

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе  
на тему

ДИСПЕТЧЕР ПРОЦЕССОВ И ПОТОКОВ

БГУИР КР 1-40 02 01 118 ПЗ

Студент

Д.А. Снитко

Руководитель

А.О. Игнатович

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: «Диспетчер процессов и потоков»
- 2 Срок сдачи студентом законченного проекта: 10 мая 2024 г.
- 3 Исходные данные к проекту: нет.
- 4 Содержание пояснительной записки (перечень подлежащих разработке вопросов):
  - Титульный лист.
  - Реферат.
  - Введение.
  - 1. Обзор литературы.
  - 2. Системное проектирование.
  - 3. Функциональное проектирование.
  - 4. Разработка программных модулей.
  - 5. Руководство пользователя.
  - Заключение.
  - Список использованных источников.
  - Приложения.
- 5 Перечень графического материала (с точным указанием обязательных чертежей):
  - 5.1 Структурная схема.
  - 5.2 Схема алгоритма `switch_color_mode`
  - 5.3 Схема алгоритма `kill_process_by_pid`
  - 5.4 Ведомость документов

## КАЛЕНДАРНЫЙ ПЛАН

| Наименование этапов<br>курсового проекта                        | Объем<br>этапа,<br>% | Срок<br>выполнения<br>этапа | Примечания               |
|---|----------------------|-----------------------------|--------------------------|
| Выбор темы курсового проекта                                    | 5                    | 17.02 –<br>01.03            |                          |
| Начальный этап ПЗ   | 30                   | 01.03 –<br>01.04            |                          |
| Основная часть кода   | 50                   | 01.04 –<br>01.05            |                          |
| Оформление пояснительной<br>записки и графического<br>материала | 15                   | 01.05 –<br>10.05            | с выполнением<br>чертежа |
| Защита курсового проекта  |                      | 28.05 –<br>10.06            |                          |

Дата выдачи задания: 20.02.2024 г.

Руководитель

А.О. Игнатович

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ

\_\_\_\_\_

## СОДЕРЖАНИЕ

|   |    |
|---|----|
| ВВЕДЕНИЕ .....  | 5  |
| 1 ОБЗОР ЛИТЕРАТУРЫ .....  | 6  |
| 1.1 Основные шаги разработки диспетчера процессов и потоков ..... | 6  |
| 1.2 Постановка задачи.....  | 7  |
| 1.3 Обзор существующих аналогов.....                              | 7  |
| 1.4 Сравнительный анализ.....                                     | 22 |
| 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ.....                                   | 24 |
| 2.1 Блок ввода-вывода .....                                       | 25 |
| 2.2 Блок чтения данных .....                                      | 25 |
| 2.3 Блок сортировки .....   | 25 |
| 2.4 Блок управления процессами и потоками .....                   | 25 |
| 2.5 Блок обработки сигналов.....                                  | 25 |
| 2.6 Блок главного цикла программы .....                           | 26 |
| 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ .....                             | 27 |
| 3.1 Описание основных структур данных программы.....              | 27 |
| 3.2 Описание основных функций программы.....                      | 28 |
| 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ.....                             | 32 |
| 4.1 Разработка структурной схемы.....                             | 32 |
| 4.2 Схемы алгоритмов .....  | 32 |
| 4.3 Разработка алгоритмов .....                                   | 32 |
| 4.3.1 Алгоритм функции read_sysinfo .....                         | 32 |
| 4.3.2 Алгоритм функции read_processes .....                       | 33 |
| 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ .....                                  | 35 |
| 5.1 Требования к программному и аппаратному обеспечению .....     | 35 |
| 5.2 Руководство по использованию.....                             | 00 |
| 5.3 Код программы.....  | 00 |
| ЗАКЛЮЧЕНИЕ.....   | 00 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....                             | 00 |
| ПРИЛОЖЕНИЕ А .....  | 00 |
| ПРИЛОЖЕНИЕ Б .....  | 00 |
| ПРИЛОЖЕНИЕ В.....   | 00 |
| ПРИЛОЖЕНИЕ Г .....  | 00 |
| ПРИЛОЖЕНИЕ Д.....   | 00 |

## ВВЕДЕНИЕ

Диспетчеры процессов и потоков – компоненты, утилиты операционных системы, предназначенные для управления выполнением задач и системными ресурсами, и мониторингом. В Unix-подобных системах, таких как Linux, процессы и потоки играют особую роль.

Процесс в Unix-системе - это запущенная программа или программный код с собственной областью памяти и состоянием. Каждый процесс имеет уникальный идентификатор процесса PID (Process Identifier) и может быть однозначно идентифицирован в системе. Каждый процесс может быть создан другим процессом, называемым родительским процессом. Таким образом, процессы могут образовывать иерархическую структуру.

С другой стороны, потоки - это легкие единицы выполнения внутри процесса. Поток - наборы инструкций, которые выполняются независимо друг от друга в контексте процесса. Потоки в одном процессе имеют общую область памяти и могут обмениваться данными и ресурсами. У них есть свой собственный идентификатор TID (Thread Identifier), который помогает системе управлять потоками. Идентификатор потока (TID) - это целое число, которое операционная система присваивает каждому потоку в процессе. Когда создается поток, операционная система присваивает ему уникальный TID, который остается неизменным на протяжении всего времени существования потока.

Менеджер процессов и потоков, как и утилита Top, предоставляет пользователю информацию о текущих процессах и потоках в системе. Он отображает список запущенных процессов с различными характеристиками, такими как PID, имя пользователя, приоритет, использование ресурсов оперативной памяти и процессора и другие параметры. Эта информация позволяет пользователям отслеживать активность процессов, определять их важность и эффективность использования ресурсов.

Менеджер отображает список процессов, запущенных в системе. Каждый процесс представлен отдельной строкой и содержит информацию о его идентификаторе (PID), пользователе, использовании процессора, памяти и других параметрах. Информация об использовании системных ресурсов, ЦП и памяти.

Целью курсового проекта является разработка диспетчера процессов и потоков для Unix-подобных систем, таких как Linux. Диспетчер должен предоставлять пользователю информацию о текущих процессах и потоках в системе включая PID, имя пользователя, приоритет, использование ресурсов оперативной памяти и процессора и другие параметры. Пользователи должны иметь возможность завершать, сортировать и просматривать активные процессы и потоки. А также для просматривать общую информацию о системе: текущее время, количество пользователей, процессов зомби, среднюю загрузженность системы за разное время, процент использования процессора пользователями, процент использования процессора системой и прочую информацию.

# 1 ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Основные шаги разработки диспетчера процессов и потоков

Проектирование структуры данных: Для хранения информации о процессах и потоках необходимо создать соответствующие структуры данных. В нашем случае, это структуры `Process` и `Thread`. Они должны содержать необходимую информацию, такую как идентификатор (PID или TID), имя пользователя, приоритет, использование виртуальной и резидентной памяти, использование процессора и команду, запустившую процесс.

Чтение данных о процессах и потоках: Для получения информации о процессах и потоках необходимо прочитать данные из системных файлов. В Unix-подобных системах, таких как Linux, эта информация хранится в файлах в каталоге `/proc`. Функции `read_processes` и `read_threads` отвечают за чтение этих данных и заполнение структур `Process` и `Thread`. Для работы с каталогами используются системные вызовы `opendir`, `readdir`, `closedir`, а для работы с файлами - `fopen`, `fgets`, `sscanf`, `fclose`.

Обработка ввода пользователя: Для взаимодействия с пользователем необходимо реализовать обработку ввода. Функция `handle_input` отвечает за это. Она обрабатывает нажатия клавиш и выполняет соответствующие действия, такие как переключение между режимами отображения процессов и потоков, сортировка по разным критериям, убийство процессов и потоков и т.д. Для обработки ввода с клавиатуры используются системные вызовы `getchar` и `kbhit`.

Отображение информации: Для отображения информации о процессах и потоках необходимо реализовать функции `display_processes` и `display_threads`. Они выводят информацию в табличном виде, используя данные из структур `Process` и `Thread`. Для ввода-вывода информации в консоль используются системные вызовы `printf` и `scanf`, а для форматированного вывода строк - `sprintf`.

Сортировка: Для сортировки процессов и потоков по разным критериям необходимо реализовать соответствующие функции. В нашем случае, это функции `sort_processes_by_` и `sort_threads_by_`. Они используют стандартную функцию `qsort` для сортировки массивов структур. Для сравнения критериев сортировки, введенных пользователем, используется системный вызов `strcmp`.

Завершение процессов и потоков: Для завершения процессов и потоков необходимо реализовать функции `kill_process_by_pid` и `kill_thread_by_tid`. Они используют системные вызовы `kill` и `pthread_cancel` для отправки сигнала завершения процессу или потоку.

Главный цикл программы: Главный цикл программы выполняет следующие действия: очистка экрана, чтение данных о процессах и потоках, отображение информации, ожидание 1 секунда и обработка ввода пользователя. Этот цикл повторяется бесконечно, пока программа не будет завершена. Для выполнения системных команд используется системный вызов `system`, а для приостановки выполнения программы на указанное количество секунд - `sleep`.

Обработка сигналов: Для корректного завершения программы необходимо обрабатывать сигнал SIGINT, который генерируется при нажатии клавиш Ctrl+C. Функция `handle_signal` отвечает за это. Она выводит сообщение о завершении программы и вызывает функцию `exit` для завершения программы. Для установки обработчика сигнала используется системный вызов `signal`.

## **1.2 Постановка задачи**

В рамках данного проекта будет разработан диспетчер процессов и потоков, обладающий функциональностью мониторинга процессов и управлением, запущенных в данный момент. Диспетчер должен предоставлять информацию о PID (идентификатор процесса), пользователе, приоритете, потреблении виртуальной и физической памяти, % времени процессора, % ОЗУ используемым процессором, название команды, инициализировавшей процесс. Для реализации данного диспетчера будет использован язык программирования высокого уровня, такой как Си. В качестве операционной системы была выбрана Fedora Workstation 39.

## **1.3 Обзор существующих аналогов**

Существует множество программ для мониторинга и управления процессами в операционных системах. Два из наиболее известных и широко используемых аналогов программы диспетчера процессов и потоков, разработанной в рамках данного проекта, это утилита `top` и программа `htop`.

### **1.3.1 top**

`Top` (table of processes) является стандартной утилитой Unix-подобных операционных систем для мониторинга процессов. Она предоставляет пользователю динамическое представление о текущем состоянии процессов в системе, включая их идентификаторы, использование ресурсов, приоритет, статус и т.д.

Программа `top` периодически обновляет информацию о процессах, сортирует ее в соответствии с выбранным критерием (по умолчанию - использование процессора) и выводит ее на экран. Пользователь может взаимодействовать с программой, используя различные команды для сортировки, фильтрации и управления процессами. Например, можно отправить сигнал для завершения процесса или изменить приоритет процесса.

`Top` широко используется системными администраторами и разработчиками для мониторинга и диагностики производительности системы. Однако, несмотря на ее популярность, у программы есть некоторые недостатки, такие как отсутствие графического интерфейса и неудобство использования для новичков.

```

top - 16:54:41 up 5:12, 2 users, load average: 0.75, 0.77, 0.35
Tasks: 116 total, 1 running, 115 sleeping, 0 stopped, 0 zombie
Cpu(s): 5.3%us, 2.7%sy, 0.0%ni, 91.7%id, 0.3%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 775540k total, 758548k used, 16992k free, 13920k buffers
Swap: 787144k total, 34724k used, 752420k free, 443552k cached

```

| PID  | USER     | PR | NI | VIRT  | RES  | SHR  | S | %CPU | %MEM | TIME+    | COMMAND      |
|------|----------|----|----|-------|------|------|---|------|------|----------|--------------|
| 6938 | funalien | 15 | 0  | 70012 | 29m  | 18m  | S | 4.0  | 3.9  | 10:40.43 | ktorrent     |
| 5375 | root     | 15 | 0  | 79060 | 55m  | 6616 | S | 2.3  | 7.3  | 4:45.84  | Xorg         |
| 7869 | funalien | 15 | 0  | 30400 | 15m  | 13m  | S | 1.0  | 2.0  | 0:00.99  | ksnapshot    |
| 5600 | funalien | 18 | 0  | 15252 | 9700 | 4528 | S | 0.3  | 1.3  | 0:19.03  | pypanel      |
| 5605 | funalien | 15 | 0  | 9704  | 3592 | 2968 | S | 0.3  | 0.5  | 1:20.99  | conky        |
| 7802 | funalien | 15 | 0  | 228m  | 75m  | 23m  | S | 0.3  | 9.9  | 0:36.56  | firefox-bin  |
| 1    | root     | 15 | 0  | 2952  | 1852 | 532  | S | 0.0  | 0.2  | 0:01.33  | init         |
| 2    | root     | 11 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00  | kthreadd     |
| 3    | root     | RT | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00  | migration/0  |
| 4    | root     | 34 | 19 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.10  | ksoftirqd/0  |
| 5    | root     | RT | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00  | watchdog/0   |
| 6    | root     | 10 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.16  | events/0     |
| 7    | root     | 10 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00  | khelper      |
| 26   | root     | 12 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00  | kblockd/0    |
| 27   | root     | 20 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00  | kacpid       |
| 28   | root     | 20 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00  | kacpi_notify |
| 108  | root     | 10 | -5 | 0     | 0    | 0    | S | 0.0  | 0.0  | 0:00.00  | kseriod      |

Рисунок 1.1 – Интерфейс утилиты top

### 1.3.2 htop

Нtop - это усовершенствованная версия программы top, которая предоставляет более удобный и функциональный интерфейс для мониторинга процессов. В отличие от top, htop использует графический интерфейс, позволяющий пользователю просматривать список процессов в виде таблицы, сортировать их по различным критериям, фильтровать по имени или идентификатору, а также управлять ими с помощью мыши или клавиатуры.

Программа htop предоставляет более подробную информацию о процессах, включая использование ресурсов в реальном времени, графики использования процессора и памяти, а также информацию о загрузке системы. Кроме того, htop позволяет пользователю отправлять сигналы процессам, менять их приоритет, завершать или замораживать процессы, а также выполнять другие операции управления.

Нtop является более удобным и функциональным инструментом для мониторинга процессов, чем top, и широко используется системными администраторами и разработчиками. Однако, несмотря на свои преимущества, htop также имеет некоторые недостатки, такие как более высокие требования к ресурсам системы и невозможность работы в текстовом режиме.



