

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Забайкальский государственный университет»  
(ФГБОУ ВО «ЗабГУ»)  
Энергетический факультет  
Кафедра информатики, вычислительной техники и прикладной математики

## КУРСОВАЯ РАБОТА

По дисциплине: Программирование

На тему: Система мониторинга ресурсов серверов Linux.

Выполнил студент группы: ИВТ–16-1 Шишмарев Павел Георгиевич

Руководитель работы: Старший преподаватель кафедры ИВТиПМ Ветров  
Сергей Владимирович

Чита 2018

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Забайкальский государственный университет»  
(ФГБОУ ВО «ЗабГУ»)  
Энергетический факультет  
Кафедра информатики, вычислительной техники и прикладной математики

ЗАДАНИЕ

на курсовую работу

По дисциплине: Программирование

Студенту: Шишмареву Павлу Георгиевичу

Специальности (направления подготовки): 09.03.01 Информатика и  
вычислительная техника

Тема курсовой работы: Система мониторинга ресурсов серверов Linux;

Срок подачи студентом законченной работы: \_\_\_\_\_;

Исходные данные к работе: Документация Python и PyQtGraph.

Перечень подлежащих разработке в курсовой работе вопросов:

Создание серверной части программы мониторинга, собирающей  
информацию о ресурсах системы;

Создание клиентской части программы мониторинга, принимающей и  
визуализирующей информацию о ресурсах серверов.

Дата выдачи задания: \_\_\_\_\_.

Руководитель курсовой работы \_\_\_\_\_/Ветров С. В./

(подпись, расшифровка подписи)

Задание принял к исполнению «\_\_» \_\_\_\_\_ 2018 г.

Подпись студента \_\_\_\_\_ / Шишмарев П. Г./

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Забайкальский государственный университет»  
(ФГБОУ ВО «ЗабГУ»)  
Энергетический факультет  
Кафедра информатики, вычислительной техники и прикладной математики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовой работе

По дисциплине: Программирование

На тему: Система мониторинга ресурсов серверов Linux.

Выполнил студент группы ИВТ-16-1 Шишмарев Павел Георгиевич

Руководитель работы: Старший преподаватель кафедры ИВТиПМ Ветров  
Сергей Владимирович

УТВЕРЖДАЮ

Зав. кафедрой \_\_\_\_\_

«\_\_\_» \_\_\_\_\_ 2018 г.

КАЛЕНДАРНЫЙ ГРАФИК  
выполнения курсовой работы

Этапы выполнения курсовой работы	Месяцы и недели											
	Февраль				Март				Апрель			
	1	2	3	4	1	2	3	4	1	2	3	4
1. Получение задания на курсовую работу												
2. Анализ предметной области												
3. Проектирование												
4. Реализация												
5. Защита работы												

План выполнен: руководитель \_\_\_\_\_

(подпись, расшифровка подписи)

«\_\_\_» \_\_\_\_\_ 2018г.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	7
1. Постановка задачи .....	8
1.1. Требования к функционалу .....	8
1.2. Перечень показателей использования ресурсов .....	8
2. Особенности реализации .....	8
2.1. Коммуникация серверной и клиентской части .....	8
2.2. Получение и обработка данных .....	10
2.3. Сигнализация о недоступности серверов .....	13
3. Пользовательский интерфейс .....	14
3.1. Интерфейс серверной части .....	14
3.2. Интерфейс клиентской части .....	16
3.3. Интерфейс системы сигнализации .....	18
4. Руководство пользователя .....	19
4.1. Основная информация .....	19
4.2. Минимальные системные требования .....	19
4.3. Процедура установки .....	19
4.4. Работа с программой .....	20
Заключение .....	23
Список использованной литературы .....	24
Приложение 1. Исходный код. ....	25

## РЕФЕРАТ

Пояснительная записка – 45 страниц, 3 рисунка, 12 листингов, 6 источников, 1 предложение.

### МОНИТОРИНГ, СЕТЬ, СЕРВЕРЫ, UNIX, LINUX, СИСТЕМА МОНИТОРИНГА, PYTHON

В курсовой работе рассматривается разработка программного обеспечения – комплексной системы мониторинга серверов под управлением операционной системы Linux, посредством высокоуровневого языка программирования Python с использованием библиотеки для создания графических пользовательских интерфейсов PyQt5. В работе были использованы методы функционального и структурного программирования.

## ВВЕДЕНИЕ

Сетевой сервер — специальный компьютер(или другое аппаратное устройство), подключенный к локальной или глобальной сети и предоставляющий в этой сети определённый сервис или совокупность сервисов(например, доступ к веб-сайту).

Зачастую в крупных организациях, ведущих деятельность в сфере информационных технологий содержится целый парк серверов, решающих различные задачи и обеспечивающих внутреннюю сетевую инфраструктуру организации, а также работу внешних сетевых сервисов, таких как веб-сайты и электронная почта.

Одной из задач при администрировании серверов является отслеживание их производительности и использования системных ресурсов, чтобы обеспечить их эффективное и бесперебойное функционирование.

## 1. Постановка задачи

### 1.1. Требования к функционалу

Для организации эффективного процесса разработки системы мониторинга необходимо определить основные задачи, которые должны будут быть осуществлены при реализации данного продукта.

При разработке данного ПО были выделены основополагающие задачи:

- определение перечня показателей нагрузки целевой системы, подлежащих сбору;
- создание алгоритма приведения собранной информации к виду, удобному для передачи и отображения.
- реализация передачи информации по сети между серверной и клиентской частью приложения
- разработка графического пользовательского интерфейса, обеспечивающего возможность эффективного отслеживания показателей использования ресурсов нескольких серверов.
- разработка системы сигнализации недоступности («падения») серверов.

### 1. 2. Перечень показателей использования ресурсов.

- нагрузка на центральный процессор
- объём занятой оперативной памяти
- использование памяти на примонтированных разделах
- скорость входящей и исходящей передачи информации через сетевые интерфейсы

## 2. Особенности реализации

### 2.1. Коммуникация серверной и клиентской части

Части приложения соединяются между собой с помощью сокетов по протоколу TCP. Серверная часть программы слушает выбранный порт



и принимает оттуда текстовые команды. Перечень доступных для исполнения команд:

- fetch
- reboot
- kill

В зависимости от команды, сервер может отправить в ответ информацию о нагрузке, перезагрузиться или отключить саму программу. Если полученная информация не соответствует ни одной из команд, она проверяется на соответствие формату MAC-адреса. Если на вход подан MAC-адрес, то программа отправит широковещательное сообщение со сформированным Wake-On-LAN magic packet в своей сети.

