**ВВЕдение**

CPU является одним из ключевых компонентов компьютера и играет важную роль в его работе. CPU обрабатывает данные, выполняет инструкции и управляет всеми операциями в компьютере.

CPU является "мозгом" компьютера, который обеспечивает выполнение всех задач, от запуска операционной системы до запуска приложений и выполнения сложных вычислений. Более мощный процессор позволяет быстрее выполнять задачи и работать с более сложными приложениями.

В современных компьютерных системах, особенно в условиях многозадачности и высокой производительности, мониторинг загруженности CPU играет важную роль в обеспечении стабильной работы и предотвращении проблем с производительностью. Он позволяет отслеживать уровень нагрузки на центральный процессор, идентифицировать возможные узкие места и проблемы в работе системы, а также принимать меры для оптимизации ее работы.

Среди преимуществ мониторинга нагрузки CPU можно выделить возможность быстро обнаруживать проблемы, повышать эффективность и производительность системы, а также улучшать ее стабильность. Однако, также существуют и некоторые недостатки, такие как нагрузка на систему при проведении мониторинга, сложность анализа большого объема данных и трудность в подборе наиболее подходящих инструментов для конкретной системы.

В данной работе будет рассмотрены технологии позволяющие отслеживать загруженность CPU, а также будет создана программа для отслеживания загруженности CPU.

**1 Обзор литературы**

* 1. **Определения**

**определение 1.** Мониторинг ­– это процесс сбора/регистрации, хранения и анализа некоторого количества информации по определенным (ключевым) параметрам.

**Определение 2.** CPU (Central Processing Unit) – Центральный процессор – основной элемент аппаратного обеспечения цифровых устройств, который используется для обработки информации.

**Определение 3.** Утилита – небольшая вспомогательная программа для решения специализированных задач по настройке, оптимизации, улучшению работы оборудования и программного обеспечения.

**Определение 4.** RAM (Random Access Memory) – Энергозависимая часть компьютерной памяти, во время работы компьютера хранится выполняемый машинный код (программы), а также входные, выходные данные.

**Определение 5.** PID (Process Identifier) – это уникальный идентификатор процесса в операционной системе. Каждый процесс, запущенный на компьютере, имеет свой уникальный PID, который можно использовать для идентификации процесса при работе с ним или мониторинге системы. PID обычно представляется целым числом и может быть использован для управления процессами, например, для отправки сигналов процессам, убийства процессов или изменения приоритетов.

**Определение 6.** Хост — это компьютер или другое устройство, подключенное к сети, которое может обеспечивать доступ к ресурсам сети или предоставлять какие-то услуги другим устройствам в сети. Хост может быть сервером, рабочей станцией, маршрутизатором, коммутатором или любым другим устройством, подключенным к сети. Каждый хост в сети имеет свой уникальный идентификатор, называемый IP-адресом, который позволяет ему обмениваться данными с другими устройствами в сети.

**Определение 7.** Парсинг — это процесс анализа строки или текстового документа для извлечения нужных данных или информации. Этот процесс может включать разбиение строки на составляющие (токены), преобразование данных в нужный формат и проверку корректности данных. Парсинг широко используется в различных областях, таких как программирование, обработка естественного языка, анализ данных, веб-разработка и многих других.

* 1. **Аналоги**

Существуют инструменты для просмотра загруженности CPU такие как htop, top, nmon, atop.

htop - это интерактивный процессорный монитор для Unix-подобных систем, который позволяет отслеживать загрузку системы и детальную информацию о запущенных процессах. htop является более продвинутой альтернативой утилите top, которая широко используется для мониторинга системных процессов в Unix-подобных системах.

Сам по себе предоставляет множество полезных функций, таких как поддержка цветной графики, возможность изменения приоритета процессов, сортировка процессов по различным критериям, фильтрация процессов по имени, PID и т.д. Кроме того, предоставляет подробную информацию о ресурсах системы, таких как использование CPU, RAM и дискового пространства.

Утилита имеет простой и понятный интерфейс, который позволяет быстро отслеживать работу системы и идентифицировать потенциальные проблемы. Утилита может быть полезна как для системных администраторов, так и для обычных пользователей, которые хотят контролировать процессы, которые запущены на их компьютере.

