Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Схемотехника

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ВИДЖЕТ ДЛЯ МОНИТОРИНГА ПАРАМЕТРОВ ОКРУЖАЮЩЕЙ СРЕДЫ

БГУИР КП 1-40 02 01 121 ПЗ

Студент: группы 250501  
Снитко Д.А.

Руководитель: ассистент каф. ЭВМ   
Стракович А.И.

Минск 2024

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: КСиС. Кафедра: ЭВМ.

Специальность: 40 02 01 «Вычислительные машины, системы и сети».

Специализация: 400201-01 «Проектирование и применение локальных компьютерных сетей».

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

\_\_\_\_\_\_\_\_\_ Б.В. Никульшин

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024 г.

ЗАДАНИЕ

по курсовому проекту студента

Снитко Даниила Александровича

**1** Тема проекта: «Виджет для мониторинга параметров окружающей среды»

**2** Срок сдачи студентом законченного проекта: 1 декабря 2024 г.

**3** Исходные данные к проекту:

**3.1** Микроконтроллер.

**3.2** Датчики – не менее 3 шт.

**3.3** Модуль управления.

**3.4** Модуль отображения информации.

**3.5** Модуль индикации.

**3.6** Модуль исполнительного устройства.

**3.7** Источник питания.

**4** Содержание пояснительной записки (перечень подлежащих разработке

вопросов):

Введение 1. Обзор литературы. 2. Разработка структуры устройства. 3. Обоснование выбора узлов, элементов функциональной схемы устройства. 4. Разработка принципиальной электрической схемы устройства. 5. Разработка программного обеспечения. Заключение. Список использованных источников. Приложения.

**5** Перечень графического материала (с точным указанием обязательных

чертежей):

**5.1** Название темы. Схема электрическая структурная.

**5.2** Название темы. Схема электрическая функциональная.

**5.3** Название темы. Схема электрическая принципиальная.

**5.4** Название темы. Схема печатной платы (кто будет делать).

КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапов  курсового проекта | Объем  этапа,  % | Срок выполнения этапа | Примечания |
| Обзор литературы | 15 | 01.09 – 20.09 |  |
| Разработка структурной схемы | 15 | 21.09 – 04.10 |  |
| Разработка функциональной схемы | 20 | 05.10 – 23.10 |  |
| Разработка принципиальной схемы | 15 | 24.10 – 05.11 |  |
| Разработка программного обеспечения | 15 | 06.11 – 15.11 |  |
| Создание макета устройства | 10 | 16.11 – 23.11 |  |
| Оформление пояснительной записки и графического материала | 10 | 24.11 – 01.12 |  |
| Защита курсового проекта |  | 07.12 – 19.12 |  |

Дата выдачи задания: 13.09.2024 г.

Руководитель А.И. Стракович

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ \_\_\_\_\_\_\_\_\_\_

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 5

1. ОБЗОР ЛИТЕРАТУРЫ 6
   1. Основные шаги разработки диспетчера процессов и потоков 6
   2. Постановка задачи 7
   3. Обзор существующих аналогов 7

1.4 Сравнительный анализ 22

1. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 24
   1. Блок ввода-вывода 25
   2. Блок чтения данных 25
   3. Блок управления процессами и потоками 25
   4. Блок главного цикла программы 25
2. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 27
   1. Описание основных структур данных программы 26
   2. Описание основных функций программы 28
3. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 31
   1. Разработка структурной схемы 31
   2. Схемы алгоритмов 31

4.2.1 Схема алгоритма get\_thread\_info 31

4.2.2 Схема алгоритма count\_cpu\_cores 31

* 1. Разработка алгоритмов 31

4.3.1 Алгоритм функции read\_sysinfo 31

4.3.2 Алгоритм функции read\_processes 32

4.4 Код программы 33

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 34

5.1 Требования к программному и аппаратному обеспечению 34

5.2 Руководство по использованию 34

ЗАКЛЮЧЕНИЕ 41

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 42

ПРИЛОЖЕНИЕ А 43

ПРИЛОЖЕНИЕ Б 44

ПРИЛОЖЕНИЕ В 45

ПРИЛОЖЕНИЕ Г 46

ПРИЛОЖЕНИЕ Д 47

**ВВЕДЕНИЕ**

Мониторинг окружающей среды является важным элементом для создания комфортных условий как в жилых, так и в рабочих пространствах. Современные устройства для отслеживания параметров, таких как температура, влажность и уровень шума, позволяют поддерживать оптимальный микроклимат, что положительно влияет на здоровье и благополучие людей, а также на работу различных технических систем.

Особенно актуальны подобные решения в условиях изменяющегося климата и роста требований к энергоэффективности зданий и производственных объектов. Устройства, способные измерять и анализировать параметры окружающей среды, имеют множество применений: от обеспечения благоприятных условий в жилых помещениях до контроля за параметрами в производственных цехах и офисах. Они позволяют следить за изменениями температуры, влажности и уровня шума, а также своевременно реагировать на их отклонения от нормальных значений. Это помогает предотвратить перегрев или переохлаждение помещений, контролировать влажность для предотвращения развития плесени и оптимизировать акустические условия для повышения комфорта и продуктивности.

Одной из ключевых задач подобных устройств является предоставление пользователю возможности не только наблюдать за текущими параметрами, но и задавать индивидуальные пороговые значения для каждого из них.

Целью разработки подобных устройств является не только предоставление информации пользователю, но и обеспечение простоты в эксплуатации. Такие устройства обычно оснащены интуитивно понятными интерфейсами и отображают измеренные параметры в реальном времени, что позволяет пользователю легко оценивать текущее состояние окружающей среды. Важной задачей является также создание системы визуальной индикации, которая наглядно демонстрирует изменения параметров, не требуя от пользователя сложных действий.

1. **ОБЗОР ЛИТЕРАТУРЫ**

**1.1 Аналоги устройства**

На рынке существует множество устройств для мониторинга параметров окружающей среды, которые измеряют такие показатели, как температура, влажность и уровень шума. Эти устройства варьируются от простых бытовых датчиков до сложных систем для промышленного использования, предоставляющих точные данные в реальном времени.

* + 1. **Xiaomi Mi Temperature and Humidity Monitor 2**

Это компактное устройство, предназначенное для мониторинга температуры и влажности в помещении. Дисплей выполнен на основе электронных чернил, что позволяет экономить заряд батареи и обеспечивает хорошую видимость даже при ярком свете. Устройство подключается к экосистеме Xiaomi через приложение Mi Home для удобного мониторинга и хранения данных.

Достоинства – низкое энергопотребление благодаря технологии электронных чернил. Интеграция с экосистемой умного дома Xiaomi. Удобное приложение с функцией истории данных.

Недостатки – отсутствие встроенного датчика шума. Ограниченные настройки без использования приложения. Не поддерживает внешние датчики.

На рисунке 1 изображено устройство



Рисунок 1 – Xiaomi Mi Temperature and Humidity Monitor 2

**1.1.2** **Govee Temperature Humidity Monitor**

Устройство от Govee предлагает мониторинг температуры и влажности с возможностью беспроводного подключения через Bluetooth или Wi-Fi. Оно синхронизируется с приложением Govee Home, которое предоставляет графики и уведомления о состоянии окружающей среды в реальном времени. Имеет LED-дисплей для локального отображения данных.

Достоинства – беспроводное подключение через Bluetooth или Wi-Fi. Удобное приложение для мониторинга в реальном времени. Возможность настройки пороговых значений с уведомлениями.

Недостатки – нет встроенного датчика для мониторинга уровня шума. Требует частой подзарядки в зависимости от режима работы. Зависимость от приложения для расширенных функций.

На рисунке 2 изображено устройство Govee Temperature Humidity Monitor



Рисунок 2 – Govee Temperature Humidity Monitor

**1.1.3 SensorPush Wireless Thermometer/Hygrometer**

SensorPush — это беспроводной термометр-гигрометр, который синхронизируется с мобильным приложением для мониторинга температуры и влажности. Он использует Bluetooth и может хранить данные в облаке с помощью дополнительного Wi-Fi-шлюза. Устройство обладает высокой точностью измерений и рекомендуется для использования в местах, требующих точного контроля микроклимата.

Достоинства – высокая точность измерений. Хранение данных в облаке при наличии Wi-Fi-шлюза. Компактные размеры и простота использования.

Недостатки – не отображает данные на дисплее, только через приложение. Отсутствие датчика шума. Требует дополнительного шлюза для полноценной работы с Wi-Fi.

На рисунке 3 изображено устройство SensorPush Wireless Thermometer/Hygrometer



Рисунок 1.10 – Интерфейс утилиты xrestop

**1.2 Основные компоненты**

Основные компоненты устройства:

– Плата с микроконтроллером;

– Датчик температуры и влажности;

– Датчик уровня шума;

– Дисплей;

– 3 кнопки для управления.