Когда пользователь в интерфейсе программы выбирает сервер из списка, происходит подключение к серверу, после чего клиент начинает посылать команду «fetch» в заданный промежуток времени(по умолчанию – каждую секунду). Команды «kill» и «reboot» отправляются при нажатии на кнопки «Kill server monitor» и «Reboot server» соответственно.

## 2.2. Получение и обработка данных

Данные собираются в системе только при помощи стандартных консольных утилит, нет необходимости в доустановке каких-либо программных пакетов для сбора информации. В процессе сбора информации о нагрузке использованы следующие утилиты UNIX:

**uptime:** Эта утилита предоставляет информацию о том, сколько времени прошло от момента старта системы, какова средняя нагрузка(load average), а также отображает текущее время системы.

Пример вывода:

```
$ uptime
12:05:36 up 16:33,  1 user,  load average: 0.21, 0.38, 0.42
```

**hostname:** Эта утилита просто выводит доменное имя системы.

Пример вывода:

```
$ hostname
pashawnn-HP-Notebook
```

**vmstat** с флагом **-s**: выводит информацию об использовании оперативной памяти.

Пример вывода:

```
$ vmstat -s
3963676 K total memory
916208 K used memory
...
```

**ip -s link:** выводит подробную информацию об использовании сетевых интерфейсов.

Также используется утилита **cat**, выводящая содержимое файла в консоль для считывания содержимого системного файла **/proc/stat**,

содержащего в себе информацию об использовании центрального процессора.

Пример вывода cat /proc/stat:

```
$ cat /proc/stat
cpu 405582 7512 116590 2057230 16069 0 5614 0 0 0
cpu0 101460 5358 28681 1841681 14884 0 900 0 0 0
...
```

df: Выводит таблицу, показывающую информацию о всех примонтированных разделах запоминающих устройств.

Пример вывода:

```
$ df
Filesystem      1K-blocks      Used Available Use% Mounted on
udev            1959476         0    1959476   0% /dev
tmpfs           396368    11444    384924   3% /run
/dev/sda4       192054884 9991756 172284236   6% /
tmpfs           1981836    10420    1971416   1% /dev/shm
tmpfs           5120         4        5116   1% /run/lock
tmpfs           1981836         0    1981836   0% /sys/fs/cgroup
/dev/sda2       23900028 2542280 20120656  12% /opt
/dev/sda3       217066460 82825960 123191080  41% /home
```

Серверный скрипт выполняет все вышеуказанные команды при сборе информации и сохраняет их вывод в переменные, но изначально вывод команд непригоден для непосредственной передачи и отображения в окне клиента, так как содержит много лишнего и не оптимизирован для удобного просмотра, поэтому перед передачей нам нужно вычлениить из всего полученного текста. Эту задачу помогает

решить Python-модуль «re», предоставляющий возможности работы с регулярными выражениями.

Регулярные выражения представляют собой формальный язык поиска подстрок в строке с использованием метасимволов(wildcard characters) и позволяют извлекать из строк необходимую информацию, основываясь на её расположении в оригинальном тексте.

Создаём метод, обрабатывающий вывод утилит и приводящий его к виду конечной информации, которая будет передана клиенту.

Утилита	Регулярное выражение	Информация
uptime	up (.+),[ 0-9]+user	Время со старта системы
uptime	([0-9+]+:[0-9+]+:[0-9+]+) up	Текущее время
uptime	([0-9.]+, [0-9.]+, [0-9.]+)	Load average.
cat /proc/stat	^cpu +([0-9]+) ([0-9]+) ([0-9]+) ([0-9]+)	Информация о нагрузке на ЦП
vmstat -s	([0-9]+) K total memory	Занято оперативной памяти
vmstat -s	([0-9]+) K used memory	Использовано оперативной памяти
hostname	-Не нужно-	Доменное имя
ip -s link	mcast +\n +([0-9]+).\n.\n +([0-9]+)	Список данных о трафике через каждый сетевой интерфейс
df	((^ \n\\+)[ ]+([0-9]+)[ ]+([0-9]+)[ ]+([0-9]+)[ ]+([0-9]+)%[ ]+([ ]+))	Список строк таблицы примонтированных разделов

Таблица 1. Регулярные выражения

После того, как вся информация будет очищена от лишнего текста, с некоторыми полями всё ещё нужно провести некоторые операции. Так как в файле /proc/stat нет нагрузки на процессор в чистом виде в форме процентного соотношения от максимальной, нужно сложить первые три полученных столбца, чтобы получить время работы процессора с

момента старта ядра, а четвёртый столбец показывает время простоя процессора. Затем эти два значения будут передаваться клиенту, а уже в клиентском скрипте будет производиться вычитание предыдущего значения из текущего, чтобы получить информацию только за время между моментами обновления информации.

Также нужно сложить все значения трафика на разных сетевых интерфейсах, чтобы получить общие значения. Данные о трафике тоже отображаются за всё время с момента запуска(ip) интерфейса, поэтому данные, по аналогии с нагрузкой на процессор, будут обрабатываться в клиентском скрипте, чтобы определить скорость исходящего и входящего трафика за текущий момент.

### 2.3. Сигнализация о недоступности серверов

Помимо мониторинга нагрузки на сервера часто возникает необходимость оперативного реагирования в случае неожиданного выхода сервера из строя под действием повышенных нагрузок, атаки или аппаратных неисправностей. Для того, чтобы клиентская часть этой системы была автономной, было решено в её качестве использовать мессенджер Telegram, так как в данном случае нет никаких проблем с кроссплатформенностью и мобильностью клиентского приложения, а гибкий API позволяет очень легко создавать серверную часть.

Был написан отдельный скрипт для слежения за доступностью серверов. По аналогии с клиентской частью, можно управлять списком отслеживаемых серверов, добавляя и удаляя их из списка и сохраняя изменения в списке в файл настроек. Скрипт в активном режиме работы каждые 10 секунд опрашивает сервера из списка, пытаясь установить TCP-соединение на указанный порт. В случае успеха, сразу же происходит разрыв установленного соединения и переход к следующему серверу, а в случае ошибки подключения, эта ошибка и адрес сервера отправляются администратору в мессенджер.

### 3. Пользовательский интерфейс

#### 3.1. Интерфейс серверной части

Нет смысла реализовывать графический интерфейс пользователя для серверной части программы и на это есть несколько причин:

- 1) На серверах очень редко используется графический интерфейс и, чаще всего, там вообще нет поддержки графических программ.
- 2) Нет никакой необходимости во взаимодействии с серверной частью напрямую после того, как она запущена.
- 3) Так как для серверов всегда критична производительность, не стоит нагружать дополнительными сторонними библиотеками(такими как PyQt5) запускаемые на этих серверах программы.

