Федеральное бюджетное образовательное учреждение  
высшего образования

«Уфимский университет науки и технологий»

Кафедра технической кибернетики

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 90 |  |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  |  |  |  |

|  |
| --- |
| Проектирование гипотетической |
| операционной системы |
| с заданными характеристиками |

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту по дисциплине «**Операционные системы**»

|  |
| --- |
| 3271.202190.000 ПЗ |
| (обозначение документа) |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Группа |  |  | Фамилия И. О. | Подпись | Дата | Оценка |
| ИВТ–ПО–201Б |  |
|  |  |
| Студент | | | Кумушбаева Г. А. |  |  |  |
| Консультант | | | Рыжов Г. И. |  |  |  |
| Принял | | | Рыжов Г. И. |  |  |  |

**Уфа 2024**

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Уфимский университет науки и технологий»

Кафедра технической кибернетики

**ЗАДАНИЕ**

на курсовой проект по дисциплине «**Операционные системы**»

Студент \_Кумушбаева Г. А.\_\_\_ Группа \_ИВТ-ПО-201Б\_\_ Консультант \_Рыжов Г. И.\_\_\_\_

1. **Тема курсового проекта**

\_\_\_Проектирование гипотетической операционной системы с заданными характеристиками\_\_\_

1. **Основное содержание:** на основе заданного индивидуального задания спроектировать гипотетическую компьютерную операционную систему.

**Индивидуальное задание:**

**Режим работы компьютера:** однопрограммный

**Архитектура операционной системы:** экзоядро

**Подсистема управления процессами:**

– алгоритм планирования процессов: вытесняющий – циклический *LIFO (Last In, First Out)*;

– алгоритм организации взаимодействия процессов: алгоритм пекарни Лэмпорта;

– задача синхронизации процессов: задача «Обедающие философы»;

– способ борьбы с тупиками: предотвращение тупиков – алгоритм банкира.

**Подсистема управления памятью:** сегментное распределение памяти

**Подсистема управления файлами:**

– иерархия каталогов: дерево;

– логическая организация файловой системы: индексная логическая организация;

– физическая организация файловой системы: связанный список индексов.

**Загрузка операционной системы:** загрузка с жесткого диска компьютера.

**Запуск прикладных программ:** запуск с жесткого диска компьютера.

**Структура управляющих блоков базы данных операционной системы:**

– формирование базы данных операционной системы;

– схема взаимодействия управляющих блоков базы данных операционной системы.

1. **Требования к оформлению.** Пояснительная записка должна быть оформлена в редакторе *Microsoft Word* в соответствии с требованиями ГОСТ.

В пояснительной записке должны содержаться следующие разделы:

Титульный лист.

Задание на курсовой проект.

Реферат.

Содержание.

Введение.

Выбор архитектуры вычислительной системы.

Описание режима работы компьютера.

Описание архитектуры операционной системы.

Описание подсистемы управления процессами.

Описание подсистемы управления памятью.

Описание подсистемы управления файлами.

Проектирование графического интерфейса.

Проектирование командного языка.

Загрузка операционной системы.

Запуск прикладных программ.

Структура управляющих блоков базы данных операционной системы.

Описание функционирования операционной системы.

Заключение.

Список использованной литературы.

Дата выдачи \_\_\_14.10.2024\_\_\_\_\_\_\_\_\_ Дата окончания \_\_\_\_\_\_21.12.2024\_\_\_\_

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Рыжов Г. И.)

**РЕФЕРАТ**

Пояснительная записка – 49 с, 10 рис., 9 табл., 14 источн.

ОПЕРАЦИОНННАЯ СИСТЕМА, АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ, БАЗА ДАННЫХ, ПРОЦЕССЫ, ПАМЯТЬ, ФАЙЛОВАЯ СИСТЕМА, *LINUX*, ЯДРО, ГРАФИЧЕСКИЙ ИНТЕРФЕЙС, КОМАНДНЫЙ ЯЗЫК

Объект исследования – гипотетическая компьютерная операционная система.

Предмет исследования – процессы в гипотетической компьютерной операционной системе.

Цель курсового проекта:

* закрепление и углубление теоретических знаний в области современных операционных систем, активное применение знаний, полученных в лекционном курсе, на практических и лабораторных занятиях;
* формирование систематизированного представления о основополагающих принципах работы операционных систем, о структурах и механизмах, лежащих внутри операционных системы.

Методами описания операционной системы:

* выделены ключевые составляющие: абстракции (процессы, адресное пространство, файлы, потоки); функции (создание, запись, управление, открытие, распределение); конкретные реализации (архитектуры, алгоритмы);
* определены основные функции: управление устройствами компьютера (его ресурсами), непосредственное управление процессами, ведение файловой структуры, пользовательский интерфейс;
* рассмотрены основные компоненты операционной системы: управление файловой системой, командный язык, графический интерфейс.

Результатом работы является гипотетическая компьютерная операционная система с заданными характеристиками.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc187250009)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc187250010)

[1.1 Цель работы 6](#_Toc187250011)

[1.2 Задачи работы 6](#_Toc187250012)

[2 ОСНОВАЯ ЧАСТЬ 7](#_Toc187250013)

[2.1 Архитектура вычислительной системы 7](#_Toc187250014)

[2.2 Режим работы компьютера 10](#_Toc187250015)

[2.3 Архитектура операционной системы 12](#_Toc187250016)

[2.4 Подсистема управления процессами 13](#_Toc187250017)

[2.5 Подсистема управления памятью 24](#_Toc187250018)

[2.6 Подсистема управления файлами 26](#_Toc187250019)

[3 ПРОЕКТНАЯ ЧАСТЬ 33](#_Toc187250020)

[3.1 Проектирование графического интерфейса 33](#_Toc187250021)

[3.2 Проектирование командного языка 35](#_Toc187250022)

[3.4 Запуск прикладных программ 39](#_Toc187250023)

[3.5 Структура управляющих блоков базы данных операционной системы 40](#_Toc187250024)

[4 ОПИСАНИЕ ФУНКЦИОНИРОВАНИЯ ОПЕРАЦИОННОЙ СИСТЕМЫ 45](#_Toc187250025)

[ЗАКЛЮЧЕНИЕ 48](#_Toc187250026)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 49](#_Toc187250027)

**ВВЕДЕНИЕ**

Современные операционные системы стремятся к балансу между функциональностью, производительностью и эстетикой пользовательского интерфейса. В условиях роста требований к ресурсоэффективности и персонализации рабочего пространства всё большую популярность приобретают тайловые оконные менеджеры (tiling window managers, TWM) — программные компоненты, автоматически организующие окна приложений на экране без перекрытий и с минимальным участием пользователя. В отличие от традиционных сред рабочего стола (desktop environments, DE), таких как GNOME или KDE Plasma, тайловые менеджеры предоставляют полный контроль над компоновкой окон, управлением ресурсами и внешним видом системы, что особенно ценно для разработчиков, системных администраторов и пользователей, стремящихся к минимализму и высокой производительности.

С переходом от устаревшего X11-стека к современному протоколу Wayland экосистема Linux получила новые возможности в области безопасности, производительности и интеграции графических компонентов. Однако этот переход потребовал создания новых композиторов, сочетающих функции дисплейного сервера и оконного менеджера. Одним из наиболее перспективных решений в этой области стал Hyprland — динамический тайловый композитор, построенный на библиотеке wlroots, поддерживающий как строгий тайлинг, так и плавающие окна, а также современные визуальные эффекты: размытие (blur), тени, анимации и прозрачность.

Данная работа посвящена анализу пользовательской конфигурации Hyprland в среде Arch Linux — дистрибутива, известного своей гибкостью, минимализмом и ориентацией на опытного пользователя. В отчёте рассматриваются архитектурные особенности сборки, включающей:

композитор Hyprland с настроенным мультимониторным режимом и привязкой рабочих пространств,

информационную панель Waybar с кастомными модулями (раскладка клавиатуры, управление питанием, Bluetooth),

систему автоматизации через скрипты и автозапуск ключевых сервисов.

Цель отчёта — не только продемонстрировать работоспособность и эстетическую привлекательность данной конфигурации, но и проанализировать её с точки зрения архитектурной целостности, масштабируемости и соответствия современным практикам настройки Wayland-сред.

1. **ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

Предметной областью курсового проекта является проектирование гипотетической операционной системы, предназначенной для обеспечения эффективного управления аппаратными ресурсами компьютера.

## **Цель работы**

Целью проекта является разработка гипотетической операционной системы с базовым функционалом, обеспечивающим выполнение основных задач работы системы.