* 1. **Плата с микроконтроллером**

Arduino Nano — компактная и доступная плата. Она имеет достаточно цифровых и аналоговых входов/выходов для подключения датчиков и дисплея, что делает её универсальной для множества проектов. Характеристики:

– Микроконтроллер: ATmega328P;

– Рабочее напряжение: 5V;

– Количество цифровых пинов: 14;

– Количество аналоговых входов: 8;

– Память: 32 KB Flash, 2 KB SRAM, 1 KB EEPROM.

* 1. **Датчик температуры и влажности**

DHT11 — достаточно точен для обычных проектов. Он хорошо подходит для измерения температуры и влажности, особенно в тех случаях, когда высокая точность не является критической. Характеристики:

– Диапазон измерения температуры: 0°C до +50°C (точность ±2°C);

– Диапазон влажности: 20% до 80% (точность ±5%);

– Напряжение питания: 3.3V-5V.

* 1. **Датчик уровня шума**

KY-037 — это чувствительный микрофонный модуль, способный измерять уровень шума в окружающей среде. Он прост в использовании и достаточно чувствителен для бытового мониторинга звука. Характеристики:

– Напряжение питания: 3.3V-5V;

– Аналоговый выход: пропорционален уровню звукового давления;

– Цифровой выход: активируется при достижении установленного порога – шума;

* 1. **Дисплей**

OLED SSD1306 — это стандартный и широко используемый дисплей для вывода данных. Выбор модуля I2C уменьшает количество необходимых для подключения пинов и упрощает схему. Он удобен для отображения температуры, влажности, уровня шума и времени. Характеристики:

– Разрешение: 128х64 точек;

– Напряжение питания: 5V;

– Интерфейс: I2C;

– Низкое энергопотребление.

* 1. **Элементы управления**

Для управления устройством было рассмотрено добавление трех кнопок, с помощью которых пользователь может регулировать граничные значения параметров и настройки времени.

**2 РАЗРАБОТКА СТРУКТУРЫ УСТРОЙСТВА**

Структурная схема устройства приведена в приложении А.

**2.1 Перечень блоков**

Можно выделить 4 основных блока:

1. Блок датчиков;
2. Микроконтроллер;
3. Блок управления
4. Блок отображения данных;

Блок датчиков содержит датчики температуры и влажности (DHT11) и датчик уровня шума (KY-037). Этот блок отвечает за измерение параметров окружающей среды и передачу этих данных микроконтроллеру для дальнейшей обработки.

Микроконтроллер – основной управляющий элемент системы (Arduino Nano), который принимает данные от датчиков, обрабатывает их, контролирует работу всех компонентов устройства, включая блок управления и блок отображения данных.

Блок управления включает три кнопки (Вверх, Вниз, ОК), используемые для навигации по интерфейсу, настройки пороговых значений и установки времени. Этот блок обеспечивает взаимодействие пользователя с системой.

Блок отображения данных включает TFT-дисплей, на котором отображаются текущие параметры окружающей среды, такие как температура, время, влажность и уровень шума, а также другая информация о состоянии системы.

**2.2 Взаимодействие блоков**

Устройство, построенное на основе микроконтроллера Arduino Nano, взаимодействует между своими блоками для выполнения задач мониторинга и отображения данных о параметрах окружающей среды. Рассмотрим, как каждый блок работает в системе и каким образом происходит взаимодействие между ними.

Микроконтроллер как центральный узел взаимодействия

Микроконтроллер (Arduino Nano) играет роль управляющего центра, который координирует работу всех остальных блоков. Вся информация, поступающая с датчиков и кнопок, обрабатывается микроконтроллером, который затем принимает решения о выводе данных на дисплей или других действиях.

Взаимодействие с блоком датчиков

Датчики температуры, влажности и уровня шума подключены к аналоговым или цифровым входам микроконтроллера. Датчик DHT11 передаёт значения температуры и влажности с помощью цифрового сигнала через один из входов, используя протокол данных, который Arduino считывает через специализированную библиотеку. KY-037, представляющий собой микрофонный датчик, передаёт аналоговые сигналы, которые микроконтроллер интерпретирует как уровень звука.

Микроконтроллер запрашивает данные с этих датчиков поочередно, например, каждые несколько секунд, чтобы обновить показания окружающей среды. Эти данные сохраняются в переменных микроконтроллера для дальнейшей обработки и отображения.

Взаимодействие с блоком управления

Блок управления состоит из трёх кнопок: Вверх, Вниз и ОК. Эти кнопки подключены к цифровым входам микроконтроллера, и каждая кнопка работает по принципу замыкания цепи, что регистрируется микроконтроллером как нажатие. При нажатии кнопок микроконтроллер интерпретирует команды, поступающие от пользователя. Например, кнопки Вверх и Вниз могут использоваться для навигации по меню настроек или изменения пороговых значений температуры и влажности. Кнопка ОК подтверждает выбор или запускает изменение параметров. Микроконтроллер обрабатывает каждое нажатие, изменяя отображение на дисплее в соответствии с действиями пользователя, а также устанавливая новые значения порогов, если это необходимо.

Взаимодействие с блоком отображения данных

Дисплей получает команды от микроконтроллера, что позволяет ему отображать информацию. Микроконтроллер обновляет отображаемую информацию в зависимости от данных, полученных с датчиков или действий пользователя через блок управления. Например, если уровень шума превышает установленный порог, микроконтроллер может отобразить предупреждение на экране. Точно так же изменения температуры и влажности также моментально отображаются на дисплее. При настройке пороговых значений интерфейс управления позволяет пользователю видеть текущие настройки и изменять их в реальном времени.

Автоматический режим и таймеры

В случае бездействия пользователя в течение определённого времени (например, 5 секунд), микроконтроллер автоматически выходит из режима настройки и возвращает устройство в режим обычного мониторинга. Для этого микроконтроллер использует встроенные таймеры, которые следят за временем между нажатиями кнопок.

Это взаимодействие всех блоков позволяет устройству выполнять свою задачу по мониторингу окружающей среды, предоставляя пользователю возможность настроить и получить данные через интерфейс.

**3. ОБОСНОВАНИЕ ВЫБОРА УЗЛОВ, ЭЛЕМЕНТОВ ФУНКЦИОНАЛЬНОЙ СХЕМЫ УСТРОЙСТВА**

Функциональная схема устройства приведена в приложении Б.

**3.1. Аппаратная платформа**

Факторами выбора аппаратной платформы были: доступность, стоимость, простота загрузки прошивки, распространённость платы, наличие большого количества инструкций и материалов по использованию.

С учётом данных факторов было решено использовать микроконтроллер Arduino Nano V3.0 с чипом CH340, так как он обладает достаточной функциональностью для работы с датчиками и модулями через интерфейсы I2C и цифровые входы. Легко прошивается через USB-интерфейс и имеет широкое сообщество, что упрощает разработку устройства.

**3.2. Датчик температуры и влажности**

Для измерения температуры и влажности был выбран датчик DHT11. Этот датчик обеспечивает достаточную точность для реализации текущего проекта и легко подключается к микроконтроллеру. Он передаёт данные в цифровом формате через один из входов Arduino, что упрощает обработку и минимизирует количество подключений. Несмотря на то, что точность DHT11 ниже, чем у других датчиков, его доступность и простота интеграции делают его оптимальным выбором для данного устройства.

**3.3. Датчик уровня шума**

В качестве датчика уровня шума был выбран микрофонный модуль KY-037 (HW-485). Этот модуль предоставляет аналоговый сигнал, который может быть интерпретирован Arduino Nano как уровень шума. Основными причинами выбора KY-037 стали его низкая стоимость, простота интеграции и возможность измерения уровня звука, что соответствует целям данного проекта. Датчик реагирует на шумы в окружающей среде, и его показания используются для мониторинга уровня шума.

**3.4. Дисплей**

Для отображения данных выбран OLED-дисплей размером 0.96 дюйма с разрешением 128x64 пикселей и интерфейсом I2C. Дисплей компактен и обладает хорошей читаемостью благодаря высокой контрастности. Разрешение 128x64 пикселя позволяет выводить все основные параметры (температура, влажность, уровень шума) и состояние устройства и состояния смайлика. Так же интерфейс I2C минимизирует количество подключений к Arduino.

**4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

В данном разделе представлены схемы алгоритмов и алгоритмы по шагам основных функций разработанной в рамках курсового проекта.

**4.1 Разработка структурной схемы**

Структурная схема программы приведена в приложении А.