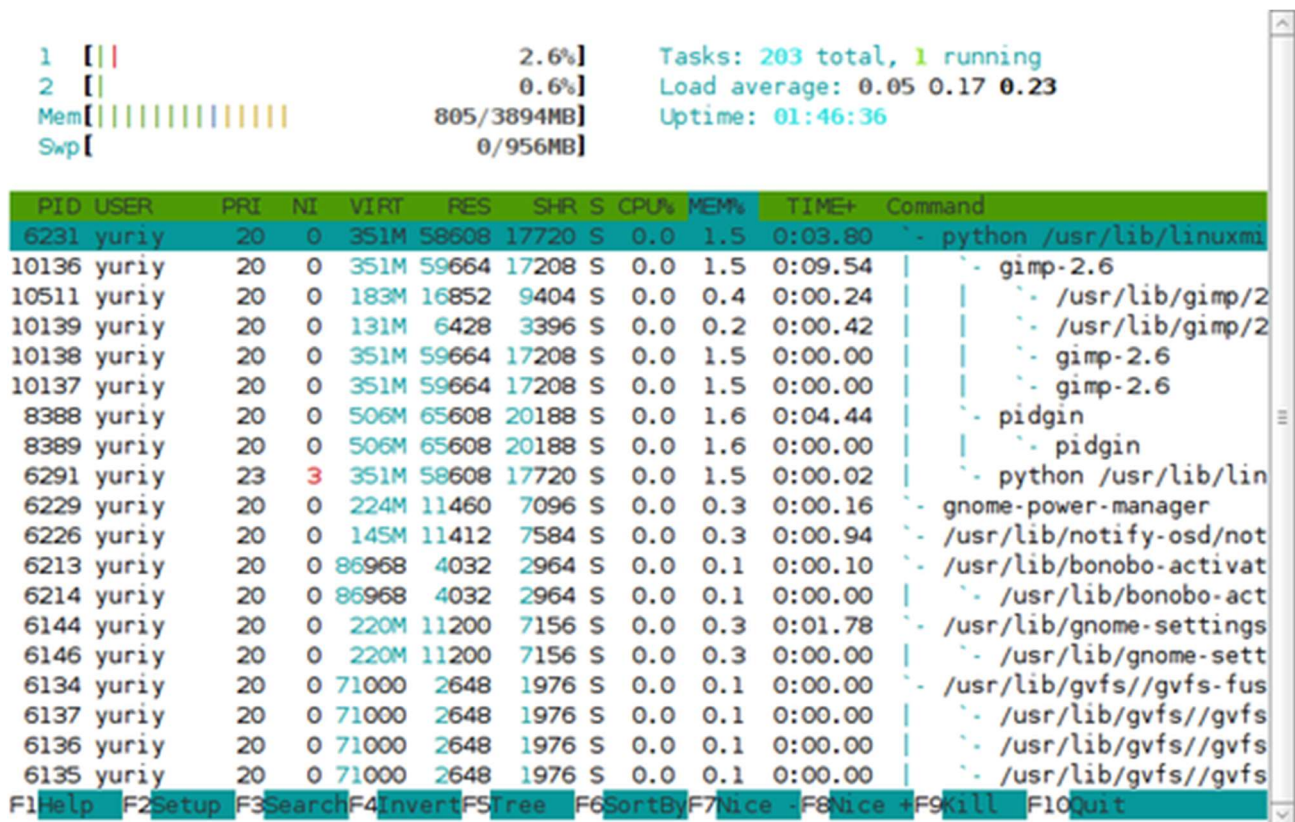


Рисунок 1.2 – Интерфейс утилиты htop

### 1.3.3 atop

Atop имеет два режима работы — сбор статистики и наблюдение за системой в реальном времени. В режиме сбора статистики atop запускается как демон и раз в N времени (обычно 10 мин) скидывает состояние в двоичный журнал. Потом по этому журналу atop'ом же (ключ -r и имя лог-файла) можно бегать вперёд-назад кнопками T и t, наблюдая показания atop'a с усреднением за 10 минут в любой интересный момент времени.

В отличие от top отлично знает про существование блочных устройств и сетевых интерфейсов, способен показывать их загрузку в процентах (на 10G, правда, процентов не получается, но хотя бы показывается количество мегабит).

Незаменимое средство для поиска источников лагов на сервере, так как сохраняет не только статистику загрузки системы, но и показатели каждого процесса — то есть «долистав» до нужного момента времени можно увидеть, кто этот счастливый момент с LA > 30 создал. И что именно было причиной — IO программ, своп (нехватка памяти), процессор или что-то ещё. Помимо большого количества информации ещё способен двумя цветами подсказывать, какие параметры выходят за разумные пределы.

|       |           |        |       |          |       |         |       |          |        |          |       |
|-------|-----------|--------|-------|----------|-------|---------|-------|----------|--------|----------|-------|
| cpu   | sys       | 0%     | user  | 0%       | irq   | 0%      | idle  | 100%     | cpu005 | w        | 0%    |
| cpu   | sys       | 0%     | user  | 0%       | irq   | 0%      | idle  | 100%     | cpu015 | w        | 0%    |
| cpu   | sys       | 1%     | user  | 0%       | irq   | 0%      | idle  | 82%      | cpu011 | w        | 17%   |
| cpu   | sys       | 0%     | user  | 0%       | irq   | 0%      | idle  | 100%     | cpu004 | w        | 0%    |
| cpu   | sys       | 0%     | user  | 0%       | irq   | 0%      | idle  | 100%     | cpu007 | w        | 0%    |
| cpu   | sys       | 0%     | user  | 0%       | irq   | 0%      | idle  | 100%     | cpu012 | w        | 0%    |
| CPL   | avg1      | 4.51   | avg5  | 5.31     | avg15 | 4.77    | csw   | 12122271 | intr   | 2640151  |       |
| MEM   | tot       | 70.9G  | free  | 44.9G    | cache | 61.3M   | buff  | 24.3G    | slab   | 847.7M   |       |
| SWP   | tot       | 7.4G   | free  | 7.4G     |       |         | vmcom | 1.9G     | vmlim  | 42.9G    |       |
| DSK   |           | sdc    | busy  | 80%      | read  | 232646  | write | 76155    | avio   | 1 ms     |       |
| DSK   |           | sdb    | busy  | 69%      | read  | 124651  | write | 147480   | avio   | 1 ms     |       |
| DSK   |           | sdd    | busy  | 13%      | read  | 43      | write | 60369    | avio   | 1 ms     |       |
| DSK   |           | sda    | busy  | 4%       | read  | 845     | write | 12592    | avio   | 1 ms     |       |
| NET   | transport |        | tcp_i | 11399e3  | tcp_o | 4895002 | udp_i | 356      | udp_o  | 356      |       |
| NET   | network   |        | ip_i  | 11399548 | ip_o  | 4895538 | ipfrw | 64       | deliv  | 1140e4   |       |
| NET   | eth1      | 0%     | pck_i | 7333     | pck_o | 5209    | si    | 9 Kbps   | so     | 59 Kbps  |       |
| NET   | eth3      | ----   | pck_i | 11395e3  | pck_o | 8104798 | si    | 268 Mbps | so     | 123 Mbps |       |
| NET   | lo        | ----   | pck_i | 694      | pck_o | 694     | si    | 1 Kbps   | so     | 1 Kbps   |       |
|       |           |        |       |          |       |         |       |          |        |          |       |
| PID   | SYSCPU    | USRCPU | VGROW | RGROW    | RDDSK | WRDSK   | ST    | EXC      | S      | CPU      | CMD   |
| 20267 | 1m44s     | 0.00s  | OK    | OK       | OK    | OK      | --    | -        | R      | 17%      | istd1 |
| 6170  | 0.57s     | 38.65s | OK    | OK       | OK    | OK      | --    | -        | S      | 6%       | java  |

Рисунок 1.3 – Интерфейс утилиты atop

### 1.3.4 iotop

Iotop является утилитой для мониторинга дисковой активности в системах Linux. Эта утилита показывает, какие процессы в настоящее время выполняют ввод-вывод с диском, сколько байт они читают или записывают, а также другие полезные сведения.

Iotop похож на утилиту top, которая используется для мониторинга использования процессора и памяти, но вместо этого концентрируется на дисковой активности. Это может быть полезно при диагностике проблем с производительностью, вызванных избыточной нагрузкой на диск, или при идентификации процессов, которые выполняют слишком много операций ввода-вывода.

Iotop может работать в двух режимах: батчевом и интерактивном. В батчевом режиме iotop выводит информацию о дисковой активности один раз и завершает работу, тогда как в интерактивном режиме он обновляет информацию в реальном времени.

Некоторые из ключевых параметров, которые можно использовать с iotop, включают:

- \* -o: отсортировать вывод по указанному столбцу.
- \* -p: мониторить только указанные процессы.
- \* -q: запустить iotop в бесшумном режиме, без вывода статистики по умолчанию.
- \* -t: отображать время простоя для каждого процесса.

Iotop является очень полезным инструментом для администрирования систем Linux, и его использование может помочь выявить и устранить проблемы с производительностью, связанные с дисковой активностью.

| Total DISK READ: 24.84 M/s   Total DISK WRITE: 22.61 M/s |      |        |            |            |        |         |                      |
|--|------|--------|------------|------------|--------|---------|----------------------|
| TID  | PRI  | USER   | DISK READ  | DISK WRITE | SWAPIN | IO>     | COMMAND              |
| 24310  | be/4 | root   | 249.73 K/s | 0.00 B/s   | 0.00 % | 11.67 % | pvmove /dev/sdb1     |
| 1266   | be/4 | root   | 3.87 K/s   | 0.00 B/s   | 0.00 % | 6.74 %  | [k.journald]         |
| 3027   | be/4 | amarao | 278.77 K/s | 0.00 B/s   | 0.00 % | 0.90 %  | python /u~deluge-gtk |
| 1268   | be/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.08 %  | [k.journald]         |
| 11871  | be/4 | amarao | 0.00 B/s   | 7.74 K/s   | 0.00 % | 0.00 %  | gnome-terminal       |
| 24314  | be/4 | root   | 22.50 M/s  | 0.00 B/s   | 0.00 % | 0.00 %  | [kcopyd]             |
| 1  | be/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | init [2]             |
| 2  | be/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [kthreadd]           |
| 3  | rt/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [migration/0]        |
| 4  | be/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [ksoftirqd/0]        |
| 5  | rt/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [watchdog/0]         |
| 6  | rt/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [migration/1]        |
| 7  | be/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [ksoftirqd/1]        |
| 8  | rt/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [watchdog/1]         |
| 9  | rt/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [migration/2]        |
| 10   | be/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [ksoftirqd/2]        |
| 11   | rt/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [watchdog/2]         |
| 12   | rt/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [migration/3]        |
| 13   | be/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [ksoftirqd/3]        |
| 14   | rt/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [watchdog/3]         |
| 15   | be/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [events/0]           |
| 16   | be/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [events/1]           |
| 17   | be/4 | root   | 0.00 B/s   | 0.00 B/s   | 0.00 % | 0.00 %  | [events/2]           |

Рисунок 1.4 – Интерфейс утилиты iotop

### 1.3.5 iftop

Iftop - это утилита командной строки для мониторинга трафика сети в реальном времени. Эта программа отображает количество передаваемых и принимаемых байтов для каждого сетевого соединения и позволяет отслеживать использование пропускной способности сети.

Iftop работает путем анализа пакетов, проходящих через сетевой интерфейс, и отображает информацию о них в удобном для чтения формате. По умолчанию, iftop сортирует соединения по скорости передачи данных, но также можно использовать различные параметры для сортировки по другим критериям, таким как использование пропускной способности, количество пакетов и т.д.

Некоторые из ключевых параметров, которые можно использовать с iftop, включают:

- i: выбрать сетевой интерфейс для мониторинга.
- f: использовать фильтр пакетов для отображения только определенных соединений.
- n: отображать IP-адреса вместо имен хостов.



-p: указать порт для мониторинга.  
 -B: отображать скорость передачи данных в битах в секунду вместо байт.  
 Iftop является очень полезным инструментом для диагностики проблем с сетью, таких как перегрузка пропускной способности, несанкционированное использование сети и т.д. Эта утилита доступна для большинства дистрибутивов Linux и может быть установлена с помощью стандартного менеджера пакетов.

|               | 19.1Mb                       | 38.1Mb        | 57.2Mb       | 76.3Mb        | 95.4Mb        |               |               |               |
|---------------|------------------------------|---------------|--------------|---------------|---------------|---------------|---------------|---------------|
| desunote.ru   | => CPE-58-161-224-211.iqla1. | 0b            | 6.61Mb       | 5.36Mb        |               |               |               |               |
|               | <=                           | 0b            | 75.3Kb       | 65.4Kb        |               |               |               |               |
| desunote.ru   | => 254.134.159.110.tm-hsbb.t | 2.62Mb        | 4.14Mb       | 3.68Mb        |               |               |               |               |
|               | <=                           | 49.3Kb        | 100Kb        | 85.6Kb        |               |               |               |               |
| desunote.ru   | => 74.101.47.138             | 3.87Mb        | 3.29Mb       | 3.33Mb        |               |               |               |               |
|               | <=                           | 96.3Kb        | 81.4Kb       | 83.9Kb        |               |               |               |               |
| desunote.ru   | => 76.166.195.195            | 2.27Mb        | 2.56Mb       | 2.14Mb        |               |               |               |               |
|               | <=                           | 83.3Kb        | 67.8Kb       | 51.5Kb        |               |               |               |               |
| desunote.ru   | => customer-189-217-42-126.c | 2.04Mb        | 1.92Mb       | 1.58Mb        |               |               |               |               |
|               | <=                           | 27.7Kb        | 26.9Kb       | 23.1Kb        |               |               |               |               |
| desunote.ru   | => 72.52.102.74              | 1.88Mb        | 1.72Mb       | 1.03Mb        |               |               |               |               |
|               | <=                           | 44.3Kb        | 45.4Kb       | 28.3Kb        |               |               |               |               |
| desunote.ru   | => 71.67.143.88              | 803Kb         | 750Kb        | 743Kb         |               |               |               |               |
|               | <=                           | 13.0Kb        | 13.2Kb       | 13.4Kb        |               |               |               |               |
| desunote.ru   | => 75-134-53-106.dhcp.stls.m | 211Kb         | 193Kb        | 172Kb         |               |               |               |               |
|               | <=                           | 869Kb         | 531Kb        | 440Kb         |               |               |               |               |
| desunote.ru   | => 77.249.17.139             | 148Kb         | 187Kb        | 179Kb         |               |               |               |               |
|               | <=                           | 487Kb         | 501Kb        | 457Kb         |               |               |               |               |
| <b>TX:</b>    | <b>cumm:</b>                 | <b>143MB</b>  | <b>peak:</b> | <b>47.3Mb</b> | <b>rates:</b> | <b>25.7Mb</b> | <b>34.0Mb</b> | <b>35.8Mb</b> |
| <b>RX:</b>    |                              | <b>11.2MB</b> |              | <b>5.35Mb</b> |               | <b>2.69Mb</b> | <b>2.57Mb</b> | <b>2.79Mb</b> |
| <b>TOTAL:</b> |                              | <b>154MB</b>  |              | <b>49.7Mb</b> |               | <b>28.4Mb</b> | <b>36.5Mb</b> | <b>38.6Mb</b> |

Рисунок 1.5 – Интерфейс утилиты iftop

### 1.3.6 powertop

Powertop - это утилита для мониторинга и оптимизации энергопотребления ноутбуков и других мобильных устройств с операционной системой Linux. Эта программа помогает выявить процессы и устройства, которые наиболее интенсивно используют энергию, и предлагает рекомендации по их оптимизации.

Powertop работает путем анализа активности процессов и устройств, а также измерения энергопотребления системы. Затем программа отображает список процессов и устройств, отсортированный по уровню энергопотребления, и предоставляет рекомендации по их оптимизации. Например, powertop может предложить снизить яркость экрана, отключить неиспользуемые устройства или изменить параметры режима энергосбережения для определенных процессов.