## **Задачи работы**

Задачи, которые должны быть выполнены в рамках курсового проекта:

* Анализ и выбор архитектуры операционной системы;
* Описание режима работы компьютера.;
* Описание архитектуры операционной системы;
* Описание подсистемы управления процессами;
* Описание подсистемы управления памятью;
* Описание подсистемы управления файлами;
* Проектирование графического интерфейса;
* Проектирование командного языка;
* Загрузка операционной системы и запуск прикладных программ с жесткого диска;
* Описать структуру управляющих блоков операционной системы;
* Описание функционирования операционной системы.

# **ОСНОВАЯ ЧАСТЬ**

## **Архитектура вычислительной системы**

Проектирование гипотетической операционной системы требует определения базовой архитектуры вычислительной системы, которая задаёт набор минимальных аппаратных средств. Архитектура описывает основные компоненты вычислительной техники, их параметры и взаимодействие. В данном проекте аппаратная платформа выбрана с учётом баланса между простотой реализации и достаточной производительностью. Базовая архитектура вычислительной системы представлена Рисунок 1.

Изображение выглядит как текст, диаграмма, План, Технический чертеж

Автоматически созданное описание

Рисунок 1 – Базовая архитектура вычислительной системы

### **Процессор**

**Процессор (или центральный процессор, ЦП)** — это основное вычислительное устройство в компьютере, которое выполняет команды программного обеспечения и обрабатывает данные. Его основная задача — управлять всеми вычислительными и логическими операциями системы**.**

В качестве центрального процессора используется архитектура x86, совместимая с процессорами семейства *Pentium*. Основные характеристики процессора:

* **тактовая частота**: 1.5 *GHz*. Это обеспечивает достаточную скорость обработки данных и выполнения системных задач;
* **количество ядер**: 1 ядро — для однопрограммного режима с минимальными требованиями;
* **набор команд**: поддержка x86 инструкций и расширений семейства *Pentium* (*MMX, SSE);*
* **размер кэш-памяти**: 256 Кб L2 кэш, что обеспечит быстрое выполнение часто используемых операций;
* **режим работы**: 16/32-битный (поддержка базовых операций для совместимости с простыми программами и задачами).

### **Память**

Для успешного функционирования гипотетической операционной системы необходимо правильно организованное управление памятью, включающее как оперативную память (*RAM – Random Access Memory*), так и долговременные хранилища данных, такие как жёсткий диск (*HDD – Hard Disk Drive*).

#### **Оперативная память**

**Оперативная память** является основным хранилищем для временных данных, с которыми активно работает процессор. Для гипотетической операционной системы выбраны следующие характеристики оперативной памяти:

* **объём:** 4 Гб – достаточный объем для загрузки операционной системы и работы базовых программ;
* **тип**: *DDR3* или аналогичный. Память *DDR3* обеспечивает оптимальное соотношение скорости и стоимости, что подходит для начальных этапов работы операционной системы;
* **задержки (*CAS Latency*)**: Сбалансированные задержки для стабильной работы системы.

#### **Жесткий диск**

Жёсткий диск (*HDD*) является важным компонентом вычислительной системы, который выполняет роль основного хранилища данных. На нём размещаются операционная система, программы, а также данные пользователя. В данном проекте жёсткий диск играет ключевую роль в обеспечении стабильной работы операционной системы, хранении файлов и организации виртуальной памяти.

* **Объём**: 128 Гб — выбранный объём предоставляет достаточно места для хранения операционной системы, драйверов, приложений, пользовательских данных, а также журналов системы и временных файлов;
* **тип интерфейса**: для подключения жёсткого диска выбран стандарт *SATA* (*Serial ATA*). Это современный и распространённый интерфейс, который обеспечивает надёжную и достаточно быструю передачу данных между диском и другими компонентами системы.

### **Внешние устройства**

**Внешние устройства (ВУ)** играют важную роль во взаимодействии пользователя с операционной системой. Операционная система должна поддерживать работу с основными устройствами ввода и вывода, обеспечивая эффективное взаимодействие с пользователем.

Внешние устройства, используемые для взаимодействия с операционной системой, включают: клавиатуру, мышь, монитор (видеотерминал), флеш-диск, принтер

Каждое из этих устройств выполняет важную роль в обеспечении удобного и эффективного взаимодействия пользователя с системой:

* **клавиатура**: это основное устройство ввода, предназначенное для ввода команд и текстовых данных. Операционная система должна поддерживать работу с клавиатурой, обеспечивая точную обработку клавишных событий и поддержку различных типов клавишных комбинаций;
* **мышь**: устройство для указания положения курсора и выполнения действий с элементами интерфейса, такими как окна, кнопки и другие объекты. Мышь позволяет пользователю легко и быстро управлять системой. Операционная система должна обеспечить точное отслеживание движений и кликов мыши;
* **монитор (видеотерминал)**: это устройство вывода, которое отображает информацию, такую как текст, графика и другие визуальные данные. Операционная система должна поддерживать разнообразные разрешения экрана и обеспечивать правильный вывод информации с учётом характеристик монитора;
* **флеш-диск**: внешний носитель данных, который используется для хранения и переноса файлов между системами. Операционная система должна поддерживать работу с флеш-дисками, обеспечивая автоматическое распознавание подключённых устройств и возможность чтения/записи данных;
* **принтер**: устройство вывода, предназначенное для печати документов на бумаге. Операционная система должна обеспечивать взаимодействие с принтерами, предоставляя пользователю возможность настроить параметры печати и отправить документы на печать.

## **Режим работы компьютера**

### **Однопрограммный**

**Однопрограммный режим** — это режим работы компьютера, при котором в оперативной памяти и на процессоре одновременно выполняется только одна программа. Он может иметь модификации: однопрограммный режим непосредственного доступа и однопрограммный режим косвенного доступа.

**Режим непосредственного доступа.**

Пользователь получает электронную вычислительную машину (ЭВМ) в полное распоряжение: он сам подготавливает ЭВМ к работе, загружает задания, инициирует их, наблюдает за ходом решения и выводом результатов. По завершении работы одного пользователя все ресурсы ЭВМ передаются в распоряжение следующего, что показано на Рисунок 2. Этот тип режима характеризуется низкой полезной загрузкой технических средств. Снижение производительности ЭВМ происходит из-за простоев процессора, вызванных затратами времени на подготовку ЭВМ к работе (включение, проверка, загрузка ОС, ввод заданий и т.д.) и длительным временем реакции пользователя. По этим причинам данный режим практически не используется в универсальных ЭВМ. В то же время, в ПЭВМ этот режим является основным, так как в этих типах ЭВМ главным критерием эффективной работы считается обеспечение максимальных удобств пользователю [4].

Изображение выглядит как линия, диаграмма, текст, Шрифт

Автоматически созданное описание

Рисунок 2 – Режим непосредственного доступа

**Режим косвенного доступа**

Пользователь не имеет прямого контакта с ЭВМ. Этот режим был предшественником многопрограммных режимов в ЭВМ высокой и средней производительности и имел целью обеспечить более полную загрузку процессора за счет сокращения непроизводительных простоев. В настоящее время он практически не используется, так как время работы процессоров в современных ЭВМ не является главным ресурсом системы. Однако принципы построения этого режима помогают лучше понять суть многопрограммной обработки.

Суть режима состоит в следующем: из подготовленных заданий пользователей формируется пакет заданий. Процессор обслуживает программы пользователей строго в порядке их следования в пакете. Процесс выполнения очередной программы не прерывается до полного ее завершения. Только после этого процессор передается в монопольное владение следующей программе. Доступ пользователя к ресурсам ЭВМ осуществляется косвенно через средства ОС, которые организуют автоматический переход от обслуживания одного задания к другому. Благодаря этому режим часто называют последовательной пакетной обработкой. При этом обеспечивается параллельная работа устройств ввода-вывода и процессора (Рисунок 3), что позволяет значительно повысить производительность ЭВМ за счет сокращения простоев [5].

Изображение выглядит как текст, линия, Шрифт, диаграмма

Автоматически созданное описание

Рисунок 3 – Режим косвенного доступа

## **Архитектура операционной системы**

Архитектура операционной системы представляет собой основу, которая определяет способ взаимодействия её компонентов, а также распределение обязанностей между ядром и пользовательским пространством. От выбора архитектуры зависит производительность системы, её масштабируемость, безопасность и возможность адаптации к различным аппаратным платформам.