* 1. **Схемы алгоритмов**
     1. **Схема алгоритма get\_thread\_info**

Функция получает информацию о конкретном потоке в процессе. Она принимает на вход идентификатор процесса (PID) и идентификатор потока (TID). Затем она открывает соответствующие файлы в каталоге /proc/[pid]/task/[tid] для чтения информации о потоке. Схема алгоритма get\_thread\_info приведена в приложении Б.

* + 1. **Схема алгоритма count\_cpu\_cores**

Функция подсчитывает количество ядер процессора, на которых может выполняться процесс, исходя из списка ядер, на которых разрешено выполнение (список cpu\_list). Она принимает этот список в качестве входного параметра и возвращает количество ядер. Функция сканирует список и подсчитывает количество запятых, что позволяет определить число ядер. Схема алгоритма count\_cpu\_cores приведена в приложении В.

* 1. **Разработка алгоритмов**
     1. **Алгоритм функции kill\_process\_or\_thread**

Функция kill\_process\_or\_thread обеспечивает пользователю возможность завершения процесса или потока. Шаги выполнения этой функции:

1. Включение отображения вводимых символов и курсора: Функция вызывает echo() для включения отображения вводимых символов и curs\_set(1) для включения курсора.

2. Установка времени ожидания ввода на 15 секунд: Используется timeout(15000) для установки времени ожидания ввода на 15 секунд.

3. Ввод идентификатора (TID или PID): Пользователю предлагается ввести идентификатор процесса или потока.

4. Получение идентификатора и преобразование в целое число: Введенная строка преобразуется в целое число с помощью atoi().

5. Проверка на выход: Если введенное значение равно 0, функция завершает свою работу.

6. Попытка завершения процесса или потока: Вызывается функция kill() для отправки сигнала SIGKILL процессу или потоку с введенным идентификатором.

7. Вывод результата попытки завершения: На экран выводится сообщение о результате попытки завершения.

8. Повторение процесса ввода и попытки завершения: Цикл повторяется, пока время ожидания не истекло или пока пользователь не введет 0.

9. Выключение отображения вводимых символов и курсора: После завершения работы цикла функция отключает отображение вводимых символов и курсор.

10. Очистка экрана и завершение функции: Экран очищается с помощью clear(), и функция завершает свою работу.

* + 1. **Алгоритм функции get\_process\_info**

Функция read\_processes() отвечает за чтение информации о процессах и сохранение ее в структуре Process. Она читает информацию из файла /proc/[pid]/status, где [pid] - идентификатор процесса. Шаги выполнения этой функции:

1. Получение информации о процессе:

Функция получает в качестве аргументов указатель на структуру ProcessInfo и идентификатор процесса (pid).

2. Инициализация переменных и открытие файлов:

Инициализируются переменные path и buffer. Открывается файл /proc/[pid]/status для чтения информации о процессе.

3. Чтение информации из /proc/[pid]/status:

В цикле while считывается каждая строка из файла. Если строка начинается с Uid:, из нее извлекается идентификатор пользователя (uid), который используется для получения имени пользователя с помощью функции getpwuid(). Если строка начинается с VmSize:, из нее извлекается размер виртуальной памяти процесса (vm\_size). Если строка начинается с State:, из нее извлекается состояние процесса (state). Если строка начинается с Name:, из нее извлекается имя исполняемого файла процесса (command). Если строка начинается с Cpus\_allowed\_list:, из нее извлекается строка, содержащая список доступных ядер процессора, из которой определяется количество ядер (cpu\_cores). Если строка начинается с Threads:, из нее извлекается количество потоков (threads).

4. Чтение информации из /proc/[pid]/stat:

Открывается файл /proc/[pid]/stat для получения информации о времени работы процесса и потреблении физической памяти. Считываются необходимые значения: utime, stime, starttime, rss.

5. Расчет времени запуска процесса:

Вычисляется текущее время в секундах (now). Получается время работы системы в секундах (uptime) с помощью функции get\_system\_uptime(). Вычисляется время запуска процесса (start\_time). Для форматирования времени используется функция localtime\_r().

6. Вычисление объема занимаемой физической памяти:

Открывается файл /proc/[pid]/statm. Считывается количество страниц резидентной памяти (rss), занимаемых процессом. Вычисляется объем физической памяти в мегабайтах (resident\_memory).

7. Заполнение структуры ProcessInfo:

Полученные данные записываются в поля структуры ProcessInfo.

8. Закрытие файлов:

Все открытые файлы закрываются.

9. Завершение работы функции:

**4.4 Код программы**

Код программы представлен в приложении Г.

**5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

**5.1 Требования к программному и аппаратному обеспечению**

Процессор: любой совместимый с архитектурой x86 или x86-64

Оперативная память: не менее 128 МБ

Жесткий диск: не менее 10 МБ свободного места

**5.2 Руководство по использованию**

При запуске программы пользователю будет выведен заголовок со столбцами PID, USER, STATE, RES\_MEM, VIRT\_MEM, CORES, START, COMMAND и заголовок с текущими датой и временем, под столбцами заголовка выведена информация для каждого из процессов.

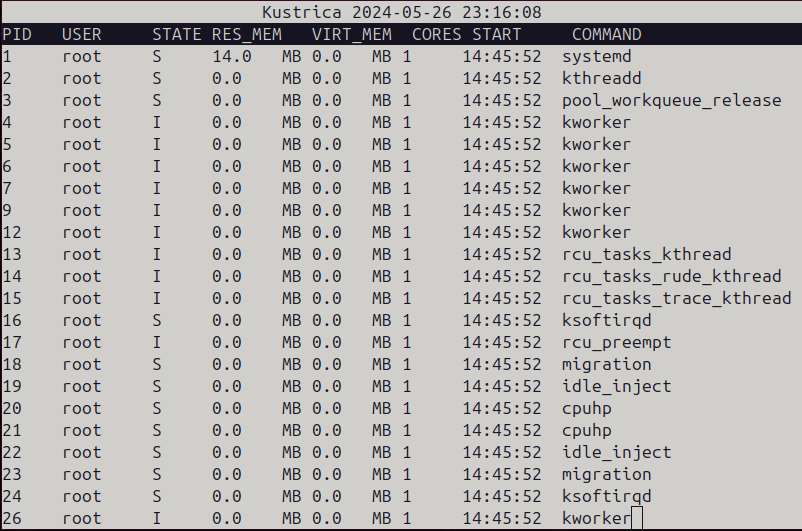


Рисунок 5.2.1 – Список процессов

Функции, которые пользователь может вызвать нажатием клавиш:

По нажатию на клавиатуре клавиши t или T, раскроется древовидный список потоков. Под каждым и процессов будут показаны его потоки и их идентификаторы.

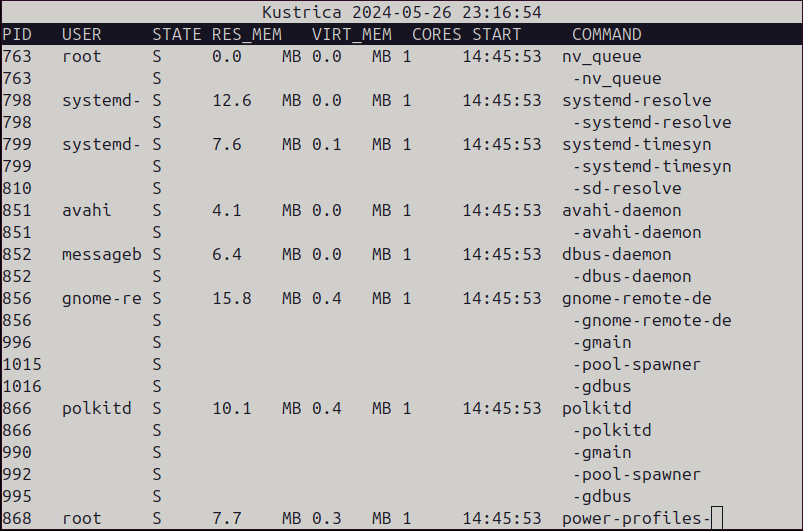


Рисунок 5.2.2 – Список процессов и потоков

По нажатию на клавиатуре клавиши p или P, произойдет сортировка процессов по PID по возрастанию.

