNU Debugger): Для відлагодження програм, особливо тих, що працюють із складними функціональними структурами даних, GDB є важливим інструментом. Він надає можливість крокування коду, вивчення значень змінних та інших корисних функцій для ефективного аналізу програми.

Ресурси у галузі функціональних структур даних:

- Linux Kernel Source Code: Вивчення вихідного коду ядра Linux є непересічним ресурсом для розробників функціональних структур даних. Відкритий код надає можливість вивчення реалізацій та оптимізацій, які використовуються в одній з найпоширеніших операційних систем.
- Linux Documentation Project: Проект з документації Linux надає розробникам доступ до докладних пояснень і прикладів щодо роботи з функціональними структурами даних в середовищі Linux.

- LWN.net (Linux Weekly News): Як інтернет-ресурс, LWN.net забезпечує розробників останніми новинами, статтями та обговореннями, пов'язаними із розробкою для Linux. Це джерело інформації може бути важливим для відстеження останніх тенденцій у галузі функціональних структур даних[11].

Мови програмування C та C++ є вибором багатьох розробників для роботи з функціональними структурами даних у Linux. Вони забезпечують прямий доступ до системних ресурсів та низькорівневих операцій, сприяючи ефективній реалізації складних алгоритмів.

Використання спеціалізованих бібліотек, таких як libavl, liblfds, чи libcds, може значно полегшити розробку та оптимізацію функціональних структур даних. Ці бібліотеки містять реалізації різних структур, таких як дерева, графи, та списки, які можуть бути використані у різноманітних сценаріях.

Застосування інструментів для аналізу та візуалізації даних, таких як DTrace або SystemTap, може сприяти відстеженню та вирішенню проблем у функціональних структурах даних. Ці інструменти надають можливість спостерігати за виконанням програми та виявляти можливі дефекти чи недоліки.

Використання Just-In-Time (JIT) компіляції може значно покращити продуктивність функціональних структур даних. Вона дозволяє програмам перетворювати вихідний код на машинний в процесі виконання, адаптуючись до конкретного середовища та оптимізуючи його для конкретної задачі.

Розробники можуть активно взаємодіяти та обмінюватися знаннями через спільноти розробників Linux, форуми, або соціальні мережі, такі як GitHub. Це дозволяє виявляти найкращі практики, розвивати ідеї та спільно розв'язувати проблеми, пов'язані з функціональними структурами даних.

Загальний внесок у розвиток функціональних структур даних у Linux залежить від поєднання вищезазначених інструментів та ресурсів, а також від активної участі розробників у спільноті. Передові підходи та найновіші технології допомагають підтримувати високий рівень ефективності та функціональності у програмному забезпеченні, що працює на операційній системі Linux.

З урахуванням росту кількості ядер процесорів та розподілених обчислень, розробники функціональних структур даних повинні вдосконалювати свої рішення для підтримки паралельної обробки та ефективної роботи в розподілених середовищах.

З урахуванням зростання кількості кіберзагроз та атак на програмне забезпечення, розробники повинні приділяти увагу аспектам безпеки та відмовостійкості своїх функціональних структур даних.

З ростом обсягів даних у сучасних застосунках, розробники повинні шукати оптимальні рішення для ефективної роботи з великими обсягами інформації, використовуючи функціональні структури даних.

Нові мови програмування та інструменти можуть з'явитися, пропонуючи розробникам більш ефективні та зручні засоби для створення функціональних структур даних.

Інтеграція функціональних структур даних із системами машинного навчання та штучного інтелекту відкриває нові можливості для розвитку інтелектуальних та адаптивних програм.

Розробники повинні прагнути до стандартизації своїх функціональних структур даних для забезпечення можливості взаємодії з іншими платформами та середовищами.