### **Экзоядро**

В качестве прототипа для проектируемой гипотетической операционной системы выбрана архитектура экзоядра, основанная на современных операционных системах семейства *Linux*. **Экзоядро (*Exokernel*)** — это минималистическая архитектура операционной системы, при которой ядро выполняет только базовые функции по управлению аппаратными ресурсами, предоставляя приложениям прямой доступ к этим ресурсам. Такая структура отличается высокой производительностью за счёт минимизации накладных расходов на обработку системных вызовов и гибкостью в настройке. Это позволяет:

* **увеличить стабильность и безопасность**. Поскольку меньше кода выполняется в режиме ядра, вероятность возникновения ошибок и уязвимостей снижается. В случае сбоя пользовательского процесса ядро остается нетронутым, что повышает общую устойчивость системы;
* **упрощение разработки и тестирования.** Модульная структура экзоядра позволяет разработчикам тестировать и обновлять отдельные компоненты без необходимости перезагрузки всего ядра. Это упрощает процесс отладки и внедрения новых функций;
* **гибкость в управлении ресурсами**. Экзоядро позволяет динамически загружать и выгружать модули, что дает возможность адаптировать систему под конкретные задачи и требования пользователя.

Операционная система *Linux*, особенно в своих современных дистрибутивах, демонстрирует многие принципы экзоядра. Она использует модульную архитектуру, позволяя загружать драйверы и другие компоненты по мере необходимости. Это обеспечивает:

* **поддержку широкого спектра аппаратного обеспечения.** Пользователи могут добавлять поддержку новых устройств без необходимости обновления всего ядра;
* **разделение функций.** Системные службы, такие как управление процессами, файловыми системами и сетевыми соединениями, могут быть реализованы как отдельные модули, что упрощает их обновление и поддержку;
* **сообщество и открытость.** Открытый исходный код Linux позволяет разработчикам со всего мира вносить свой вклад в улучшение системы, что способствует быстрому развитию и внедрению новых технологий.

## **Подсистема управления процессами**

Важнейшей частью операционной системы, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами. **Процесс** (или по-другому, задача) — абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

### **Алгоритмы планирования процессов**

Алгоритм планирования процессов — это метод, который используется операционной системой для решения задачи, какой процесс должен быть выполнен в данный момент. В данном проекте выбран вытесняющий алгоритм циклического типа, реализованный с использованием принципа *Last In, First Out (LIFO).*

#### **Вытесняющий – циклический *LIFO (Last In, First Out)***

Алгоритм ***LIFO*** **(*Last In, First Out*)** предполагает, что процесс, который был помещён в очередь последним, будет первым, кто получит доступ к процессору. Это характерно для вытесняющего планирования, где текущий процесс может быть приостановлен и вытеснен новым процессом, если последний был помещён в очередь позже. Принцип *LIFO* подразумевает, что в случае многозадачности, более новые процессы получают приоритет, и, если система запускает процесс, а затем появляется новый с более высоким приоритетом, то старый процесс будет остановлен и вернётся в очередь после завершения нового. Выталкивание старого процесса и немедленное выполнение более нового в большинстве случаев ведет к улучшению скорости реакции системы и увеличению её производительности в условиях ограниченных ресурсов.

**Пример работы алгоритма** ***LIFO***:

Предположим, в системе есть четыре процесса с идентификаторами *P*1, *P*2, *P*3 и *P*4. Пусть процессы добавляются в очередь в следующем порядке:

1. Процесс *P*1 запускается и начинает выполнение.
2. Процесс *P*2 появляется и добавляется в очередь.
3. Процесс *P*3 появляется и добавляется в очередь.
4. Процесс *P*4 появляется и добавляется в очередь.

Теперь, очередь выглядит так: *P*1, *P*2, *P*3, *P*4. В системе используется алгоритм *LIFO*, то есть последний добавленный процесс будет первым выполнен.

1. Поскольку процесс *P*4 был добавлен последним, он первым будет запущен на процессоре.
2. После завершения *P*4, следующим будет процесс *P*3, так как он был добавлен перед *P*4.
3. Затем очередь продолжит с *P*2 и, наконец, с *P*1.

Таким образом несмотря на то, что процесс *P*1 был первым в очереди, он будет выполнен последним. Схему работы алгоритма можно увидеть Рисунок 4

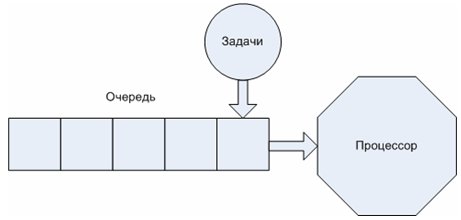


Рисунок 4 – Схема работы алгоритма *LIFO*

Схема иллюстрируют статическое планирование исполнения процессов в однопроцессорной системе под управлением операционной системы с монопольным доступом к процессору [6].

**Преимущества и недостатки**

**Преимущества алгоритма *LIFO***:

* простота реализации;
* быстрое выполнение более новых процессов;
* подходит для систем с короткими задачами, где более новые процессы имеют больший приоритет.

**Недостатки алгоритма *LIFO***:

* + неэффективен при равных приоритетах процессов;
  + может привести к задержкам в выполнении старых процессов (эффект "starvation");
  + не гарантирует справедливое распределение ресурсов между процессами.

### **Алгоритмы организации взаимодействия процессов**

Взаимодействие между процессами (*IPC, Inter-Process Communication*) является важным элементом многозадачной операционной системы. Оно позволяет процессам обмениваться данными и синхронизировать свою работу, что важно для обеспечения координации и эффективного использования ресурсов.

#### **Алгоритм пекарни Лэмпорта**

Алгоритм пекарни Лэмпорта является одним из наиболее известных алгоритмов для синхронизации процессов в многозадачных системах. Он был предложен Робертом Лэмпорта в 1974 году и решает задачу координации процессов, которые обращаются к общим ресурсам в условиях параллельной работы. Алгоритм получил своё название благодаря аналогии с очередью в пекарне, где люди берут номер и поочередно обслуживаются в порядке очереди.

**Принцип работы алгоритма**

Каждый процесс перед тем, как получить доступ к общему ресурсу, должен взять «номер». Процесс выбирает себе минимальный номер, который ещё не был занят другим процессом, и ожидает своей очереди для выполнения. Алгоритм пекарни Лэмпорта гарантирует, что два процесса не будут одновременно выполнять критическую секцию, если они хотят сделать это одновременно. Кроме того, он обеспечивает справедливость: процессы всегда будут обслуживаться в порядке их «номера».

Алгоритм работает следующим образом:

1. каждый процесс имеет свой номер, который он выбирает перед доступом к общему ресурсу;

2. каждый процесс ставит метку, которая указывает, что он хочет войти в критическую секцию;

3. процесс с наименьшим номером в очереди получает доступ к ресурсу первым, после чего его номер становится «недоступным» для других;

4. после завершения работы в критической секции процесс снимает свою метку и освобождает ресурсы для других;

Этот алгоритм является решением проблемы взаимного исключения и используется в многозадачных системах для предотвращения гонок и обеспечения корректного доступа к разделяемым данным. Алгоритм пекарни Лэмпорта имеет важное преимущество в том, что его работа не зависит от приоритетов процессов, обеспечивая справедливое распределение времени процессора.

### **Задача синхронизации процессов**

**Синхронизация процессов** — это механизм, который обеспечивает согласованное выполнение параллельных или мультизадачных процессов, предотвращая ситуации, которые могут привести к ошибкам или конфликтам при совместном использовании ресурсов. В однопрограммном режиме операционная система работает с одним процессом за раз, но синхронизация все равно может быть важной, если система будет расширена для обработки нескольких процессов или виртуализации.

Задача синхронизации процессов необходима в системах с несколькими параллельно работающими процессами, особенно когда процессы взаимодействуют через общие ресурсы (например, память, порты ввода-вывода или файлы). Синхронизация регулирует, когда и как процессы могут взаимодействовать или совместно использовать ресурсы без возникновения конфликтов, таких как гонки или взаимные блокировки.

**Основные задачи синхронизации процессов включают**:

* обмен данных между процессами (например, один процесс записывает данные, а другой их считывает);
* управление доступом к разделяемым ресурсам, чтобы избежать конфликтов (например, при попытке доступа к файлам или портам ввода-вывода одновременно несколькими процессами);
* синхронизация действий с внешними событиями, например, реакции на пользовательские вводы или системные сигналы.

Если задачи синхронизации процессов не решены должным образом, могут возникнуть проблемы, такие как:

* **гонки** — ситуация, когда два или более процесса изменяют или используют общие данные, и результат зависит от порядка их выполнения;
* **тупики** — когда процессы блокируют друг друга, удерживая ресурсы, которые необходимы для продолжения работы других процессов.

#### **Задача обедающие философы**

Задача "Обедающие философы" была предложена Эдсгером Дейкстрой в 1965 году и представляет собой классическую задачу синхронизации, которая используется для моделирования ситуации, когда несколько процессов (или, в этом случае, философов) пытаются получить доступ к общим ресурсам (вилки) для выполнения своих задач.