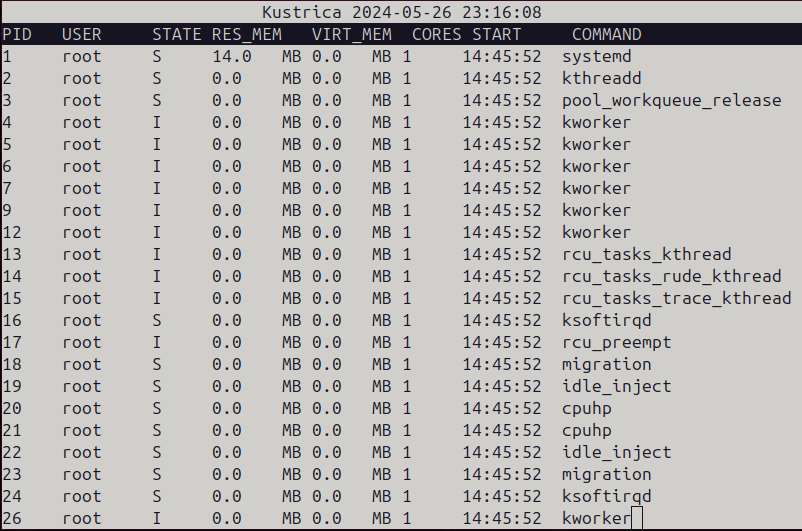


Рисунок 5.2.3 – Список процессов отсортированный по возрастанию PID

По нажатию на клавиатуре клавиши p или P повторно, произойдет сортировка процессов по PID по убыванию.

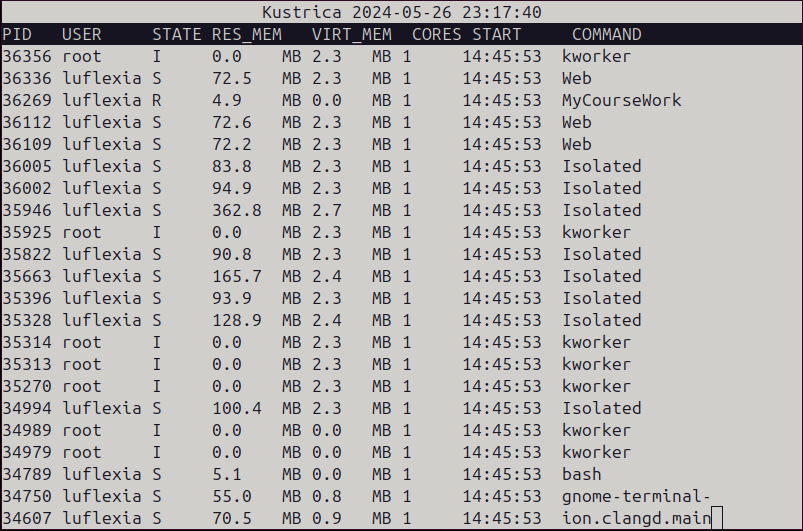


Рисунок 5.2.4 – Список процессов отсортированный по убыванию PID

По нажатию на клавиатуре клавиши r или R, произойдет сортировка процессов по RES\_MEM по убыванию.

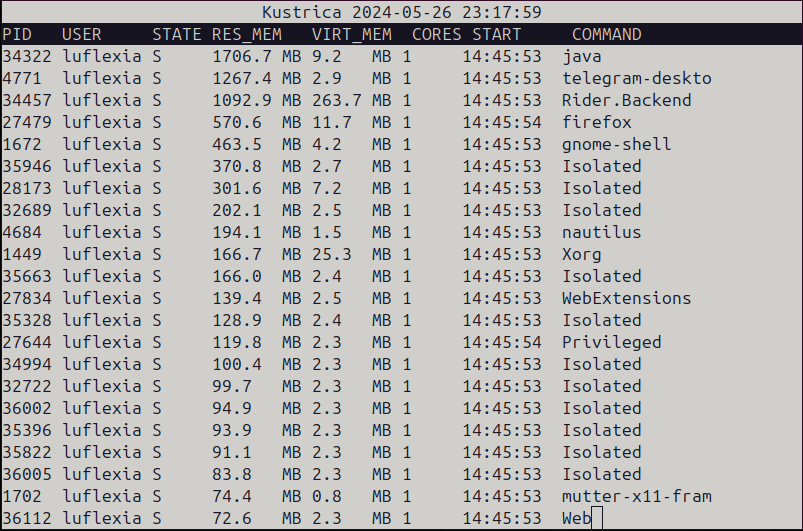


Рисунок 5.2.5 – Список процессов отсортированный по убыванию RES\_MEM

По нажатию на клавиатуре клавиши v или V, произойдет сортировка процессов по VIRT\_MEM по убыванию.

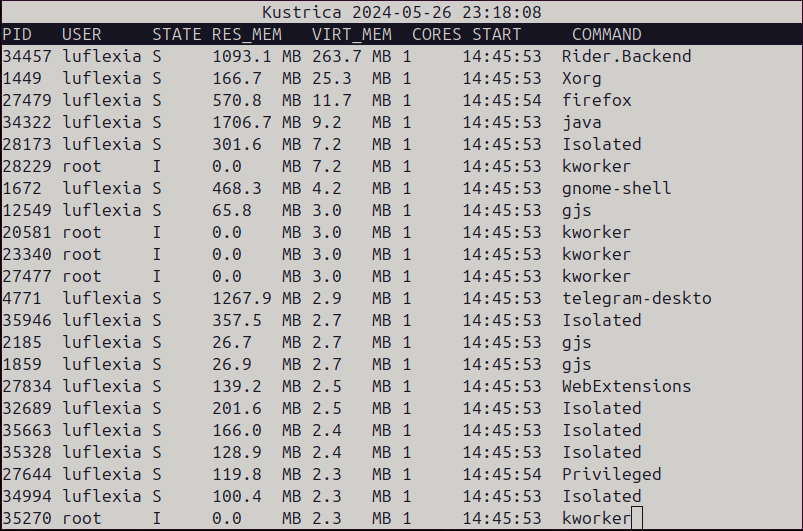


Рисунок 5.2.6 – Список процессов отсортированный по убыванию VIRT\_MEM

По нажатию на клавиатуре клавиши h или H, откроется справка, которая покажет функциональные клавиши, выйти из нее можно нажав любую клавишу.

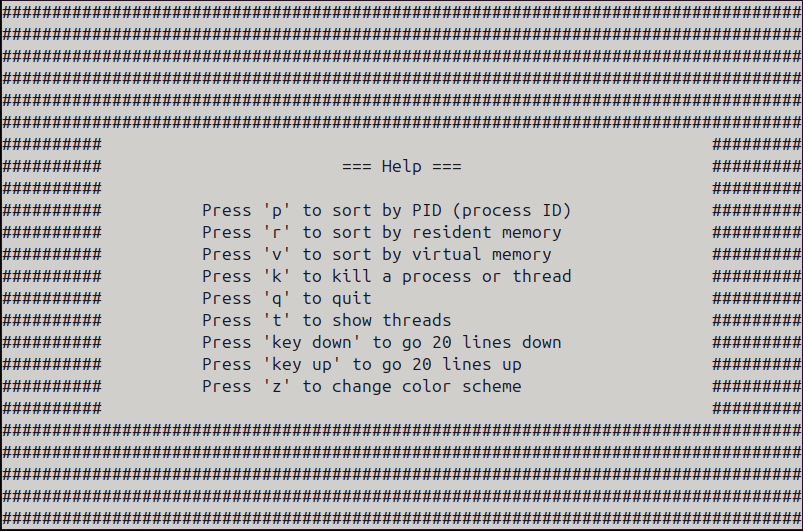


Рисунок 5.2.7 – Справка в светлом режиме отображения

По нажатию на клавиатуре клавиши z или Z, цвета в консоли инвертируются, фон станет черным, а цвет текста белым.

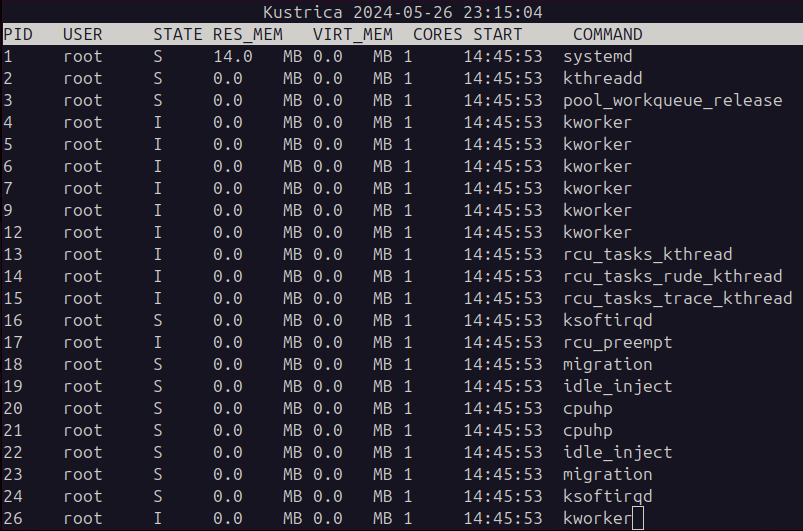


Рисунок 5.2.8 – Список процессов в темном режиме отображения

По нажатию на клавиатуре клавиши h или H, в темном режиме откроется справка, которая покажет функциональные клавиши.



