

Разработка приложения для мониторинга системных ресурсов с использованием Python и библиотеки psutil

Малик Александр Игоревич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассматривается разработка приложения для мониторинга системных ресурсов (ЦП, память, диск, сеть) с использованием языка программирования Python и библиотеки psutil. Приложение позволяет в реальном времени отслеживать показатели работы системы, а также предоставляет пользователю возможность сохранять отчеты о текущем состоянии ресурсов и сбрасывать историю. Разработанное приложение предназначено для системных администраторов и пользователей, желающих эффективно следить за состоянием своего оборудования и предотвращать перегрузки. В статье представлены методы работы с системными ресурсами, а также особенности реализации графического интерфейса с использованием библиотеки Tkinter.

Ключевые слова: Мониторинг системных ресурсов, python, psutil, графический интерфейс, tkinter, цп, память, диск, сеть, приложение, отчеты, системный администратор.

Developing an application for monitoring system resources using Python and the psutil library

Malik Aleksandr Igorevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article discusses the development of an application for monitoring system resources (CPU, memory, disk, network) using the Python programming language and the psutil library. The application allows you to monitor system performance in real time, and also provides the user with the ability to save reports on the current status of resources and reset the history. The developed application is intended for system administrators and users who want to effectively monitor the condition of their equipment and prevent overloads. The article presents methods of working with system resources, as well as the specifics of implementing a graphical interface using the Tkinter library.

Keywords: System resource monitoring, python, psutil, gui, tkinter, cpu, memory, disk, network, application, reports, system administrator.

1 Введение

1.1 Актуальность

Современные информационные системы требуют постоянного мониторинга их состояния для обеспечения бесперебойной работы. Особенно это важно для серверов, рабочих станций и других вычислительных устройств, где перегрузка ресурсов может привести к снижению производительности и сбоям. В связи с этим существует необходимость в создании инструментов, позволяющих оперативно отслеживать параметры работы системы и выявлять возможные проблемы на ранних стадиях. Разработка приложения для мониторинга системных ресурсов на Python с использованием библиотеки `psutil` актуальна как для профессиональных системных администраторов, так и для широкого круга пользователей, стремящихся лучше понимать состояние своих устройств.

1.2 Обзор исследований

В.А. Рыжков в своём исследовании представил методику создания универсальной системы мониторинга API функций программных систем, основанную на объектной модели компонентов [1]. Д.И. Понамаренко и А.М. Ломакина в своей работе исследовали возможности использования языка программирования Python для автоматизации сбора данных [2]. А.А. Новичихина в своём исследовании уделила внимание анализу и описанию способов создания графического интерфейса на языке Python с использованием библиотеки Tkinter [3]. И.А. Баланов и Д.В. Кондратов изучали разнообразные программные продукты, предназначенные для сбора диагностических данных с персональных компьютеров [4]. В исследовании Ю.В. Редькина, И.И. Бузенкова и Е.М. Чернышевой представлен новый подход, направленный на улучшение качества систем сбора и статистической обработки данных [5].

1.3 Цель исследования

Целью данного исследования является разработка простого, но эффективного приложения для мониторинга системных ресурсов (ЦП, память, диск, сеть) с использованием языка программирования Python и библиотеки `psutil`, а также создание графического интерфейса для удобства взаимодействия пользователя с программой.

2 Материалы и методы

Язык программирования Python — выбор Python обусловлен его популярностью, доступностью библиотек и удобством работы с системными ресурсами.

Библиотека `psutil` — использована для получения информации о системных ресурсах, таких как использование процессора, памяти, диска и сети.

Tkinter — библиотека для разработки графического интерфейса, что позволило создать интуитивно понятный и удобный инструмент для взаимодействия с пользователем.

Matplotlib — библиотека для построения графиков, которая используется для отображения динамики использования ресурсов в реальном времени.

CSV-файлы — использовалась возможность сохранения отчетов в формате CSV для дальнейшего анализа данных.

Методы сбора статистики — периодическая съемка показателей системы с использованием библиотеки psutil через интервалы времени.