У галузі функціональних структур даних у Linux інструменти та ресурси грають ключову роль у вдосконаленні та оптимізації розробки програмного забезпечення. Використання вищезазначених інструментів та

ресурсів допомагає розробникам ефективно впроваджувати функціональні структури даних та створювати продуктивні програми для операційної системи Linux.

3.3. Вплив використання функціональних структур даних на продуктивність системи

Сучасні операційні системи, зокрема Linux, стикаються з постійним підвищенням вимог до продуктивності та ефективності. Одним з ключових аспектів оптимізації є використання ефективних структур даних для забезпечення швидкодії та оптимального використання ресурсів. У цьому контексті, функціональні структури даних відіграють важливу роль у покращенні продуктивності системи.

Функціональні структури даних - це спеціальні типи даних та алгоритми, спроектовані для оптимізації певних операцій обробки інформації. У порівнянні з традиційними структурами даних, такими як масиви чи списки, функціональні структури даних можуть забезпечувати більш ефективний доступ та модифікацію даних.

У Linux, функціональні структури даних широко використовуються у різноманітних компонентах ядра та системних програм. Наприклад, дерева пошуку, такі як Red-Black trees, використовуються для швидкого пошуку та вставки даних. Бінарні кучі використовуються для ефективного управління пам'яттю, а хеш-таблиці - для прискорення пошуку за ключем.

Використання функціональних структур даних може значно поліпшити продуктивність системи у багатьох аспектах. Швидкість доступу до даних, операції пошуку та сортування можуть бути оптимізовані завдяки використанню відповідних функціональних структур. Крім того, ефективне управління ресурсами, таким як пам'ять, може призвести до зменшення затрат і підвищення загальної ефективності системи.

Незважаючи на позитивний вплив використання функціональних структур даних, існують виклики та перспективи. Важливо збалансувати вибір конкретної структури даних залежно від конкретних потреб системи. Додатково, розробники повинні уникати надмірного використання складних структур даних, які можуть вплинути на читабельність та підтримку коду.

Одним із прикладів використання функціональних структур даних у Linux є реалізація системного виклику для роботи з файловою системою. Використання бінарних дерев пошуку для ефективного індексування та пошуку файлів може значно покращити швидкодію операцій з файлами, забезпечуючи швидкий доступ та модифікацію даних.

Функціональні структури даних також виявляють свою корисність у роботі з мережевими операціями в Linux. Наприклад, графи можуть використовуватися для представлення та оптимізації мережевих зв'язків між процесами та ресурсами системи, що дозволяє швидко та ефективно керувати мережевим трафіком та забезпечувати високу продуктивність[5].

Здатність вирішувати проблеми та вдосконалювати продуктивність системи у Linux вимагає поєднання правильного вибору функціональних структур даних з аналізом та оптимізацією коду. Наприклад, уникати зайвих операцій копіювання даних, правильне використання кеш-пам'яті та розумне керування пам'яттю можуть внести суттєвий внесок у загальну продуктивність системи.

Розробка нових алгоритмів та структур даних, придатних для високо навантажених систем, є ключовим завданням для забезпечення стабільної та швидкої роботи операційних систем.

Забезпечення безпеки є іншим аспектом, в якому функціональні структури даних можуть відігравати ключову роль у Linux. Використання надійних структур для зберігання та обробки конфіденційних даних, таких як

хеш-таблиці та криптографічні алгоритми, може зменшити ризики безпеки та підвищити стійкість системи до атак.

У світі, де багатоядерні системи стають стандартом, функціональні структури даних мають адаптуватися до паралельних обчислень. Розробка структур, які ефективно використовують можливості багатоядерних архітектур, може значно покращити загальну продуктивність системи.

Використання методів машинного навчання для автоматизації оптимізації функціональних структур даних є перспективним напрямком досліджень. Алгоритми машинного навчання можуть аналізувати поведінку системи та автоматично адаптувати вибір структур даних для максимальної ефективності.