**Условия задачи:**

Представим, что пять философов сидят за круглым столом. Каждый философ размышляет и иногда берёт две вилки, расположенные по бокам его тарелки, чтобы поесть. Вилки лежат между философами, и каждый философ может взять только одну вилку с обеих сторон. После того как философ поел, он кладёт вилки обратно на стол и снова начинает размышлять.

Проблема возникает, когда несколько философов одновременно пытаются взять вилки. Если они не синхронизированы должным образом, может возникнуть ситуация, в которой каждый философ держит одну вилку, но не может взять вторую, что приведет к взаимной блокировке (*deadlock*) [7].

**Основные требования:**

1. мутуальное исключение *(Mutual Exclusion*): вилки — это общие ресурсы, и каждый философ должен использовать две вилки одновременно. При этом два философа не могут использовать одну вилку одновременно;

2. отсутствие взаимной блокировки (*Deadlock-freedom*): все философы должны иметь возможность завершить свои действия (поесть). Важно избежать ситуации, когда философы никогда не смогут поесть, потому что все они застряли, пытаясь захватить вилки;

3. отсутствие голодания (*Starvation-freedom*): не должно быть ситуации, когда какой-то философ никогда не может поесть, так как всегда есть философы, которые забирают вилки раньше него.

**Решение задачи:**

Для решения этой задачи используется подход синхронизации с использованием мьютексов (или семафоров). Каждый философ имеет доступ к двум вилкам, которые могут быть захвачены с помощью мьютексов. Алгоритм может выглядеть следующим образом:

1. каждый философ поочередно думает;

2. когда философ решает поесть, он пытается захватить вилку слева и вилку справа;

3. если обе вилки свободны, философ поедает, затем кладёт вилки обратно;

4. если хотя бы одна вилка занята, философ ждёт, пока обе вилки не будут свободны;

5. чтобы избежать взаимной блокировки, философы могут договориться о том, в каком порядке захватывать вилки (например, всегда захватывать вилку с меньшим номером первым).

### **Способ борьбы с тупиками**

**Тупик (*deadlock*)** — это ситуация в операционной системе, когда несколько процессов одновременно ожидают освобождения ресурсов, удерживаемых друг другом, что приводит к взаимной блокировке и невозможности их дальнейшего выполнения. В операционных системах, где одновременно выполняются множество процессов, тупики могут привести к значительным потерям в производительности.

На Рисунок 5 показан пример тупиковой ситуации с двумя процессами (A и B) и двумя ресурсами (*R*1 и *R*2). Процессы представлены прямоугольниками, а ресурсы — окружностями. Стрелки от ресурса к процессу показывают, что процесс владеет ресурсом, а стрелки от процесса к ресурсу — что процесс ожидает его.

Процесс A удерживает ресурс *R*1, но ему необходим *R*2, а процесс B удерживает *R*2, но ему нужен *R*1. Оба процесса находятся в состоянии ожидания: каждый ждет, чтобы другой освободил нужный ему ресурс, что приводит к тупику, в котором оба процесса не могут продолжить выполнение [8].

Изображение выглядит как текст, диаграмма, снимок экрана, круг

Автоматически созданное описание

Рисунок 5 – Пример тупиковой ситуации

#### **Предотвращение тупиков – алгоритм банкира**

Алгоритм банкира был предложен Эдсгером Дейкстрой и используется для предотвращения тупиков в ОС, где ресурсы (например, процессорное время, память или устройства ввода/вывода) ограничены и могут быть разделяемыми между несколькими процессами. Основная идея алгоритма заключается в проверке безопасности системы, моделируя поведение банка, который выдает кредиты клиентам. Перед предоставлением доступа к ресурсу проверяется, что система останется в безопасном состоянии при любом допустимом распределении ресурсов.

**Безопасное состояние** – состояние системы, при котором существует последовательность выполнения процессов, позволяющая каждому из них завершиться без возникновения тупика.

**Принцип работы алгоритма:**

* предварительное объявление потребностей: каждый процесс сообщает максимальное количество ресурсов, которое ему может понадобиться для завершения работы;
* проверка доступности ресурсов: при поступлении запроса на ресурсы система проверяет, достаточно ли доступных ресурсов для удовлетворения запроса;
* оценка безопасности состояния: если ресурсы доступны, система временно предполагает их выделение и проверяет, останется ли система в безопасном состоянии после этого;
* выделение ресурсов: если после предполагаемого выделения система остается в безопасном состоянии, ресурсы действительно выделяются процессу. В противном случае процесс ожидает, пока ресурсы не станут доступными без нарушения безопасности системы.

**Пример алгоритма Банкира:**

предположим, что у нас есть следующие ресурсы:

* 5 флэш-накопителей;
* 2 принтера;
* 5 сканеров;
* 3 жестких диска.

Здесь мы создали вектор, представляющий общее количество ресурсов: доступно = (5, 2, 4, 3).

Предположим, что имеется четыре процесса. В Таблица 1 представлены распределенные ресурсы.

Таблица 1 – Доступные ресурсы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя процесса | Пен Диски | Принтер | Сканер | Жесткий диск |
| *P* | 2 | 0 | 1 | 1 |
| *Q* | 0 | 1 | 0 | 0 |
| *R* | 1 | 0 | 1 | 1 |
| *S* | 1 | 1 | 0 | 1 |
| Всего | 4 | 2 | 2 | 3 |

Здесь выделенные ресурсы — это сумма этих столбцов:

Выделено = (4, 2, 2, 3).

Мы также создаем матрицу для отображения количества каждого ресурса, необходимого для всех процессов (Таблица 2). Эта матрица называется необходимость= (3,0,2,2)

Таблица 2 – Необходимость ресурсов для процессов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя процесса | Пен Диски | Принтер | Сканер | Жесткий диск |
| *P* | 1 | 1 | 0 | 0 |
| *Q* | 0 | 1 | 1 | 2 |
| *R* | 2 | 1 | 0 | 0 |
| *S* | 0 | 0 | 1 | 0 |

Доступный вектор будет:

Доступно = Доступно - Выделено = (5, 2, 4, 3) - (4, 2, 2, 3) = (1, 0, 2, 0)

**Алгоритм запроса ресурсов**

Алгоритм запроса ресурсов в контексте алгоритма Банкира представляет собой процесс, который моделирует поведение системы, когда определенный процесс выполняет запрос на ресурс. Этот процесс используется для предотвращения тупиков, гарантируя, что ресурсы распределяются только в случае, если это не приведет к блокировке системы [9].

**Шаги алгоритма запроса ресурсов**:

**Шаг 1: Проверка допустимости запроса**

Процесс делает запрос на определенные ресурсы. В этом шаге проверяется, не запрашивает ли процесс больше ресурсов, чем указано в его максимальной потребности. Алгоритм сравнивает общее количество запрашиваемых ресурсов с максимальной потребностью процесса и переходит ко второму шагу, если запрос не превышает максимальную потребность.

**Шаг 2: Проверка доступности ресурсов**

Далее алгоритм проверяет, доступны ли все запрашиваемые ресурсы в системе. Если количество доступных ресурсов больше или равно запрашиваемому количеству, система продолжает обработку запроса. В противном случае процесс будет вынужден ожидать, пока ресурсы станут доступны.

**Шаг 3: Выделение ресурсов**

Если запрос прошел все проверки, ресурсы выделяются процессу. Это происходит следующим образом:

* обновляется вектор доступных ресурсов: *Available* = *Available* – *Request;*
* увеличивается количество ресурсов, выделенных процессу: *Allocation(x) = Allocation(x) + Request(x);*
* обновляется необходимость ресурса для процесса: *Need(x) = Need(x) - Request(x).*

Этот шаг подразумевает, что система "предполагает", что ресурсы были выделены, и обновляет состояние системы соответственно. После этого количество доступных ресурсов уменьшается.

**Характеристики алгоритма Банкира:**

* **множество ресурсов:** система должна иметь достаточно ресурсов для удовлетворения хотя бы одного процесса;
* **ограничение времени:** после получения всех необходимых ресурсов процесс должен вернуть их в течение ограниченного времени;
* **ожидание:** если процесс не может сразу получить нужный ресурс, он должен подождать;
* **ограниченность ресурсов:** в системе имеется фиксированное количество ресурсов;
* **максимальное распределение:** Каждый процесс заранее указывает максимальное количество ресурсов, которое ему может понадобиться.