3 Результаты и обсуждения

1. Импорт библиотек (рис.1).

```
import psutil
import time
import tkinter as tk
from tkinter import messagebox, filedialog
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
import numpy as np
import csv
from datetime import datetime
```

Рисунок 1. Импорт библиотек

- **psutil** — библиотека для работы с системными ресурсами (ЦП, память, диск, сеть). Мы используем её для получения информации о процессоре, памяти, сети и диске.
- **tkinter** — стандартная библиотека Python для создания графических интерфейсов. Она используется для создания окна приложения, кнопок и отображения информации.
- **messagebox** и **filedialog** — используются для вывода окон с предупреждениями и для диалоговых окон для сохранения файлов.
- **matplotlib** — используется для создания графиков, на которых отображаются данные о системных ресурсах.
- **csv** — библиотека для работы с CSV-файлами, используется для сохранения отчета.
- **datetime** — библиотека для получения текущего времени и меток времени для отчетов.

2. Определение структуры для хранения данных (рис.2).

```
history_data = {  
    "cpu_percent": [],  
    "memory_percent": [],  
    "disk_percent": [],  
    "network_send_speed": [],  
    "network_recv_speed": [],  
    "timestamp": []  
}
```

Рисунок 2. Структура хранения данных

Создается структура для хранения исторических данных о системных ресурсах:

- **cpu_percent** — список с данными о загрузке процессора.
- **memory_percent** — список с данными о использовании памяти.
- **disk_percent** — данные о загрузке диска.
- **network_send_speed** — данные о скорости отправки данных по сети.
- **network_recv_speed** — данные о скорости приема данных по сети.
- **timestamp** — временные метки, когда были собраны данные.

3. Константы для порогов (рис.3).

```
CPU_THRESHOLD = 100 #  
MEMORY_THRESHOLD = 100  
DISK_THRESHOLD = 100 #
```

Рисунок 3. Пороги значений

Здесь задаются пороговые значения для различных ресурсов. Если использование ресурса (например, процессора) превышает 80%, то будет сгенерировано предупреждение.

4. Функция для получения системных данных (рис.4).

```
def get_system_info(): 1 usage
    # Получение информации о ЦП
    cpu_percent = psutil.cpu_percent(interval=0.1)
    cpu_per_core = psutil.cpu_percent(interval=0.1, percpu=True)

    # Получение информации о памяти
    memory = psutil.virtual_memory()
    memory_percent = memory.percent

    # Получение информации о диске
    disk = psutil.disk_usage('/')
    disk_percent = disk.percent

    # Используем psutil.disk_io_counters для получения скорости
    disk_io = psutil.disk_io_counters()
    disk_read = disk_io.read_bytes
    disk_write = disk_io.write_bytes
    disk_read_speed = (disk_read) / (1024 * 1024) # скорость чтения
    disk_write_speed = (disk_write) / (1024 * 1024) # скорость записи

    # Получение информации о сети
    network = psutil.net_io_counters()
    bytes_sent = network.bytes_sent
    bytes_recv = network.bytes_recv
    network_send_speed = (bytes_sent) / (1024 * 1024) # скорость отправки
    network_recv_speed = (bytes_recv) / (1024 * 1024) # скорость приема

    # Формируем вывод
    system_info = {
        'cpu_percent': cpu_percent,
        'memory_percent': memory_percent,
        'disk_percent': disk_percent,
        'disk_read_speed': disk_read_speed,
        'disk_write_speed': disk_write_speed,
        'network_send_speed': network_send_speed,
        'network_recv_speed': network_recv_speed,
        'cpu_per_core': cpu_per_core # Использование каждого ядра
    }

    return system_info
```

Рисунок 4. Код функции для получения системных данных

В этой функции происходит сбор данных о системных ресурсах с использованием библиотеки psutil.

- `psutil.cpu_percent(interval=0.1)` — получает процент загрузки процессора за время 0.1 секунды.
- `psutil.virtual_memory()` — собирает данные о текущем использовании памяти.
- `psutil.disk_usage('/')` — собирает информацию о занятии пространства на основном диске.

- `psutil.disk_io_counters()` — собирает данные о количестве прочитанных и записанных байтов на диске.
- `psutil.net_io_counters()` — собирает данные о переданных и полученных байтах через сеть.