Некоторые из ключевых параметров, которые можно использовать с powertop, включают:

\* -с: показать список процессов и устройств в двух столбцах для удобства просмотра.

\* -d: запустить powertop в режиме мониторинга, без рекомендаций по оптимизации.

\* -t: отображать только процессы, превышающие заданный порог энергопотребления.

\* -w: запустить powertop в режиме калибровки, для более точного измерения энергопотребления.

Powertop является очень полезным инструментом для оптимизации энергопотребления ноутбуков и других мобильных устройств с операционной системой Linux. Эта утилита доступна для большинства дистрибутивов Linux и может быть установлена с помощью стандартного менеджера пакетов.

| PowerTOP 1.97  |          |           |                           |  |
|--|----------|-----------|---------------------------|--|
| Overview   |          |           |                           |  |
| Idle stats   |          |           |                           |  |
| Frequency stats  |          |           |                           |  |
| Device stats   |          |           |                           |  |
| Tunab  |          |           |                           |  |
| Summary: 9582,2 wakeups/second, 0,0 GPU ops/second and 0,0 VFS ops/sec |          |           |                           |  |
| Usage  | Events/s | Category  | Description               |  |
| 29,7 ms/s  | 1928,0   | Interrupt | [3] net_rx(softirq)       |  |
| 18,1 ms/s  | 1696,6   | Process   | [usb-storage]             |  |
| 235,4 ms/s   | 1542,4   | Process   | ./el.x86_64.linux.bin     |  |
| 14,2 ms/s  | 1133,4   | Timer     |                           |  |
| 13,1 ms/s  | 819,0    | Process   | [kcopyd]                  |  |
| 18,9 ms/s  | 727,3    | Interrupt | [23] ehci_hcd:usb2        |  |
| 57,7 ms/s  | 445,1    | Process   | /usr/bin/python /usr/bin/ |  |
| 13,1 ms/s  | 404,1    | Interrupt | [4] block(softirq)        |  |
| 253,5 ms/s   | 130,8    | Process   | /usr/lib/opera/opera      |  |
| 3,3 ms/s   | 138,6    | Interrupt | [27] SATA controller      |  |
| 2,0 ms/s   | 109,3    | Process   | /usr/bin/mplayer -noquiet |  |
| 12,1 ms/s  | 93,7     | Process   | /opt/google/chrome/chrome |  |
| 8,6 ms/s   | 44,9     | Process   | /usr/bin/Xorg :0 -br -ver |  |
| 1,2 ms/s   | 45,9     | Process   | [kmirror]                 |  |
| 153,7 μs/s   | 44,9     | Process   | [ksoftirqd/3]             |  |
| 1,1 ms/s   | 31,2     | Process   | /usr/lib/gnome-applets/ge |  |
| 66,5 ms/s  | 3,9      | Process   | /usr/bin/gkrellmd --pidfi |  |
| <ESC> Exit   |          |           |                           |  |

Рисунок 1.6 – Интерфейс утилиты powertop

### 1.3.7 itop

Itop - это утилита командной строки для мониторинга использования ресурсов системы в операционной системе Linux. Эта программа отображает список запущенных процессов и их использование ресурсов, таких как ЦП, память, диск и сеть, в реальном времени.

Itop работает путем анализа информации о процессах из /proc и отображает ее в удобном для чтения формате. По умолчанию, itop сортирует процессы по использованию ЦП, но также можно использовать различные параметры для

сортировки по другим критериям, таким как использование памяти, диска или сети.

Некоторые из ключевых параметров, которые можно использовать с `itop`, включают:

- \* `-o`: отсортировать процессы по указанному столбцу.
- \* `-p`: мониторить только указанные процессы.
- \* `-u`: отображать только процессы, запущенные от имени указанного пользователя.
- \* `-d`: указать интервал обновления списка процессов в секундах.
- \* `-a`: отображать все процессы, включая те, которые не используют ресурсы системы.

`Itop` является очень полезным инструментом для диагностики проблем с производительностью системы и оптимизации использования ресурсов. Эта утилита доступна для большинства дистрибутивов Linux и может быть установлена с помощью стандартного менеджера пакетов.

| INT |       | NAME      |   | RATE       |       | MAX   |
|-----|-------|-----------|---|------------|-------|-------|
| 0   | [7785 | 66931927  | ] | 702 Ints/s | (max: | 723)  |
| 17  | [2585 | 178771929 | ] | 781 Ints/s | (max: | 1007) |
| 22  | [0074 | 200094    | ] | 8 Ints/s   | (max: | 16)   |
| 23  | [4801 | 2630331   | ] | 349 Ints/s | (max: | 369)  |
| 27  | [7516 | 2075910   | ] | 117 Ints/s | (max: | 233)  |

Рисунок 1.7 – Интерфейс утилиты `itop`

### 1.3.8 `dnstop`

`Dnstop` - это утилита командной строки для мониторинга DNS-трафика в операционной системе Linux. Эта программа отображает статистику по DNS-запросам и ответам, а также позволяет отслеживать активность конкретных хостов и доменов в реальном времени.

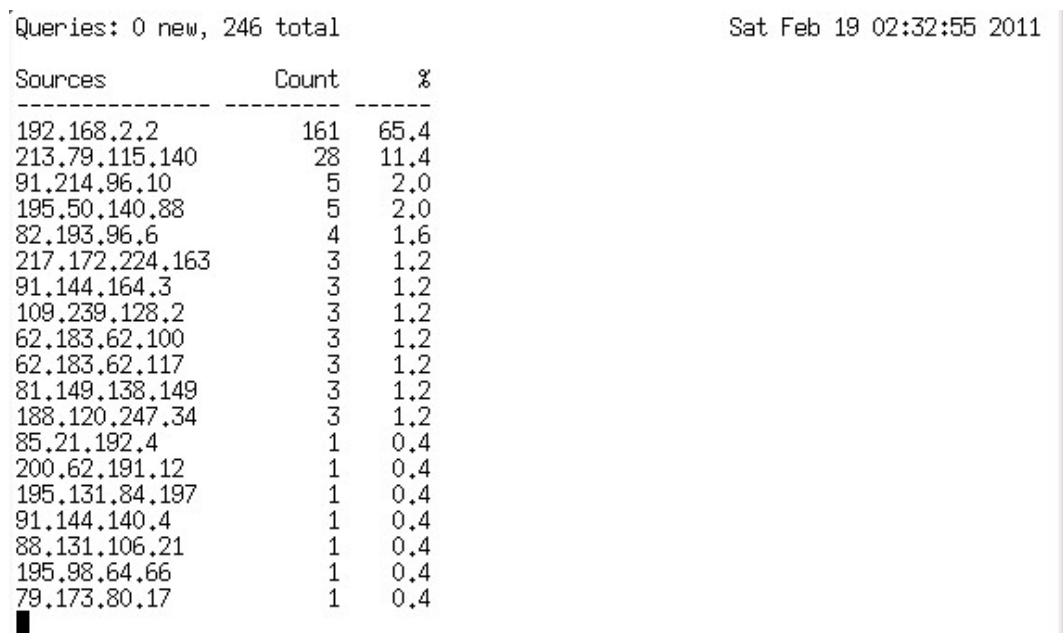
`Dnstop` работает путем анализа пакетов, проходящих через сетевой интерфейс, и извлечения из них информации о DNS-запросах и ответах. Затем программа отображает статистику по количеству запросов и ответов, типам запросов, кодам ответов, используемым протоколам и другим параметрам. Кроме того, `dnstop` позволяет отслеживать активность конкретных хостов и доменов, отображая список наиболее активных из них и их статистику.

Некоторые из ключевых параметров, которые можно использовать с `dnstop`, включают:

- \* `-i`: указать сетевой интерфейс для мониторинга.
- \* `-f`: использовать фильтр пакетов для отображения только определенных DNS-запросов и ответов.
- \* `-n`: отображать IP-адреса вместо имен хостов.
- \* `-p`: указать порт для мониторинга (по умолчанию используется порт 53).

\* -l: записывать статистику в файл для последующего анализа.

Dnstop является очень полезным инструментом для диагностики проблем с DNS-трафиком, таких как замедление работы сети, несанкционированное использование DNS-сервера и т.д. Эта утилита доступна для большинства дистрибутивов Linux и может быть установлена с помощью стандартного менеджера пакетов.



The screenshot shows the output of the dnstop utility. At the top, it says 'Queries: 0 new, 246 total' and 'Sat Feb 19 02:32:55 2011'. Below this is a table with three columns: 'Sources', 'Count', and '%'. The table lists various IP addresses as sources and their corresponding query counts and percentages.

| Sources         | Count | %    |
|-----------------|-------|------|
| 192.168.2.2     | 161   | 65.4 |
| 213.79.115.140  | 28    | 11.4 |
| 91.214.96.10    | 5     | 2.0  |
| 195.50.140.88   | 5     | 2.0  |
| 82.193.96.6     | 4     | 1.6  |
| 217.172.224.163 | 3     | 1.2  |
| 91.144.164.3    | 3     | 1.2  |
| 109.239.128.2   | 3     | 1.2  |
| 62.183.62.100   | 3     | 1.2  |
| 62.183.62.117   | 3     | 1.2  |
| 81.149.138.149  | 3     | 1.2  |
| 188.120.247.34  | 3     | 1.2  |
| 85.21.192.4     | 1     | 0.4  |
| 200.62.191.12   | 1     | 0.4  |
| 195.131.84.197  | 1     | 0.4  |
| 91.144.140.4    | 1     | 0.4  |
| 88.131.106.21   | 1     | 0.4  |
| 195.98.64.66    | 1     | 0.4  |
| 79.173.80.17    | 1     | 0.4  |

Рисунок 1.8 – Интерфейс утилиты dnstop

### 1.3.9 jnettop

Jnettop - это утилита командной строки для мониторинга сетевой активности Java-приложений в операционной системе Linux. Эта программа отображает список активных сетевых соединений, используемых Java-приложениями, и их статистику, такую как скорость передачи данных, объем переданных данных и время активности.

Jnettop работает путем анализа информации о сетевых соединениях, предоставляемой виртуальной машиной Java (JVM), и отображает ее в удобном для чтения формате. По умолчанию, jnettop сортирует соединения по скорости передачи данных, но также можно использовать различные параметры для сортировки по другим критериям, таким как объем переданных данных или время активности.

Некоторые из ключевых параметров, которые можно использовать с jnettop, включают:

\* -l: отображать информацию о локальных соединениях (только для соединений, установленных на локальном хосте).

- \* -r: отображать информацию о удаленных соединениях (только для соединений, установленных с удаленных хостов).
- \* -p: фильтровать список соединений по указанному порту.
- \* -i: указать интервал обновления списка соединений в секундах.
- \* -m: отображать статистику по использованию памяти для каждого Java-приложения.

Jnettop является очень полезным инструментом для диагностики проблем с сетевой активностью Java-приложений, таких как замедление работы приложения, несанкционированное использование сети и т.д. Эта утилита доступна для большинства дистрибутивов Linux и может быть установлена с помощью стандартного менеджера пакетов. Однако, для ее работы может потребоваться установка и настройка соответствующего агента мониторинга Java-приложений.

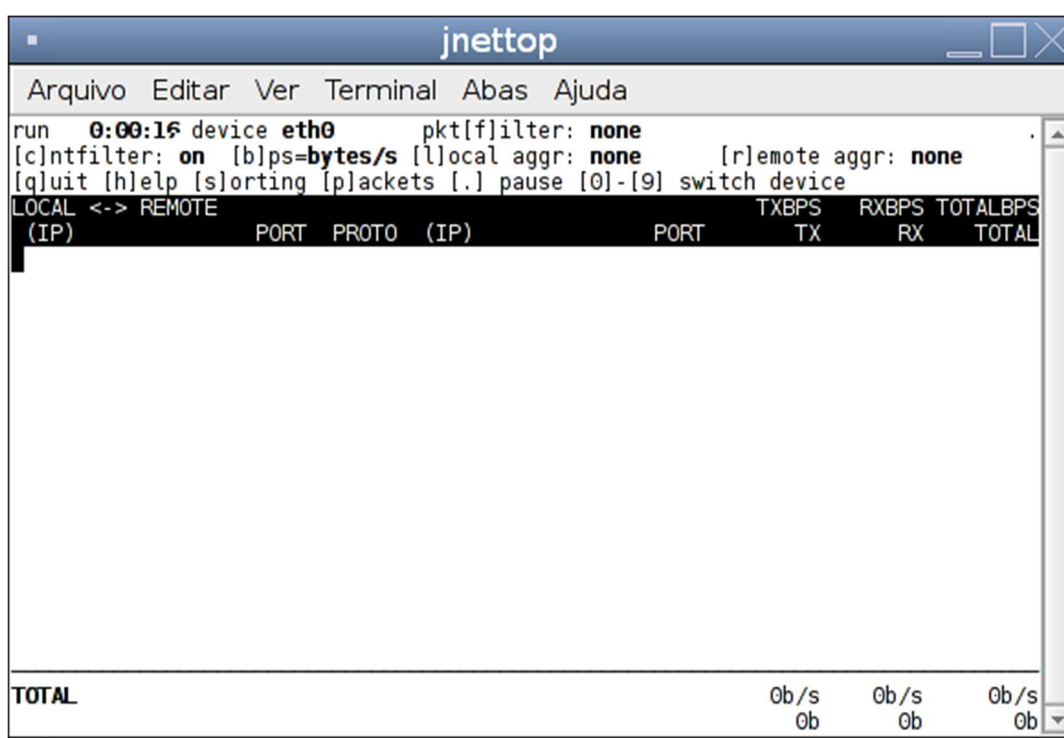


Рисунок 1.9 – Интерфейс утилиты jnettop

### 1.3.10 xrestop

Xrestop - это графическая утилита для мониторинга использования ресурсов X-сервера в операционной системе Linux. Эта программа отображает список запущенных клиентских приложений и их использование ресурсов, таких как ЦП, память и сеть, в реальном времени.

Xrestop работает путем анализа информации о клиентских приложениях, предоставляемой X-сервером, и отображает ее в удобном для чтения формате. По умолчанию, xrestop сортирует приложения по использованию ЦП, но также



можно использовать различные параметры для сортировки по другим критериям, таким как использование памяти или сети.

Некоторые из ключевых параметров, которые можно использовать с xrestop, включают:

- \* -display: указать адрес и номер дисплея X-сервера для мониторинга.
- \* -update: указать интервал обновления списка приложений в секундах.
- \* -geometry: указать геометрию окна программы (ширину и высоту в пикселях).
- \* -font: указать шрифт для отображения текста в программе.
- \* -help: отобразить справку по использованию программы.