Кроме того, htop является открытым исходным кодом и доступен бесплатно для загрузки и использования. Он может быть установлен на большинство Unix-подобных систем, таких как Linux, FreeBSD и macOS. Пример использование утилиты htop приведен на рисунке 1.1.

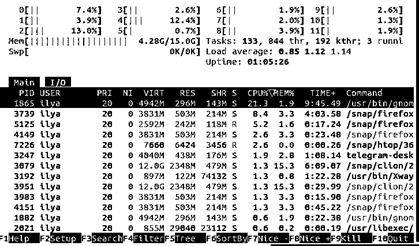


Рисунок 1.1 – Интерфейс утилиты htop

top - это утилита командной строки для мониторинга процессов и загрузки системы в Unix-подобных операционных системах. Она позволяет отслеживать использование CPU, RAM, дискового пространства и других ресурсов системы.

Утилита отслеживает список процессов, запущенных на системе, с информацией о PID, потреблении CPU и памяти, времени запуска и других характеристиках процесса. Пользователь может отсортировать список процессов по различным критериям, таким как потребление CPU, потребление памяти или имя процесса. Также выводит общую информацию о системе, такую как загрузка CPU, общее количество свободной и используемой памяти, а также информацию о состоянии системы и активных задачах.

top стандартна в большинстве Unix-подобных операционных систем, таких как Linux, macOS и FreeBSD. Утилита может быть полезна для системных администраторов и других пользователей, которые хотят контролировать процессы, запущенные на их системе и мониторить использование ресурсов. Пример утилиты atop приведен на рисунке 1.2.

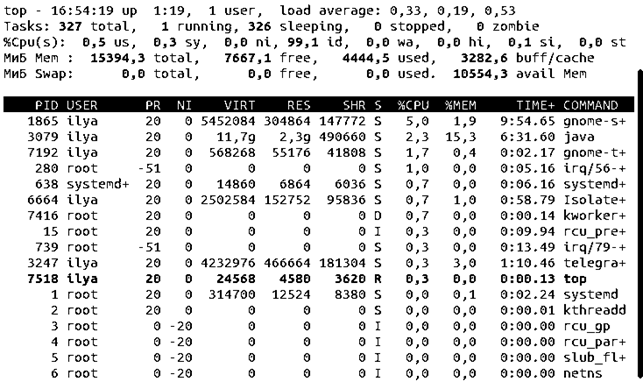


Рисунок 1.2 – Интерфейс утилиты top

nmon - это утилита командной строки для мониторинга системы в Unix-подобных операционных системах. Она предоставляет информацию о загрузке CPU, RAM, сети, дисковой активности, процессах и других ресурсах системы.

Среди множества функций есть те, которые делают ее полезной для системных администраторов и других пользователей. Утилита отображает информацию в реальном времени и может сохранять данные в файлы для последующего анализа. Кроме того, nmon имеет графический интерфейс и может работать в интерактивном режиме. Также позволяет пользователю отслеживать процессы, используемые на системе, и управлять ими. Он также отображает информацию о потреблении процессора, памяти, ввода-вывода и других ресурсах для каждого процесса.

Утилита может быть установлена на большинство Unix-подобных операционных систем, таких как Linux, AIX, Solaris, HP-UX, IBM и другие. nmon является открытым исходным кодом и бесплатно доступна для загрузки и использования. Пример интерфейса nmon приведен на рисунке 1.3.