5. Функция для обновления данных (рис.5).

```
def update_system_info(): 2 usages
    system_info = get_system_info()
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    # Добавляем текущие данные в историю
    history_data["cpu_percent"].append(system_info['cpu_percent'])
    history_data["memory_percent"].append(system_info['memory_percent'])
    history_data["disk_percent"].append(system_info['disk_percent'])
    history_data["network_send_speed"].append(system_info['network_send_speed'])
    history_data["network_recv_speed"].append(system_info['network_recv_speed'])
    history_data["timestamp"].append(timestamp)

    # Обновляем метки на экране
    cpu_label.config(text=f"ЦП: {system_info['cpu_percent']}%\n"
                      f"Ядра: {' '.join([f'{x}%' for x in system_info['cpu_per_core']])}")
    memory_label.config(text=f"Память: {system_info['memory_percent']}%")
    disk_label.config(text=f"Диск: {system_info['disk_percent']}%")
    network_label.config(
        text=f"Отправлено: {system_info['network_send_speed']:.2f} МБ/с\nПолучено: {system_info['network_recv_speed']:.2f} МБ/с")

    # Проверка порогов
    if system_info['cpu_percent'] > CPU_THRESHOLD:
        show_warning(resource="CPU", system_info['cpu_percent'])
    if system_info['memory_percent'] > MEMORY_THRESHOLD:
        show_warning(resource="Memory", system_info['memory_percent'])
    if system_info['disk_percent'] > DISK_THRESHOLD:
        show_warning(resource="Disk", system_info['disk_percent'])

    # Обновляем графики
    update_graph(cpu_graph, system_info['cpu_percent'], title="Использование процессора", cpu_data)
    update_graph(memory_graph, system_info['memory_percent'], title="Использование памяти", memory_data)
    update_graph(disk_graph, system_info['disk_percent'], title="Использование диска", disk_data)
    update_graph(network_send_graph, system_info['network_send_speed'], title="Сеть: Отправлено (МБ/с)", network_send_data)
    update_graph(network_recv_graph, system_info['network_recv_speed'], title="Сеть: Получено (МБ/с)", network_recv_data)

    # Отображаем скорость чтения и записи на диск
    disk_speed_label.config(
        text=f"Чтение: {system_info['disk_read_speed']:.2f} МБ/с\nЗапись: {system_info['disk_write_speed']:.2f} МБ/с")

    # Ожидаем 1 секунду перед обновлением
    root.after(ms=1000, update_system_info)
```

Рисунок 5. Код функции обновления данных

В этой функции происходит обновление значений для всех системных показателей в реальном времени.

Данные добавляются в историю (`history_data`), обновляются метки на интерфейсе (`cpu_label`, `memory_label`, `disk_label`, `network_label`), а также проверяются пороговые значения. Если какое-либо значение превышает порог, срабатывает предупреждение с помощью функции `show_warning`.

Также обновляются графики для различных метрик.

6. Функция для отображения предупреждения (рис.6).

```
def show_warning(resource, value): 3 usages
    messagebox.showwarning(title=f"Внимание: {resource}!", message=f"{resource} использует {value}% ресурсов!")
```

Рисунок 6. Код функции отображения предупреждения

Эта функция выводит окно с предупреждением, если использование ресурса превышает пороговое значение.

7. Функция для обновления графиков (рис.7).

```
def update_graph(graph, value, title, data_list):
    # Добавляем новые данные на график
    data_list.append(value)
    if len(data_list) > max_data_points:
        data_list.pop(0)

    # Обновляем график
    graph.clear()
    graph.plot(data_list, label=title, color='b')
    graph.set_ylim(0, 100) # Устанавливаем диапазон
    graph.set_title(title)
    graph.legend(loc="upper right")
    canvas.draw()
```

Рисунок 7. Код функции для отображения графиков

- Эта функция обновляет данные на графиках.
- Вначале она добавляет новое значение в список данных.
- Если данных больше, чем разрешено по количеству (в данном случае, max_data_points), старые данные удаляются.
- Затем происходит очистка и перерисовка графика с новыми данными.