Xrestop является очень полезным инструментом для диагностики проблем с производительностью X-сервера и оптимизации использования ресурсов клиентскими приложениями. Эта утилита доступна для большинства дистрибутивов Linux и может быть установлена с помощью стандартного менеджера пакетов. Однако, для ее работы может потребоваться настройка доступа к X-серверу с использованием механизма аутентификации xauth.

```
xrestop - Display: :0.0
Monitoring 58 clients. XErrors: 0
Pixmap: 89950K total, Other: 275K total, All: 90226K total
```

| res-base | Uins | GCs | Fnts | Pxms | Misc | Pxm    | mem | Other | Total  | PID   | Identifier     |
|----------|------|-----|------|------|------|--------|-----|-------|--------|-------|----------------|
| 1a00000  | 7    | 37  | 1    | 21   | 45   | 29056K |     | 3K    | 29059K | 2944  | x-nautilus-des |
| 1000000  | 6    | 28  | 0    | 19   | 293  | 28800K |     | 7K    | 28807K | 2922  | gnome-settings |
| 3000000  | 17   | 55  | 1    | 34   | 82   | 13219K |     | 4K    | 13223K | 9219  | ajaxterm.py at |
| 3800000  | 13   | 3   | 1    | 1919 | 1996 | 8699K  |     | 48K   | 8747K  | 9318  | 'Krusader'     |
| 4c00000  | 1    | 4   | 0    | 453  | 10   | 3021K  |     | 360B  | 3021K  | ?     | rdesktop - 192 |
| 5800000  | 77   | 118 | 1    | 70   | 126  | 1393K  |     | 8K    | 1402K  | 5976  | atop.png-3.0 ( |
| 4800000  | 14   | 2   | 0    | 97   | 122  | 1210K  |     | 3K    | 1213K  | 25104 | [ARR] Ore wa T |
| 3400000  | 12   | 28  | 0    | 5    | 19   | 521K   |     | 1K    | 523K   | 9272  | exe            |
| 1600000  | 45   | 67  | 1    | 24   | 142  | 479K   |     | 6K    | 486K   | 2932  | Panel          |
| 2000000  | 8    | 43  | 1    | 20   | 52   | 331K   |     | 3K    | 334K   | 3009  | Deluge         |
| 5000000  | 23   | 30  | 1    | 19   | 66   | 256K   |     | 3K    | 259K   | 11871 | 1cmail@ag-srv- |
| 7000000  | 5    | 44  | 1    | 19   | 47   | 256K   |     | 3K    | 259K   | 9955  | 0i0xN~EN~0D°Dx |
| 6e00000  | 5    | 44  | 1    | 19   | 47   | 256K   |     | 3K    | 259K   | 9501  | 0i0xN~EN~0D°Dx |
| 6c00000  | 5    | 44  | 1    | 19   | 47   | 256K   |     | 3K    | 259K   | 8621  | 0i0xN~EN~0D°Dx |
| 6a00000  | 5    | 44  | 1    | 19   | 47   | 256K   |     | 3K    | 259K   | 8274  | 0i0xN~EN~0D°Dx |
| 6800000  | 5    | 44  | 1    | 19   | 47   | 256K   |     | 3K    | 259K   | 7934  | 0i0xN~EN~0D°Dx |
| 6400000  | 5    | 44  | 1    | 19   | 47   | 256K   |     | 3K    | 259K   | 7776  | 0i0xN~EN~0D°Dx |
| 6200000  | 5    | 44  | 1    | 19   | 47   | 256K   |     | 3K    | 259K   | 7175  | 0i0xN~EN~0D°Dx |
| 6000000  | 5    | 44  | 1    | 19   | 47   | 256K   |     | 3K    | 259K   | 7046  | 0i0xN~EN~0D°Dx |
| 5e00000  | 5    | 44  | 1    | 19   | 47   | 256K   |     | 3K    | 259K   | 6814  | 0i0xN~EN~0D°Dx |

Рисунок 1.10 – Интерфейс утилиты xrestop

### 1.3.11 slabtop

Slabtop - это утилита командной строки для мониторинга использования кэша ядра (слабов) в операционной системе Linux. Эта программа отображает список кэшей, используемых ядром для хранения часто используемых объектов,

и их статистику, такую как количество используемых и свободных объектов, объем памяти, занимаемый кэшами, и т.д.

Slabtop работает путем анализа информации о кэшах, предоставляемой ядром, и отображает ее в удобном для чтения формате. По умолчанию, slabtop сортирует кэши по объему используемой памяти, но также можно использовать различные параметры для сортировки по другим критериям, таким как количество используемых объектов или фрагментация памяти.

Некоторые из ключевых параметров, которые можно использовать с slabtop, включают:

- \* -o: отсортировать кэши по указанному столбцу.
- \* -s: отсортировать кэши по указанному критерию (например, по объему памяти или количеству объектов).
- \* -d: указать интервал обновления списка кэшей в секундах.
- \* -c: отображать информацию о фрагментации памяти для каждого кэша.
- \* -h: отобразить справку по использованию программы.

Slabtop является очень полезным инструментом для диагностики проблем с использованием памяти в системе, вызванных фрагментацией или нехваткой памяти в кэшах ядра. Эта утилита доступна для большинства дистрибутивов Linux и может быть установлена с помощью стандартного менеджера пакетов. Однако, для ее работы может потребоваться настройка ядра с включением поддержки slab-дебаггера (SLUB debugging).

```
Active / Total Objects (% used) : 18330624 / 18493941 (99.1%)
Active / Total Slabs (% used)   : 569316 / 569538 (100.0%)
Active / Total Caches (% used)  : 103 / 169 (60.9%)
Active / Total Size (% used)    : 2108425.57K / 2145540.83K (98.3%)
Minimum / Average / Maximum Object : 0.02K / 0.12K / 4096.00K
```

| OBJ      | ACTIVE   | USE | OBJ SIZE | SLABS  | OBJ/SLAB | CACHE    | SIZE              | NAME |
|----------|----------|-----|----------|--------|----------|----------|-------------------|------|
| 17830670 | 17744282 | 99% | 0.10K    | 481910 | 37       | 1927640K | buffer_head       |      |
| 483938   | 443326   | 91% | 0.54K    | 69134  | 7        | 276536K  | radix_tree_node   |      |
| 28560    | 27418    | 96% | 0.03K    | 255    | 112      | 1020K    | size-32           |      |
| 25016    | 21298    | 85% | 0.06K    | 424    | 59       | 1696K    | size-64           |      |
| 14304    | 14015    | 97% | 0.08K    | 298    | 48       | 1192K    | sysfs_dir_cache   |      |
| 13500    | 10902    | 80% | 0.25K    | 900    | 15       | 3600K    | skbuff_head_cache |      |
| 10887    | 7640     | 70% | 0.20K    | 573    | 19       | 2292K    | dentry            |      |
| 9180     | 7873     | 85% | 0.12K    | 306    | 30       | 1224K    | size-128          |      |
| 9094     | 8900     | 97% | 4.00K    | 9094   | 1        | 36376K   | size-4096         |      |
| 7200     | 4662     | 64% | 0.02K    | 50     | 144      | 200K     | dm_target_io      |      |
| 7176     | 4638     | 64% | 0.04K    | 78     | 92       | 312K     | dm_io             |      |
| 4870     | 4501     | 92% | 0.73K    | 974    | 5        | 3896K    | ext2_inode_cache  |      |
| 4444     | 1454     | 32% | 0.02K    | 22     | 202      | 88K      | biovec-1          |      |
| 4263     | 2682     | 62% | 0.18K    | 203    | 21       | 812K     | vm_area_struct    |      |
| 3021     | 2157     | 71% | 0.07K    | 57     | 53       | 228K     | Acpi-Operand      |      |
| 2970     | 1395     | 46% | 0.12K    | 99     | 30       | 396K     | bio               |      |
| 2640     | 737      | 27% | 0.19K    | 132    | 20       | 528K     | filp              |      |
| 2564     | 2533     | 98% | 2.00K    | 1282   | 2        | 5128K    | size-2048         |      |
| 2440     | 2401     | 98% | 1.00K    | 610    | 4        | 2440K    | size-1024         |      |
| 2280     | 2218     | 97% | 0.77K    | 456    | 5        | 1824K    | shmem_inode_cache |      |
| 2040     | 1567     | 76% | 0.25K    | 136    | 15       | 544K     | size-256          |      |
| 1920     | 1069     | 55% | 0.74K    | 384    | 5        | 1536K    | ext3_inode_cache  |      |
| 1872     | 715      | 38% | 0.02K    | 13     | 144      | 52K      | anon_vma          |      |
| 1452     | 1290     | 88% | 0.58K    | 242    | 6        | 968K     | proc_inode_cache  |      |
| 1400     | 1294     | 92% | 0.50K    | 175    | 8        | 700K     | size-512          |      |
| 1299     | 330      | 26% | 0.06K    | 21     | 59       | 84K      | task_delay_info   |      |

Рисунок 1.11 – Интерфейс утилиты slabtop

### 1.3.12 mytop

Mytop - это утилита командной строки для мониторинга использования ресурсов базы данных MySQL в операционной системе Linux. Эта программа

отображает список запущенных потоков MySQL и их использование ресурсов, таких как ЦП, память и диск, в реальном времени.

Mytop работает путем анализа информации о потоках, предоставляемой MySQL, и отображает ее в удобном для чтения формате. По умолчанию, mytop сортирует потоки по использованию ЦП, но также можно использовать различные параметры для сортировки по другим критериям, таким как время выполнения запроса или использование памяти.

Некоторые из ключевых параметров, которые можно использовать с mytop, включают:

- \* -h: указать адрес хоста, на котором запущена база данных MySQL.
- \* -u: указать имя пользователя для подключения к базе данных MySQL.
- \* -p: указать пароль для подключения к базе данных MySQL.
- \* -d: указать имя базы данных для мониторинга (по умолчанию используется текущая база данных).
- \* -i: указать интервал обновления списка потоков в секундах.
- \* -s: отсортировать потоки по указанному критерию (например, по времени выполнения запроса или использованию памяти).

Mytop является очень полезным инструментом для диагностики проблем с производительностью базы данных MySQL и оптимизации использования ресурсов запросами. Эта утилита доступна для большинства дистрибутивов Linux и может быть установлена с помощью стандартного менеджера пакетов. Однако, для ее работы может потребоваться настройка доступа к базе данных MySQL с использованием соответствующих прав доступа.

```
MySQL on localhost (5.0.51a-24+lenny4) up 54+15:15:23 [02:44:32]
Queries: 39.2k qps: 0 Slow: 0.0 Se/In/Up/De(%): 17/00/00/00
qps now: 0 Slow qps: 0.0 Threads: 2 ( 1/ 6) 00/00/00/00
Key Efficiency: 99.3% Bps in/out: 0.0/ 0.0 Now in/out: 8.4/ 1.3k

  Id      User      Host/IP      DB      Time      Cmd Query or State
  --      ----      -
2078     root      localhost
1609     atslog     localhost     atslog    18519     Sleep
```

Рисунок 1.12 – Интерфейс утилиты mytop

### 1.3.13 xentop

Xentop - это утилита командной строки для мониторинга использования ресурсов виртуальных машин (VM) в среде Xen. Эта программа отображает

список запущенных ВМ и их использование ресурсов, таких как ЦП, память, диск и сеть, в реальном времени.

Xentop работает путем анализа информации о ВМ, предоставляемой гипервизором Xen, и отображает ее в удобном для чтения формате. По умолчанию, xentop сортирует ВМ по использованию ЦП, но также можно использовать различные параметры для сортировки по другим критериям, таким как использование памяти или сети.

Некоторые из ключевых параметров, которые можно использовать с xentop, включают:

- \* -s: указать адрес хоста, на котором запущен гипервизор Xen.
- \* -p: указать номер порта, используемый для подключения к гипервизору Xen (по умолчанию используется порт 26).
- \* -u: указать имя пользователя для подключения к гипервизору Xen.
- \* -d: указать интервал обновления списка ВМ в секундах.
- \* -c: отображать информацию о использовании ЦП для каждого ядра (CPU core) гипервизора Xen.
- \* -m: отображать информацию о использовании памяти для каждой ВМ.

Xentop является очень полезным инструментом для диагностики проблем с производительностью виртуальных машин в среде Xen и оптимизации использования ресурсов. Эта утилита доступна для большинства дистрибутивов Linux и может быть установлена с помощью стандартного менеджера пакетов. Однако, для ее работы может потребоваться настройка доступа к гипервизору Xen с использованием соответствующих прав доступа.

```
xentop - 02:25:26 Xen 3.4.2
39 domains: 2 running, 37 blocked, 0 paused, 0 crashed, 0 dying, 0 shutdown
Mem: total, 22498456k used, free CPUs: 8 @ 2266MHz
```