Рисунок 5.2.9 – Справка в темном режиме отображения

По нажатию на клавиатуре клавиши k или K, на строке с заголовком появится поле для ввода идентификатор потока или процесса.

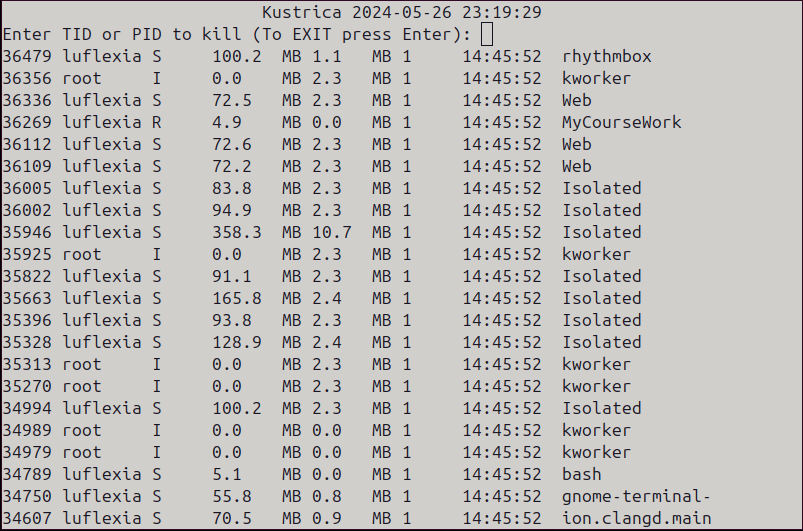


Рисунок 5.2.10 – Поле для ввода идентификатора потока или процесса

При попытке завершить процесс или поток, в случае успеха будет выведено соответствующее сообщение.

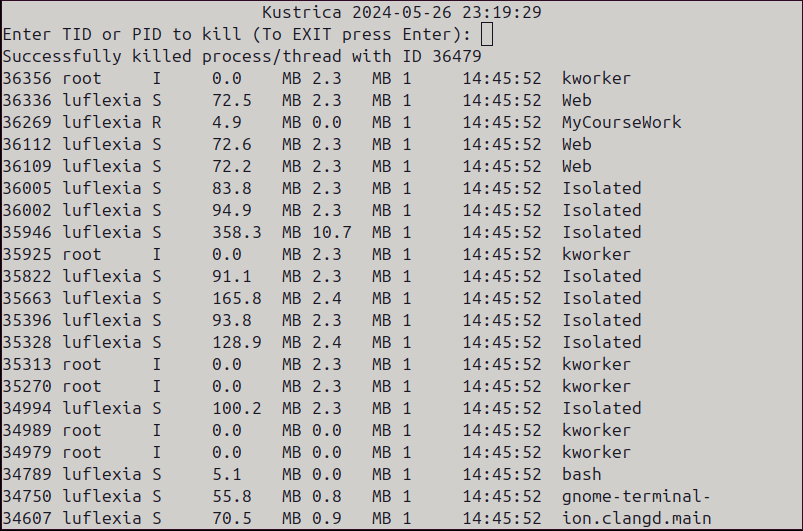


Рисунок 5.2.11 – Успешное завершение потока или процесса

В случае если процесс или поток завершить не удалось выведется соответствующее сообщение.

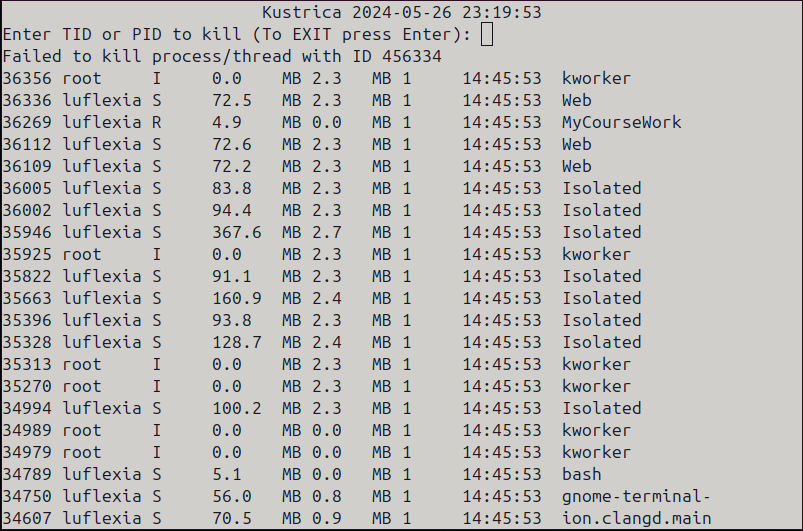


Рисунок 5.2.12 – Неудачное завершение потока или процесса

**ЗАКЛЮЧЕНИЕ**

В рамках курсового проекта была разработана программа диспетчера процессов и потоков, представляющая собой аналог утилиты top, которая предоставляет пользователю важный инструмент для мониторинга, анализа и мониторинга потоков и процессов, запущенных в системе.

Разработка диспетчера процессов и потоков включает в себя рассмотрение основных этапов, таких как получение списка процессов и потоков, обновление информации о них и отображение на экране. В процессе разработки необходимо было изучить системные вызовы и функции для работы с процессами и потоками операционной системы.

Программа предоставляет пользователю информацию о процессах, такую как идентификатор процесса (PID\TID), пользователь, потребление физической и виртуальной памяти, количество ядер процессора, которые использует процесс, какой командой был запущен процесс или поток. Кроме того, пользователь может сортировать и управлять процессами и потоками, что позволяет эффективно мониторить и анализировать работу системы.

Разработанный диспетчер процессов и потоков предоставляет пользователю возможность наблюдать текущие процессы, анализировать их характеристики и принимать решения на основе полученных данных. Это позволяет оптимизировать работу системы, выявлять и устранять проблемы с производительностью, а также обеспечивать безопасность системы.

В процессе разработки был использован язык программирования Си, а также библиотеки для работы с процессами и потоками, а так же библиотека ncurses, которая предоставляет набор функций для создания текстовых пользовательских интерфейсов (TUI) в терминальном окне. Были реализованы такие функции, как получение списка процессов и потоков, обновление информации о них, сортировка процессов, обработка пользовательского ввода, а также управление процессами и потоками.

Программа может быть дополнена и расширена для поддержки дополнительных функциональных возможностей, таких как мониторинг сетевой активности, анализ дискового пространства и мониторинг температуры компонентов системы.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

Брайан Керниган, Деннис Ритчи. Язык программирования Си. Издательство: «Вильямс», 2019 г.

Ричард Стивенс, Стивен Раго. UNIX. Профессиональное программирование. Издательство: «Вильямс», 2017 г.

Ричард Стивенс, Стивен Раго. Разработка приложений для UNIX. Издательство: «Питер», 2011 г.

The C Programming Language. Издательство: Prentice Hall, 1988 г.

top, htop, atop определение загрузки ОС (Load average, LA) [Электронный ресурс] – Режим доступа: https://wiki.dieg.info/top

Analysis with top in Linux [Электронный ресурс] – Режим доступа: https://prowse.tech/top/

**ПРИЛОЖЕНИЕ А**

(Обязательное)

Схема структурная

**ПРИЛОЖЕНИЕ Б**

(Обязательное)

Схема алгоритма get\_thread\_info

**ПРИЛОЖЕНИЕ В**

(Обязательное)

Схема алгоритма count\_cpu\_cores

**ПРИЛОЖЕНИЕ Г**

(Обязательное)

Код программы

Файл control.c

#include <curses.h>

#include <unistd.h>

#include <stdlib.h>

#include <signal.h>

#include "display.h"

#include "control.h"

SortCriteria current\_sort = SORT\_BY\_PID; // начальное значение

SortOrder sort\_order = SORT\_ORDER\_ASCENDING; // начальный порядок сортировки

void kill\_process\_or\_thread() {

char input[10];

int id = -1;

echo(); // Включаем отображение вводимых символов

curs\_set(1); // Включаем курсор

timeout(15000); // Устанавливаем время ожидания ввода 15 секунд

while (1) {

// Очищаем строку ввода и выводим приглашение к вводу

move(1, 0);

clrtoeol();

printw("Enter TID or PID to kill (To EXIT press Enter): ");

refresh();

// Ввод ID

if (getnstr(input, sizeof(input) - 1) == ERR) {

// Если время истекло и пользователь не ввел ID, выходим из функции

break;

}

id = atoi(input);

// Проверка на выход

if (id == 0) {

break;

}

// Попытка убить процесс/поток

move(2, 0); // Перемещаем курсор под строку ввода

clrtoeol(); // Очищаем строку

if (id > 0 && kill(id, SIGKILL) == 0) {

printw("Successfully killed process/thread with ID %d\n", id);

} else {

printw("Failed to kill process/thread with ID %d\n", id);

}

refresh();

}

noecho(); // Отключаем отображение вводимых символов

curs\_set(0); // Выключаем курсор

clear(); // Очищаем экран после завершения режима

refresh();

}

int compare\_by\_pid(const void \*a, const void \*b) {

const ProcessData \*p1 = (const ProcessData \*)a;

const ProcessData \*p2 = (const ProcessData \*)b;

return (sort\_order == SORT\_ORDER\_ASCENDING) ?