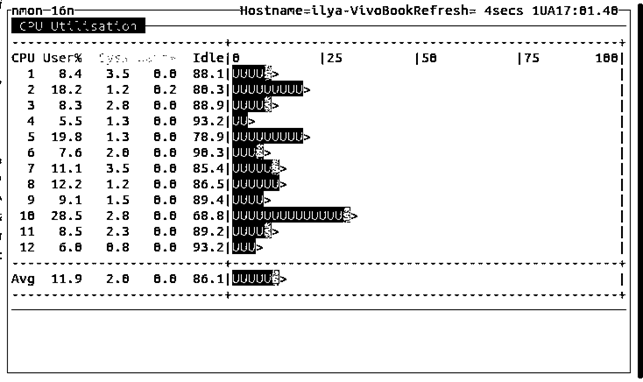


Рисунок 1.3 – Интерфейс утилиты nmon

atop — это утилита мониторинга производительности, которая позволяет отслеживать нагрузку на различные ресурсы компьютера, включая процессор, память, дисковую подсистему и сетевые интерфейсы. Она предоставляет детальную информацию о процессах, запущенных на компьютере, и позволяет быстро определить, какие процессы занимают большую часть ресурсов. Она предоставляет возможность просмотра и анализа исторических данных, что позволяет определить причины возможных проблем производительности и принять меры по их устранению. Она может быть использована для отслеживания нагрузки на серверах и виртуальных машинах, а также на рабочих станциях и ноутбуках. atop доступна для большинства популярных дистрибутивов Linux.

Утилита также предоставляет детальную информацию о процессах, запущенных на системе, и их ресурсоемкости, что позволяет быстро идентифицировать проблемные процессы и оптимизировать их работу. Кроме того, atop может анализировать данные о системной активности в режиме реального времени и сохранять их в журнал для последующего анализа.

Одной из особенностей atop является возможность установки пользовательских правил, которые позволяют настраивать сбор и анализ данных в соответствии с потребностями пользователя. В целом, это мощный инструмент мониторинга, который позволяет быстро определить проблемы с производительностью системы и принимать меры для их решения. Пример интерфейса atop приведен на рисунке 1.4.

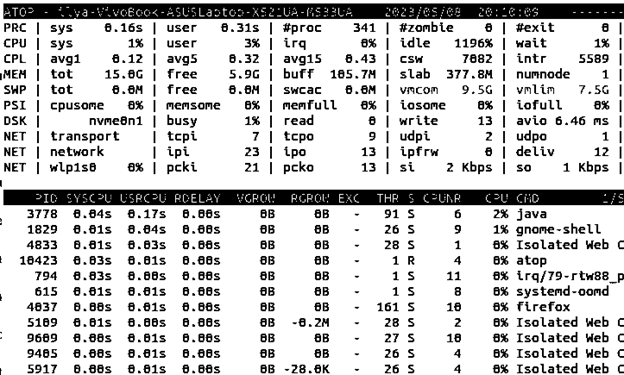
****

Рисунок 1.4 – Интерфейс утилиты atop

**2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

Перед реализацией требуется поставить конкретные требования к программе, которые в будущем должны быть реализованы. При проектировании данного приложения требуется реализовать следующие модули:

- Вычисление загруженности CPU

- Вывод информации о процессоре

- Вывод информации о нагрузке на процессор с использованием псевдографики

- Вывод системной информации о пользователе

- Использование различных флагов при работе программы

- Вывод итоговой статистики

Структурная схема взаимодействия модулей программы приведена в приложении А.

**2.1 Модуль вычисления загруженности CPU**

Основная цель выполнения курсового проекта, создание программы для мониторинга загруженности CPU. Поэтому главный модуль должен производить расчеты нагрузки на процессор и сохранять данные. Для этого и будет разработан данный модуль

**2.2 Модуль получения вывода информации о процессоре**

В данном модуле будет реализован получения информации о процессоре. Данный модуль предназначен для того, чтобы выводить информацию в терминал в простейшем для пользователя виде.

**2.3 Модуль вывода информации о нагрузке на процессор с использованием псевдографики**

Данный модуль предназначен для более удобного вывода информации для пользователя, чтобы можно было визуально оценить нагрузку на процессор. Для этого будет реализован псевдографическое представление с использованием графиков.

**2.4 Модуль вывода системной информации о пользователе**

Модуль, используемый как дополнительный модуль. Представляет собой функцию сбора информации о системе и предоставления ее пользователю для ознакомления.

**2.5 Модуль использования различных флагов при работе программы**

Модуль, предназначенный для более удобных стартовых настроек, в нем будет реализованы различные формы запуска программы, такие как вывод нагрузки в простой форме или псевдографической, также задания количества итераций сбора информации, задержка между выводом информации.

**2.6 Модуль вывода итоговой статистики**

Данный модуль выводит итоговую статистику, собранную за весь промежуток мониторинга. Предназначен для того, чтобы пользователь мог ознакомиться с итогами мониторинга и получить всю нужную информацию. Вывод информации будет в той форме, которая будет задана пользователем упрощенном/графическом.