| NAME     | STATE | CPU(sec) | CPU(%)  | MEM(k) | MEM(%)  | PRIVATE(k) | PRIVATE(%) | VCPUS | NETS    | NETTX(k) | NETRX(k) | VDS  | VDS_OO   | VDS_RD  | VDS_WR | SSID |
|----------|-------|----------|---------|--------|---------|------------|------------|-------|---------|----------|----------|------|----------|---------|--------|------|
| --b----  | 466   | 0.0      | 262144  | 0.5    | 262144  | 0.5        | 8          | 1     | 38      | 182241   | 2        | 115  | 1        | 35707   | 0      |      |
| --b----  | 1577  | 1.2      | 368640  | 0.7    | 331776  | 0.7        | 8          | 1     | 31578   | 207328   | 1        | 54   | 19534    | 126922  | 0      |      |
| --b----  | 548   | 0.0      | 189232  | 0.4    | 189232  | 0.4        | 8          | 1     | 2178    | 186231   | 2        | 97   | 8        | 42973   | 0      |      |
| --b----  | 49481 | 0.0      | 565248  | 1.1    | 524288  | 1.0        | 8          | 1     | 1023836 | 2131908  | 1        | 347  | 742845   | 801503  | 0      |      |
| --b----  | 976   | 0.0      | 317188  | 0.6    | 317188  | 0.6        | 8          | 1     | 30443   | 284194   | 1        | 12   | 7294     | 183022  | 0      |      |
| --b----  | 1412  | 0.0      | 368640  | 0.7    | 331776  | 0.7        | 8          | 1     | 430726  | 276800   | 2        | 533  | 8098     | 130394  | 0      |      |
| --b----  | 658   | 0.0      | 262144  | 0.5    | 262144  | 0.5        | 8          | 1     | 8303    | 1921445  | 1        | 768  | 3588     | 84210   | 0      |      |
| --b----  | 2239  | 0.8      | 1153468 | 2.3    | 1048576 | 2.1        | 8          | 1     | 10123   | 194412   | 1        | 65   | 16361    | 122700  | 0      |      |
| --b----  | 5825  | 0.2      | 524288  | 1.0    | 524288  | 1.0        | 8          | 1     | 16949   | 198058   | 1        | 965  | 99928    | 320427  | 0      |      |
| --b----  | 4943  | 0.0      | 349184  | 0.7    | 314264  | 0.6        | 8          | 1     | 334477  | 557799   | 1        | 607  | 106659   | 413236  | 0      |      |
| --b----  | 1207  | 0.0      | 560020  | 1.1    | 560020  | 1.1        | 8          | 1     | 133478  | 227154   | 1        | 541  | 18947    | 77850   | 0      |      |
| --b----  | 189   | 0.0      | 174448  | 0.3    | 174448  | 0.3        | 8          | 1     | 10440   | 18642    | 1        | 0    | 102      | 12729   | 0      |      |
| --b----  | 680   | 0.0      | 131072  | 0.3    | 131072  | 0.3        | 1          | 1     | 12627   | 196166   | 1        | 549  | 3138     | 48811   | 0      |      |
| --b----  | 41046 | 0.8      | 2097152 | 4.2    | 2097152 | 4.2        | 8          | 1     | 1904189 | 884356   | 1        | 115  | 1        | 291041  | 0      |      |
| --b----  | 942   | 0.0      | 524288  | 1.0    | 524288  | 1.0        | 8          | 1     | 17197   | 276355   | 1        | 204  | 4356     | 76414   | 0      |      |
| --b----  | 1877  | 0.4      | 524288  | 1.0    | 524288  | 1.0        | 8          | 1     | 446976  | 367794   | 3        | 142  | 27047    | 641968  | 0      |      |
| --b----  | 1033  | 0.0      | 317440  | 0.6    | 285696  | 0.6        | 8          | 1     | 16595   | 200006   | 1        | 702  | 2303     | 58893   | 0      |      |
| --b----  | 3983  | 0.0      | 628240  | 1.2    | 628240  | 1.2        | 8          | 1     | 559322  | 285972   | 1        | 735  | 128925   | 309032  | 0      |      |
| --b----  | 107   | 0.0      | 348944  | 0.7    | 348944  | 0.7        | 8          | 1     | 1899    | 140498   | 1        | 0    | 14225    | 51977   | 0      |      |
| --b----  | 680   | 0.0      | 173908  | 0.3    | 173908  | 0.3        | 1          | 1     | 48257   | 186113   | 1        | 562  | 15040    | 73964   | 0      |      |
| --b----  | 14170 | 3.7      | 310868  | 0.6    | 310868  | 0.6        | 8          | 1     | 6180    | 210732   | 1        | 571  | 52698    | 153168  | 0      |      |
| --b----  | 6081  | 0.6      | 498104  | 1.0    | 498104  | 1.0        | 8          | 1     | 782837  | 760241   | 1        | 901  | 1219637  | 441230  | 0      |      |
| --b----  | 2909  | 1.6      | 1048576 | 2.1    | 1048576 | 2.1        | 8          | 1     | 57685   | 239654   | 1        | 607  | 2824     | 159293  | 0      |      |
| --b----  | 1859  | 0.0      | 317188  | 0.6    | 317188  | 0.6        | 8          | 1     | 13774   | 193541   | 1        | 244  | 244      | 166260  | 0      |      |
| --b----  | 5339  | 0.0      | 307200  | 0.6    | 307200  | 0.6        | 8          | 1     | 185390  | 806934   | 1        | 169  | 110099   | 693536  | 0      |      |
| --b----  | 40288 | 0.4      | 1048576 | 2.1    | 1048576 | 2.1        | 8          | 1     | 366408  | 587602   | 1        | 360  | 183623   | 1535394 | 0      |      |
| --b----  | 512   | 0.0      | 262144  | 0.5    | 262144  | 0.5        | 8          | 1     | 16395   | 98991    | 1        | 131  | 9309     | 37333   | 0      |      |
| --b----  | 2314  | 0.0      | 262144  | 0.5    | 262144  | 0.5        | 8          | 1     | 31302   | 202988   | 1        | 406  | 73009    | 340742  | 0      |      |
| --b----  | 2123  | 0.0      | 288768  | 0.6    | 262144  | 0.5        | 8          | 1     | 1375    | 195376   | 1        | 1008 | 7699     | 224920  | 0      |      |
| --b----  | 3415  | 0.0      | 112640  | 0.2    | 112640  | 0.2        | 1          | 1     | 14010   | 274271   | 1        | 524  | 42925    | 258769  | 0      |      |
| --b----  | 1493  | 0.0      | 216268  | 0.4    | 216268  | 0.4        | 8          | 1     | 824733  | 1030976  | 1        | 53   | 250181   | 171667  | 0      |      |
| -----f   | 692   | 98.3     | 262144  | 0.5    | 262144  | 0.5        | 8          | 1     | 65533   | 184839   | 1        | 120  | 8871     | 44710   | 0      |      |
| --b----  | 924   | 0.0      | 237892  | 0.5    | 237892  | 0.5        | 8          | 1     | 2763    | 187614   | 1        | 56   | 18946    | 173344  | 0      |      |
| --b----  | 395   | 0.0      | 131072  | 0.3    | 131072  | 0.3        | 1          | 1     | 1770    | 182937   | 1        | 92   | 0        | 34439   | 0      |      |
| --b----  | 1515  | 0.0      | 166692  | 0.3    | 166692  | 0.3        | 1          | 1     | 198937  | 257972   | 1        | 407  | 2918     | 43959   | 0      |      |
| --b----  | 66523 | 1.1      | 2097152 | 4.2    | 2097152 | 4.2        | 8          | 1     | 3984489 | 531761   | 1        | 902  | 26387565 | 2689018 | 0      |      |
| --b----  | 3351  | 2.5      | 1102096 | 2.2    | 1102096 | 2.2        | 1          | 1     | 818902  | 112100   | 1        | 0    | 153293   | 48450   | 0      |      |
| Domain-0 | 84294 | 98.8     | 1262592 | 2.5    | 1262592 | 2.5        | 8          | 0     | 0       | 0        | 0        | 0    | 0        | 0       | 0      |      |
| --b----  | 1670  | 1.6      | 2097152 | 4.2    | 2097152 | 4.2        | 2          | 1     | 12871   | 21343    | 1        | 0    | 2        | 60717   | 0      |      |

Delay Networks vds CPUs Repeat header Sort order Quit

Рисунок 1.13 – Интерфейс утилиты xentop



### 1.3.14 nethogs

Nethogs - это утилита командной строки для мониторинга использования сетевого трафика отдельными процессами в операционной системе Linux. Эта программа отображает список запущенных процессов и их использование сетевого трафика в реальном времени.

Nethogs работает путем анализа информации о сетевых пакетах, проходящих через сетевую карту, и сопоставления их с соответствующими процессами. Затем программа отображает список процессов, отсортированный по использованию сетевого трафика, и показывает статистику по переданным и полученным байтам, а также по текущей скорости передачи данных.

Некоторые из ключевых параметров, которые можно использовать с nethogs, включают:

- \* -d: указать интервал обновления списка процессов в секундах.
- \* -t: отображать статистику по переданным и полученным байтам в виде таблицы.
- \* -p: фильтровать список процессов по указанному порту.
- \* -u: отображать информацию о владельце процесса (UID).
- \* -h: отобразить справку по использованию программы.

Nethogs является очень полезным инструментом для диагностики проблем с сетевым трафиком, вызванных отдельными процессами в системе. Эта утилита доступна для большинства дистрибутивов Linux и может быть установлена с помощью стандартного менеджера пакетов. Однако, для ее работы может потребоваться настройка доступа к сетевой карте с использованием соответствующих прав доступа.

| PID   | USER     | PROGRAM                     | DEV  | SENT     | RECEIVED      |
|-------|----------|-----------------------------|------|----------|---------------|
| 3009  | amarao   | /usr/bin/python             | eth2 | 3392.547 | 49.332 KB/sec |
| 4458  | www-data | /usr/sbin/apache2           | eth2 | 18.372   | 1.497 KB/sec  |
| 31001 | amarao   | ./el.x86_64.linux.bin       | eth2 | 0.193    | 0.218 KB/sec  |
| 0     | root     | ..6:80-85.118.226.108:38600 |      | 1.149    | 0.190 KB/sec  |
| 0     | root     | ..6:80-109.110.40.176:49426 |      | 2.473    | 0.172 KB/sec  |
| 0     | root     | ..2.76:80-109.254.49.8:3325 |      | 0.077    | 0.151 KB/sec  |
| 0     | root     | ..6:80-88.204.125.101:59399 |      | 2.484    | 0.146 KB/sec  |
| 0     | root     | ..6:80-85.118.226.108:56970 |      | 0.914    | 0.139 KB/sec  |
| 0     | root     | ..57776-94.100.19.196:49462 |      | 0.178    | 0.129 KB/sec  |
| 0     | root     | ..6:1c25:aed:2c1a:b318:6552 |      | 0.000    | 0.054 KB/sec  |
| 0     | root     | ..57776-98.109.218.42:57671 |      | 0.033    | 0.048 KB/sec  |
| 0     | root     | ..:57776-124.104.97.58:1443 |      | 0.033    | 0.048 KB/sec  |
| 0     | root     | ..:57776-24.37.115.49:55605 |      | 0.033    | 0.048 KB/sec  |
| 0     | root     | ..6:1c25:aed:2c1a:b318:6552 |      | 0.031    | 0.038 KB/sec  |
| 7171  | root     | pptp                        | eth2 | 0.024    | 0.026 KB/sec  |
| 4708  | www-data | /usr/sbin/apache2           | eth2 | 0.013    | 0.013 KB/sec  |
| 0     | root     | ..49589-220.237.130.63:9904 |      | 0.011    | 0.012 KB/sec  |
| 0     | root     | ..:57776-64.217.18.183:3600 |      | 0.000    | 0.012 KB/sec  |
| 4698  | www-data | /usr/sbin/apache2           | eth2 | 0.011    | 0.012 KB/sec  |
| 0     | root     | ..:57776-64.217.18.183:3599 |      | 0.000    | 0.012 KB/sec  |
| 0     | root     | ..:57776-184.56.20.44:54224 |      | 0.000    | 0.000 KB/sec  |
| 0     | root     | ..6:80-209.121.54.212:59156 |      | 0.000    | 0.000 KB/sec  |
| 0     | root     | unknown TCP                 |      | 0.000    | 0.000 KB/sec  |
| TOTAL |          |                             |      | 3418.577 | 52.296 KB/sec |

Рисунок 1.14 – Интерфейс утилиты nethogs

## 1.4 Сравнительный анализ

Тор и htop являются двумя самыми популярными утилитами для мониторинга процессов в Unix-подобных операционных системах. Обе программы предоставляют пользователю информацию о процессах, запущенных в системе, и позволяют управлять ими. В этом разделе будет проведен сравнительный анализ этих двух программ, а также выделены важные моменты, которые необходимо учесть при реализации собственного проекта.

Тор имеет текстовый интерфейс, который обновляется каждые несколько секунд. Он отображает список процессов в табличном виде, содержащем информацию о PID, пользователе, приоритете, использовании памяти и процессора, времени выполнения и других параметрах. Пользователь может отсортировать процессы по любому из этих параметров, а также фильтровать их по различным критериям.

Нтор также имеет текстовый интерфейс, но он более визуальный и интуитивно понятный. Программа отображает список процессов в виде таблицы, где каждая строка содержит цветную индикацию использования ресурсов. Пользователь может прокручивать список вверх и вниз, а также использовать мышь для выделения процессов и выполнения действий с ними. Кроме того, htop предоставляет графическое представление использования процессора, памяти и swap-памяти.

Функциональность:

Тор и htop предоставляют схожую функциональность, но есть некоторые отличия. Тор предоставляет больше опций для настройки отображения информации о процессах, например, можно выбрать, какие столбцы отображать в таблице. Кроме того, тор позволяет выполнять некоторые действия с процессами, такие как убийство процесса или изменение его приоритета.

Нтор также предоставляет возможность управлять процессами, но он делает это более удобным способом. Например, пользователь может выделить несколько процессов и выполнить с ними одно действие, или просто нажать клавишу F9, чтобы убить выделенный процесс. Кроме того, htop предоставляет возможность отправлять сигналы процессам, например, SIGTERM или SIGKILL.

Тор и htop имеют разную производительность. Тор использует меньше ресурсов системы, так как он обновляет информацию о процессах каждые несколько секунд.

Нтор обновляет информацию в реальном времени, что требует большего количества ресурсов системы.

Кроссплатформенность

Тор доступен на большинстве Unix-подобных операционных систем, включая Linux, macOS и BSD.

Нтор также доступен на большинстве этих систем, но его нет в стандартной поставке macOS.

На основе проведённого анализа следует выделить важные моменты, которые необходимо учесть при реализации собственного проекта:

Интерфейс должен быть интуитивно понятным и удобным для пользователя. Функциональность должна соответствовать потребностям пользователя и предоставлять необходимые возможности для управления процессами. Производительность должна быть оптимизирована для минимизации нагрузки на систему. Необходимо предусмотреть возможность настройки отображения информации о процессах в соответствии с потребностями пользователя. Предусмотреть возможность отправки сигналов процессам для управления ими. Предусмотреть возможность фильтрации процессов по различным критериям для удобства пользователя. Предусмотреть возможность сортировки процессов по различным параметрам для удобства пользователя. Предусмотреть возможность настройки интервала обновления информации о процессах для оптимизации производительности.

## 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Установка требований к функционалу разрабатываемой в рамках курсового проекта программы позволяет провести разделение всего алгоритма работы приложения на функциональные блоки. Функциональные блоки – это блоки программного компонента, которые ответственны за определенную задачу, а совокупность функциональных блоков позволяет реализовать полноценную работу программы. Наличие функциональных блоков сокращает количество времени на понимание внутреннего устройства программы, обеспечивая гибкость и масштабируемость приложения с целью последующей возможной доработки путем добавления дополнительных программных блоков.

Программу диспетчера процессов и потоков можно разделить на 6 функциональных блоков:

Блок ввода-вывода: этот блок отвечает за взаимодействие с пользователем и отображение информации о процессах и потоках. Он включает в себя функции для отображения списка процессов и потоков, обработки ввода пользователя и вывода сообщений об ошибках.

Блок чтения данных: этот блок отвечает за чтение данных о процессах и потоках из системных файлов. Он включает в себя функции для чтения информации из каталога /proc и заполнения структур Process и Thread.

Блок сортировки: этот блок отвечает за сортировку процессов и потоков по разным критериям. Он включает в себя функции для сортировки массивов структур Process и Thread с помощью стандартной функции qsort.

Блок управления процессами и потоками: этот блок отвечает за управление процессами и потоками, включая их запуск, остановку и убийство. Он включает в себя функции для отправки сигналов процессам и потокам с помощью системных вызовов kill и pthread\_cancel.

Блок обработки сигналов: этот блок отвечает за обработку сигналов, генерируемых операционной системой. Он включает в себя функции для установки обработчиков сигналов и обработки сигнала SIGINT.

Блок главного цикла программы: этот блок отвечает за управление основным циклом программы, включая очистку экрана, обновление данных о процессах и потоках, отображение информации и обработку ввода пользователя.

Взаимодействие между этими блоками происходит следующим образом:

Блок ввода-вывода получает команды от пользователя и передает их в соответствующие блоки для обработки. Блок чтения данных читает информацию о процессах и потоках из системных файлов и заполняет структуры Process и Thread. Блок сортировки сортирует массивы структур Process и Thread в соответствии с критериями, заданными пользователем. Блок управления процессами и потоками отправляет сигналы процессам и потокам в соответствии с командами, полученными от блока ввода-вывода. Блок обработки сигналов обрабатывает сигналы, генерируемые операционной системой, и выполняет необходимые действия, такие как завершение программы при получении сигнала SIGINT. Блок главного цикла программы управляет основным циклом



программы, вызывая функции из других блоков для обновления данных, отображения информации и обработки ввода пользователя.