8. Функция для сохранения отчета (рис.8).

```
def save_report(): 1 usage
    file_path = filedialog.asksaveasfilename(defaultextension=".csv",
                                              filetype=[("CSV files", "*.csv"), ("Text files", "*.txt")])

    if file_path:
        with open(file_path, 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(
                ["Timestamp", "CPU (%)", "Memory (%)", "Disk (%)", "Network Send (MB/s)", "Network Recv (MB/s)"])
            for i in range(len(history_data["timestamp"])):
                writer.writerow([
                    history_data["timestamp"][i],
                    history_data["cpu_percent"][i],
                    history_data["memory_percent"][i],
                    history_data["disk_percent"][i],
                    history_data["network_send_speed"][i],
                    history_data["network_recv_speed"][i]
                ])
        messagebox.showinfo(title="Готово", message="Отчет успешно сохранен!")
```

Рисунок 8. Код функции сохранения отчета

- Эта функция сохраняет отчет о текущем состоянии ресурсов в CSV-файл.
- Файл сохраняется в выбранном пользователем пути.

9. Функция для сброса истории системных ресурсов (рис.9).

```
def reset_history(): 1 usage
    global history_data, cpu_data, memory_data, disk_data, network_send_data, network_recv_data
    history_data = {
        "cpu_percent": [],
        "memory_percent": [],
        "disk_percent": [],
        "network_send_speed": [],
        "network_recv_speed": [],
        "timestamp": []
    }
    cpu_data = []
    memory_data = []
    disk_data = []
    network_send_data = []
    network_recv_data = []
    messagebox.showinfo( title: "История сброшена", message: "История системных ресурсов была сброшена.")
```

Рисунок 9. Код функции сброса истории

10. Запуск главного окна приложения (рис.10).

```

root = tk.Tk()
root.title("Мониторинг системных ресурсов")

# Настроим размеры окна
root.geometry("800x700")

# Создание фрейма для кнопок
button_frame = tk.Frame(root)
button_frame.pack(side="top", pady=10)

# Кнопки для сохранения и сброса истории
save_button = tk.Button(button_frame, text="Сохранить отчет", command=save_report, font=("Helvetica", 12))
save_button.pack(side="left", padx=10)

reset_button = tk.Button(button_frame, text="Сбросить историю", command=reset_history, font=("Helvetica", 12))
reset_button.pack(side="left", padx=10)

# Создание меток для отображения значений
cpu_label = tk.Label(root, font=("Helvetica", 14))
cpu_label.pack(pady=10)

memory_label = tk.Label(root, font=("Helvetica", 14))
memory_label.pack(pady=10)

disk_label = tk.Label(root, font=("Helvetica", 14))
disk_label.pack(pady=10)

network_label = tk.Label(root, font=("Helvetica", 14))
network_label.pack(pady=10)

# Метка для отображения скорости чтения и записи диска
disk_speed_label = tk.Label(root, font=("Helvetica", 14))
disk_speed_label.pack(pady=10)

# Настройка графиков для отображения данных
fig, (cpu_graph, memory_graph, disk_graph, network_send_graph, network_recv_graph) = plt.subplots(nrows=5, ncols=1, figsize=(8, 10))
fig.tight_layout(pad=3.0)

# Список для хранения данных для каждого графика
max_data_points = 60 # Количество точек на графике

# Создание холста для отображения графиков
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.get_tk_widget().pack(pady=20)

# Запуск обновления данных
root.after(ms=1000, update_system_info)

# Запуск главного цикла программы
root.mainloop()

```

Рисунок 10. Код запуска главного окна приложения

- В этом фрагменте создается основное окно с помощью библиотеки Tkinter.
- Создаются кнопки, метки и графики для отображения системных данных.
- Запускается бесконечный цикл для обновления данных с интервалом 1 секунда.

11. Внешний вид программы (рис.11).