У світі швидко розвиваючихся технологій, де продуктивність є ключовим аспектом функціонування операційних систем, використання функціональних структур даних у Linux залишається невід'ємною частиною стратегії оптимізації. Забезпечуючи ефективний доступ, оптимізацію роботи з ресурсами та підтримку безпеки, ці структури допомагають забезпечити високий рівень продуктивності та стійкість у Linux-середовищі. Розвиток та вдосконалення цих підходів визначатиме ефективність операційних систем у майбутньому.

3.4. Висновки до розділу 3

Використання функціональних структур даних у Linux є важливим елементом оптимізації продуктивності системи. Правильний вибір та ефективне використання цих структур можуть призвести до значущих поліпшень у швидкодії та ресурсоємності системи. Дослідження та вдосконалення функціональних структур даних залишаються актуальним завданням для розвитку операційних систем та їхнього ядра.

Розробка функціональних структур даних у Linux — це постійно зростаючий процес, який вимагає від розробників не лише технічної експертизи, але й здатності адаптуватися до змін у технологічному ландшафті. Використання розширених інструментів та активна участь у спільноті дозволяють розробникам досягати нових висот у розробці програмного забезпечення для Linux.

Використання функціональних структур даних в Linux є необхідним компонентом стратегії оптимізації продуктивності. Відправною точкою є вибір правильних структур для конкретного завдання, адаптація їх до потреб системи та постійна оптимізація для вирішення викликів сучасного інформаційного середовища.

Застосування функціональних структур даних у власних проектах для Linux дозволяє досягти більшої ефективності та швидкодії. Вони сприяють створенню більш стабільних та гнучких рішень, що відповідають вимогам сучасного програмного забезпечення.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Екосистема Linux, визнана своєю відкритістю та гнучкістю, заснована на складній системі функціональних структур даних, що визначають її унікальні характеристики та можливості.

Аналіз функціональних структур даних в екосистемі Linux дозволяє зрозуміти, як система досягає високої продуктивності та надійності. Використання різноманітних структур, таких як списки, дерева та хеш-таблиці, сприяє ефективному управлінню ресурсами та оптимізованому доступу до даних. Синхронізація та механізми блокування гарантують коректну обробку даних у багатозадачних та багатокористувацьких середовищах.

Також слід відзначити, що функціональні структури даних в екосистемі Linux є ключовим елементом для забезпечення її стабільності та високої продуктивності. Подальший розвиток цих структур та їхнє вдосконалення важливі для забезпечення майбутньої ефективності та розширення функціональності Linux.

Однією з ключових перспектив розвитку функціональних структур даних в екосистемі Linux є пошук оптимальних рішень для забезпечення масштабованості системи. Спрощення та оптимізація алгоритмів, використаних у функціональних структурах, можуть покращити продуктивність системи на великих об'ємах даних та в умовах інтенсивного навантаження.

					ДТЕУ 121 02-12.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		01.11.23	*Функціональні структури даних в екосистемі Linux*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Тищенко Д.О.		01.11.23		*ВП*	*56*	*60*
Гарант		Котенко Н.О.		01.11.23		*Факультет інформаційних технологій 2м курс, 2 група*		
Розробив		Москаленко В. В.		01.11.23	*Висновки та пропозиції*			

Функціональні структури даних повинні підтримувати відкритий інтерфейс для сприяння співпраці та розвитку великої кількості додаткових функцій та модулів. Забезпечення стандартів та відкритості сприятиме створенню розширюваних та інтегрованих рішень у межах екосистеми Linux.

Узагальнюючи, слід відзначити, що функціональні структури даних в екосистемі Linux відіграють важливу роль у забезпеченні її функціональності та ефективності. Забезпечення безпеки, масштабованості та відкритості є ключовими викликами для майбутнього розвитку.