## **2.1 Блок ввода-вывода**

Блок ввода-вывода отвечает за взаимодействие с пользователем и отображение информации о процессах и потоках. Он предоставляет пользователю интерфейс для ввода команд и выводит результаты их выполнения. В частности, этот блок включает в себя функции для отображения списка процессов и потоков, обработки ввода пользователя и вывода сообщений об ошибках.

## **2.2 Блок чтения данных**

Блок чтения данных отвечает за чтение данных о процессах и потоках из системных файлов. Он читает информацию из каталога /proc и заполняет структуры Process и Thread. В частности, этот блок включает в себя функции для чтения информации о процессах и потоках, а также для преобразования этой информации в формат, подходящий для отображения и сортировки.

## **2.3 Блок сортировки**

Блок сортировки отвечает за сортировку процессов и потоков по разным критериям. Он предоставляет пользователю возможность сортировать список процессов и потоков по различным параметрам, таким как идентификатор, имя пользователя, приоритет, использование ресурсов и т.д. В частности, этот блок включает в себя функции для сортировки массивов структур Process и Thread с помощью стандартной функции qsort.

## **2.4 Блок управления процессами и потоками**

Блок управления процессами и потоками отвечает за управление процессами и потоками, включая их запуск, остановку и убийство. Он предоставляет пользователю возможность управлять процессами и потоками, отправляя им соответствующие сигналы. В частности, этот блок включает в себя функции для отправки сигналов процессам и потокам с помощью системных вызовов kill и pthread\_cancel..

## **2.5 Блок обработки сигналов**

Блок обработки сигналов отвечает за обработку сигналов, генерируемых операционной системой. Он предоставляет пользователю возможность обрабатывать сигналы, такие как SIGINT, и выполнять необходимые действия,

такие как завершение программы. В частности, этот блок включает в себя функции для установки обработчиков сигналов и обработки сигнала SIGINT.

## **2.6 Блок главного цикла программы**

Блок главного цикла программы отвечает за управление основным циклом программы, включая очистку экрана, обновление данных о процессах и потоках, отображение информации и обработку ввода пользователя. Он обеспечивает взаимодействие между всеми другими блоками и управляет потоком выполнения программы. В частности, этот блок включает в себя функции для очистки экрана, обновления данных о процессах и потоках, отображения информации и обработки ввода пользователя.

Каждый из этих блоков выполняет определенную задачу и взаимодействует с другими блоками для обеспечения полноценной работы программы. Наличие функциональных блоков позволяет сократить количество времени на понимание внутреннего устройства программы и обеспечить гибкость и масштабируемость приложения с целью последующей возможной доработки путем добавления дополнительных программных блоков.

### 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается структура разрабатываемой в рамках курсового проекта программы с точки зрения описания данных и обрабатывающих их подпрограмм – функций.

#### 3.1 Описание основных структур данных программы

В программе используются следующие основные структуры данных:

Структура `Process` содержит информацию о процессе, включая его идентификатор (PID), имя пользователя, приоритет, потребление памяти и ресурсов процессора, команду, запустившую процесс, и т.д. Эта структура используется для хранения информации о каждом процессе в системе.

```
typedef struct {
    int pid;
    char user[50];
    int priority;
    long virtual_memory;
    long resident_memory;
    float cpu_usage;
    char command[100];
} Process;
```

Структура `Thread`: содержит информацию о потоке, включая его идентификатор (TID), идентификатор процесса, к которому он относится (PID), имя пользователя, приоритет, потребление памяти и ресурсов процессора, команду, запустившую поток, и т.д. Эта структура используется для хранения информации о каждом потоке в системе.

```
typedef struct {
    pid_t tid;
    pid_t pid;
    char user[50];
    int priority;
    long memory;
    long cpu_usage;
    char command[100];
} Thread;
```

Структура `ProcessList` представляет собой список процессов, содержащий указатель на массив структур `Process` и количество элементов в этом массиве. Эта структура используется для хранения списка процессов, полученного из системы.

```
typedef struct {
    Process* processes;
    int num_processes;
} ProcessList;
```

Структура ThreadList представляет собой список потоков, содержащий указатель на массив структур Thread и количество элементов в этом массиве. Эта структура используется для хранения списка потоков, полученного из системы.

```
typedef struct {
    Thread* threads;
    int num_threads;
} ThreadList;
```

Структура SysInfo содержит информацию о системе, включая общее количество процессов, количество запущенных процессов, количество потоков, количество пользователей, среднюю загрузку системы за разное время, процент использования процессора пользователями и системой, и т.д. Эта структура используется для хранения информации о системе, полученной из системы.

```
typedef struct {
    int total_processes;
    int running_processes;
    int total_threads;
    int num_users;
    float load_avg[3];
    float cpu_usage_user;
    float cpu_usage_system;
} SysInfo;
```

Эти структуры данных используются для хранения и обработки информации о процессах, потоках и системе в программе. Они являются основой для реализации функционала программы, такого как отображение списка процессов и потоков, сортировка и фильтрация списка, управление процессами и потоками, и т.д.

## **3.2 Описание основных функций программы**

### **3.2.1 Файл control.h**

Файл содержит прототипы функций, используемых для управления программой. Эти функции обрабатывают ввод пользователя и сигналы, переключают режим отображения, убивают процессы и потоки, а также сортируют список процессов или потоков по выбранному критерию.

`void handle_signal(int signal)` - это обработчик сигналов. Он регистрируется при запуске программы и вызывается при получении сигнала SIGINT (Ctrl+C). В данной функции выводится сообщение "Программа завершена." и выполняется завершение программы.

`void handle_input(Process *processes, int num_processes, Thread *threads, int num_threads)` - это функция обработки ввода пользователя. Она проверяет, есть ли доступные символы во входном буфере, и, если они есть, считывает их и выполняет соответствующие действия. Например, если пользователь нажал клавишу 'z', то функция переключает режим цвета, если нажата клавиша 't', то она переключает режим отображения на потоки, и т.д.

`void switch_display_mode()` - это функция переключения режима отображения. Если текущий режим отображения - процессы, то она переключает его на потоки, и наоборот.

`void kill_process_by_pid(int pid)` - это функция убивает процесс по его идентификатору. Она находит процесс в списке процессов по его PID и вызывает системную функцию `kill()` для его удаления.

`void kill_thread_by_tid(int tid)` - это функция убивает поток по его идентификатору. Она находит поток в списке потоков по его TID и вызывает системную функцию `pthread_cancel()` для его удаления.

`void sort_by_criterion()` - это функция сортировки списка процессов или потоков по выбранному критерию. Она выводит сообщение "Введите критерий для сортировки (pid/tid, user, priority, virtual\_memory, resident\_memory, cpu\_usage): " и считывает критерий сортировки. Затем она вызывает соответствующую функцию сортировки из файла `sort.c`.

### 3.2.2 Файл `display.h`

Этот файл содержит прототипы функций, используемых для отображения информации о системе, процессах и потоках.

`void display_sysinfo(SysInfo *sysinfo)` - это функция отображения общих сведений о системе, таких как общее количество процессов, количество активных процессов, общее количество потоков, количество пользователей, средняя нагрузка на систему (за последние 1, 5 и 15 минут), использование ЦП пользователем и системой.

`void display_processes(Process *processes, int num_processes)` - это функция отображения списка процессов. Она выводит заголовок таблицы с информацией о процессах и затем выводит информацию о каждом процессе в отдельной строке таблицы.

`void display_threads(Thread *threads, int num_threads)` - это функция отображения списка потоков. Она выводит заголовок таблицы с информацией о потоках и затем выводит информацию о каждом потоке в отдельной строке таблицы.

`void display_process_info(Process *process)` - это функция отображения подробной информации о процессе. Она выводит информацию о процессе в формате "Имя пользователя: [имя пользователя], Приоритет: [приоритет], Виртуальная память: [виртуальная память] КБ, Физическая память: [физическая память] КБ, CPU: [CPU]%, Команда: [команда]".

`void display_thread_info(Thread *thread)` - это функция отображения подробной информации о потоке. Она выводит информацию о потоке в формате "Имя пользователя: [имя пользователя], Приоритет: [приоритет], Виртуальная память: [виртуальная память] КБ, Физическая память: [физическая память] КБ, CPU: [CPU]%, Команда: [команда]".

### 3.2.3 Файл `read.h`

Этот файл содержит прототипы функций, используемых для чтения информации о системе, процессах и потоках.

`void read_sysinfo(SysInfo *sysinfo)` - это функция чтения общих сведений о системе. Она заполняет структуру `sysinfo` информацией о количестве процессов, количестве активных процессов, количестве потоков, количестве пользователей, средней нагрузке на систему (за последние 1, 5 и 15 минут), использовании ЦП пользователем и системой.

`void read_processes(Process *processes, int *num_processes)` - это функция чтения списка процессов. Она заполняет массив `processes` информацией о процессах и записывает количество процессов в переменную `num_processes`.

`void read_threads(Thread *threads, int *num_threads)` - это функция чтения списка потоков. Она заполняет массив `threads` информацией о потоках и записывает количество потоков в переменную `num_threads`.

`void read_process_info(Process *process, int pid)` - это функция чтения подробной информации о процессе. Она заполняет структуру `process` информацией о процессе с заданным PID.

`void read_thread_info(Thread *thread, int tid)` - это функция чтения подробной информации о потоке. Она заполняет структуру `thread` информацией о потоке с заданным TID.

### 3.2.4 Файл `sort.h`

Этот файл содержит прототипы функций, используемых для сортировки списка процессов или потоков по различным критериям.

`void sort_processes_by_pid(Process *processes, int num_processes)` - это функция сортировки списка процессов по идентификатору процесса (PID).

`void sort_processes_by_user(Process *processes, int num_processes)` - это функция сортировки списка процессов по имени пользователя.

`void sort_processes_by_priority(Process *processes, int num_processes)` - это функция сортировки списка процессов по приоритету.

`void sort_processes_by_virtual_memory(Process *processes, int num_processes)` - это функция сортировки списка процессов по потреблению виртуальной памяти.

`void sort_processes_by_resident_memory(Process *processes, int num_processes)` - это функция сортировки списка процессов по потреблению физической памяти.

`void sort_processes_by_cpu_usage(Process *processes, int num_processes)` - это функция сортировки списка процессов по использованию ЦП.

`void sort_threads_by_pid(Thread *threads, int num_threads)` - это функция сортировки списка потоков по идентификатору процесса (PID).

`void sort_threads_by_user(Thread *threads, int num_threads)` - это функция сортировки списка потоков по имени пользователя.

`void sort_threads_by_priority(Thread *threads, int num_threads)` - это функция сортировки списка потоков по приоритету.

`void sort_threads_by_virtual_memory(Thread *threads, int num_threads)` - это функция сортировки списка потоков по потреблению виртуальной памяти.

`void sort_threads_by_resident_memory(Thread *threads, int num_threads)` - это функция сортировки списка потоков по потреблению физической памяти.

`void sort_threads_by_cpu_usage(Thread *threads, int num_threads)` - это функция сортировки списка потоков по использованию ЦП.

## 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.

### 4.2 Схемы алгоритмов

#### 4.2.1 Схема алгоритма `switch_color_mode`

Функция `switch_color_mode` переключает режим отображения цвета текста в консоли. Схема алгоритма `switch_color_mode` приведена в приложении Б.

#### 4.2.2 Схема алгоритма `kill_process_by_pid`

Функция `kill_process_by_pid` убивает процесс с указанным идентификатором (PID). Она принимает три аргумента. `Process *processes` - указатель на массив структур `Process`, содержащий информацию о процессах. `int num_processes` - количество процессов в массиве `processes`. `int pid` - идентификатор процесса, который нужно убить. Схема алгоритма `kill_process_by_pid` приведена в приложении В.

### 4.3 Разработка алгоритмов

#### 4.3.1 Алгоритм функции `read_sysinfo`

Функция `read_sysinfo()` отвечает за чтение информации о системе и сохранение ее в структуре `SysInfo`. Она использует системные вызовы для получения информации о системе, такие как `sysinfo()` и `getloadavg()`. Шаги выполнения этой функции:

1. Объявляется структура `SysInfo` с именем `sysinfo` и выделяется память для нее с помощью функции `malloc()`.
2. Открывается файл `/proc/stat` с помощью функции `fopen()`. Этот файл содержит информацию о системе, включая количество процессов, количество потоков, использование ЦП и т.д.
3. Считывается первая строка из файла `/proc/stat` с помощью функции `fgets()`. Эта строка содержит информацию о использовании ЦП.



4. Извлекается информация о использовании ЦП из считанной строки с помощью функции `sscanf()`. Эта информация сохраняется в структуре `sysinfo`.

5. Закрывается файл `/proc/stat` с помощью функции `fclose()`.

6. Открывается файл `/proc/loadavg` с помощью функции `fopen()`. Этот файл содержит информацию о средней нагрузке на систему.

7. Считывается первая строка из файла `/proc/loadavg` с помощью функции `fgets()`. Эта строка содержит информацию о средней нагрузке на систему.

8. Извлекается информация о средней нагрузке на систему из считанной строки с помощью функции `sscanf()`. Эта информация сохраняется в структуре `sysinfo`.

9. Закрывается файл `/proc/loadavg` с помощью функции `fclose()`.

10. Открывается файл `/proc/meminfo` с помощью функции `fopen()`. Этот файл содержит информацию о использовании памяти.

11. Считываются строки из файла `/proc/meminfo` с помощью функции `fgets()`, пока не будет найдена строка, содержащая информацию о количестве свободной памяти.

12. Извлекается информация о количестве свободной памяти из считанной строки с помощью функции `sscanf()`. Эта информация сохраняется в структуре `sysinfo`.

13. Закрывается файл `/proc/meminfo` с помощью функции `fclose()`.

14. Возвращается указатель на структуру `sysinfo`.

### 4.3.2 Алгоритм функции `read_processes`

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. Объявляется структура `Process` с именем `processes` и выделяется память для нее с помощью функции `malloc()`.

2. Объявляется переменная `num_processes` типа `int` и инициализируется значением 0. Эта переменная будет использоваться для хранения количества процессов.

3. Открывается каталог `/proc` с помощью функции `opendir()`.

4. Создается цикл, который проходит через все файлы в каталоге `/proc`.

5. Внутри цикла проверяется, является ли текущий файл директорией, соответствующей процессу. Для этого используется функция `isdigit()`, которая проверяет, состоит ли имя файла из цифр (идентификатор процесса).

6. Если текущий файл является директорией процесса, то открывается файл `/proc/[pid]/stat`, где `[pid]` - это идентификатор процесса. Этот файл содержит

информацию о процессе, такую как идентификатор процесса, имя пользователя, приоритет, использование памяти и т.д.

7. Из файла `/proc/[pid]/stat` считывается информация о процессе с помощью функции `fscanf()`. Эта информация сохраняется в структуре `Process`, которая была определена ранее.

8. После того, как информация о процессе была считана и сохранена, закрывается файл `/proc/[pid]/stat` с помощью функции `fclose()`.

9. Цикл продолжает проходить через все файлы в каталоге `/proc`, пока не будут просмотрены все файлы.

10. После того, как все процессы были прочитаны и сохранены в структуре `Process`, закрывается каталог `/proc` с помощью функции `closedir()`.

11. Возвращается указатель на структуру `Process` и количество прочитанных процессов.