**3 ФУНЦИОНАЛЬНАЕ ПРОЕКТИРОВАНИЕ**

В данном блоке будут рассмотрены функции и структуры, реализующие функционал программы для мониторинга.

**3.1 Реализация структуры LinckedListNode**

Структура включает в себя два поля:

- char str [1024] – массив, в котором будет храниться информация

- struct LinckedListNode \*next – указатель на следующий элемент списка

Также создан объект Node типа struct LinckedListNode

**3.2 Реализация структуры UsageInformationLinckedLists**

Структура включает в себя следующие поля:

- Node \*cpu\_usage\_list\_head – указатель на голову связанного списка, используемого для работы с CPU

- Node \*cpu\_usage\_list\_tail – указатель на хвост связанного списка, используемого для работы с CPU

- int lastTotal – общее время безотказной работы в последний промежуток измерения.

- int lastIdle – общее время простоя в последний промежуток измерения.

Также создан объект UsageInfoLL типа struct UsageInformationLinckedLists

**3.3 Реализация модуля вычисления загруженности CPU**

float calculateCPUUsage(int \*lastTotal, int \*lastIdle) – функция вычисления процентной нагрузки на процессор. В качестве параметров принимает:

- int \*lastTotal – общее время безотказной работы в последний промежуток измерения.

- int \*lastIdle – общее время простоя в последний промежуток измерения.

**3.4 Реализация модуля вывода информации о процессоре**

void generateCPUUsage(int samples, int tdelay, UsageInfoLL \*usageInfo, int i) – функция вывода инфомации о нагрузке на процессор без использования графики, в простейшем для пользователя виде. В качестве параметров принимает:

- int samples – количество итераций по считыванию показаний процессора.

- int tdelay – задержка между считываниями показаний.

- UsageInfoLL \*usageInfo – указатель на структуру, хранящую список для вывода информации с использованием псевдографики

- int i – номер итерации по выводу информации на экран

**3.5 Реализация модуля вывода информации о нагрузке на процессор с использованием псевдографики**

void generateCPUUsageGraphics(int samples, int tdelay, UsageInfoLL \*usageInfo, int i) – функция для вывода информации с использованием графики, информация выводится в виде графиков. В качестве параметров принимает:

- int samples – количество итераций по считыванию показаний процессора.

- int tdelay – задержка между считываниями показаний.

- UsageInfoLL \*usageInfo – указатель на структуру, хранящую список для вывода информации с использованием псевдографики

- int i – номер итерации по выводу информации на экран

**3.6 Реализация модуля выводы системной информации о пользователе**

void displaySystemInfo() – функция вывода информации о пользователе. В качестве праметров ничего не принимает.

**3.7 Реализация модуля использования различных флагов при работе программы**

bool parseArguments(int argc, char \*\*argv, int \*samples, int \*tdelay, bool \*systemFlagPresent, bool \*graphicsFlagPresent) – функция выполняющая обработку входных флагов от пользователя и выполняющая генерацию итоговых выходных данных. Возвращает true если аргументы были переданы а правильной форме. В качестве параметров принимает:

- int argc – количество аргументов командной строки.

- char \*\*argv – список аргументов командной строки.

- int \*samples – количество итераций для сбора и вывода информации.

- int \*tdelay – задержка между измерениями информации о процессоре.

- bool \*systemFlagPresent – флаг разрешающий вывод итоговой системной информации.

- bool \*graphicsFlagPresent – флаг разрешающий функции вывод информации с помощью графики.

**3.8 Реализация модуля вывода итоговой статистики**

void printReport(int samples, int tdelay, bool systemFlagPresent, bool graphicsFlagPresent) - функция выполняющая вывод финальной информации на экран. В качестве параметров принимает:

- int samples – количество итераций для сбора и вывода информации.

- int tdelay – задержка между измерениями информации о процессоре.

- bool systemFlagPresent – флаг разрешающий вывод итоговой системной информации.

- bool graphicsFlagPresent – флаг разрешающий функции вывод информации с помощью графики.