В итоге, всё взаимодействие с сервером заключается в передаче некоторых параметров запуска посредством конфигурационного файла или аргументов при запуске.

Пример конфигурационного файла для сервера:

```
{  
  "port": 8080  
}
```

Список доступных аргументов командной строки:

```
$ ./server.py -h  
usage: server.py [-h] [-p PORT] [-n]  
optional arguments:  
  -h, --help            show this help message and exit  
  -p PORT, --port PORT  Run monitoring server on a specific port  
  -n, --no-reboot       Don't really reboot system
```

### 3.2. Интерфейс клиентской части системы мониторинга

Для построения пользовательского интерфейса клиентской части программы был использован фреймворк PyQt, реализующий возможности создания кроссплатформенных программ с классическим графическим интерфейсом. Помимо этого программа поддерживает дополнительные аргументы запуска:

```
$ ./client.py -h
usage: client.py [-h] [-a ADDRESS] [-c CONFIG] [-t]

optional arguments:
  -h, --help            show this help message and exit
  -a ADDRESS, --address ADDRESS
                        Connect to specific address instead of
                        listed in
                        config E.g. 127.0.0.1:8000
  -c CONFIG, --config CONFIG
                        Load config from specific file. Default
                        is
                        client_config.json
  -t, --text            Don't initialize UI, work in terminal
```

Программа состоит из одного основного окна, на котором отображается вся информация о сервере, подключение с которым установлено в текущий момент. Также на главном окне присутствуют элементы управления, позволяющие осуществлять изменение списка контролируемых серверов, а также выполнять некоторые действия с подключенным сервером: перезагрузка, завершение резидентной серверной программы, а также отправка Wake-On-Lan magic packet на указанный MAC-адрес в локальной сети сервера.



Для подключения к серверу нужно просто выбрать его в выпадающем списке в верхней части программы. Для отключения служит кнопка «Disconnect».

Помимо обычного графического интерфейса, поддерживается текстовый режим подключения в целях отладки. Подключение осуществляется с помощью флага —text согласно справке по командам(-h). После установления подключения терминал ожидает ввода пользователя, который напрямую отправляется серверу посредством ТСП-подключения, в ответ на ввод пользователя приходит ответ сервера на введённую команду. Сервер распознаёт в качестве команд слова, описанные в части 2.1, т. е. «fetch», «kill», «reboot» и MAC-адрес.

### 3.3. Интерфейс системы сигнализации

Резидентная программа для сигнализации выхода серверов из строя реализована в виде скрипта, подключающегося к API Telegram посредством заранее полученного токена, который необходим, чтобы осуществлять обработку команд и вывод сообщений через конкретного бота. При запуске проверяется наличие сохранённого токена, и если таковой не найден, необходимо запустить скрипт с флагом `--set-token`, указав после него действительный токен, после чего будет создан конфигурационный файл.

```
$ ./monitor-bot.py
Error reading config file!
Error initializing API: No API Token in config file.
Ask @BotFather for it and use --set-token argument.

$ ./monitor-bot.py --set-token
555858328:AAHsPgokz24sCZ5hLs1hb0rJhSbAHn7A0hc
Error reading config file!
Config saved.
```

Также необходимо сменить пароль доступа флагом `—set-password`.

Всё дальнейшее взаимодействие производится через бота Telegram. Вывод списка доступных команд происходит при вводе `«/help»`. Для начала мониторинга следует ввести команду `«/poll»`, для остановки — `«/stoppoll»`. Все команды, влияющие на список серверов, требуют предварительной авторизации с помощью команды `«/password»`.

## 4. Руководство пользователя

### 4.1. Основная информация

Данная программа предназначена для отслеживания нагрузки на сервера под управлением операционных систем на основе Debian/Red Hat Linux.

Программа состоит из трёх компонентов:

- Серверная часть — server.py

Предназначена для запуска на серверах для предоставления возможности подключения к ним для дальнейшего отслеживания производительности и работы.

- Клиентская часть — client.py

Графическое кроссплатформенное приложение, позволяющее подключаться к серверам для отслеживания их производительности и использования ими ресурсов в данный момент.

- Telegram-bot — monitor-bot.py

Предназначен для запуска на отдельном выделенном сервере, чтобы сигнализировать о недоступности подконтрольных серверов посредством мессенджера «Telegram»

### 4.2. Минимальные системные требования

Программа тестировалась на машине со следующими характеристиками:

Процессор 0,5 ГГц x 1 ядро

512 Мб оперативной памяти

Жёсткий диск 5 Гб

Заметной нагрузки на систему при этом замечено не было.

### 4.3. Процедура установки

Для работы всех частей программы необходимо наличие в системе установленного Python версии 3.

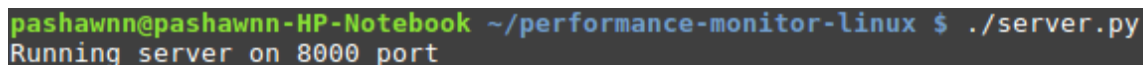
Серверный скрипт `server.py` поместить на сервера или в сетевую папку, доступную серверам. Настроить автоматический запуск посредством файла `rc.local` или любым другим способом. При необходимости указать нестандартный порт с помощью аргумента `-p` или `—port`.

Для запуска клиентской части необходимо установить библиотеки `PyQt5` и `pyqtgraph`, после чего `client.py` сразу готов к запуску.

Для запуска `monitor-bot.py` необходимо установить пакет `python-telegram-bot`. Для работы Telegram-бота необходимо сначала создать своего бота с помощью стандартного бота «@BotFather», после чего однократно запустить `monitor-bot.py` с флагом `-s` или `--set-token` и токеном после него. Затем можно добавить `monitor-bot.py` в автозагрузку.