(p1->process\_info.pid - p2->process\_info.pid) :

(p2->process\_info.pid - p1->process\_info.pid);

}

int compare\_by\_resident\_memory(const void \*a, const void \*b) {

const ProcessData \*p1 = (const ProcessData \*)a;

const ProcessData \*p2 = (const ProcessData \*)b;

if (p1->process\_info.resident\_memory < p2->process\_info.resident\_memory)

return (sort\_order == SORT\_ORDER\_ASCENDING) ? -1 : 1;

if (p1->process\_info.resident\_memory > p2->process\_info.resident\_memory)

return (sort\_order == SORT\_ORDER\_ASCENDING) ? 1 : -1;

return 0;

}

int compare\_by\_virtual\_memory(const void \*a, const void \*b) {

const ProcessData \*p1 = (const ProcessData \*)a;

const ProcessData \*p2 = (const ProcessData \*)b;

if (p1->process\_info.virtual\_memory < p2->process\_info.virtual\_memory)

return (sort\_order == SORT\_ORDER\_ASCENDING) ? -1 : 1;

if (p1->process\_info.virtual\_memory > p2->process\_info.virtual\_memory)

return (sort\_order == SORT\_ORDER\_ASCENDING) ? 1 : -1;

return 0;

}

void handle\_user\_input(int ch) {

switch (ch) {

case 'p':

case 'P':

if (current\_sort == SORT\_BY\_PID) {

sort\_order = (sort\_order == SORT\_ORDER\_ASCENDING) ? SORT\_ORDER\_DESCENDING : SORT\_ORDER\_ASCENDING;

} else {

current\_sort = SORT\_BY\_PID;

sort\_order = SORT\_ORDER\_ASCENDING;

}

break;

case 'r':

case 'R':

if (current\_sort == SORT\_BY\_RESIDENT\_MEMORY) {

sort\_order = (sort\_order == SORT\_ORDER\_ASCENDING) ? SORT\_ORDER\_DESCENDING : SORT\_ORDER\_ASCENDING;

} else {

current\_sort = SORT\_BY\_RESIDENT\_MEMORY;

sort\_order = SORT\_ORDER\_ASCENDING;

}

break;

case 'v':

case 'V':

if (current\_sort == SORT\_BY\_VIRTUAL\_MEMORY) {

sort\_order = (sort\_order == SORT\_ORDER\_ASCENDING) ? SORT\_ORDER\_DESCENDING : SORT\_ORDER\_ASCENDING;

} else {

current\_sort = SORT\_BY\_VIRTUAL\_MEMORY;

sort\_order = SORT\_ORDER\_ASCENDING;

}

break;

case 'k':

case 'K':

kill\_process\_or\_thread();

break;

case 'q':

case 'Q':

endwin();

exit(0);

default:

break;

}

}

Файл control.h

#ifndef CONTROL\_H

#define CONTROL\_H

typedef enum {

SORT\_BY\_PID,

SORT\_BY\_RESIDENT\_MEMORY,

SORT\_BY\_VIRTUAL\_MEMORY

} SortCriteria;

typedef enum {

SORT\_ORDER\_ASCENDING,

SORT\_ORDER\_DESCENDING

} SortOrder;

extern SortCriteria current\_sort;

void handle\_user\_input(int ch);

void kill\_process\_or\_thread();

int compare\_by\_pid(const void \*a, const void \*b);

int compare\_by\_resident\_memory(const void \*a, const void \*b);

int compare\_by\_virtual\_memory(const void \*a, const void \*b);

#endif //CONTROL\_H

Файл display.c

#include "display.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <ncurses.h>

#include <unistd.h>

#include "processes.h"

#include "threads.h"

#include "control.h"

void display\_thread\_info(const ThreadInfo \*thread\_info) {

printw("%-5d %-8s %-5c %-9s %-9s %-4s %-10s -%s\n",

thread\_info->tid, "", thread\_info->state, "", "", "", "", thread\_info->name);

}

void display\_header(ColorScheme color\_scheme) {

int max\_y, max\_x; // Объявляем переменные для хранения размеров экрана

getmaxyx(stdscr, max\_y, max\_x); // Получаем размеры окна

char datetime[50];

time\_t rawtime;

struct tm \*timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

strftime(datetime, 50, "%Y-%m-%d %H:%M:%S", timeinfo); // Форматируем дату и время

int datetime\_length = strlen(datetime);

const char \*prefix = "Kustrica ";

int prefix\_length = strlen(prefix);

int total\_length = prefix\_length + datetime\_length;

int padding = (max\_x - total\_length) / 2;

// Выводим текст "Kustrica " и дату/время по центру первой строки без цветовой пары

mvprintw(0, padding, "%s%s", prefix, datetime);

// Устанавливаем цветовую пару для второй строки в зависимости от схемы

if (color\_scheme == COLOR\_SCHEME\_INVERTED) {

attron(COLOR\_PAIR(1)); // Черный текст на белом фоне

} else {

attron(COLOR\_PAIR(2)); // Белый текст на черном фоне

}

mvprintw(1, 0, "%\*s", max\_x, ""); // Заполняем всю вторую строку пробелами с цветовой парой

mvprintw(1, 0, "%-5s %-8s %-5s %-9s %-9s %-5s %-9s %s",

"PID", "USER", "STATE", "RES\_MEM", "VIRT\_MEM", "CORES", "START", "COMMAND");

// Отключаем цветовую пару после второй строки

if (color\_scheme == COLOR\_SCHEME\_INVERTED) {

attroff(COLOR\_PAIR(1));

} else {

attroff(COLOR\_PAIR(2));

}

}

void display\_process\_info(const ProcessInfo \*proc\_info) {

// Обрезаем имя пользователя до 8 символов

printw("%-5d %-8.8s %-5c %-6.1f MB %-5.1f MB %-5d %-9.8s ",

proc\_info->pid, proc\_info->user, proc\_info->state,

proc\_info->resident\_memory, proc\_info->virtual\_memory,

proc\_info->cpu\_cores, proc\_info->start\_time + 11);

// Обрезаем имя команды до символа '/'

char \*slash\_pos = strchr(proc\_info->command, '/');

if (slash\_pos) {

\*slash\_pos = '\0'; // Устанавливаем символ '/' как конец строки

}

printw("%s\n", proc\_info->command);

}

void update\_display(int start\_line, int total\_lines, ProcessData \*process\_data, int process\_count, DisplayMode mode, ColorScheme color\_scheme) {

// Сортировка данных перед отображением

switch (current\_sort) {

case SORT\_BY\_PID:

qsort(process\_data, process\_count, sizeof(ProcessData), compare\_by\_pid);

break;

case SORT\_BY\_RESIDENT\_MEMORY:

qsort(process\_data, process\_count, sizeof(ProcessData), compare\_by\_resident\_memory);

break;

case SORT\_BY\_VIRTUAL\_MEMORY:

qsort(process\_data, process\_count, sizeof(ProcessData), compare\_by\_virtual\_memory);

break;

}

clear(); // Очистка окна перед новым выводом

display\_header(color\_scheme);

int y = 2; // Текущая строка для вывода информации о процессе/потоке

int line\_count = 2; // Счетчик строк для вывода (учитываем строку заголовка)

for (int i = 0; i < process\_count && line\_count < total\_lines; i++) {

if (line\_count >= start\_line && y < LINES) {

move(y, 0);

if (color\_scheme == COLOR\_SCHEME\_INVERTED) {

attron(COLOR\_PAIR(2));

} else {

attron(COLOR\_PAIR(1));

}

display\_process\_info(&process\_data[i].process\_info);

y++;

}

line\_count++;

if (mode == SHOW\_PROCESSES\_AND\_THREADS) {

for (int j = 0; j < process\_data[i].thread\_count && line\_count < total\_lines; j++) {

if (line\_count >= start\_line && y < LINES) {

move(y, 0);

if (color\_scheme == COLOR\_SCHEME\_INVERTED) {

attron(COLOR\_PAIR(2));

} else {

attron(COLOR\_PAIR(1));

}

display\_thread\_info(&process\_data[i].threads[j]);

y++;

}

line\_count++;

}

}

}

refresh();