Пропозиції:

- Оптимізація роботи з файловою системою: Подальше вдосконалення функціональних структур для оптимізації роботи з файловою системою, щоб забезпечити ефективний доступ до даних та підтримку великих об'ємів інформації.
- Розширення можливостей мережі: Дослідження та розробка нових функціональних структур для підтримки високопродуктивних мережевих операцій, зокрема для обробки пакетів та оптимізації передачі даних.
- Адаптація до сучасних архітектур: Вивчення та розробка функціональних структур, які адаптуються до сучасних архітектур, таких як мультіядерні процесори та архітектури з великою кількістю ядер, для оптимізації використання обчислювальних ресурсів.
- Розвиток засобів віртуалізації: Дослідження можливостей функціональних структур для підтримки та оптимізації технологій віртуалізації, зокрема для контейнеризації та управління віртуальними середовищами.
- Удосконалення механізмів синхронізації: Подальший розвиток та оптимізація механізмів синхронізації для забезпечення ефективної роботи в

умовах великої кількості паралельних операцій та високозавантажених систем.

- Підтримка енергоефективності: Розробка функціональних структур, спрямованих на оптимізацію використання енергії та підтримку енергоефективних технологій, особливо для портативних та вбудованих систем.
- Вдосконалення засобів роботи з пам'яттю: Дослідження нових підходів до роботи з пам'яттю та розробка функціональних структур, які ефективно управляють та оптимізують використання системної пам'яті.

Цілком очевидно, що функціональні структури даних в екосистемі Linux є основним фундаментом, на якому ґрунтується її успіх. Розуміння та удосконалення цих структур є важливою метою для розробників та спільноти Linux, які призначаються не лише продовжити традиції відкритості та гнучкості, але й визначити нові стандарти у світі операційних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Garcia, M., & Lins, R. "Functional Data Structures in Linux: An In-depth Analysis." – Proceedings of the International Conference on Linux Systems, 2022-150-165p.
2. Garcia, M., & Lins, R. (2022). "Functional Data Structures in Linux: An In-depth Analysis." – Proceedings of the International Conference on Linux Systems,-2022-150-165p.
3. Smith, J. (2023). "Optimizing Linux Kernel Data Structures for Performance." – Journal of Open Source Software Engineering, 10(2), 2023-45-60p.
4. Wang, H., & Li, Y. (2022). "Functional Data Structures in Practice: Case Studies from the Linux Ecosystem." – Conference on Computer Systems, 2022-250-265p
5. Білоус, А. І. Оптимізація та використання функціональних структур даних в Linux API. – Київ: Видавництво "Технічна література", 2021–209с.
6. Григоренко, С. В. Функціональні структури даних в контексті розробки драйверів для Linux. – Одеса: Видавництво "Наукова Освіта",2021 – 168с.
7. Ковальчук, Н. В. Сучасні техніки реалізації функціональних структур даних в програмах для Linux. – Дніпро: Видавництво "Професіонал", 2021–197с.

					ДТЕУ 121 02-12.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		01.11.23	*Функціональні структури даних в екосистемі Linux*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Тищенко Д.О.		01.11.23		*СВД*	*59*	*60*
Гарант		Котенко Н.О.		01.11.23		*Факультет інформаційних технологій 2м курс, 2 група*		
Розробив		Москаленко В. В.		01.11.23	*Список використаних джерел*			

8. Молдер, Ж., Лавердюр, Ж., & Стиверс, У. Лінійні структури даних і їхнє використання. – Київ: Видавництво "TechBooks", 2021р.-243с.
9. Офіційна документація Linux Kernel [Електронний ресурс]. – Режим доступу: https://www.kernel.org/doc/html/latest/.
10. Петренко, О. В. Основи розробки ядра Linux: функціональні структури даних та їх застосування. – Київ: Видавництво "Університетська книга", 2022 –218с.
11. Сидоров, І. М. (2023). Лінійні та функціональні структури даних у ядрі Linux: аналіз та оптимізація. – Львів: Видавництво "Техніка", 2023 – 184с.
12. Шевченко, Ю. О. Розвиток функціональних структур даних у сучасному Linux Kernel. – Харків: Видавництво "Наукова думка", 2022 –199с.