**Недостатки алгоритма Банкира:**

* невозможность изменения максимальной потребности процесса;
* требует от всех процессов заранее указать свои максимальные потребности, что ограничивает гибкость системы.
  1. **Подсистема управления памятью**

Память является важнейшим ресурсом, требующим тщательного управления со стороны мультипрограммной операционной системы. Распределению подлежит вся оперативная память, не занятая операционной системой. Обычно ОС располагается в самых младших адресах, однако может занимать и самые старшие адреса. Функциями ОС по управлению памятью являются:

* **отслеживание свободной и занятой памяти**. Система должна поддерживать информацию о состоянии каждого сегмента памяти, чтобы понимать, какие участки свободны, а какие заняты;
* **выделение памяти процессам**. При запуске процесса ОС выделяет ему определённый объём памяти, основываясь на требованиях и доступных ресурсах;
* **освобождение памяти при завершении процессов.** Когда процесс завершает свою работу, операционная система освобождает занимаемую им память, чтобы она могла быть использована другими процессами;
* **вытеснение процессов из оперативной памяти на диск**. Если оперативной памяти недостаточно для размещения всех процессов, система может выгрузить часть процессов на диск, освобождая место для других;
* **возвращение процессов в оперативную память**. Когда в памяти освобождается место, система может вернуть вытесненные процессы в оперативную память;
* **настройка адресов программы на конкретную область физической памяти.** Каждый процесс имеет свой логический адрес, который преобразуется в физический адрес с помощью механизмов, таких как страничная или сегментная адресация.

### **Сегментное распределение памяти**

**Сегментное распределение памяти** — это метод организации виртуального адресного пространства процесса, при котором оно делится на логические блоки, называемые сегментами.

Сегментное распределение памяти, представленное на Рисунок 6, предполагает деление виртуального адресного пространства процесса на сегменты. Размеры сегментов задаются программистом с учетом смыслового значения данных, которые они содержат. Сегмент может представлять подпрограмму, массив данных и т. п., иногда сегментацию выполняет компилятор автоматически. При загрузке процесса часть сегментов размещается в оперативной памяти (операционная система выбирает подходящие свободные участки), а часть — на диске. Сегменты программы могут занимать несмежные участки памяти. Система создает таблицу сегментов процесса, где указываются начальный физический адрес, размер, правила доступа, признаки модификации и обращения, а также другая информация. Если сегмент используется несколькими процессами, он загружается в память в единственном экземпляре, а таблицы сегментов содержат ссылки на него.

Изображение выглядит как текст, диаграмма, линия, Параллельный

Автоматически созданное описание

Рисунок 6 – Сегментное распределение памяти

Система с сегментной организацией памяти функционирует следующим образом: периодически возникают прерывания, связанные с отсутствием необходимых сегментов в оперативной памяти. При необходимости освобождения памяти некоторые сегменты выгружаются на диск. При каждом обращении к памяти выполняется преобразование виртуального адреса в физический. Также проверяется, разрешен ли доступ требуемого типа к указанному сегменту [10].

Виртуальный адрес в такой системе представляется парой (*g, s*), где *g* — номер сегмента, а s — смещение внутри сегмента. Физический адрес вычисляется путем сложения начального физического адреса сегмента, определенного по номеру g из таблицы сегментов, и смещения s.

Основным недостатком сегментной организации памяти является фрагментация на уровне сегментов, а также более низкая скорость преобразования адресов по сравнению с другими методами.

## **Подсистема управления файлами**

**Подсистема управления файлами** в операционной системе отвечает за организацию, хранение, доступ и управление файлами на носителе данных. Она включает в себя различные компоненты, такие как каталоги, файлы, их атрибуты и права доступа. Основные функции подсистемы управления файлами включают:

* организацию файлов на диске и определение структуры хранения данных;
* управление доступом к файлам и каталогам, включая права на чтение, запись и выполнение;
* обеспечение совместного доступа к файлам несколькими процессами;
* обработку запросов на ввод/вывод данных и кэширование данных для повышения производительности;
* обеспечение целостности данных, контроль их повреждений и восстановление.

Подсистема управления файлами также должна обеспечивать интерфейс для работы с файлами через команды операционной системы, что позволяет пользователю и программам эффективно работать с данными.

### **Иерархия каталогов**

**Иерархия каталогов** — это структура, в которой файлы и папки организованы в виде дерева. Каталог, также называемый директорией, представляет собой контейнер для хранения файлов и других каталогов. Иерархическая структура каталогов позволяет пользователю и операционной системе легко управлять файлами, обеспечивая удобный доступ и поиск. Каждый каталог может содержать файлы и подкаталоги, создавая многослойную структуру.

Иерархия каталогов имеет следующие ключевые элементы:

**корневой каталог:** это начальный уровень иерархии, который является основой для организации всех других каталогов;

**подкаталоги:** это каталоги, которые находятся внутри других каталогов, образуя вложенные уровни;

**файлы:** хранятся внутри каталогов, могут быть любыми данными, такими как текстовые документы, изображения и программы.

Основным преимуществом иерархической структуры является простота навигации по файлам и каталогам, а также возможность эффективно организовывать и разделять различные типы данных.

#### **Дерево**

**Дерево каталогов** представляет собой графическую или логическую модель иерархии, в которой файлы и каталоги структурированы как узлы, соединенные ребрами. В этой модели каждый каталог — это узел, и каталоги, расположенные внутри другого каталога, становятся дочерними узлами. Такая организация данных даёт возможность эффективно управлять файлами, особенно в системах с большим количеством информации. Каждый файл или каталог может быть однозначно идентифицирован с помощью пути, который может быть либо абсолютным, либо относительным. Чтобы продемонстрировать, как работает подобная запись, рассмотрим дерево каталогов, изображенный на Рисунок 7.

Изображение выглядит как диаграмма, Технический чертеж, План, линия

Автоматически созданное описание

Рисунок 7 - Дерево каталогов

**Абсолютный путь** начинается с корня / и указывает полное местоположение файла или каталога. Например, путь */usr/ast/mailbox* указывает на файл *mailbox* в каталоге *ast*, который в свою очередь находится в каталоге *usr*. Абсолютный путь используется для точного указания местоположения файла.

**Относительный путь** указывается относительно текущего рабочего каталога. Например, если рабочий каталог — */usr/ast*, файл *mailbox* можно указать как *mailbox*, без необходимости повторять полный путь. Для навигации используются специальные записи: . (одна точка) — для текущего каталога и .. (две точки) — для родительского.

Каждый процесс имеет свой рабочий каталог, что позволяет гибко изменять местоположение файлов без влияния на другие процессы. Абсолютные пути полезны для точного указания местоположения, а относительные — для сокращения записи и удобства работы.

Дерево каталогов упрощает навигацию и управление файлами, позволяя эффективно организовывать данные и выполнять операции как пользователям, так и программам.

### **Логическая организация файловой системы**

**Логическая организация файловой системы** — это способ организации и управления данными на уровне операционной системы, который обеспечивает удобный и эффективный доступ к файлам. Важной задачей логической организации является управление метаданными, такими как структура каталогов, индексы и связи между файлами. Она определяет, как файлы и каталоги представлены в системе и как осуществляется доступ к данным, что позволяет операционной системе эффективно управлять ресурсами хранения. Основные элементы логической организации файловой системы включают:

* **каталоги**: они группируют файлы в иерархическую структуру, которая упрощает навигацию и управление;
* **индексы**: используются для ускорения поиска файлов и других данных в системе;
* **метаданные**: содержат информацию о файлах, такую как размер, дата создания, права доступа и другие характеристики.

#### **Индексная логическая организация**

**Индексная логическая организация файловой системы** — метод управления данными, при котором для быстрого доступа к данным используется специальная индексная таблица. В этой таблице значения ключевых полей (индексов) соответствуют адресам местоположения данных на диске.

**Преимущества индексной логической организации:**

* **быстрый доступ к данным** — благодаря прямому доступу к блокам данных через указатели;
* **универсальность** — подходит для файлов разных размеров;
* **гибкость управления файлами** — легко изменять структуру и размер файла за счёт добавления новых указателей;
* **минимизация фрагментации** — блоки данных могут располагаться не подряд, что позволяет оптимально использовать свободное место.

**Недостатки индексной логической организации**

* **дополнительные затраты на хранение метаданных** — требуется значительное место для индексных таблиц;
* **ограничение на размер файла** — если таблица не поддерживает многоуровневую структуру, максимальный размер файла ограничен числом записей в таблице;
* **сложность реализации** — требует сложных алгоритмов управления.

**Схема индексной логической организации**

На Рисунок 8 показана схема индексной логической организации файловой системы, которая наглядно объясняет, как она работает, подчеркивая связь между значениями индексов и адресами данных, что имеет решающее значение для эффективного извлечения данных [11].



Рисунок 8 – Схема индексной логической организации файловой системы

### **Физическая организация файловой системы**

**Физическая организация файловой системы** — способ, с помощью которого данные о файлах и их содержимом хранятся на физическом носителе — жестком диске, твердотельном накопителе или другом устройстве хранения. Это важная часть файловой системы, определяющая, как именно данные распределяются по блокам на устройстве и как осуществляется доступ к этим блокам на физическом уровне.

