

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 загрузки 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 загрузенность CPU, а также будет создана программа для отслеживания загрузки CPU.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 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.

1.2 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 использования утилиты htop приведен на рисунке 1.1.

The screenshot displays the htop utility interface. At the top, it shows system statistics: CPU usage (0%, 1%, 2%, 3%, 4%, 5%, 6%, 7%, 8%, 9%, 10%, 11%), Memory usage (Mem: 4.28G/15.0G, Swap: 0K/0K), Tasks (133), Threads (844), Processes (192), and Load average (0.85, 1.12, 1.14). The uptime is 01:05:26. Below the statistics, there is a table of running processes. The table has columns: PID, USER, PRI, NI, VIRT, RES, SHR, S, CPU%, MEM%, TIME+, and Command. The processes listed include /usr/bin/gnome, /snap/firefox, /snap/htop/36, telegram-desk, /snap/clicon/2, /usr/bin/Xway, /snap/clicon/2, /snap/firefox, /snap/firefox, /usr/bin/gnome, and /usr/libexec/.

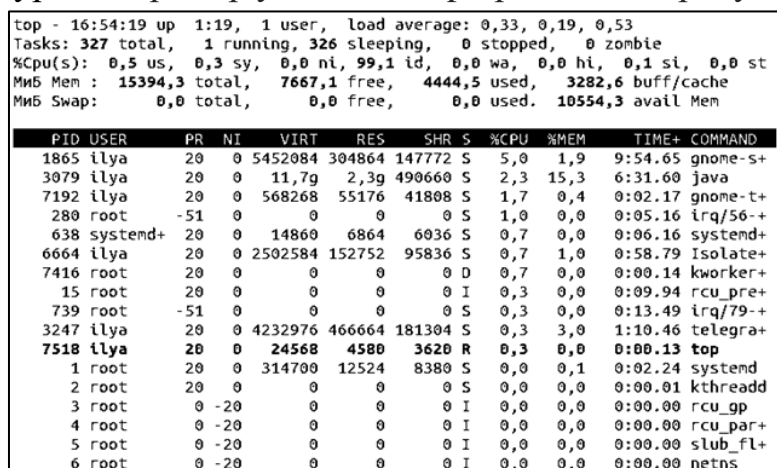
PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1865	ilya	20	0	4942M	296M	143M	S	21.3	1.9	9:45.49	/usr/bin/gnome
3739	ilya	20	0	3831M	503M	214M	S	8.4	3.3	4:03.58	/snap/firefox
5125	ilya	20	0	2592M	242M	118M	R	5.2	1.6	0:17.24	/snap/firefox
4149	ilya	20	0	3831M	503M	214M	S	2.6	3.3	0:23.48	/snap/firefox
7226	ilya	20	0	7668	6424	3456	R	2.6	0.0	0:00.26	/snap/htop/36
3247	ilya	20	0	4040M	438M	176M	S	1.9	2.8	1:08.14	telegram-desk
3079	ilya	20	0	12.0G	2348M	479M	S	1.3	15.3	6:09.07	/snap/clicon/2
3192	ilya	20	0	897M	122M	74132	S	1.3	0.8	1:22.28	/usr/bin/Xway
3951	ilya	20	0	12.0G	2348M	479M	S	1.3	15.3	0:29.99	/snap/clicon/2
3983	ilya	20	0	3831M	503M	214M	S	1.3	3.3	0:15.90	/snap/firefox
4151	ilya	20	0	3831M	503M	214M	S	1.3	3.3	0:45.22	/snap/firefox
1882	ilya	20	0	4942M	296M	143M	S	0.6	1.9	0:22.38	/usr/bin/gnome
2021	ilya	20	0	855M	29848	23112	S	0.6	0.2	0:00.19	/usr/libexec/