**4 РАЗРАБОТКА ПРОГРАМНЫХ МОДУЛЕЙ**

**4.1 Реализация алгоритма по шагам функции bool parseArguments(int argc, char \*\*argv, int \*samples, int \*tdelay, bool \*systemFlagPresent, bool \*graphicsFlagPresent)**

Шаг 1. Инициализировать переменные *samplesSpecified* и *tdelaySpecified* в false, чтобы отслеживать, были ли они установлены пользователем.

Шаг 2. Если количество аргументов командной строки больше единицы, перейти к следующему шагу, иначе пропустить.

Шаг 3. Пройти по каждому аргументу командной строки, начиная со второго *(argv[1]),* с помощью цикла while.

Шаг 4. Разбить каждый аргумент на части с помощью функции *strtok()*, используя "=" как разделитель. Это позволит определить, какой флаг был указан, если таковой имеется.

Шаг 5. Если флаг *--samples* был указан, сохранить значение, указанное пользователем, в переменную samples с помощью функции *atoi()*. Установить *samplesSpecified* в true, чтобы отметить, что пользователь указал количество выборок.

Шаг 6. Если флаг *--tdelay* был указан, сохранить значение, указанное пользователем, в переменную *tdelay* с помощью функции *atoi()*. Установить *tdelaySpecified* в true, чтобы отметить, что пользователь указал задержку между выборками.

Шаг 7. Если флаг --system был указан, установить значение *systemFlagPresent* в true.

Шаг 8. Если флаг --graphics или -g был указан, установить значение *graphicsFlagPresent* в true.

Шаг 9. Если аргумент не соответствует ни одному из флагов, проверить, является ли он числом, и если это так, определить, какому параметру он соответствует (samples или tdelay). Если samples и tdelay не были указаны ранее, сохранить эти значения. Инкрементировать счётчик цикла i на единицу, чтобы пропустить следующий аргумент, который уже был обработан.

Шаг 10. Если аргумент является единственным числовым значением, определить, какому параметру он соответствует (samples) и сохранить его значение. Установить *samplesSpecified* в true, чтобы отметить, что пользователь указал количество выборок.

Шаг 11. Если аргумент не соответствует ни одному из флагов или числовых значений, вывести сообщение об ошибке и вернуть false.

Шаг 12. Инкрементировать счётчик цикла i на единицу для перехода к следующему аргументу.

Шаг 13. Повторять шаги 4-12 для каждого аргумента командной строки.

Шаг 14. Если успешно обработаны все аргументы командной строки, вернуть true.

**4.2 Описание алгоритма работы программы.**

Шаг 1. Сбор общей информации от системы

- Для сбора информации о количестве ядер в CPU, происходит вызов системной функции *sysconf(\_SC\_NPROCESSORS\_ONLN)* принадлежащий библиотеке *unistd.h*

- Для того, чтобы собрать информацию о CPU, считываем системный файл */proc/stat* информацию из которого будем использовать в Шаге 2.

- Для получения информации об активных сессиях и пользователях, авторизованных в них, используем структуру *utmp* из библиотеки *utmp.h*. Для сбора информации воспользуемся вызовом системной функции *getutent().* После вызова функции в поля структуры будут записаны следующие значения:

*- ut\_user*: имя авторизованного пользователя

*- ut\_line*: название устройства

*- host*: имя хоста

- Для получения подробной инфомрации о системе воспользуемся структурой *ustname* из библиотеки *<sys/ustname.h>*

*- sysname*: имя системы

*- nodename*: название машины

*- version*: версия операционной системы

*- release*: выпуск операционной системы

*- machine:* информация об архитектуре машины на котором запущена программа

Шаг 2. Вычисления

Нагрузка на CPU рассчитывается за промежуток и расчитывается по формуле:

CPU\_Usage = 100 – (*idle* - *\*lastIdle*) \* 100 / (*total\_time* - *\*lastTotal*)

где:

*- idle* – время простоя

*- \*lastTotal* – общее время безотказной работы

*- \*lastIdle* – общее время простоя

*- total\_time* – полное время, считанное из файла */proc/stat*

Шаг 3. Хранение данных

С целью сохранения данных, полученных в результате считывания из файла */proc/stat*, было решено хранить данные в односвязном списке. Каждый момент времени представлен как узел в связном списке, и каждый узел содержит строку, содержащую всю соответствующую информацию для этого момента времени. Также в связи с использованием списков была создана структура *UsageInfoLL* типа *struct UsageInformationLinckedLists*, которая хранит в себе указатели на голову и хвост каждого связанного списка.