ТЕХНІЧНЕ ЗАВДАННЯ

Технічне завдання для знаходження найбільш віддаленого елементу в масиві за допомогою наявних індексів

1. Мета проекту: Створення функції для знаходження найбільш віддаленого елементу в масиві за допомогою наявних індексів.

2. Функціональні вимоги:

- Розробка функції для пошуку найбільш віддаленого елементу.
- Можливість використання функції з існуючими індексами масиву.
- Підтримка різних типів даних в масиві.

3. Нефункціональні вимоги:

- Ефективність: Функція повинна працювати швидко навіть при великих розмірах масиву.
- Надійність: Програмний код повинен бути стійким до помилок і винятків.
- Читабельність: Код повинен бути читабельним та добре документованим.

4. Технічні вимоги:

- Мова програмування: Реалізація повинна бути виконана мовою програмування Java.
- Структура даних: Використовувати оптимальні структури даних для забезпечення ефективності алгоритму.

					ДТЕУ 121 02-12.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		01.11.23	*Функціональні структури даних в екосистемі Linux*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Тищенко Д.О.		01.11.23		*ТЗ*	*62*	*60*
Гарант		Котенко Н.О.		01.11.23		*Факультет інформаційних технологій 2м курс, 2 група*		
Розробив		Москаленко В. В.		01.11.23	*Технічне завдання*			

- Алгоритм: Розробити оптимальний алгоритм для знаходження найбільш віддаленого елементу.

5. Технічні вимоги:

- Мова програмування: Java
- Версія Java: Використовувати Java 8 або вище.
- Стандарт коду: Дотримання стандартів коду Java Coding Conventions.

6. Процес розробки:

- Версіонування коду: Використовувати систему контролю версій.
- Тестування: Розробити тестові набори для валідації коректності та швидкодії алгоритму.

7. Тестування:

- Одиничне тестування: Розробити тести для перевірки окремих функцій.
- Інтеграційне тестування: Провести тести для перевірки взаємодії з іншими частинами системи.

ПРОГРАМА ТА МЕТОДИКА ТЕСТУВАННЯ

Дане програмне забезпечення призначене для вирішення конкретної задачі - знаходження найбільш віддаленого елемента в масиві, до якого можна дістатися за допомогою наявних малих значень. Однією з основних функцій цієї програми є знаходження елемента в масиві, який знаходиться на найбільшому віддаленні від початкового пункту, і до якого можна дістатися, використовуючи лише наявні малі значення.

Функціонал програми

Програмне забезпечення виконує наступні ключові задачі:

- Зчитування вхідних даних: програма отримує вхідні дані, такі як масив чисел та малі значення, які вказують на обмеження допустимої відстані.
- Обчислення відстаней: для кожного елемента масиву програма обчислює відстань від початкової точки або елементу, використовуючи надані малі значення.
- Визначення найбільш віддаленого елемента: програма визначає елемент, який має найбільшу відстань від початкової точки та є доступним за допомогою заданих малих значень.
- Вивід результату: результатом виконання програми є визначений найбільш віддалений елемент, який можна досягти за допомогою наявних малих значень.

- Обробка помилок: програма враховує можливість помилок у вхідних даних, таких як некоректні типи або порожні масиви, та надає адекватні повідомлення про помилки.
- Оптимізація ресурсів: з урахуванням обмежень малих значень програма може використовувати оптимізовані алгоритми для швидкого визначення найбільш віддалених елементів.

Мета тестування

Метою тестування є перевірка того, що програма правильно визначає найбільший елемент масиву, до якого можна дістатися за допомогою маленьких значень. А також визначити відповідність програми вимогам та очікуванням.