## **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**



### **5.3 Код программы**

Код программы представлен в приложении Г.

## ЗАКЛЮЧЕНИЕ

В рамках курсового проекта была разработана программа диспетчера процессов и потоков, представляющая собой аналог утилиты `top`, которая предоставляет пользователю важный инструмент для мониторинга и анализа 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, приоритет, потребление памяти и ресурсов процессора, какой командой был запущен процесс или поток, а так же общую, подробную информацию о системе. 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. 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.

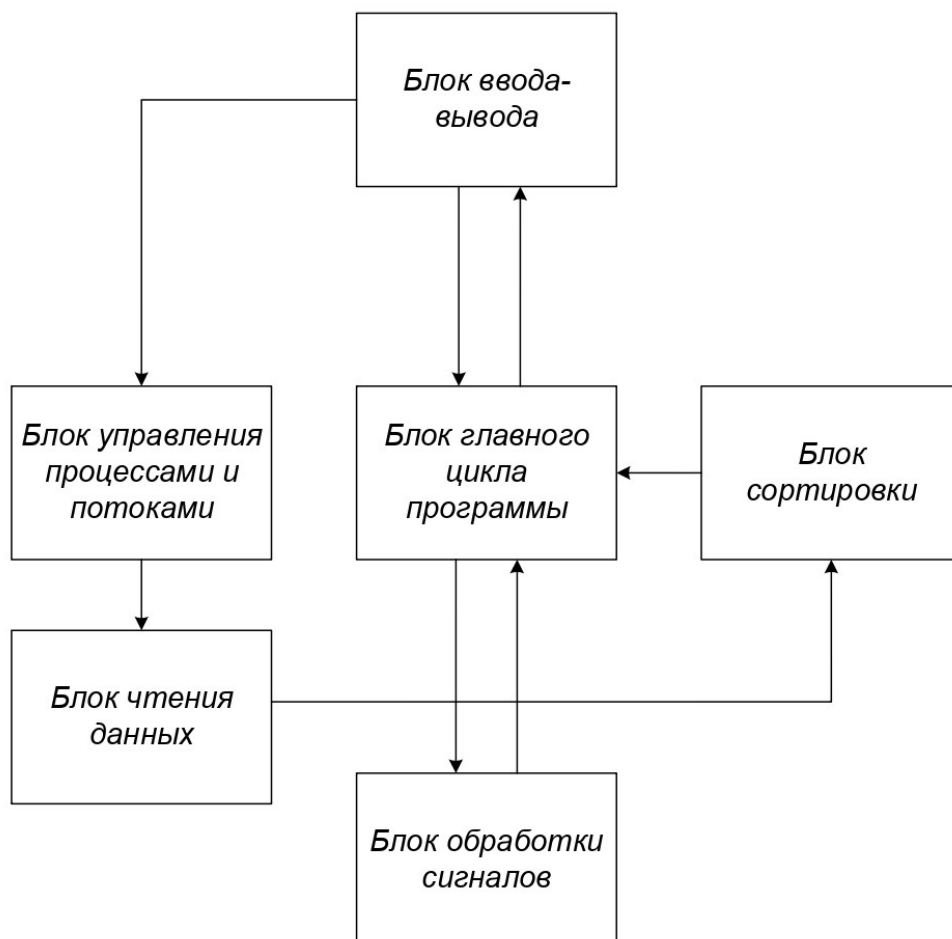
The C Programming Language. Издательство: Prentice Hall, 1988 г.

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



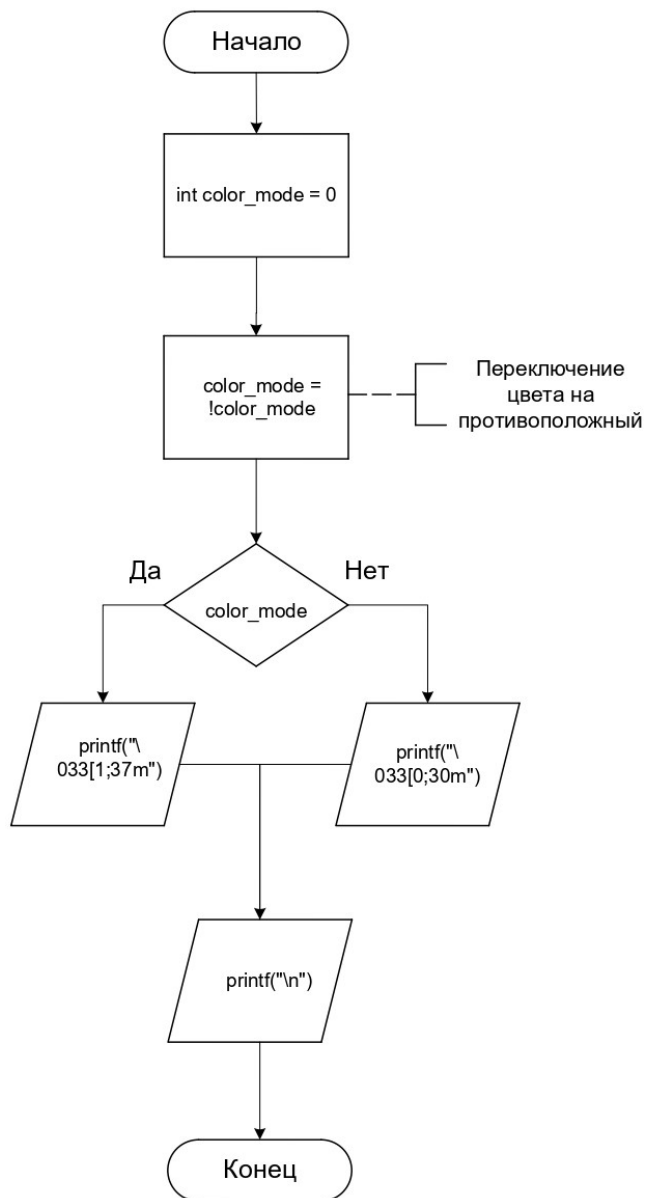
|         |      |           |       |      |  |                |  |          |         |  |
|---------|------|-----------|-------|------|--|----------------|--|----------|---------|--|
|         |      |           |       |      | ГУИР.400201.118 С1                                 |                |  |          |         |  |
|         |      |           |       |      | Диспетчер процессов и потоков<br>Структурная схема | Лит.           |  | Масса    | Масштаб |  |
| Изм.    | Лист | № докум.  | Подп. | Дата |  |                |  |          |         |  |
| Разраб. |      | Снитко    |       |      |  | у              |  |          |         |  |
| Пров.   |      | Игнатович |       |      |  |                |  |          |         |  |
|         |      |           |       |      |  |                |  |          |         |  |
|         |      |           |       |      |  | Лист           |  | Листов 1 |         |  |
|         |      |           |       |      |  | ЭВМ, ар.250501 |  |          |         |  |

## **ПРИЛОЖЕНИЕ Б**

(Обязательное)

Схема алгоритма `switch_color_mode`

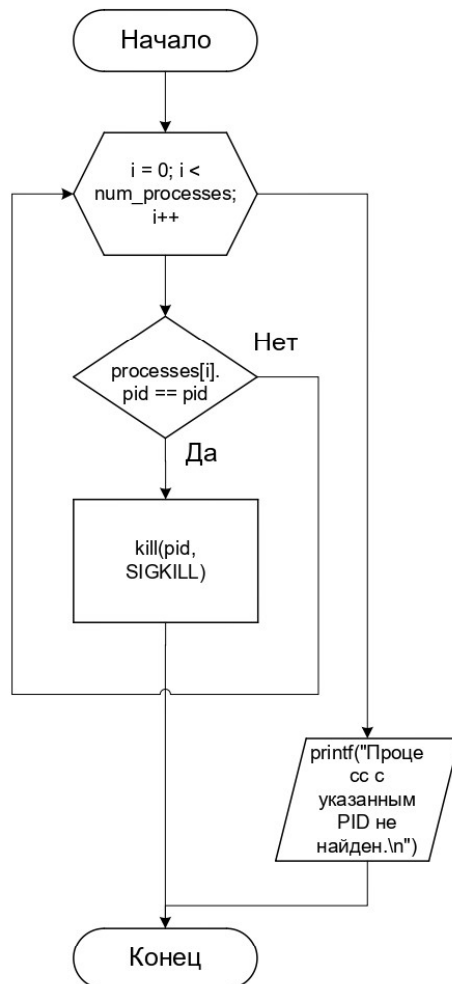




|         |           |          |       |      |  |                 |          |         |
|---------|-----------|----------|-------|------|--|-----------------|----------|---------|
|         |           |          |       |      | ГУИР.400201.118 Б                            |                 |          |         |
|         |           |          |       |      | Схема алгоритма функции<br>switch_color_mode | Лит.            | Масса    | Масштаб |
| Изм.    | Лист      | № докум. | Подп. | Дата |  | У               |          |         |
| Разраб. | Снитко    |          |       |      |  |                 |          |         |
| Пров.   | Игнатович |          |       |      |  |                 |          |         |
|         |           |          |       |      |  | Лист            | Листов 1 |         |
|         |           |          |       |      |  | ЭВМ, гр. 250501 |          |         |

**ПРИЛОЖЕНИЕ В**  
(Обязательное)

Схема алгоритма `kill_process_by_pid`



|         |      |           |       |      |  |                 |          |         |
|---------|------|-----------|-------|------|--|-----------------|----------|---------|
|         |      |           |       |      | ГУИР.400201.118 В                              |                 |          |         |
|         |      |           |       |      | Схема алгоритма функции<br>kill_process_by_pid | Лит.            | Масса    | Масштаб |
| Изм.    | Лист | № докум.  | Подп. | Дата |  | у               |          |         |
| Разраб. |      | Снитко    |       |      |  |                 |          |         |
| Пров.   |      | Игнатович |       |      |  |                 |          |         |
|         |      |           |       |      |  | Лист            | Листов 1 |         |
|         |      |           |       |      |  | ЭВМ, гр. 250501 |          |         |

## ПРИЛОЖЕНИЕ Г (Обязательное)

### Код программы

```
#ifndef CONTROL_H
#define CONTROL_H

#include "process.h"
#include "threads.h"

void handle_signal(int signal);
void handle_input(Process *processes, int num_processes, Thread *threads, int num_threads);
void switch_color_mode();
void kill_process_by_pid(int pid);
void kill_thread_by_tid(int tid);
void sort_by_criterion();

#endif /* CONTROL_H */
#include "control.h"
#include "display.h"
#include "input.h"
#include "sort.h"
#include "read.h"
#include "sysinfo.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <termios.h>
#include <fcntl.h>
#include "process.h"
#include "threads.h"

#define MAX_PROC 1000
#define MAX_THREADS 1000

Process processes[MAX_PROC];
Thread threads[MAX_THREADS];

int num_processes = 0;
int num_threads = 0;

int display_mode = 0; // 0 - процессы, 1 - потоки

void handle_signal(int signal) {
    if (signal == SIGINT) {
        printf("Программа завершена.\n");
        exit(0);
    }
}

void handle_input(Process *processes, int num_processes, Thread *threads, int num_threads) {
    if (kbhit()) {
        char c = getchar();
        switch (c) {
            case 'z':
                switch_color_mode();
                break;
            case 't':
                display_mode = 1;
                break;
            case 'p':
                display_mode = 0;
                break;
            case 'k': {
                int id;
                char prompt[50];
                if (display_mode == 0) {
                    sprintf(prompt, "Введите PID процесса для удаления: ");
                } else {
                    sprintf(prompt, "Введите TID потока для удаления: ");
                }
                printf("%s", prompt);
                scanf("%d", &id);
                if (display_mode == 0) {
```

```

        kill_process_by_pid(processes, num_processes, id);
    } else {
        kill_thread_by_tid(threads, num_threads, id);
    }
    break;
}
case 'q':
    handle_signal(SIGINT);
    break;
case 's': {
    char criterion[50];
    printf("Введите критерий для сортировки (pid/tid, user, priority, virtual_memory,
resident_memory, cpu_usage): ");
    scanf("%s", criterion);
    if (display_mode == 0) {
        if (strcmp(criterion, "pid") == 0) {
            sort_processes_by_pid(processes, num_processes);
        } else if (strcmp(criterion, "user") == 0) {
            sort_processes_by_user(processes, num_processes);
        } else if (strcmp(criterion, "priority") == 0) {
            sort_processes_by_priority(processes, num_processes);
        } else if (strcmp(criterion, "virtual_memory") == 0) {
            sort_processes_by_virtual_memory(processes, num_processes);
        } else if (strcmp(criterion, "resident_memory") == 0) {
            sort_processes_by_resident_memory(processes, num_processes);
        } else if (strcmp(criterion, "cpu_usage") == 0) {
            sort_processes_by_cpu_usage(processes, num_processes);
        } else {
            printf("Неверный критерий сортировки.\n");
        }
    } else {
        if (strcmp(criterion, "tid") == 0) {
            sort_threads_by_pid(threads, num_threads);
        } else if (strcmp(criterion, "user") == 0) {
            sort_threads_by_user(threads, num_threads);
        } else if (strcmp(criterion, "priority") == 0) {
            sort_threads_by_priority(threads, num_threads);
        } else if (strcmp(criterion, "virtual_memory") == 0) {
            sort_threads_by_virtual_memory(threads, num_threads);
        } else if (strcmp(criterion, "resident_memory") == 0) {
            sort_threads_by_resident_memory(threads, num_threads);
        } else if (strcmp(criterion, "cpu_usage") == 0) {
            sort_threads_by_cpu_usage(threads, num_threads);
        } else {
            printf("Неверный критерий сортировки.\n");
        }
    }
    break;
}
default:
    break;
}
}

void switch_color_mode() {
    static int color_mode = 0;
    color_mode = !color_mode;
    if (color_mode) {
        printf("\033[1;37m"); // white on black
    } else {
        printf("\033[0;30m"); // black on white
    }
    printf("\n");
}

void kill_process_by_pid(Process *processes, int num_processes, int pid) {
    // find process with given PID and kill it
    for (int i = 0; i < num_processes; i++) {
        if (processes[i].pid == pid) {
            kill(pid, SIGKILL);
            return;
        }
    }
    printf("Процесс с указанным PID не найден.\n");
}

void kill_thread_by_tid(Thread *threads, int num_threads, int tid) {