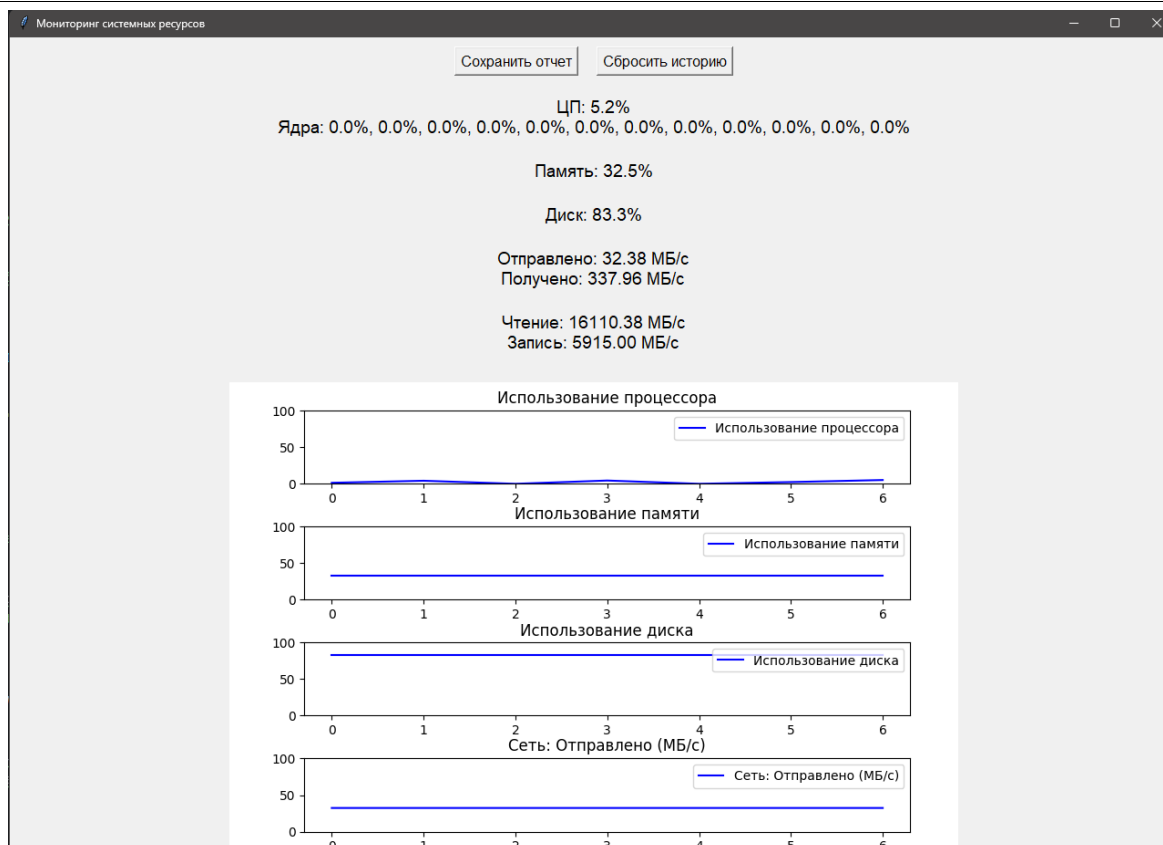


Рисунок 11. Внешний вид программы

Выводы

Разработанное приложение для мониторинга системных ресурсов эффективно решает задачи по отслеживанию состояния вычислительных устройств в реальном времени. Использование библиотеки `psutil` позволяет получать актуальные данные о состоянии процессора, памяти, диска и сети, а графический интерфейс на базе Tkinter обеспечивает удобство работы пользователя. Приложение имеет возможность сохранять отчеты и сбрасывать историю, что повышает его гибкость и функциональность. Разработка данного инструмента является важным шагом в направлении улучшения мониторинга и управления системными ресурсами, а также может быть использована как для личных нужд, так и в профессиональной сфере для администрирования серверных и пользовательских машин.

Библиографический список

1. Рыжков В.А. Разработка универсальной системы мониторинга API функций программных систем на основе объектной модели компонентов // В сборнике: Информатика: проблемы, методология, технологии. материалы XV международной научно-методической конференции. 2015. С. 367-370.
2. Понамаренко Д.И., Ломакина А.М. Применение языка Python в автоматизации сбора данных // Наука и образование транспорту. 2022. № 1. С. 333-336.

3. Новичихина А.А. Разработка простых приложений средствами tkinter в Python // Актуальные исследования. 2023. № 4-1 (134). С. 32-35.
4. Баланов И.А., Кондратов Д.В. Проблемы автоматического сбора диагностических данных о персональном компьютере // Математическое моделирование, компьютерный и натурный эксперимент в естественных науках. 2023. № 2. С. 45-51.
5. Редькин Ю.В., Бузенков И.И., Чернышева Е.М. Решение задачи сбора и обработки данных в целях повышения качества контроля // В сборнике: Современные информационные технологии в управлении качеством. сборник статей VI Международной научно-прикладной конференции. — 2018. С. 31-36.