#### **Связанный список индексов**

**Связанный список индексов** — это метод организации данных, в котором каждый блок информации (или индексный блок) связан с последующим блоком с помощью указателя. Эта структура используется для представления данных файлов в файловой системе, где каждый блок файла ссылается на следующий через указатель.

Файлу также выделяется память в виде связанного списка кластеров. Номер первого кластера запоминается в записи каталога, где хранятся характеристики этого файла. Остальная адресная информация отделена от кластеров файла. С каждым кластером диска связан индекс. Индексы располагаются в отдельной области диска — в файловых системах *FAT* это таблица *(File Allocation Table*): когда память свободна, все индексы имеют нулевое значение. Если некоторый кластер *N* назначен некоторому файлу, то индекс этого кластера становится равным либо номеру *M* следующего кластера данного файла, либо принимает специальное значение – признак того, что этот кластер является для файла последним. Индекс же предыдущего кластера файла принимает значение *N*, указывая на вновь назначенный кластер [12, 13]. Пример такой структуры связанного списка индексов показан на Рисунок 9

Изображение выглядит как текст, снимок экрана, диаграмма, Параллельный

Автоматически созданное описание

Рисунок 9 – Пример структуры связанного списка индексов

**Достоинства**:

* **минимальность адресной информации** — для каждого кластера требуется минимальное количество данных для его адресации.
* **отсутствие фрагментации на уровне кластеров** — данные файла распределяются по кластеру целиком, что исключает проблемы фрагментации на уровне кластеров;
* **отсутствие проблем при изменении размера файла** — изменение размера файла не вызывает трудностей, так как для его расширения достаточно добавить новые кластеры;
* **быстрый доступ к произвольному кластеру** — для доступа к любому кластеру файла не нужно последовательно считывать все предыдущие кластеры; достаточно прочитать таблицу индексов, отсчитать нужное количество кластеров и определить номер целевого кластера;
* **оптимизация использования памяти** — данные файла заполняют кластер целиком, что позволяет использовать кластеры, объем которых равен степени двойки, и эффективно использовать память.

**Недостатки:**

* фрагментация на уровне файлов (файл может разбиваться на несмежные фрагменты).

# **ПРОЕКТНАЯ ЧАСТЬ**

## **Проектирование графического интерфейса**

Проектирование графического интерфейса направлено на создание удобного и простого способа взаимодействия пользователя с гипотетической операционной системой. Интерфейс базируется на текстовом взаимодействии в рамках монохромного отображения, где поддерживаются только алфавитно-цифровые символы, а работа организована в построчном режиме (следует из примечания 7). Данное решение исключает использование графических элементов и обеспечивает минималистичную структуру интерфейса.

**Структура интерфейса:**

1. **приветствие и инструкции:** при старте системы пользователю выводится сообщение с приветствием и предложением ввести команду *help* для получения списка доступных команд;

**Аутентификация пользователя:** для входа в систему требуется ввести имя пользователя и пароль. Пароль отображается в виде звездочек (для безопасности).

2. **основной экран терминала:** после успешного входа, пользователь видит приглашение для ввода команд в командной строке. Информация о текущем пользователе и каталоге отображается в следующем формате: *user@LAPTOP-V6CLNC7V:~$;*

3. **основной функционал команд:** В терминале доступен набор базовых команд, которые могут быть использованы для взаимодействия с файловой системой и процессами:

***help*:** выводит список доступных команд;

***ls*:** показывает содержимое текущей директории;

***cd*:** позволяет сменить текущую директорию;

***mkdir*:** создаёт новую директорию;

***rm*:** удаляет файл или директорию;

***touch*:** создаёт новый пустой файл;

***cp*:** копирует файлы или директории;

***mv*:** перемещает или переименовывает файлы;

***echo*:** выводит текст на экран или в файл;

***ps*:** показывает текущие процессы;

***kill*:** завершает процесс;

***clear*:** очищает экран терминала;

***exit*:** завершает сеанс и выходит из системы.

1. **Пример работы с командным интерфейсом:** Пользователь может выполнять ряд операций, таких как создание и удаление файлов, копирование данных, управление процессами и завершение сессии.

Графический интерфейс операционной системы можно увидеть на Рисунок 10

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 10 – Графический интерфейс ОС

## **Проектирование командного языка**

Согласно заданию необходимо спроектировать командный язык, который будет включать в себя только основные команды (работа с файловой системой, вход и выход из системы). В качестве прототипов команд взяты базовые утилиты командной оболочки Linux.

**Основные команды**

Проектирование командного языка гипотетической операционной системы заключается в разработке минимального набора команд, которые позволят выполнять основные операции с файловой системой и управлять сессией пользователя. Команды будут заимствованы из базовых утилит командной оболочки Linux и адаптированы под нужды гипотетической ОС. Основной акцент сделан на команды для работы с файлами и каталогами, а также для входа и выхода из системы. Этот набор команд обеспечит базовую функциональность операционной системы, необходимую для её работы в рамках заданных требований.

В Таблица 3 перечислены основные команды командного языка гипотетической операционной системы, их описание и примеры использования.

Таблица 3 – Основные команды системы

|  |  |  |
| --- | --- | --- |
| Команда | Описание | Пример использования |
| Команды управления сеансом | | |
| *login* | Авторизация в системе, вход пользователя. | *$ login*  Имя пользователя: *user* Пароль: \*\*\*\*\*\*\*\* |
| *logout* | Завершение сеанса и выход из системы. | *$ logout*  Выход… |
| Команды для работы с файлами | | |
| *touch* | Создает новый файл. | *$ touch file.txt* |
| *cat* | Отображает содержимое указанного файла. | *$ cat file.txt* |
| *echo* | Добавляет текст в файл или выводит его на экран. | *$ echo "*Добро пожаловать в *TaeOS!" > NewFile.txt* |
| *cp* | Копирует содержимое одного файла в другой. | *$ cp file.txt fileCopy.txt* |
| *mv* | Переименовывает файл или перемещает его в другой каталог. | *$ mv file.txt NEW\_file.txt* |
| *rm* | Удаляет файл. | *$ rm file.txt* |

Продолжение таблицы Таблица 3

|  |  |  |
| --- | --- | --- |
| Команда | Описание | Пример использования |
| Команды для работы с каталогами | | |
| *ls* | Выводит список файлов и папок в текущем каталоге. | *$ ls*  *file.txt fileCopy.txt* |
| *mkdir* | Создает новый каталог. | *$ mkdir GUZdir* |
| *rmdir* | Удаляет пустой каталог. | *$ rmdir empty\_dir* |
| *cd* | Переключает текущий рабочий каталог. | *$ cd Documents* |
| *ps* | Показывает список запущенных процессов. | *$ ps*  *PID TTY TIME CMD*  *a2b3 pts/0 00:00:02 bash*  *b9c4 pts/0 00:00:06 editor* |
| *kill* | Завершается выполнение указанного процесса. | *$ kill b9c4* |
| *clear* | Очищает экран терминала. | *$ clear* |

**Дополнительные команды**

В Таблица 4 перечислены команды, обеспечивающие средства защиты на уровне файлов и каталогов между пользователями гипотетической операционной системы.

Таблица 4 – Команды системы защиты

|  |  |  |
| --- | --- | --- |
| Команда | Описание | Пример использования |
| *chmod* | Управляет правами доступа к файлам или каталогам (чтение, запись, выполнение). | *$ chmod 644 file.txt* |
| *chown* | Изменяет владельца или группу файла или каталога. | *$ chown user1 file.txt* |

Продолжение Таблица 4

|  |  |  |
| --- | --- | --- |
| Команда | Описание | Пример использования |
| *umask* | Устанавливает права доступа по умолчанию для новых файлов и каталогов. | *$ umask 022* |

* 1. **Загрузка операционной системы**

**Загрузка операционной системы** — это процесс инициализации и активации операционной системы на компьютере, который начинается при включении устройства и заканчивается предоставлением пользователю рабочего интерфейса. В ходе загрузки операционная система загружает необходимые системные компоненты, инициализирует аппаратные средства, а также подготавливает среду для работы пользовательских процессов [14].

### **Загрузка с жесткого диска компьютера**

**Загрузка операционной системы с жёсткого диска** — это процесс, при котором компьютер загружается для обычной работы, используя загрузочный образ, находящийся на локальном диске и созданный при установке операционной системы.