```

```

    // find thread with given TID and kill it
    for (int i = 0; i < num_threads; i++) {
        if (threads[i].tid == tid) {
            pthread_cancel(threads[i].tid);
            return;
        }
    }
    printf("Поток с указанным TID не найден.\n");
}

#include "display.h"
#include "sysinfo.h"
#include <stdio.h>

void display_sysinfo(SysInfo *sysinfo) {
    printf("Общее количество процессов: %d\n", sysinfo->total_processes);
    printf("Количество активных процессов: %d\n", sysinfo->running_processes);
    printf("Общее количество потоков: %d\n", sysinfo->total_threads);
    printf("Общее количество пользователей: %d\n", sysinfo->num_users);
    printf("Средняя нагрузка на систему (за последние 1, 5 и 15 минут): %.2f, %.2f, %.2f\n",
sysinfo->load_avg[0], sysinfo->load_avg[1], sysinfo->load_avg[2]);
    printf("Использование ЦП пользователем: %.2f%%\n", sysinfo->cpu_usage_user);
    printf("Использование ЦП системой: %.2f%%\n", sysinfo->cpu_usage_system);
}

void display_processes(Process *processes, int num_processes) {
    printf("%-8s %-15s %-8s %-12s %-12s %-8s %-8s %s\n",
        "PID", "USER", "PRIORITY", "VM(KB)", "RM(KB)", "CPU(%)", "COMMAND");

    for (int i = 0; i < num_processes; ++i) {
        printf("%-8d %-15s %-8d %-12ld %-12ld %-8.2f %-8s\n",
            processes[i].pid, processes[i].user, processes[i].priority,
            processes[i].virtual_memory, processes[i].resident_memory,
            processes[i].cpu_usage, processes[i].command);
    }
}

void display_threads(Thread *threads, int num_threads) {
    printf("%-8s %-15s %-8s %-12s %-12s %-8s %-8s %s\n",
        "PID", "USER", "TID", "VM(KB)", "RM(KB)", "CPU(%)", "COMMAND");

    for (int i = 0; i < num_threads; ++i) {
        printf("%-8d %-15s %-8d %-12ld %-12ld %-8.2f %-8s\n",
            threads[i].pid, threads[i].user, threads[i].tid,
            threads[i].virtual_memory, threads[i].resident_memory,
            threads[i].cpu_usage, threads[i].command);
    }
}

#ifdef DISPLAY_H
#define DISPLAY_H

#include "process.h"

void display_sysinfo(SysInfo *sysinfo);
void display_processes(Process *processes, int num_processes);
void display_threads(Thread *threads, int num_threads);

#endif /* DISPLAY_H */

#include "read.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <dirent.h>
#include <unistd.h>
#include <errno.h>

void read_sysinfo(SysInfo *sysinfo) {
    struct sysinfo info;
    if (sysinfo_mem_info(&info) == 0) {
        sysinfo->total_processes = info.procs;
        sysinfo->running_processes = info.procs_running;
        sysinfo->total_threads = info.totalhigh;
        sysinfo->num_users = info.users;
        sysinfo->load_avg[0] = info.loads[0];
        sysinfo->load_avg[1] = info.loads[1];
        sysinfo->load_avg[2] = info.loads[2];
    }
}

```

```

        sysinfo->cpu_usage_user = info.totalcpu * 100.0 / (1 << SI_CPU_SHIFT);
        sysinfo->cpu_usage_system = info.totalsystemcpu * 100.0 / (1 << SI_CPU_SHIFT);
    } else {
        printf("Ошибка чтения информации о системе\n");
    }
}

int read_processes(Process *processes) {
    DIR *dir;
    struct dirent *entry;
    int count = 0;

    if ((dir = opendir("/proc")) == NULL) {
        perror("Ошибка opendir()");
        return -1;
    }

    while ((entry = readdir(dir)) != NULL && count < MAX_PROC) {
        if (isdigit(*entry->d_name)) {
            char path[256];
            sprintf(path, "/proc/%s/status", entry->d_name);

            FILE *fp;
            if ((fp = fopen(path, "r")) != NULL) {
                char line[256];
                while (fgets(line, sizeof(line), fp)) {
                    if (strncmp(line, "Pid:", 4) == 0) {
                        sscanf(line, "%s %d", &processes[count].pid);
                    } else if (strncmp(line, "Uid:", 4) == 0) {
                        sscanf(line, "%s %s", &processes[count].user);
                    } else if (strncmp(line, "VmSize:", 7) == 0) {
                        sscanf(line, "%s %ld", &processes[count].virtual_memory);
                    } else if (strncmp(line, "VmRSS:", 6) == 0) {
                        sscanf(line, "%s %ld", &processes[count].resident_memory);
                    } else if (strncmp(line, "Cpu(s):", 7) == 0) {
                        sscanf(line, "%s %lf", &processes[count].cpu_usage);
                    } else if (strncmp(line, "Cmdline:", 8) == 0) {
                        sscanf(line, "%s %[^\\n]", &processes[count].command);
                    }
                }
                fclose(fp);
                count++;
            }
        }
    }

    closedir(dir);
    return count;
}

int read_threads(Thread *threads) {
    DIR *dir;
    struct dirent *entry;
    int count = 0;

    if ((dir = opendir("/proc")) == NULL) {
        perror("Ошибка opendir()");
        return -1;
    }

    while ((entry = readdir(dir)) != NULL && count < MAX_THREADS) {
        if (isdigit(*entry->d_name)) {
            char path[256];
            sprintf(path, "/proc/%s/task/%s/status", entry->d_name, entry->d_name);

            FILE *fp;
            if ((fp = fopen(path, "r")) != NULL) {
                char line[256];
                while (fgets(line, sizeof(line), fp)) {
                    if (strncmp(line, "Pid:", 4) == 0) {
                        sscanf(line, "%s %d", &threads[count].pid);
                    } else if (strncmp(line, "Tid:", 4) == 0) {
                        sscanf(line, "%s %d", &threads[count].tid);
                    } else if (strncmp(line, "Uid:", 4) == 0) {
                        sscanf(line, "%s %s", &threads[count].user);
                    } else if (strncmp(line, "VmSize:", 7) == 0) {
                        sscanf(line, "%s %ld", &threads[count].virtual_memory);
                    } else if (strncmp(line, "VmRSS:", 6) == 0) {

```

```

        sscanf(line, "%s %ld", &threads[count].resident_memory);
    } else if (strncmp(line, "Cpu(s):", 7) == 0) {
        sscanf(line, "%s %lf", &threads[count].cpu_usage);
    } else if (strncmp(line, "Cmdline:", 8) == 0) {
        sscanf(line, "%s %[^\\n]", threads[count].command);
    }
}
fclose(fp);
count++;
}
}

closedir(dir);
return count;
}

#include "control.h"
#include "display.h"
#include "sysinfo.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

int main() {
    // Установка обработчика сигнала SIGINT (Ctrl+C)
    signal(SIGINT, handle_signal);

    SysInfo sysinfo;

    // Чтение данных о системе
    read_sysinfo(&sysinfo);

    // Чтение данных при запуске
    num_processes = read_processes(processes);
    if (num_processes < 0) {
        fprintf(stderr, "Ошибка чтения процессов\\n");
        return 1;
    }

    num_threads = read_threads(threads);
    if (num_threads < 0) {
        fprintf(stderr, "Ошибка чтения потоков\\n");
        return 1;
    }

    // Бесконечный цикл обновления данных
    while (1) {
        // Очистка экрана перед обновлением информации
        system("clear");

        // Отображение информации о системе
        display_sysinfo(&sysinfo);

        // Чтение данных о процессах и потоках
        num_processes = read_processes(processes);
        if (num_processes < 0) {
            fprintf(stderr, "Ошибка чтения процессов\\n");
            return 1;
        }

        num_threads = read_threads(threads);
        if (num_threads < 0) {
            fprintf(stderr, "Ошибка чтения потоков\\n");
            return 1;
        }

        // Отображение информации о процессах или потоках в зависимости от выбранного режима
        if (display_mode == 0) {
            display_processes(processes, num_processes);
        } else {
            display_threads(threads, num_threads);
        }

        // Ожидание 1 секунда перед следующим обновлением данных
        sleep(1);
    }
}

```



```

        // Обработка нажатия клавиш
        handle_input(processes, num_processes, threads, num_threads);
    }

    return 0;
}

#ifdef PROCESS_H
#define PROCESS_H

#define MAX_PROC 1000

typedef struct {
    int pid;                // Идентификатор процесса
    char user[50];          // Имя пользователя
    int priority;           // Приоритет процесса
    long virtual_memory;    // Потребление виртуальной памяти (КВ)
    long resident_memory;   // Потребление физической памяти (КВ)
    float cpu_usage;        // % использования CPU
    char command[100];      // Название команды, инициализировавшей процесс
} Process;

#endif /* PROCESS_H */

#ifdef READ_H
#define READ_H

#include "process.h"
#include "threads.h"

void read_sysinfo(SysInfo *sysinfo);
int read_processes(Process *processes);
int read_threads(Thread *threads);

#endif /* READ_H */

#include "sort.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int compare_processes_by_pid(const void *a, const void *b) {
    const Process *process_a = a;
    const Process *process_b = b;
    return process_a->pid - process_b->pid;
}

int compare_processes_by_user(const void *a, const void *b) {
    const Process *process_a = a;
    const Process *process_b = b;
    return strcmp(process_a->user, process_b->user);
}

int compare_processes_by_priority(const void *a, const void *b) {
    const Process *process_a = a;
    const Process *process_b = b;
    return process_a->priority - process_b->priority;
}

int compare_processes_by_virtual_memory(const void *a, const void *b) {
    const Process *process_a = a;
    const Process *process_b = b;
    return process_a->virtual_memory - process_b->virtual_memory;
}

int compare_processes_by_resident_memory(const void *a, const void *b) {
    const Process *process_a = a;
    const Process *process_b = b;
    return process_a->resident_memory - process_b->resident_memory;
}

int compare_processes_by_cpu_usage(const void *a, const void *b) {
    const Process *process_a = a;
    const Process *process_b = b;
    return (int)(process_a->cpu_usage - process_b->cpu_usage);
}

void sort_processes_by_pid(Process *processes, int num_processes) {
    qsort(processes, num_processes, sizeof(Process), compare_processes_by_pid);
}

```

```

}

void sort_processes_by_user(Process *processes, int num_processes) {
    qsort(processes, num_processes, sizeof(Process), compare_processes_by_user);
}

void sort_processes_by_priority(Process *processes, int num_processes) {
    qsort(processes, num_processes, sizeof(Process), compare_processes_by_priority);
}

void sort_processes_by_virtual_memory(Process *processes, int num_processes) {
    qsort(processes, num_processes, sizeof(Process), compare_processes_by_virtual_memory);
}

void sort_processes_by_resident_memory(Process *processes, int num_processes) {
    qsort(processes, num_processes, sizeof(Process), compare_processes_by_resident_memory);
}

void sort_processes_by_cpu_usage(Process *processes, int num_processes) {
    qsort(processes, num_processes, sizeof(Process), compare_processes_by_cpu_usage);
}

int compare_threads_by_pid(const void *a, const void *b) {
    const Thread *thread_a = a;
    const Thread *thread_b = b;
    return thread_a->pid - thread_b->pid;
}

int compare_threads_by_user(const void *a, const void *b) {
    const Thread *thread_a = a;
    const Thread *thread_b = b;
    return strcmp(thread_a->user, thread_b->user);
}

int compare_threads_by_priority(const void *a, const void *b) {
    const Thread *thread_a = a;
    const Thread *thread_b = b;
    return thread_a->priority - thread_b->priority;
}

int compare_threads_by_virtual_memory(const void *a, const void *b) {
    const Thread *thread_a = a;
    const Thread *thread_b = b;
    return thread_a->virtual_memory - thread_b->virtual_memory;
}

int compare_threads_by_resident_memory(const void *a, const void *b) {
    const Thread *thread_a = a;
    const Thread *thread_b = b;
    return thread_a->resident_memory - thread_b->resident_memory;
}

int compare_threads_by_cpu_usage(const void *a, const void *b) {
    const Thread *thread_a = a;
    const Thread *thread_b = b;
    return (int)(thread_a->cpu_usage - thread_b->cpu_usage);
}

void sort_threads_by_pid(Thread *threads, int num_threads) {
    qsort(threads, num_threads, sizeof(Thread), compare_threads_by_pid);
}

void sort_threads_by_user(Thread *threads, int num_threads) {
    qsort(threads, num_threads, sizeof(Thread), compare_threads_by_user);
}

void sort_threads_by_priority(Thread *threads, int num_threads) {
    qsort(threads, num_threads, sizeof(Thread), compare_threads_by_priority);
}

void sort_threads_by_virtual_memory(Thread *threads, int num_threads) {
    qsort(threads, num_threads, sizeof(Thread), compare_threads_by_virtual_memory);
}

void sort_threads_by_resident_memory(Thread *threads, int num_threads) {
    qsort(threads, num_threads, sizeof(Thread), compare_threads_by_resident_memory);
}

```

```

void sort_threads_by_cpu_usage(Thread *threads, int num_threads) {
    qsort(threads, num_threads, sizeof(Thread), compare_threads_by_cpu_usage);
}

#ifndef SORT_H
#define SORT_H

#include "process.h"
#include "threads.h"

void sort_processes_by_pid(Process *processes, int num_processes);
void sort_processes_by_user(Process *processes, int num_processes);
void sort_processes_by_priority(Process *processes, int num_processes);
void sort_processes_by_virtual_memory(Process *processes, int num_processes);
void sort_processes_by_resident_memory(Process *processes, int num_processes);
void sort_processes_by_cpu_usage(Process *processes, int num_processes);

void sort_threads_by_pid(Thread *threads, int num_threads);
void sort_threads_by_user(Thread *threads, int num_threads);
void sort_threads_by_priority(Thread *threads, int num_threads);
void sort_threads_by_virtual_memory(Thread *threads, int num_threads);
void sort_threads_by_resident_memory(Thread *threads, int num_threads);
void sort_threads_by_cpu_usage(Thread *threads, int num_threads);

#endif /* SORT_H */

// sysinfo.h
#ifndef SYSINFO_H
#define SYSINFO_H

typedef struct {
    int total_processes; // общее количество процессов
    int running_processes; // количество активных процессов
    int total_threads; // общее количество потоков
    int num_users; // количество пользователей
    float load_avg[3]; // средняя нагрузка на систему за последние 1, 5 и 15 минут
    float cpu_usage_user; // использование ЦП пользователем
    float cpu_usage_system; // использование ЦП системой
} SysInfo;

#endif // SYSINFO_H

#ifndef THREADS_H
#define THREADS_H

#define MAX_THREADS 1000

typedef struct {
    int pid; // Идентификатор процесса
    int tid; // Идентификатор потока
    char user[50]; // Имя пользователя
    int priority; // Приоритет потока
    long virtual_memory; // Потребление виртуальной памяти (KB)
    long resident_memory; // Потребление физической памяти (KB)
    float cpu_usage; // % использования CPU
    char command[100]; // Название команды, инициализировавшей процесс
} Thread;

#endif /* THREADS_H */

```

**ПРИЛОЖЕНИЕ Д**  
**(Обязательное)**

Ведомость документов

[illegible]

|         |           |             |         |      |   |                 |  |          |         |
|---------|-----------|-------------|---------|------|---|-----------------|--|----------|---------|
|         |           |             |         |      | ГУИР.400201.118 Д1                                    |                 |  |          |         |
|         |           |             |         |      | Диспетчер процессов и потоков<br>Ведомость документов | Лит.            |  | Масса    | Масштаб |
| Изм.    | Лист      | № документа | Подпись | Дата |   | У               |  |          |         |
| Разраб. | Снитко    |             |         |      |   |                 |  |          |         |
| Пров.   | Игнатович |             |         |      |   |                 |  |          |         |
|         |           |             |         |      |   | Лист            |  | Листов 1 |         |
|         |           |             |         |      |   | ЭВМ, зр. 250501 |  |          |         |
|         |           |             |         |      |   |                 |  |          |         |