}

void display\_help(ColorScheme color\_scheme) {

clear(); // Очищаем окно перед выводом справки

// Определяем символы для заполнения пространства вокруг текста справки

char fill\_char = ' ';

if (color\_scheme == COLOR\_SCHEME\_INVERTED) {

fill\_char = '#'; // Для инвертированной схемы используем пробел

} else {

fill\_char = '#'; // Для стандартной схемы используем решетку

}

// Определяем цвет заполнения в зависимости от выбранной темы

int fill\_color\_pair = 1;

if (color\_scheme == COLOR\_SCHEME\_INVERTED) {

fill\_color\_pair = 2; // В светлой теме цвет черный

}

// Получаем размеры окна

int max\_y, max\_x;

getmaxyx(stdscr, max\_y, max\_x);

// Определяем координаты вывода для каждой строки

int y\_center = (max\_y / 2)-2; // Центральная координата по вертикали

int x\_cetner = max\_x / 2; // Центральная координата по горизонтали

// Вывод справки

mvprintw(y\_center - 3, x\_cetner - 6, "=== Help ===");

mvprintw(y\_center - 1, x\_cetner - 20, "Press 'p' to sort by PID (process ID)");

mvprintw(y\_center, x\_cetner - 20, "Press 'r' to sort by resident memory");

mvprintw(y\_center + 1, x\_cetner - 20, "Press 'v' to sort by virtual memory");

mvprintw(y\_center + 2, x\_cetner - 20, "Press 'k' to kill a process or thread");

mvprintw(y\_center + 3, x\_cetner - 20, "Press 'q' to quit");

// Добавленные строки

mvprintw(y\_center + 4, x\_cetner - 20, "Press 't' to show threads");

mvprintw(y\_center + 5, x\_cetner - 20, "Press 'key down' to go 20 lines down");

mvprintw(y\_center + 6, x\_cetner - 20, "Press 'key up' to go 20 lines up");

mvprintw(y\_center + 7, x\_cetner - 20, "Press 'z' to change color scheme");

// Пустая строка

mvprintw(y\_center + 8, x\_cetner - 20, "");

// Заполняем пространство вокруг текста справки

attron(COLOR\_PAIR(fill\_color\_pair)); // Устанавливаем цвет для заполнения

for (int i = 0; i < max\_y; i++) {

for (int j = 0; j < max\_x; j++) {

if (i < y\_center - 4 || i > y\_center + 8 || j < x\_cetner - 30 || j > x\_cetner + 30) {

mvaddch(i, j, fill\_char);

}

}

}

attroff(COLOR\_PAIR(fill\_color\_pair)); // Отключаем цвет заполнения

refresh();

// Ожидаем нажатия любой клавиши или истечения времени

timeout(30000); // Ожидание 30 секунд

getch(); // Ожидаем нажатия клавиши

clear(); // Очищаем экран после завершения отображения справки

refresh();

}

Файл display.h

#ifndef DISPLAY\_H

#define DISPLAY\_H

#include "display.h"

#include "processes.h"

#define MAX\_THREADS\_PER\_PROCESS 100 // Максимальное количество потоков на процесс

typedef enum {

SHOW\_PROCESSES,

SHOW\_PROCESSES\_AND\_THREADS

} DisplayMode;

typedef enum {

COLOR\_SCHEME\_DEFAULT,

COLOR\_SCHEME\_INVERTED

} ColorScheme;

void display\_help(ColorScheme color\_scheme);

void display\_thread\_info();

void display\_header(ColorScheme color\_scheme);

void display\_time(ColorScheme color\_scheme);

void display\_process\_info(const ProcessInfo \*proc\_info);

void update\_display(int start\_line, int total\_lines, ProcessData \*process\_data, int process\_count, DisplayMode mode, ColorScheme color\_scheme);

void kill\_process\_or\_thread();

#endif /\* DISPLAY\_H \*/

Файл main.c

#include <curses.h>

#include <unistd.h>

#include <stdlib.h>

#include <signal.h>

#include <dirent.h>

#include <string.h>

#include "processes.h"

#include "display.h"

#include "threads.h"

#include "control.h"

#include "read.h"

#define MAX\_PROCESSES 1000 // Максимальное количество процессов для отображения

#define REFRESH\_INTERVAL 1000 // Интервал обновления в миллисекундах

#define SCROLL\_LINES 20 // Количество строк для прокрутки по стрелкам

int main() {

initscr(); // Инициализация ncurses

start\_color(); // Включение поддержки цвета

noecho(); // Отключение эха вводимых символов

cbreak(); // Включение режима cbreak

keypad(stdscr, TRUE); // Включение поддержки клавиш

mousemask(ALL\_MOUSE\_EVENTS, NULL); // Включение обработки всех событий мыши

// Инициализация цветовых пар

init\_pair(1, COLOR\_WHITE, COLOR\_BLACK); // Стандартная цветовая схема (белый текст на черном фоне)

init\_pair(2, COLOR\_BLACK, COLOR\_WHITE); // Инвертированная цветовая схема (черный текст на белом фоне)

int start\_line = 0; // Первая видимая строка

int total\_lines = 0; // Общее количество строк для отображения

DisplayMode mode = SHOW\_PROCESSES; // Режим отображения по умолчанию

ColorScheme color\_scheme = COLOR\_SCHEME\_DEFAULT; // Цветовая схема по умолчанию

while (1) {

DIR \*dir;

struct dirent \*entry;

dir = opendir("/proc");

if (!dir) {

perror("opendir(/proc)");

endwin();

return 1;

}

ProcessData process\_data[MAX\_PROCESSES]; // Сохранение информации о процессах и потоках

int process\_count = 0; // Обнуляем количество процессов

total\_lines = 1; // Обнуляем количество строк (учитываем строку заголовка)

while ((entry = readdir(dir)) != NULL) {

if (entry->d\_type == DT\_DIR) { // Проверяем тип элемента и его идентификатор

int pid = atoi(entry->d\_name); // Преобразуем имя каталога в целочисленный PID

if (pid > 0 && process\_count < MAX\_PROCESSES) { // Проверяем, что PID положительный и количество процессов еще не достигло максимума

ProcessInfo proc\_info;

get\_process\_info(&proc\_info, pid); // Получаем информацию о процессе по его PID

process\_data[process\_count].process\_info = proc\_info; // Сохраняем информацию о процессе в массиве process\_data

process\_data[process\_count].thread\_count = 0; // Обнуляем счетчик потоков для данного процесса

total\_lines++; // Увеличиваем общее количество строк для отображения

if (mode == SHOW\_PROCESSES\_AND\_THREADS) { // Если режим отображения включает информацию о потоках, получаем ее

char path[256];

DIR \*task\_dir;

struct dirent \*task\_entry;

snprintf(path, sizeof(path), "/proc/%d/task", proc\_info.pid); // Формируем путь к каталогу с потоками процесса

task\_dir = opendir(path); // Открываем каталог

if (task\_dir) {

while ((task\_entry = readdir(task\_dir)) != NULL) { // Перебираем все элементы в каталоге потоков

// Проверяем тип элемента и его имя

if (task\_entry->d\_type == DT\_DIR && task\_entry->d\_name[0] !=

'.' && process\_data[process\_count].thread\_count < MAX\_THREADS\_PER\_PROCESS) {

int tid = atoi(task\_entry->d\_name); // Преобразуем имя каталога в целочисленный TID

ThreadInfo thread\_info;

get\_thread\_info(&thread\_info, proc\_info.pid, tid); // Получаем информацию о потоке по его PID и TID

process\_data[process\_count].threads[process\_data[process\_count].thread\_count] = thread\_info;

process\_data[process\_count].thread\_count++; // Увеличиваем счетчик потоков

total\_lines++; // Увеличиваем общее количество строк для отображения

}

}

closedir(task\_dir);

}

}

process\_count++;

}

}

}

closedir(dir);

update\_display(start\_line, total\_lines, process\_data, process\_count, mode, color\_scheme);

timeout(REFRESH\_INTERVAL); // Устанавливаем таймаут ожидания ввода

int ch = getch();

if (ch == KEY\_MOUSE) {

MEVENT event;

if (getmouse(&event) == OK) {

if (event.bstate & BUTTON4\_PRESSED) { // Прокрутка вверх

if (start\_line > 0) start\_line--;

} else if (event.bstate & BUTTON5\_PRESSED) { // Прокрутка вниз

// Увеличиваем начальную строку, если это возможно

if (start\_line < total\_lines - (LINES - 1)) start\_line++;

}

update\_display(start\_line, total\_lines, process\_data, process\_count, mode, color\_scheme);

}

} else if (ch == KEY\_UP) {

if (start\_line > SCROLL\_LINES) {

start\_line -= SCROLL\_LINES;