**Алгоритм загрузки с жесткого диска:**

1. **включение питания:** при включении компьютера подается питание на все компоненты системы;
2. **инициализация *BIOS/UEFI*:** базовая система ввода-вывода (*BIOS*) или ее современный аналог *UEFI* инициализирует аппаратное обеспечение, включая процессор, память и устройства ввода/вывода;
3. **поиск загрузочного устройства:** после успешного завершения *POST BIOS/UEFI* ищет загрузочные устройства в соответствии с установленным порядком загрузки. Если жесткий диск настроен как первое загрузочное устройство, *BIOS/UEFI* передает управление загрузочному сектору жесткого диска;
4. **чтение главной загрузочной записи (*MBR*):** главная загрузочная запись (*Master Boot Record, MBR*) находится в первом секторе жесткого диска. Она содержит код, необходимый для загрузки операционной системы, и таблицу разделов диска. *BIOS/UEFI* загружает этот код в оперативную память и передает ему управление;
5. **передача управления загрузчику операционной системы:** Загрузчик, указанный в *MBR*, отвечает за дальнейшую загрузку операционной системы;
6. **загрузка ядра операционной системы:** загрузчик операционной системы загружает ядро ОС в оперативную память и передает ему управление;
7. **инициализация операционной системы:** ядро ОС инициализирует все необходимые компоненты, включая драйверы устройств, и запускает системные процессы;
8. **Запуск пользовательской оболочки:** когда основные системные службы запущены, ОС начинает загрузку пользовательской оболочки (интерфейса).В этом этапе ОС становится доступной для взаимодействия с пользователем.

После загрузки всех компонентов ОС она готова к полноценному использованию. Пользователь может запускать программы, работать с файлами и выполнять другие задачи.

## **Запуск прикладных программ**

**Запуск прикладных программ** — это процесс загрузки и выполнения программного обеспечения, предназначенного для решения специфических задач пользователя, таких как обработка текста, создание графики или работа с данными. Эти программы запускаются на операционной системе и используют аппаратные ресурсы компьютера для выполнения своих функций.

### **Запуск с жесткого диска компьютера**

Запуск прикладных программ с жесткого диска начинается с момента, когда пользователь выбирает нужное приложение. Это может быть выполнено через графический интерфейс (например, кликом по иконке программы) или через командную строку.

После выбора программа находится на жестком диске, где хранятся все файлы и приложения. Операционная система ищет исполняемый файл программы в заранее определенных каталогах. Найдя файл, система проверяет, доступен ли он для выполнения, учитывая установленные права доступа. Если программа доступна для запуска, операционная система начинает процесс её загрузки.

На этом этапе исполняемый файл программы, а также необходимые вспомогательные файлы и библиотеки, которые обеспечивают корректную работу программы, считываются с жесткого диска и загружаются в оперативную память компьютера. Система создает для программы отдельный процесс, который получает необходимые ресурсы для выполнения — память, процессорное время, доступ к устройствам ввода/вывода и другие.

После завершения подготовки программа начинает выполнение. В зависимости от типа приложения это может быть графический интерфейс, окно терминала или выполнение в фоновом режиме. Пользователь взаимодействует с программой через доступный интерфейс, а система обеспечивает стабильную и безопасную работу приложения.

Эффективный запуск программ с жесткого диска требует оптимизации взаимодействия с файловой системой, правильного управления правами доступа, а также надежного механизма обработки ошибок, чтобы исключить сбои при запуске.

## **Структура управляющих блоков базы данных операционной системы**

**База данных операционной системы** — это ключевой компонент, обеспечивающий хранение и управление всей необходимой информацией для работы системы.

### **Формирование базы данных операционной системы**

База данных операционной системы состоит из нескольких управляющих блоков, каждый из которых отвечает за определенные аспекты работы системы [15]. Основные блоки включают:

* блок управления пользователями;
* блок управления процессами;
* блок управления памятью;
* блок управления файлами;
* блок управления устройствами.

**Структура управляющих блоков базы данных**

Каждый из блоков базы данных представлен таблицей, содержащей набор полей. Эти таблицы отражают структуру и организацию информации, необходимой для работы системы.

**Блок управления пользователями** обрабатывает информацию о зарегистрированных пользователях (Таблица 5). При входе пользователя в систему в таблице пользователей создаётся запись с данными об идентификаторе, группе и правах доступа. При выходе из системы соответствующая запись удаляется.

Таблица 5 – Блок управления пользователями

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| *U\_ID* | *INT* | Идентификатор пользователя |
| *U\_Name* | *VARCHAR* | Имя пользователя |
| *U\_GROUP* | *VARCHAR* | Группа пользователя |
| *U\_RIGHT* | *VARCHAR* | Права пользователя |

**Блок управления процессами** учитывает текущие процессы. Каждому процессу присваиваются идентификатор, имя, приоритет и статус. Фиксируется принадлежность процесса конкретному пользователю. Этот блок управляет состоянием процессов: исполнение, ожидание или готовность (Таблица 6).

Таблица 6 – Блок управления процессами

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| *PID* | *INT* | Идентификатор процесса |
| *NAME* | *VARCHAR* | Имя процесса |
| *QID* | *INT* | Идентификатор очереди |
| *ID* | *INT* | Идентификатор пользователя |
| *PRI* | *INT* | Приоритет процесса |

Продолжение Таблица 6

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| *ST* | *INT* | Текущее состояние процесса |

**Блок управления памятью** фиксирует и распределяет ресурсы оперативной памяти между процессами. При обращении к памяти блок проверяет возможность доступа на основании прав пользователя. По завершении работы процесс освобождает занятые ресурсы, которые становятся доступными для других. Подробная структура блока управления памятью представлена в **Таблица 7**

Таблица 7 – Блок управления памятью

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| RCB\_HANDLE | INT | Дескриптор блока |
| RCB\_SIZE | INT | Размер блока |
| RCB\_CADR | INT | Текущий адрес блока |
| RCB\_RIGHT | VARCHAR | Права доступа |

**Управляет метаданными открытых файлов.** При открытии файла создаётся запись с его параметрами (например, идентификатор, имя и размер), а при закрытии – запись удаляется. Если с файлом одновременно работают несколько процессов, указываются все активные связи. Структура блока описана в Таблица 8

Таблица 8 – Блок управления файлами

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| *FCB\_ID* | *INT* | Идентификатор файла |
| *NAME* | *VARCHAR* | Имя файла |
| *SIZE* | *INT* | Размер файла |
| *UID* | *INT* | Идентификатор владельца |
| *TIME* | *DATETIME* | Время создания |
| *BUSYBLOCKS* | *TEXT* | Перечень блоков, занимаемых файлом |

**Блок управления устройствами** учитывает подключённые устройства, их состояния и параметры. При подключении нового устройства создаётся запись, загружается соответствующий драйвер, а в случае отключения устройство удаляется из списка, и драйвер выгружается из памяти. Полная структура блока управления устройствами приведена в Таблица 9.

Таблица 9 – Блок управления устройствами

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| *DCB\_ID* | *INT* | Идентификатор устройства |
| *DCB\_NAME* | *VARCHAR* | Имя устройства |
| *DCB\_ST* | *VARCHAR* | Статус устройства |
| *DCB\_INT* | *TEXT* | Таблица точек входа |
| *DCB\_INIT* | *TEXT* | Инициализация |
| *DCB\_WRITE* | *BOOLEAN* | Флаг записи |
| *DCB\_READ* | *BOOLEAN* | Флаг чтения |
| *DCB\_PROP* | *TEXT* | Параметры устройства |
| *DCB\_DRV* | *VARCHAR* | Адрес драйвера устройства |
| *DCB\_NEXT* | *INT* | Адрес следующего *DCB* |

### **Схема взаимодействия управляющих блоков базы данных операционной системы**

Схема взаимодействия управляющих блоков базы данных операционной системы описывает, как различные блоки базы данных взаимодействуют между собой для управления ресурсами системы. Все запросы и операции происходят через ядро операционной системы, которое координирует процесс взаимодействия между блоками.

**Обработка запросов на ресурсы**

Когда процесс или пользователь запрашивает ресурс (например, память или файл), ядро операционной системы обрабатывает запрос и направляет его в соответствующий блок.

Например, запрос на выделение памяти передается в блок управления памятью, который проверяет наличие свободных блоков и выделяет необходимую память.

**Управление процессами:**

Когда создается новый процесс, ядро добавляет его информацию в блок управления процессами. Если процесс запрашивает доступ к памяти, блок управления процессами взаимодействует с блоком управления памятью для выделения необходимого ресурса.

Блок управления процессами также взаимодействует с блоком управления устройствами, если процесс требует работы с внешними ресурсами (например, при чтении или записи данных с диска).

**Обработка операций с файлами**

Когда процесс или пользователь пытается получить доступ к файлу, запрос передается в блок управления файлами. Если файл существует и доступ к нему разрешен, блок управления файлами передает запрос в блок управления устройствами для обработки ввода-вывода.

Все операции с файлами (создание, удаление, чтение и запись) требуют взаимодействия с блоком управления устройствами, так как файлы физически хранятся на устройствах.