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.



```

top - 16:54:19 up 1:19, 1 user, load average: 0,33, 0,19, 0,53
Tasks: 327 total, 1 running, 326 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0,5 us, 0,3 sy, 0,0 ni, 99,1 id, 0,0 wa, 0,0 hi, 0,1 si, 0,0 st
MiB Mem: 15394,3 total, 7667,1 free, 4444,5 used, 3282,6 buff/cache
MiB Swap: 0,0 total, 0,0 free, 0,0 used. 10554,3 avail Mem

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1865	ilya	20	0	5452084	304864	147772	S	5,0	1,9	9:54.65	gnome-s+
3079	ilya	20	0	11,7g	2,3g	490660	S	2,3	15,3	6:31.60	java
7192	ilya	20	0	568268	55176	41808	S	1,7	0,4	0:02.17	gnome-t+
280	root	-51	0	0	0	0	S	1,0	0,0	0:05.16	irq/56-+
638	systemd+	20	0	14860	6864	6036	S	0,7	0,0	0:06.16	systemd+
6664	ilya	20	0	2502584	152752	95836	S	0,7	1,0	0:58.79	Isolate+
7416	root	20	0	0	0	0	D	0,7	0,0	0:00.14	kworker+
15	root	20	0	0	0	0	I	0,3	0,0	0:09.94	rcu_pre+
739	root	-51	0	0	0	0	S	0,3	0,0	0:13.49	irq/79-+
3247	ilya	20	0	4232976	466664	181304	S	0,3	3,0	1:10.46	telegra+
7518	ilya	20	0	24568	4580	3620	R	0,3	0,0	0:00.13	top
1	root	20	0	314700	12524	8380	S	0,0	0,1	0:02.24	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.01	kthreadd
3	root	0 -20	0	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0 -20	0	0	0	0	I	0,0	0,0	0:00.00	rcu_par+
5	root	0 -20	0	0	0	0	I	0,0	0,0	0:00.00	slub_fl+
6	root	0 -20	0	0	0	0	I	0,0	0,0	0:00.00	netns

Рисунок 1.2 – Интерфейс утилиты top

htop - 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, htop 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.

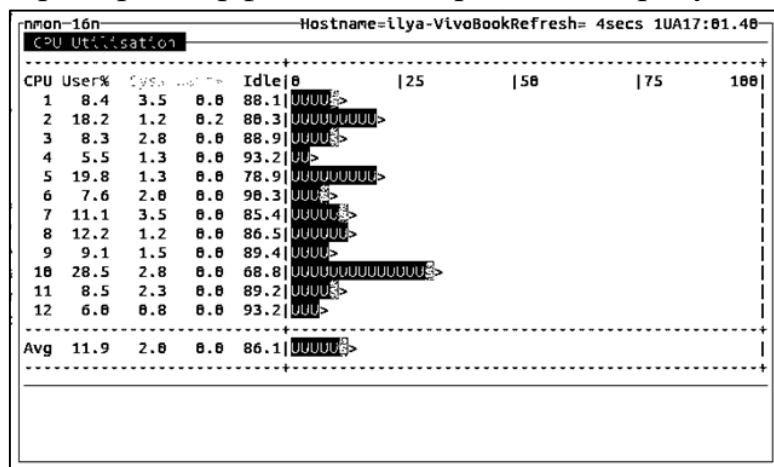


Рисунок 1.3 – Интерфейс утилиты nmon

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.

atop - /dev/vc/vcBook-ASUSLaptop-X521UA-WS33UA 2023/05/08 20:10:08 -----															
PRC		sys	0.16s		user	0.31s		#proc	341		#zombie	0		#exit	0
CPU		sys	1%		user	3%		irq	0%		idle	1196%		wait	1%
CPL		avg1	0.12		avg5	0.32		avg15	0.43		csw	7882		intr	5589
MEM		tot	15.0G		free	5.9G		buff	105.7M		slab	377.8M		numnode	1
SWP		tot	0.0M		free	0.0M		swcac	0.0M		vmcom	9.5G		vmlim	7.5G
PSI		cpusome	0%		memsome	0%		memfull	0%		iosome	0%		iofull	0%
DSK		nvme0n1	busy		1%		read	0		write	13		avio	6.46 ms	
NET		transport			tcp	7		tcpo	9		udpi	2		udpo	1
NET		network			ipl	23		lpo	13		ipfrw	0		deliv	12
NET		wlpi0	0%		pcki	21		pcko	13		si	2 Kbps		so	1 Kbps

PID	SYS	CPU	USR	CPU	RDELAY	VGROW	RGROW	EXC	THR	S	CPUNR	CPU	CHD	I/S	
3778	0.04s	0.17s	0.00s	00	00	-	91	S	6	2%	java				
1829	0.01s	0.04s	0.00s	00	00	-	26	S	9	1%	gnome-shell				
4833	0.01s	0.03s	0.00s	00	00	-	28	S	1	0%	Isolated Web C				
10423	0.03s	0.01s	0.00s	00	00	-	1	R	4	0%	atop				
794	0.03s	0.00s	0.00s	00	00	-	1	S	11	0%	irq/79-rtw88_p				
615	0.01s	0.01s	0.00s	00	00	-	1	S	8	0%	systemd-oomd				
4037	0.00s	0.01s	0.00s	00	00	-	161	S	10	0%	firefox				
5109	0.01s	0.00s	0.00s	00	-0.2M	-	28	S	2	0%	Isolated Web C				
9609	0.00s	0.01s	0.00s	00	00	-	27	S	10	0%	Isolated Web C				
9405	0.00s	0.01s	0.00s	00	00	-	26	S	4	0%	Isolated Web C				
5917	0.00s	0.01s	0.00s	00	-28.0K	-	26	S	4	0%	Isolated Web C				

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 будут 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 ФУНКЦИОНАЛЬНОЕ 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 загрузки 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 информации 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 вывода системной информации о пользователе

`void displaySystemInfo()` – функция вывода информации о пользователе. В качестве параметров 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 ПРОГРАММНЫХ 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 информации 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 рассчитывается по формуле:

$$\text{CPU_Usage} = 100 - (\text{idle} - \text{*lastIdle}) * 100 / (\text{total_time} - \text{*lastTotal})$$

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 прочесть 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 каждой 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:

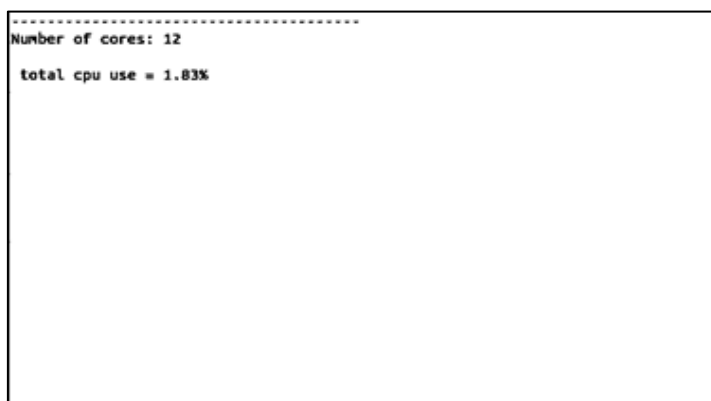
`./CW --system --samples=10 --tdelay=1`