#### 4.4. Работа с программой.

При запуске сервера(рисунок 1) он выводит сообщение о том, что он запущен и прослушивает указанный порт. После запуска более никакое взаимодействие с этой частью программы не требуется.



```
pashawnn@pashawnn-HP-Notebook ~/performance-monitor-linux $ ./server.py
Running server on 8000 port
```

Рисунок 1 – Запущенный сервер

При запуске клиентской части программы сразу же появляется главное окно программы(рисунок 2) с пустыми графиками использования ресурсов. Для подключения к серверу нужно добавить его в список с помощью кнопки «Add server»(а), после чего выбрать добавленный сервер из выпадающего меню(б). Для того, чтобы удалить сервер из списка, нажмите на кнопку «Delete server»(в). После подключения к серверу в пространстве(г) отобразится общая

информация, такая как имя сервера, показатель средней нагрузки и текущее время на сервере. Графики (д) и (е) показывают процентный коэффициент загруженности центрального процессора и оперативной памяти соответственно. На графике (ж) зелёным цветом показан исходящий трафик, а голубым — входящий. В таблице (з) отображается информация о примонтированных файловых системах. Чтобы разорвать соединение, следует нажать на кнопку «Disconnect»(и). Для перезагрузки сервера, завершения server.py и отправки Wake-On-Lan magic packet служат кнопки (к), (л) и (м) соответственно.

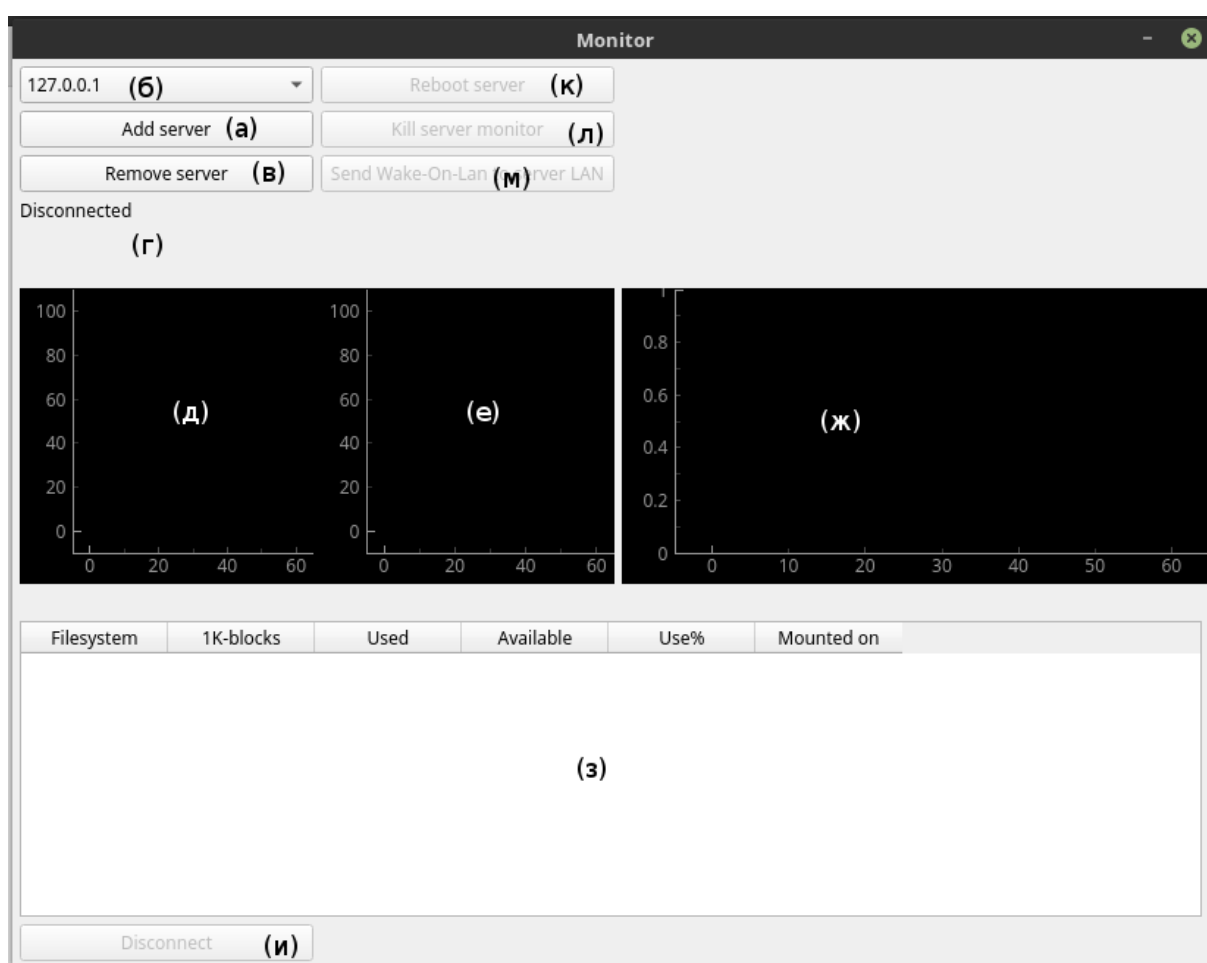


Рисунок 2. – Главное окно программы.

Для работы Telegram-бота необходимо запустить monitor-bot.py.

После запуска всё взаимодействие происходит через Telegram. Список команд всегда можно получить с помощью команды /help.

Перед работой с ботом необходимо авторизоваться, введя пароль, указанный в конфигурационном файле. Стандартный пароль – «1234». В системе может быть авторизован только один пользователь одновременно. Когда второй пользователь вводит пароль, первому приходит сообщение о том, что он деавторизован.

Полный список команд представлен на рисунке 3.

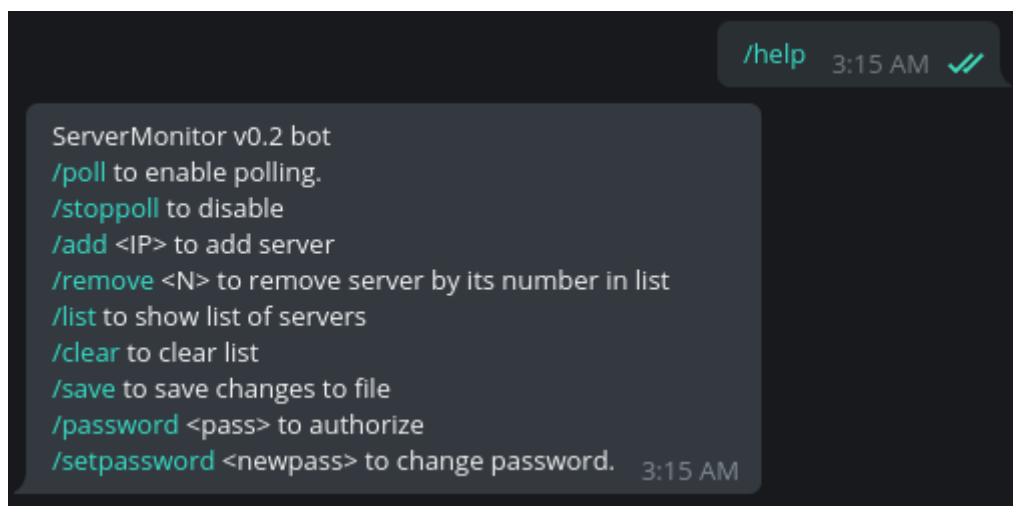


Рисунок 3. Вывод команды «/help»

## ЗАКЛЮЧЕНИЕ

Разработанный программный продукт предоставляет всю базовую функциональность, которая необходима администраторам небольших серверных парков для периодического отслеживания использования ресурсов серверами, а также постоянного слежения за доступностью сетевых ресурсов, чего вполне достаточно, так как более углублённая диагностика, производится, как правило, при непосредственном подключении к серверу по SSH и использовании гораздо более продвинутых программных средств.