**Обслуживание устройств:**

* при подключении нового устройства или при изменении состояния существующего устройства, блок управления устройствами обновляет свою таблицу, добавляя информацию о новом устройстве и его характеристиках;
* при выполнении операций ввода-вывода блок управления устройствами передает запросы на чтение/запись в соответствующие драйвера.

**Контроль прав доступа:**

* все операции с ресурсами проверяются через блок управления пользователями. При попытке выполнения операции (например, открытие файла или запуск процесса), блок управления пользователями проверяет права доступа пользователя (например, с помощью поля *U\_RIGHT*);
* если у пользователя есть соответствующие права, операция продолжается, если нет — она блокируется.

# **ОПИСАНИЕ ФУНКЦИОНИРОВАНИЯ ОПЕРАЦИОННОЙ СИСТЕМЫ**

Функционирование гипотетической операционной системы основано на четком взаимодействии различных подсистем и управлении ресурсами компьютера. Рассмотрим основные аспекты работы системы:

**Алгоритмы управления ресурсами**

Управление ресурсами в системе осуществляется с использованием следующих подходов:

* **процессорное время:** планирование процессов реализовано с применением вытесняющего алгоритма циклического *LIFO* (*Last In, First Out*). Это обеспечивает выполнение наиболее недавно добавленных задач с возможностью вытеснения менее приоритетных;
* **оперативная память:** система использует сегментное распределение памяти, что позволяет выделять память блоками, соответствующими логическим сегментам (код, данные, стек);
* **внешние устройства:** доступ к устройствам (клавиатура, принтер, флеш-диск и т.д.) осуществляется через синхронные операции ввода-вывода, что гарантирует завершение операции до перехода к следующей задаче.

**Алгоритмы управления процессами**

* **планирование процессов:** реализован вытесняющий алгоритм циклического *LIFO*. Каждый процесс имеет квант времени, после которого, если он не завершен, он помещается в конец очереди;
* **организация взаимодействия процессов:** используется алгоритм пекарни Лэмпорта, обеспечивающий координацию процессов при доступе к общим ресурсам и предотвращающий гонки данных;
* **синхронизация процессов:** реализована через решение задачи "Обедающие философы". Используются семафоры для управления доступом к ресурсам и предотвращения взаимной блокировки;
* **предотвращение тупиков:** применяется алгоритм банкира. Система оценивает, достаточно ли доступных ресурсов для безопасного выполнения нового процесса, прежде чем выделить их.

**Организация оперативной памяти**

Оперативная память организована по сегментной схеме. Каждый процесс получает доступ к отдельным сегментам, что позволяет избежать фрагментации и изоляции данных одного процесса от другого. Сегменты могут увеличиваться или уменьшаться в зависимости от текущих потребностей приложения.

**Алгоритмы управления памятью.**

* **распределение памяти:** сегментное распределение минимизирует внешнюю фрагментацию и упрощает управление;
* **освобождение памяти:** завершенные процессы возвращают выделенные сегменты в общий пул;
* **перераспределение:** при необходимости система может перемещать сегменты для оптимального использования доступной памяти.

**Общее функционирование операционной системы**

* **загрузка системы:** загрузчик ОС считывает ядро с жесткого диска в память, инициализирует основные таблицы и запускает основные системные процессы;
* **выполнение процессов:** система циклически обрабатывает запросы на выполнение приложений, управляет их состоянием (выполнение, ожидание, завершение) и освобождает ресурсы после их завершения;
* **работа с файловой системой:** доступ к файлам организован через иерархическую структуру каталогов с использованием индексной логической и связной физической организации;
* **интерфейс пользователя:** взаимодействие осуществляется через текстовый командный интерфейс, поддерживающий основные команды работы с файлами, каталогами и входом/выходом из системы.

# **ЗАКЛЮЧЕНИЕ**

В рамках курсового проекта была разработана гипотетическая операционная система с заданными характеристиками. Основная цель работы, заключающаяся в приобретении практических навыков проектирования компонентов операционных систем, была успешно достигнута.

В процессе выполнения проекта был проведен выбор архитектуры вычислительной системы, включающей процессор, оперативную память, жесткий диск и внешние устройства, такие как клавиатура, мышь, монитор, флеш-диск и принтер. Описан однопрограммный режим работы компьютера с учетом специфики управления ресурсами. Разработана архитектура операционной системы на основе экзоядра, что позволило минимизировать размер ядра и обеспечить эффективное управление ресурсами.

В рамках работы детально изучены алгоритмы управления процессами, включая вытесняющий планировщик *LIFO*, алгоритм взаимодействия процессов на основе пекарни Лэмпорта, задачи синхронизации процессов, а также способы предотвращения тупиков с использованием алгоритма банкира. Подсистема управления памятью описана с использованием сегментного распределения, что обеспечивает рациональное использование доступной оперативной памяти.

Изучена организация файловой системы, включающая иерархию каталогов, индексную логическую организацию и физическую структуру, основанную на связанном списке индексов. Разработан пользовательский интерфейс, ориентированный на текстовый терминал, и предложен упрощенный командный язык для взаимодействия с системой. Представлен алгоритм загрузки операционной системы с жесткого диска и описан процесс запуска прикладных программ.

Проект также включает проектирование структуры управляющих блоков базы данных операционной системы с описанием их взаимодействия. В работе дано комплексное описание функционирования операционной системы, включая управление процессами, памятью, файлами и ресурсами.

Выполнение данного проекта позволило не только углубить теоретические знания, но и приобрести практические навыки проектирования и анализа операционных систем. Итоговая работа демонстрирует способность применять современные подходы к разработке, адаптируя алгоритмы и структуры для выполнения конкретных задач. Данный проект стал важным этапом профессионального становления в области операционных систем.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Таненбаум, Э. С., Бос, Х. Современные операционные системы. — 4-е изд. — СПб.: Питер, 2015. — 1120 с.
2. Востокин, С. В. Архитектура операционных систем: учебное пособие. — Самара: Издательство Самарского университета, 2023. — 84 с.
3. Грейбо, С. В., Новосёлова, Т. Е., Пронькин, Н. Н., Семёнычева, И. Ф. Архитектура вычислительных систем: учебное пособие. [Электронный ресурс]. URL: <http://scipro.ru/conf/computerarchitecture.pdf> (дата обращения: 23.12.2024).
4. Studizba. Режимы работы ЭВМ. [Электронный ресурс]. URL: <https://studizba.com/lectures/informatika-i-programmirovanie/vychislitelnye-mashiny-sistemy-i-seti/3697-rezhimy-raboty-evm.html> (дата обращения: 23.12.2024).
5. Studfile. Лекция 7 Режимы работы ЭВМ. [электронный курс]. URL: <https://studfile.net/preview/1701547/> (дата обращения 23.12.2024)
6. Intuit. Основы операционных систем. Лекция 5. Режимы работы ЭВМ. [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/954/397/lecture/9153> (дата обращения: 23.12.2024).
7. HammockDale. Philosophers. [Электронный ресурс]. URL: <https://github.com/HammockDale/Philosophers> (дата обращения: 27.12.2024).
8. Intuit. Основы операционных систем. Лекция 5. Режимы работы ЭВМ. [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/941/31/lecture/980> (дата обращения: 27.12.2024).
9. Guru99. Алгоритм банкира в операционных системах. [Электронный ресурс]. URL: <https://www.guru99.com/ru/bankers-algorithm-in-operating-system.html> (дата обращения: 28.12.2024).
10. Dit.isuct. Операционные системы. Глава 7. [Электронный ресурс]. URL: <https://dit.isuct.ru/IVT/BOOKS/OPERATING_SYSTEMS/OPER12/GLAVA_7.HTM> (дата обращения: 29.12.2024).
11. Yztm.ru. Операционные системы. Лекция 6. [Электронный ресурс]. URL: <https://yztm.ru/epo/epo_l6/> (дата обращения: 28.12.2024).
12. Bibliofond.ru. Операционные системы: основные понятия и функции. [Электронный ресурс]. URL: <https://www.bibliofond.ru/view.aspx?id=519155> (дата обращения: 05.01.2025).
13. Intuit. Основы операционных систем. Лекция 6. [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/631/487/lecture/11059?page=6> (дата обращения: 05.01.2025).
14. Andreyex. Процесс загрузки в операционной системе: вопросы для собеседования. [Электронный ресурс]. URL: <https://andreyex.ru/programmnoe-obespechenie/protsess-zagruzki-v-operatsionnoj-sisteme-voprosy-dlya-sobesedovaniya-po-operatsionnoj-sisteme/> (дата обращения: 06.01.2025).
15. Studfile. Формирование базы данных ОС [Электронный ресурс]. URL: <https://studfile.net/preview/9124585/page:6/> (дата обращения: 06.01.2025)