Програма тестування

1. Тестування з правильними вхідними даними:
   - подавати програмі коректний масив та маленькі значення.
   - перевірити, чи повертає програма правильний результат.
2. Тестування з неправильними вхідними даними:
   - Подавати програмі некоректні типи даних або порожні дані.
   - Переконатися, що програма вірно обробляє помилки та повідомляє про них.
3. Тестування з різними масивами та значеннями:
   - Провести тести з різними розмірами масивів та різними значеннями маленьких.
   - Впевнитися, що програма працює коректно для різних умов.
4. Тестування меж:

Перевірити, як програма веде себе при найменших та найбільших можливих значеннях вхідних даних.

5. Тестування продуктивності:
   - Провести тести з великими масивами та виміряти час виконання.

- Переконатися, що програма працює ефективно навіть при великих обсягах даних.

6. Висновок

Програма розроблена з метою оптимізації використання ресурсів, таких як час та пам'ять, для ефективного вирішення даної задачі. Враховуючи обмеження на доступні малі значення, програма може забезпечувати ефективний підрахунок найбільш віддаленого елемента. Потенційно, це програмне забезпечення може бути корисним у ситуаціях, де важливо визначити найбільш віддалений об'єкт або точку доступу на основі обмежених ресурсів, наприклад, енергії чи часу. Програма може використовуватися в алгоритмічних завданнях, де необхідно визначити оптимальний шлях або об'єкт на основі конкретних умов та обмежень.

ДОДАТКИ

Додаток А

Лістинг програмного коду-підхід 1:мінімальна купа

```
define function furthestElements(array, smallValue, largeValue):
  for each i from 0 to array.length - 2:
    current_element = array[i]
    next_element = array[i + 1]
    diff = next_element - current_element
    if diff is 0 or diff is negative:
      continue to next iteration
    other, diff is a next desired element
```

Код

```
class Approach1 {
  public int furthestElement(int[] array, int smallValue, int largeValue) {
    // Створення черги пріоритетів з компаратором для побудови мінімальної купи
    Queue<Integer> largeValuesAllocations = new PriorityQueue<>((a, b) -> a - b);
    for (int i = 0; i < array.length - 1; i++) {
      int diff = array[i + 1] - array[i];
      // Якщо різниця між елементами негативна – продовжуємо далі цикл
      if ( diff <= 0) {
        continue;
      }
      // Інакше виділяємо велике значення
      largeValuesAllocations.add(diff);
      // Якщо не перевищено кількість largeValue, більше нічого не потрібно робити.
      if (largeValuesAllocations .size() <= largeValue) {
        continue;
      }
      // В іншому випадку нам потрібно буде піднятися з largeValuesAllocations
      smallValue -= largeValuesAllocations.remove();
```

```
            // Якщо це спричинило від'ємне значення smallValue, ми не зможемо дістатися
            // до i + 1
            if (smallValue < 0) {
                return i;
            }
        }
        // Якщо ми потрапили сюди, це означає, що у нас було достатньо матеріалів,
        // щоб покрити кожну різницю між елементами.
        return array.length - 1;
    }
```

Додаток Б

Лістинг програмного коду-підхід 2:максимальна купа

```
class Approach2 {
    public int furthestElement(int[] array, int smallValue, int largeValue) {
        // Створення черги пріоритетів з компаратором для побудови максимальної купи
        Queue<Integer> smallValuesAllocations = new PriorityQueue<>((a, b) -> b - a);
        for (int i = 0; i < array.length - 1; i++) {
            int diff = array[i + 1] - array[i];
            // Якщо різниця між елементами негативна – продовжуємо далі цикл
            if (diff <= 0) {
                continue;
            }
            // Інакше виділяємо велике значення
            smallValuesAllocations.add(diff);
            smallValue -= diff;
            // Якщо були використані усі smallValue, і немає жодного largeValue , то
            // не можливо ітеруватись далі циклом.
            if (smallValue < 0 && largeValue == 0) {
                return i;
            }
            // Інакше, якщо були використані усі smallValue, ми маємо замінити найбільше
            // smallValue на largeValue.
            if (smallValue < 0) {
                smallValue += smallValuesAllocations.remove();
                largeValue--;
            }
        }
        // Якщо ми потрапили сюди, це означає, що у нас було достатньо матеріалів,
        // щоб покрити кожну різницю між елементами.
        return array.length - 1;
    }
}
```