Однако, стоит отметить, что у данного проекта есть перспективы в модификации и улучшении в сторону, например, наращивания функционала и добавления кроссплатформенности серверной части. Для осуществления этих планов можно так же продолжить разработку на Python, изучив способы отслеживания ресурсов в системах Windows Server.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Python 3.6.5 Documentation [Электронный ресурс] / Режим доступа: <https://docs.python.org/> - Загл. с экрана.
2. Manpages [Электронный ресурс] / Режим доступа: <https://man.cx/> - Загл. с экрана
3. PyQt5 Reference Guide [Электронный ресурс] / Режим доступа: <http://pyqt.sourceforge.net/Docs/PyQt5/introduction.html> - Загл. с экрана
4. pyqtgraph Documentation [Электронный ресурс] / Режим доступа: <http://pyqtgraph.org/documentation/> - Загл. с экрана
5. StackOverflow [Электронный ресурс] / Режим доступа: <https://stackoverflow.com/> - Загл. с экрана
6. Telegram bot API [Электронный ресурс] / Режим доступа: <https://core.telegram.org/bots/api> - Загл. с экрана



## Приложение 1. Исходный код.

```
#!/usr/bin/python3

import socket
import asyncore
import json
import re
import struct

from subprocess import Popen, check_output, PIPE, run

import argparse

from pathlib import Path

parser = argparse.ArgumentParser()
parser.add_argument("-p", "--port", type=int, default=-1,
                    help="Run monitoring server on a specific port")
parser.add_argument("-n", "--no-reboot", dest='noreboot', action='store_true',
                    help="Don't really reboot system")
args = parser.parse_args()

hostname = str(check_output("hostname"), 'utf-8')

def getPort():
    #Getting port from config, command line arguments or assuming default port(8000)

    file = Path("srv_config.json")

    if file.is_file():
        with open("srv_config.json", "r") as json_file:
            j = json.load(json_file)
    else:
        j = {}

    if args.port == -1:
        try:
            port = j["port"]
```

```

except KeyError:

    port = 8000

else:

    port = args.port

return port

def reg(dic):
#Parsing utilites output with regular expressions

    result = {}

    result["uptime"] = re.search(r'up (.+),[ 0-9]+user', dic["uptime"]).group(1)

    result["time"] = re.search(r'([0-9]+):([0-9]+):([0-9]+) up', dic["uptime"]).group(1)

    result["load_avg"] = re.search(r'([0-9.]+), [0-9.]+, [0-9.]+', dic["uptime"]).group(1)

    cpuStat = re.findall(r'^cpu +([0-9]+) ([0-9]+) ([0-9]+) ([0-9]+)', dic["cpu"])[0]

    result["cpu_used"] = int(cpuStat[0]) + int(cpuStat[1]) + int(cpuStat[2])

    result["cpu_free"] = int(cpuStat[3])

    result["total_memory"] = re.search(r'([0-9]+) K total memory', dic["ram"]).group(1)

    result["used_memory"] = re.search(r'([0-9]+) K used memory', dic["ram"]).group(1)

    result["free_memory"] = re.search(r'([0-9]+) K free memory', dic["ram"]).group(1)

    result["hostname"] = dic["host"]

    disks = {}

    traf = re.findall(r'mcast +\n +([0-9]+).\n +\n +([0-9]+)', dic["net"])

    traf_rx = 0

    traf_tx = 0

    for t in traf:

        traf_rx += int(t[0])

        traf_tx += int(t[1])

    result["net_rx"] = str(traf_rx)

    result["net_tx"] = str(traf_tx)

    for i, m in enumerate(re.findall(r'([^\n\\]+) [ ]+([0-9]+) [ ]+([0-9]+) [ ]+([0-9]+) [ ]+([0-9]+)%[ ]+([^\n\\]+)\n', dic["df"]))):

        disk = {}

        disk["filesystem"] = m[0]

```

```

        disk["1k_blocks"] = m[1]

        disk["used"]      = m[2]

        disk["available"] = m[3]

        disk["use"]       = m[4]

        disk["mounted_on"] = m[5]

        disks[str(i)] = disk

    result["disks"] = disks

    return result

print("Running server on %i port" % getPort())

def wol(string):
    #Creating and sending Wake-On-LAN magic packet

    string = re.sub(r'-', ':', string)

    splitMac = str.split(string, ':')

    print('Sending WOL magic packet to %s' % string)

    # Pack together the sections of the MAC address as binary hex
    hexMac = struct.pack(b'BBBBBB', int(splitMac[0], 16),

                           int(splitMac[1], 16),

                           int(splitMac[2], 16),

                           int(splitMac[3], 16),

                           int(splitMac[4], 16),

                           int(splitMac[5], 16))

    packet = '\xff' * 6 + string * 16 #create the magic packet from MAC address

    s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    s.sendto(bytes(packet, 'utf-8'),('255.255.255.255', 65535))

    s.close()

class SrvHandler(asyncore.dispatcher_with_send):

    def handle_read(self):

        data = self.recv(512)

```

```

if data == b'fetch':

    out = {}

    out["uptime"] = str(check_output("uptime"), 'utf-8')

    out["df"]    = str(check_output("df"), 'utf-8')

    out["ram"]   = str(check_output(["vmstat", "-s"]), 'utf-8')

    out["cpu"]   = str(check_output(("cat", '/proc/stat')), 'utf-8')

    out["host"]  = hostname

    out["net"]   = str(check_output(["ip", "-s", "link"]), 'utf-8')

    self.send(bytes(json.dumps(reg(out)), 'utf-8'))

elif data == b'reboot':

    if not args.noreboot:

        check_output(['reboot'])

        self.send(bytes('{"reply": "Rebooting..."}', 'utf-8'))

elif data == b'kill':

    self.send(bytes('{"reply": "Terminated by client."}', 'utf-8'))

    exit()

    elif re.match(r'^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$', str(data, 'utf-8')): #If received
MAC-address-like string

        self.send(bytes('{"reply": "Sending WOL."}', 'utf-8'))

        wol(str(data, 'utf-8'))

else:

    print(data)

class MonServer(asyncore.dispatcher):

    def __init__(self, host, port):

        asyncore.dispatcher.__init__(self)

        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)

        self.set_reuse_addr()

        self.bind((host, port))

        self.listen(5)

    def handle_accept(self):

        pair = self.accept()