Шаг 4. Парсинг пользовательского ввода входных флагов

В функцию main передаются аргументы командной строки *argv* и *argc*, в которых находится информация о пользовательском вводе флагов. Чтобы получить доступ к аргументам и правильно их распарсить была написана функция *parseArguments*. Поскольку некоторые флаги имеют посередине знак = была использована функция *strtok()* из библиотеки *string.h*, чтобы разделить каждый аргумент на =. Таким образом, мы сможем прочитать введенное пользователем значение после =. В случае если строка после знака = не может быть преобразована в целое число, появится сообщение об ошибке, и программа завершится. Для того чтобы узнать, какие флаги были введены пользователем, использована функция *strcmp()* из библиотеки *string.h* и использованы логические переменные для хранения того, был ли введен каждый флаг. В зависимости от того, какая комбинация флагов была введена, программа выводит соответствующую информацию (используя серию операторов *if/else*).

Шаг 5. Печать отчета

Чтобы убедиться, вывод обновляется каждый момент времени, перед выводом кажой выборки происходит сохранение курсора, используя управляющий код *\x1b7*. После распечатки соответствующей информации, используется escape-код *\x1b8*, который размораживает курсор, чтобы он мог вернуться в сохранённую позицию. На следующей итерации предыдущий вывод перезаписывается. Таким образом, вывод обновляется в каждый момент времени.

**4.3 Реализация функции void printReport**

Блок-схема алгоритма приведена в приложении Б.

**5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

В данном разделе приведены результаты работы приложения. Функционал приложения приведен на скрин-шотах ниже.

Перед началом использования приложения пользователю нужно решить в каком режиме он будет работать и указать это используя флаги при запуске приложения.

При запуске приложения со стандартными настройками пользователю требуется просто запустить приложение. Для этого в терминале ввести *./CW.*

Программой такой вызов будет эквивалентен:

*./CW --system --samples=10 --tdelay=1*



Рисунок 5.1 – Запуск программы со стандартными флагами

При стандартном запуске, будет произведено 10 считываний статистики с промежутком в одну секунду. Каждую секунду на экране будет появляться информация о нагрузке на процессор в считанный момент. Вывод на экран показан на рисунке 5.2.

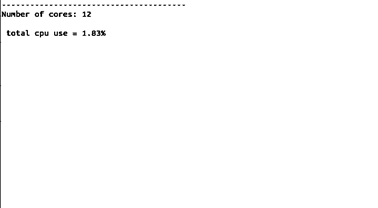


Рисунок 5.2 – Вывод информации во время работы программы

После завершения работы программы на экран будет выведен финальный результат считывания, а также системная информация о системе, активных сессиях и пользователях.

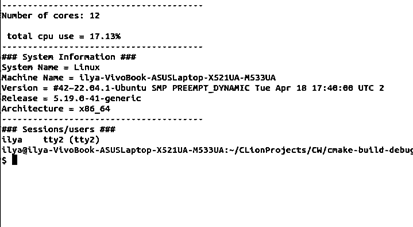


Рисунок 5.3 – Итоговая статистика

В случае если пользователь хочет увидеть статистику нагрузки на CPU за промежуток в графическом виде, он должен использовать флаг *-g*, либо же флаг *-graphics*. При вызове программы это будет выглядеть так.



Рисунок 5.4 – Использование флага -g



Рисунок 5.5 – Использования флага -graphics

После запуска программы с таким флагом будет выводить статистику графически обновляя ее каждую секунду.