} else {

start\_line = 0; // Сбрасываем начало строки при переключении режима

}

update\_display(start\_line, total\_lines, process\_data, process\_count, mode, color\_scheme);

} else if (ch == KEY\_DOWN) {

if (start\_line + SCROLL\_LINES < total\_lines - (LINES - 1)) {

start\_line += SCROLL\_LINES;

} else {

start\_line = total\_lines - (LINES - 1);

if (start\_line < 0) start\_line = 0;

}

update\_display(start\_line, total\_lines, process\_data, process\_count, mode, color\_scheme);

} else if (ch == 't') {

if (mode == SHOW\_PROCESSES) {

mode = SHOW\_PROCESSES\_AND\_THREADS;

} else {

mode = SHOW\_PROCESSES;

}

start\_line = 0; // Сбрасываем начало строки при переключении режима

update\_display(start\_line, total\_lines, process\_data, process\_count, mode, color\_scheme);

} else if (ch == 'h' || ch == 'H') {

display\_help(color\_scheme);

}else if (ch == 'z' || ch == 'Z') {

if (color\_scheme == COLOR\_SCHEME\_DEFAULT) {

// Включение инвертированной цветовой схемы для всего окна

wbkgd(stdscr, COLOR\_PAIR(2));

color\_scheme = COLOR\_SCHEME\_INVERTED;

} else {

// Включение стандартной цветовой схемы для всего окна

wbkgd(stdscr, COLOR\_PAIR(1));

color\_scheme = COLOR\_SCHEME\_DEFAULT;

}

start\_line = 0; // Сбрасываем начало строки при переключении цветовой схемы

update\_display(start\_line, total\_lines, process\_data, process\_count, mode, color\_scheme);

} else if (ch == 'q' || ch == 'Q') {

break; // Завершение программы при нажатии 'q'

} else if (ch == ERR) {

// Таймаут достигнут, обновляем экран

update\_display(start\_line, total\_lines, process\_data, process\_count, mode, color\_scheme);

}

handle\_user\_input(ch);

update\_display(start\_line, total\_lines, process\_data, process\_count, mode, color\_scheme);

}

endwin(); // Завершение работы ncurses

return 0;

}

Файл processes.h

#ifndef PROCESSES\_H

#define PROCESSES\_H

#include "threads.h"

#define MAX\_THREADS\_PER\_PROCESS 100

typedef struct {

int pid; // ID процесса

char user[50]; // Имя пользователя вызвавшего процесс

char state; // Состояние процесса

double resident\_memory; // Потребление физической памяти (MB)

double virtual\_memory; // Виртуальная память, используемая процессом (MB)

int cpu\_cores; // Количество ядер процессора, используемых процессом

int threads; // Количество потоков

char start\_time[20]; // Дата и время запуска

char command[100]; // Имя команды, запустившей процесс

} ProcessInfo;

typedef struct {

ProcessInfo process\_info; // Информация о процессе

ThreadInfo threads[MAX\_THREADS\_PER\_PROCESS]; // Информация о потоках процесса

int thread\_count; // Количество потоков процесса

} ProcessData; // Структура для хранения информации о процессе и его потоках

int compare\_by\_pid(const void \*a, const void \*b);

int compare\_by\_resident\_memory(const void \*a, const void \*b);

int compare\_by\_virtual\_memory(const void \*a, const void \*b);

#endif // PROCESSES\_H

Файл read.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <pwd.h>

#include <time.h>

#include "processes.h"

#include "threads.h"

void get\_thread\_info(ThreadInfo \*thread\_info, int pid, int tid) {

char path[256];

FILE \*file;

thread\_info->tid = tid;

snprintf(path, sizeof(path), "/proc/%d/task/%d/comm", pid, tid);

file = fopen(path, "r");

if (file) {

if (fgets(thread\_info->name, sizeof(thread\_info->name), file)) {

thread\_info->name[strcspn(thread\_info->name, "\n")] = '\0';

}

fclose(file);

}

snprintf(path, sizeof(path), "/proc/%d/task/%d/status", pid, tid);

file = fopen(path, "r");

if (file) {

char buffer[256];

while (fgets(buffer, sizeof(buffer), file)) {

if (strncmp("State:", buffer, 6) == 0) {

sscanf(buffer, "State: %c", &thread\_info->state);

break;

}

}

fclose(file);

}

}

// Функция для подсчета количества ядер

int count\_cpu\_cores(const char \*cpu\_list) {

int cores = 1;

for (const char \*p = cpu\_list; \*p; p++) {

if (\*p == ',') {

cores++;

}

}

return cores;

}

// Функция для чтения времени запуска системы

time\_t get\_system\_uptime() {

FILE \*file = fopen("/proc/uptime", "r");

if (!file) {

perror("Failed to open /proc/uptime");

return 0;

}

double uptime;

fscanf(file, "%lf", &uptime);

fclose(file);

return (time\_t)uptime;

}

// Функция для чтения информации о процессе

void get\_process\_info(ProcessInfo \*proc\_info, int pid) {

char path[256];

char buffer[256];

FILE \*file;

struct passwd \*pw;

// Заполняем PID

proc\_info->pid = pid;

// Чтение информации из /proc/[pid]/status для получения имени пользователя, состояния процесса, виртуальной памяти и команды

snprintf(path, sizeof(path), "/proc/%d/status", pid);

file = fopen(path, "r");

if (file) {

while (fgets(buffer, sizeof(buffer), file)) {

if (strncmp(buffer, "Uid:", 4) == 0) {

int uid;

sscanf(buffer, "Uid: %d", &uid);

pw = getpwuid(uid);

if (pw) {

strncpy(proc\_info->user, pw->pw\_name, sizeof(proc\_info->user) - 1);

}

} else if (strncmp(buffer, "VmSize:", 7) == 0) {

unsigned long vm\_size;

sscanf(buffer, "VmSize: %lu kB", &vm\_size);

proc\_info->virtual\_memory = vm\_size / 1024.0 / 1024.0; // Конвертируем в MB

} else if (strncmp(buffer, "State:", 6) == 0) {

sscanf(buffer, "State: %c", &proc\_info->state);

} else if (strncmp(buffer, "Name:", 5) == 0) {

sscanf(buffer, "Name: %s", proc\_info->command);

} else if (strncmp(buffer, "Cpus\_allowed\_list:", 18) == 0) {

// Здесь считываем информацию о привязке к ядрам

char cpu\_list[256];

sscanf(buffer, "Cpus\_allowed\_list: %s", cpu\_list);

proc\_info->cpu\_cores = count\_cpu\_cores(cpu\_list); // Подсчитываем количество ядер

} else if (strncmp(buffer, "Threads:", 8) == 0) {

sscanf(buffer, "Threads: %d", &proc\_info->threads);

}

}

fclose(file);

}

// Чтение информации из /proc/[pid]/stat для получения потребления физической памяти и времени запуска

snprintf(path, sizeof(path), "/proc/%d/stat", pid);

file = fopen(path, "r");

if (file) {

long rss;

unsigned long utime, stime, starttime;

fscanf(file, "%\*d %\*s %\*c %\*d %\*d %\*d %\*d %\*d %\*d %\*u %\*u %\*u %\*u %lu %lu %\*d %\*d %\*d %\*d %llu", &utime, &stime, &starttime);

fclose(file);

// Получаем текущее время в секундах

time\_t now = time(NULL);

// Время работы системы в секундах

time\_t uptime = get\_system\_uptime();

// Время запуска процесса (текущее время - (время работы системы - время старта процесса))

time\_t start\_time = now - (uptime - (starttime / sysconf(\_SC\_CLK\_TCK)));

struct tm start\_tm;

localtime\_r(&start\_time, &start\_tm);

strftime(proc\_info->start\_time, sizeof(proc\_info->start\_time), "%Y-%m-%d %H:%M:%S", &start\_tm);

snprintf(path, sizeof(path), "/proc/%d/statm", pid);

file = fopen(path, "r");

if (file) {

fscanf(file, "%\*d %ld", &rss);

fclose(file);

proc\_info->resident\_memory = rss \* (sysconf(\_SC\_PAGESIZE) / 1024.0 / 1024.0); // Конвертируем в MB

}

}

}

Файл read.h

#ifndef READ\_H

#define READ\_H

#include "processes.h"

#include "threads.h"

void get\_thread\_info(ThreadInfo \*thread\_info, int pid, int tid);

void get\_process\_info(ProcessInfo \*proc\_info, int pid);

#endif // READ\_H

Файл threads.h

#ifndef THREADS\_H

#define THREADS\_H

typedef struct {

int tid; // ID потока

char state; // Сстояние потока

char name[16]; // Имя потока

} ThreadInfo;

#endif //THREADS\_H

**ПРИЛОЖЕНИЕ Д**

(Обязательное)

Ведомость документов