```

```
    if pair is not None:
        sock, addr = pair
        print('Incoming connection from %s' % repr(addr))
        handler = SrvHandler(sock)
server = MonServer('', getPort())
asyncore.loop()
```

```
#!/usr/bin/python3
import socket
import json
import sys
import argparse
import re
from PyQt5.QtWidgets import (QWidget, QLabel,
    QComboBox, QApplication, QPushButton, QInputDialog, QTableWidget, QTableWidgetItem)
from PyQt5.QtCore import QTimer
from pathlib import Path
import pyqtgraph as pg

_height = 615
_width = 820

graph = {
    'cpu': [0]*60,
    'mem': [0]*60,
    'ntx': [0]*60,
    'nrx': [0]*60,
}

curRx, curTx, curCpu, curCpuUsed, curCpuFree = 0, 0, 0, 0, 0
```

```

class MonitorUI(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def closeEvent(self, event):
        if save: #If we have to save config(it was loaded), do it.
            with open(args.config, 'w') as json_file:
                json.dump(ips, json_file)
                print("Config saved to %s." % args.config)
            event.accept()

    def initUI(self):
        self.setGeometry(50, 50, _width, _height)
        self.setWindowTitle('Monitor')
        self.setFixedSize(_width, _height)
        self.combo = QComboBox(self)
        self.combo.setGeometry(5, 5, 200, 25)
        self.combo.activated[str].connect(self.onActivated)
        self.addBtn = QPushButton('Add server', self)
        self.addBtn.setGeometry(5, 35, 200, 25)
        self.addBtn.clicked.connect(self.addServerDialog)

        self.remBtn = QPushButton('Remove server', self)
        self.remBtn.setGeometry(5, 65, 200, 25)
        self.remBtn.clicked.connect(self.remCurrentServer)

        self.disBtn = QPushButton('Disconnect', self)
        self.disBtn.setGeometry(5, 585, 200, 25)
        self.disBtn.clicked.connect(self.disconnect)

        self.rebBtn = QPushButton('Reboot server', self)
        self.rebBtn.setGeometry(210, 5, 200, 25)
        self.rebBtn.clicked.connect(lambda: self.send('reboot'))

```

```

self.kilBtn = QPushButton('Kill server monitor', self)
self.kilBtn.setGeometry(210, 35, 200, 25)
self.kilBtn.clicked.connect(lambda: self.send('kill'))

self.wolBtn = QPushButton('Send Wake-On-Lan to server LAN', self)
self.wolBtn.setGeometry(210, 65, 200, 25)
self.wolBtn.clicked.connect(self.sendwol)

self.timer = QTimer(self)
self.timer.timeout.connect(self.fetch)

self.hstLbl = QLabel('Disconnected', self)
self.hstLbl.setGeometry(5, 95, 405, 15)

self.uptLbl = QLabel('', self)
self.uptLbl.setGeometry(5, 115, 405, 15)

self.avgLbl = QLabel('', self)
self.avgLbl.setGeometry(5, 135, 405, 15)

self.cpu = pg.PlotWidget(self, name='cpu_plot')
self.cpu.setMouseEnabled(x=False, y=False)
self.cpu.setGeometry(5, 155, 200, 200)
self.cpu.setXRange(1, len(graph['cpu'])-1)
self.cpu.setYRange(0, 100)
self.cpu.hideButtons()
self.cpuPlot = self.cpu.plot()

self.mem = pg.PlotWidget(self, name='mem_plot')
self.mem.setMouseEnabled(x=False, y=False)
self.mem.setGeometry(205, 155, 205, 200)

```

```

self.mem.setXRange(1, len(graph['mem'])-1)

self.mem.setYRange(0, 100)

self.mem.hideButtons()

self.memPlot = self.mem.plot()


self.net = pg.PlotWidget(self, name='net_plot')
self.net.setMouseEnabled(x=False, y=False)
self.net.setGeometry(415, 155, 400, 200)
self.net.setXRange(1, len(graph['ntx'])-1)
self.nrxPlot = self.net.plot(pen='#3875d8')
self.ntxPlot = self.net.plot(pen='#1cb226')


self.cpuLbl = QLabel("", self)
self.cpuLbl.setGeometry(5, 360, 200, 15)


self.memLbl = QLabel("", self)
self.memLbl.setGeometry(210, 360, 200, 15)


self.nrxLbl = QLabel("", self)
self.nrxLbl.setGeometry(415, 360, 200, 15)
self.ntxLbl = QLabel("", self)
self.ntxLbl.setGeometry(615, 360, 200, 15)


self.dskTbl = QTableWidget(self)
self.dskTbl.setColumnCount(6)

    self.dskTbl.setHorizontalHeaderLabels(['Filesystem', '1K-blocks', 'Used', 'Available', 'Use%',
'Mounted on'])

self.dskTbl.setGeometry(5, 380, 805, 200)

self.setBtnEnabled(False)

self.show()

def send(self, message):

```



```

try:
    self.sock.send(bytes(message, 'utf-8'))
except socket.error:
    self.connectionLost()

result = json.loads(str(self.sock.recv(4096), 'utf-8'))

return result

def connectionLost(self):
    self.hstLbl.setText('Connection lost.')
    self.timer.stop()

def sendwol(self):
    text, ok = QInputDialog.getText(self, 'Send Wake-On-Lan magic packet', 'Enter MAC:')
    if ok:
        self.send(checkmac(text))

def fetch(self):
    global curRx
    global curTx
    global curCpu
    global curCpuFree
    global curCpuUsed
    info = self.send('fetch')
    self.hstLbl.setText(info['hostname'])
    self.uptLbl.setText(info['uptime'])
    self.avgLbl.setText('Load avg.: %s Time: %s' % (info['load_avg'], info['time']))
    self.memLbl.setText('RAM usage: %.2fM / %.2fM' % (int(info['used_memory']/1024,
int(info['total_memory']/1024))
    info['cpu'] = 0.0

    updateGraph('mem', float(info['used_memory']/float(info['total_memory'])*100) #Update
graph info about memory usage in percents