Додаток В

Лістинг програмного коду-підхід 3: двійковий пошук остаточного доступного елементу

```
class Approach3 {

    public int furthestElement(int[] array, int smallValue, int largeValue) {
        // Виконано двійковий пошук у масиві масиву, щоб знайти остаточну досяжну
        // будівлю.
        int lowBorder = 0;
        int highBorder = array.length - 1;
        while ( lowBorder < highBorder) {
            int middleValue = lowBorder + ( highBorder - lowBorder + 1) / 2;
            if (isSusceptible(middleValue, array, smallValue, largeValue)) {
                lowBorder = middleValue;
            } else {
                highBorder = middleValue - 1;
            }
        }
        return highBorder;
    }

    private boolean isSusceptible(int elementIndex, int[] array, int smallValue, int largeValue) {
        // Складено список усіх різниць між елементами, які нам потрібно зробити, щоб
        //  досягти elementIndex.
        List<Integer> diffs = new ArrayList<>();
        for (int i = 0; i < elementIndex; i++) {
            int leftHeight = array[i];
            int rightHeight = array[i + 1];
            if ( rightHeight <= leftHeight) {
                continue;
            }
            diffs.add( rightHeight - leftHeight);
        }
        Collections.sort(diffs);
```

```
        // А тепер визначимо, чи можна покрити всі ці різниці між елементами
        // за допомогою smallValue та largeValue.
        for (int diffs : diff) {
            if ( diff <= smallValue) {
                smallValue -= diff;
            } else if (largeValue >= 1) {
                largeValue -= 1;
            } else {
                return false;
            }
        }
        return true;
    }
}
```

Додаток Г

Лістинг програмного коду-підхід 4: покращено двійковий пошук для остаточно доступного елемента

```
class Approach4 {

    public int furthestElement(int[] array, int smallValue, int largeValue) {
        // Формування сортованого списку усіх значень
        List<int[]> sortedDiffs = new ArrayList<>();
        for (int i = 0; i < array.length - 1; i++) {
            int diff = array[i + 1] - array[i];
            if ( diff <= 0) {
                continue;
            }
            sortedDiffs.add(new int[]{diff, i + 1});
        }
        Collections.sort(sortedDiffs, (a,b) -> a[0] - b[0]);

        // Бінарний пошук
        int leftHeight = 0;
        int rightHeight = array.length - 1;
        while (leftHeight < rightHeight) {
            int middleElement = leftHeight + (rightHeight - leftHeight + 1) / 2;
            if (isSusceptible(middleElement, sortedDiffs, smallValue, largeValue)) {
                leftHeight = middleElement;
            } else {
                rightHeight = middleElement - 1;
            }
        }
        return rightHeight;
    }

    private boolean isSusceptible(int elementIndex, List<int[]> diffs, int smallValue, int
largeValue) {
```

```
        for (int[] diff : diffs) {
            // Отримаємо інформацію для різниці між елементами
            int diffElement = diff[0];
            int indexElement = diff[1];
            // Перевірка чи елемент знаходиться в певних межах
            if ( indexElement > elementIndex) {
                continue;
            }
            // Виділення smallValue якщо достатня кількість наявна
            // Інакше виділяємо largeValue якщо хоча б одна величина є в наявності.
            if ( diffElement <= smallValue) {
                smallValue -= diffElement;
            } else if (largeValue >= 1) {
                largeValue -= 1;
            } else {
                return false;
            }
        }
        return true;
    }
}
```