```
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW
```

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.



```
-----
Number of cores: 12
total cpu use = 1.83%
```

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.

```
-----
Number of cores: 12
total cpu use = 17.13%
-----
### System Information ###
System Name = Linux
Machine Name = ilya-VivoBook-ASUSLaptop-X521UA-M533UA
Version = #42-22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue Apr 18 17:40:00 UTC 2
Release = 5.19.0-41-generic
Architecture = x86_64
-----
### Sessions/users ###
ilya    tty2 (tty2)
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ █
```

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.

```
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW -g
```

Рисунок 5.4 – Использование флага `-g`

```
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW -graphics
```

Рисунок 5.5 – Использование флага -graphics

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

```
-----
Number of cores: 12
total cpu use = 6.16%

* 0.00%
||| 2.07%
|||| 3.07%
||||| 5.28%
||||| 6.16%
||||| 13.82%
||||| 14.66%
||||| 15.90%
||||| 21.70%
||||| 22.31%
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ █
```

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.

```
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW --system --graphics
```

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

```
-----
Number of cores: 12
total cpu use = 7.61%

* 0.00%
|| 2.00%
||| 2.32%
||||| 5.90%
||||||||||||||||| 23.50%
||||||||||||||||| 24.56%
||||||||||||||||| 18.97%
||||||| 8.29%
||||||||| 10.45%
||||||| 7.61%
-----

### System Information ###
System Name = Linux
Machine Name = ilya-VivoBook-ASUSLaptop-X521UA-M533UA
Version = #42-22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue Apr 18 17:40:00 UTC 2
Release = 5.19.0-41-generic
Architecture = x86_64
-----

### Sessions/users ###
ilya  tty2 (tty2)
```

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 установке 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.

```
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW --samples=5 --tdelay=2
```

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

```
Nbr of samples: 5 -- every 2 secs
-----
Number of cores: 12
total cpu use = 0.37%
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$
```

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.

```

ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW -system
Invalid argument entered!
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW -sys
Invalid argument entered!
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW --sys
Invalid argument entered!
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW --grap
Invalid argument entered!
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW --graphic
Invalid argument entered!
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW --g
Invalid argument entered!
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW --twantgraphics
Invalid argument entered!
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$

```

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 завершения. Запуск программы и результаты ее выполнения приведены на рисунках 5.12 и 5.13.

```

ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ ./CW --system --graphics --samples=5 --tdelay=2

```

Рисунок 5.11 – Запуск программы со всеми возможными флагами

```

Nbr of samples: 5 -- every 2 secs
-----
Number of cores: 12
total cpu use = 5.32%

    * 0.00%
    |||| 4.99%
    ||||| 12.80%
    ||||| 20.52%
    |||| 5.32%
-----
### System Information ###
System Name = Linux
Machine Name = ilya-VivoBook-ASUSLaptop-X521UA-M533UA
Version = #42-22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue Apr 18 17:40:00 UTC 2
Release = 5.19.0-41-generic
Architecture = x86_64
-----
### Sessions/users ###
ilya    tty2 (tty2)
ilya@ilya-VivoBook-ASUSLaptop-X521UA-M533UA:~/CLionProjects/CW/cmake-build-debug
$ █

```

Рисунок 5.12 – Результат выполнения программы со всеми флагами

ЗАКЛЮЧЕНИЕ

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

В ходе работы над курсовым проектом были улучшены умения разработки приложений на C. 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