```

```

self.cpuPlot.setData(y=graph['cpu'], clear=True)

self.memPlot.setData(y=graph['mem'], clear=True)

lastRx = curRx
lastTx = curTx

lastCpuFree = curCpuFree
lastCpuUsed = curCpuUsed
lastCpu = curCpu

curRx = int(info['net_rx'])
curTx = int(info['net_tx'])
curCpuUsed = int(info['cpu_used'])
curCpuFree = int(info['cpu_free'])

curCpu = (curCpuFree-lastCpuFree) / (curCpuUsed-lastCpuUsed)

if not lastRx == 0:
    updateGraph('cpu', curCpu)
    self.cpuLbl.setText('CPU usage: %.2f%%' % curCpu)
    updateGraph('nrx', (curRx-lastRx)/1024/1024)
    updateGraph('ntx', (curTx-lastTx)/1024/1024)

self.nrxPlot.setData(y=graph['nrx'])
self.ntxPlot.setData(y=graph['ntx'])

self.net.autoRange()

self.nrxLbl.setText('RX speed: {0:.2f} Mbps'.format((curRx-lastRx)/1024/1024))
self.ntxLbl.setText('TX speed: {0:.2f} Mbps'.format((curTx-lastTx)/1024/1024))

self.dskTbl.setRowCount(len(info['disks']))

for i, d in enumerate(info['disks']):
    self.dskTbl.setItem(i, 0, QTableWidgetItem(info['disks'][str(i)]['filesystem']))
    self.dskTbl.setItem(i, 1, QTableWidgetItem(info['disks'][str(i)]['1k_blocks']))
    self.dskTbl.setItem(i, 2, QTableWidgetItem(info['disks'][str(i)]['used']))
    self.dskTbl.setItem(i, 3, QTableWidgetItem(info['disks'][str(i)]['available']))
    self.dskTbl.setItem(i, 4, QTableWidgetItem(info['disks'][str(i)]['use']))
    self.dskTbl.setItem(i, 5, QTableWidgetItem(info['disks'][str(i)]['mounted_on']))

def setBtnEnabled(self, en):

```

```

self.disBtn.setEnabled(en)

self.kilBtn.setEnabled(en)

self.wolBtn.setEnabled(en)

self.rebBtn.setEnabled(en)

if en:
    self.timer.start(1000)
else:
    self.timer.stop()

def disconnect(self):
    self.sock.close()
    self.setBtnEnabled(False)

def onActivated(self, text):
    print('Connecting to ', text)
    self.sock = socket.socket()
    ip = parseIP(text)
    self.sock.connect((ip[0], 8000 if ip[1]==" else int(ip[1]) ))
    if self.sock:
        self.sock.settimeout(1)
        self.setBtnEnabled(True)

def addServerDialog(self):
    text, ok = QInputDialog.getText(self, 'Add server', 'Enter server IP:')
    if ok:
        self.addServer(parseIP(text))

def addServer(self, ip, dontInsert=False):
    string = ip[0]
    if not ip[1] == '': string += ':' + ip[1]
    if not dontInsert:

```

```

    ips['list'].append(ip)

    self.combo.addItems([string])

    print('Added %s to the list' % string)


def remCurrentServer(self):

    print('Deleting server from the list')

    self.combo.removeItem(self.combo.currentIndex())


def updateGraph(g, value):

    graph[g].pop(0)

    graph[g].append(value)


def parseIP(string):

    # Parsing IP from string to list with IP and port. If it's not matching regex, raising exception
    if re.match(r'^([0-9A-Za-z\.\.]+)?(\d{0,4})$', string):

        return re.findall(r'([0-9A-Za-z\.\.]+)?(\d{0,4})', string)[0]

    else:

        raise KeyError('Invalid IP!')


def checkmac(string):

    #Checking if input string is really MAC address(6 2-digit hex values, splitted with "-" or ":")
    #Returning input string if it does and raising exception if doesn't
    if re.match(r'^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$', string):

        return string

    else:

        raise KeyError('Invalid MAC!')


if __name__ == '__main__':

    save = False

    parser = argparse.ArgumentParser()

    parser.add_argument("-a", "--address", type=str, default='no',

                        help="Connect to specific address instead of listed in config\nE.g.

```

```

127.0.0.1:8000")

parser.add_argument("-c", "--config", type=str, default='client_config.json',
                    help="Load config from specific file.\nDefault is client_config.json")
parser.add_argument("-t", "--text", dest='textMode', action='store_true',
                    help="Don't initialize UI, work in terminal")

args = parser.parse_args()

if not args.textMode:
    app = QApplication(sys.argv)
    w = MonitorUI()

if args.address == 'no':
    file = Path(args.config)
    if file.is_file():
        with open(args.config, "r") as json_file:
            ips = json.load(json_file)
            print("Config loaded from %s" % args.config)
            save = True
    else:
        ips = {'list': []}
        save = True
else: ips = {'list': [parseIP(args.address)]}

for ip in ips['list']:
    if args.textMode:
        print(ip[0], ':', ip[1], '\n')
        sock = socket.socket()
        sock.connect((ip[0], int(ip[1])))
        print('Connected.')
        string = input('Enter command(fetch, reboot, kill or WOL MAC addr): ')
        sock.send(bytes(string, 'utf-8'))
        result = str(sock.recv(4096), 'utf-8')
        sock.close()
        print(result)
    else:

```

```
w.addServer(ip, dontInsert=True)

if not args.textMode:
    sys.exit(app.exec_())
```

```
#!/usr/bin/python3

import socket
import json
import argparse
import re
import logging
import threading

from pathlib import Path

from telegram.ext import Updater, CommandHandler, MessageHandler, Filters


chatId = 0

TIMEOUT = 1

PERIOD = 10.0


def tg_start(bot, update):

    """Greeting"""

    update.message.reply_text('ServerMonitor v0.2 bot\n/poll to enable polling.\n/stoppoll to disable\n'+\

                               '/add <IP> to add server\n/remove <N> to remove server by its number in list\n'+\

                               '/list to show list of servers\n/clear to clear list\n/save to save changes to file\n'+\

                               '/password <pass> to authorize\n/setpassword <newpass> to change password.')


def tg_authorized(update):
```

```

global chatId

if not (chatId == update.message.chat.id):
    update.message.reply_text('You are not authorized. Type /password <pass> to authorize.')
    return False
else:
    return True

def tg_poll(bot, update):
    """Starts polling servers every PERIOD seconds"""

    global threads_stopped #Store boolean to stop running threads

    if tg_authorized(update):
        threads_stopped = False #Don't stop threads
        update.message.reply_text('Servers polling enabled.')
        serversPoll() #Start new thread

def tg_pass(bot, update, args):
    """Authorization by password, specified in config"""

    global chatId

    if args[0] == config.get('password', '1234'):
        if not (chatId == 0): updater.bot.send_message(chatId, 'You\'ve been deauthorized.')
        chatId = update.message.chat.id
        update.message.reply_text('You\'re successfully authorized!\nType /poll to start polling servers.')
    else:
        update.message.reply_text('Password incorrect!')

def tg_stopPoll(bot, update):
    """Stops polling"""

    global threads_stopped