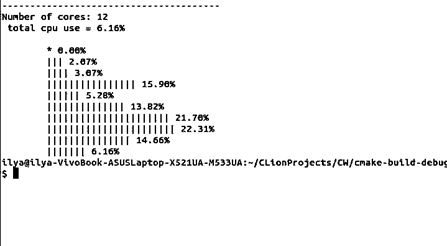


Рисунок 5.6 – Вывод информации о нагрузке CPU с использованием графики

В случае если пользователь хочет получить системную информацию об устройстве, на котором запущена программа, а также получить список активных сессий он может воспользоваться флагом *--system*. Тогда после окончания работы программы совместно со статистикой будет выведена информация об устройстве. Запуск программы с флагами --system и --graphics приведен на рисунке 5.7.



Рисунок 5.7 – Запуск программы с флагами --system и --graphics

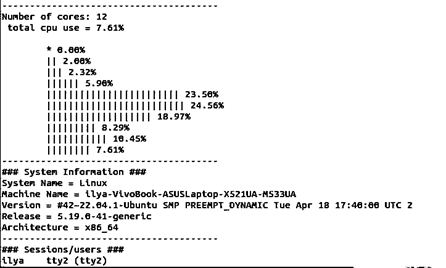


Рисунок 5.8 – Результат работы программы при запуске с флагами --system и --graphics

Может произойти ситуация что пользователю понадобится не стандартные 10 измерений, а любое другое число, а также другие промежутки между измерениями. Для этого можно воспользоваться такими флагами, как --samples и --tdelay. При установки этих флагов при вызове пользователь изменяет стандартные значения (samples = 10, tdelay = 1) на заданные им. Запуск программы с количеством и задержкой, заданной пользователем приведены на рисунке 5.9.



Рисунок 5.9 – Запуск программы с флагами --samples и --tdelay

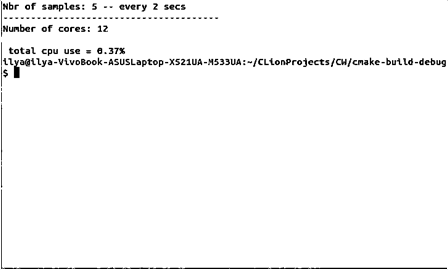


Рисунок 5.10 – Результат работы программы при работе с флагами

--samples и --tdelay

В случае не корректного ввода какого-либо аргумента будет выведено сообщение об ошибке, и программа завершится. Для правильной работы программы требуется корректный ввод флагов. Результат неправильного ввода аргументов приведен на рисунке 5.11.



Рисунок 5.11 – Результат неправильного ввода аргументов

В случае если все аргументы введены правильно и использованы все флаги, то программа будет иметь следующий вид после завершений. Запуск программы и результаты ее выполнения приведены на рисунках 5.12 и 5.13.



Рисунок 5.11 – Запуск программы со всеми возможными флагами

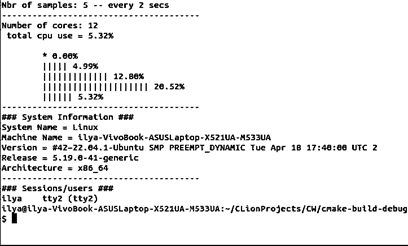


Рисунок 5.12 – Результат выполнения программы со всеми флагами

**ЗАКЛЮЧЕНИЕ**

В результате работы над данным курсовым проектом была разработана программа для мониторинга нагруженности CPU.

В ходе работы над курсовым проектом были улучшены умения разработки приложений на С. Получены теоретические и применены практические знания работы с такими библиотеками как *utmp, ustname.*

Работа была разделена на такие этапы, как анализ существующих аналогов, литературных источников, постановка требований к проектируемому программному средству, системное и функциональное проектирование, конструирование программного средства, разработка программных модулей и тестирование проекта. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее приложение.

В дальнейшем планируется усовершенствование программы, а именно,

улучшение графики, добавление дополнительного функционала в виде получения информации о RAM, а также улучшение проекта в целом.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

[1] Керниган, Б. Язык программирования С / Б. Керниган, Д. Ритчи – СПб.: Невский Диалект, 2001. - 352 с

[2] Лав Р. Linux. Системное программирование. 2-е изд. — СПб.: Питер, 2014. — 448 с.: ил. — (Серия «Бестселлеры O’Reilly»)

[3] Фленов М. Е. Linux глазами хакера. - 4-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2016 — 432 с.: ил.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Структурная схема**

**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Блок схема алгоритма void printReport**

**ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Ведомость документов**