```

```

threads_stopped = True #Stop all threads

update.message.reply_text('Stopping servers polling...')


def tg_alarm(message):
    """Send message to last user, entered /poll command"""
    if not (chatId == 0):
        updater.bot.send_message(chatId, message)


def tg_add(bot, update, args):
    """Add server to the list in config['list'] dictionary"""
    global config
    if tg_authorized(update):
        try:
            ip = parseIP(args[0])
            config['list'].append(ip)
            update.message.reply_text('%s added to servers list.' % args[0])
        except KeyError:
            update.message.reply_text('Invalid IP address!')
        except IndexError:
            update.message.reply_text('You must specify IP address after "/add "')


def tg_set_pass(bot, update, args):
    global config
    if tg_authorized(update) and (len(args) == 1):
        config['password'] = args[0]
        update.message.reply_text('Password successfully changed!')


def tg_rem(bot, update, args):

```



```

"""Remove server from the list in config['list'] dictionary"""
if tg_authorized(update):
    try:
        try:
            args[0]
        except IndexError:
            update.message.reply_text('You must specify server number after "/remove "')
            return
        config.get('list', []).pop(int(args[0]))
        update.message.reply_text('Server #%%i removed from the list.' % int(args[0]))
    except ValueError:
        update.message.reply_text('Please enter number of server.')
    except IndexError:
        update.message.reply_text('Index out of list range.')

def tg_list(bot, update):
    """Show servers list from config['list'] dictionary"""
    if tg_authorized(update):
        string = 'List of servers to poll:\n'
        if len(config['list'])<1:
            update.message.reply_text('There is no servers to poll.')
        else:
            for i, ip in enumerate(config['list']):
                s = ip[0]
                if not ip[1]=='': s += ':' + ip[1]
                string+= '%i) %s\n' % (i, s)
            update.message.reply_text(string)

def tg_clear(bot, update):
    """Clear config['list'] dictionary"""

```

```

global config

if tg_authorized(update):

    config['list'] = []

    update.message.reply_text('Servers list cleared.')


def tg_save(bot, update):

    """Save config dictionary to file"""

    if tg_authorized(update):

        saveConfig()

        update.message.reply_text('Config saved to file.')


def parseIP(string):

    """Parsing IP from string to list with IP and port. If it's not matching regex, raising exception"""

    if re.match(r'^([0-9A-Za-z\.\.]+):?(\d{0,4})$', string):

        return re.findall(r'([0-9A-Za-z\.\.]+):?(\d{0,4})', string)[0]

    else:

        raise KeyError('Invalid IP!')


def serversPoll():

    """Threading function to poll servers"""

    global t

    if threads_stopped:

        return

    t = threading.Timer(float(config.get('period', PERIOD)), serversPoll)

    t.start()

    for ip in config['list']:

        try:

            sock = socket.socket()

            sock.settimeout(int(config.get('timeout', TIMEOUT)))

            sock.connect((ip[0], 8000 if ip[1]=="" else int(ip[1]))) #Trying to connect

```

```

    sock.close()                                #Disconnecting immediately if connected

except ConnectionRefusedError:                  #What if port is closed

    tg_alarm('Error connecting to %s!\nConnection refused.' % ip[0])

except socket.timeout:                          #What if time is out

    tg_alarm('Error connecting to %s!\nConnection timed out.' % ip[0])

except socket.gaierror:                         #What if IP is incorrect(e.g. octet > 255)

    tg_alarm('Error connecting to %s!\nMaybe, invalid IP?' % ip[0])


def saveConfig():

    with open(args.config, 'w') as json_file:

        json.dump(config, json_file)

    print("Config saved.")


def loadConfig():

    global config

    file = Path(args.config)

    if file.is_file():

        with open(args.config, "r") as json_file:

            try:

                config = json.load(json_file)

                print("Config loaded from %s" % args.config)

            try:

                config['list']

            except KeyError:

                config['list'] = []

        except json.decoder.JSONDecodeError:

            print("Error parsing JSON. No config loaded")

            config = {}

            config['list'] = []

    else:

```

```

print("Error reading config file!")

config = {}

config['list'] = []

def main():
    global updater
    global config
    global args

    logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
                        level=logging.INFO)

    logger = logging.getLogger(__name__)

    parser = argparse.ArgumentParser()
    parser.add_argument("-c", "--config", type=str, default='bot_config.json',
                        help="Load config from specific file.\nDefault is bot_config.json")
    parser.add_argument("-s", "--set-token", dest='token', type=str, default='',
                        help='Set Telegram API token')

    args = parser.parse_args()

    loadConfig()

    api_token = config.get('api_token', None)

    if not (args.token == ''):
        api_token = args.token
        config['api_token'] = api_token
        saveConfig()

    if not api_token:
        if (args.token == ''):
            print("Error initializing API: No API Token in config file.\nAsk @BotFather for it and use --set-token argument.")

            return

    updater = Updater(api_token)

```

```
dp = updater.dispatcher

dp.add_handler(CommandHandler("start", tg_start))
dp.add_handler(CommandHandler("help", tg_start))
dp.add_handler(CommandHandler("poll", tg_poll))
dp.add_handler(CommandHandler("stoppoll", tg_stopPoll))
dp.add_handler(CommandHandler("add", tg_add, pass_args=True))
dp.add_handler(CommandHandler("list", tg_list))
dp.add_handler(CommandHandler("remove", tg_rem, pass_args=True))
dp.add_handler(CommandHandler("clear", tg_clear))
dp.add_handler(CommandHandler("save", tg_save))
dp.add_handler(CommandHandler("password", tg_pass, pass_args=True))
dp.add_handler(CommandHandler("setpassword", tg_set_pass, pass_args=True))

updater.start_polling()

updater.idle()

if __name__ == '__main__':
    main()
```