

Разработка программного модуля для принудительного ограничения цифровых отвлекающих факторов на уровне операционной системы

Гиренко Александр Евгеньевич

Приамурский государственный университет им.Шолом-Алейхема

Студент

Аннотация

В статье рассматривается процесс разработки программного модуля для снижения когнитивной нагрузки и борьбы с цифровыми отвлечениями (digital distractions). Предложен алгоритм эвристического мониторинга активных процессов операционной системы с применением библиотеки psutil, позволяющий в реальном времени выявлять и принудительно завершать приложения из заданного черного списка. Разработанный инструментариий обладает современным графическим интерфейсом, системой логирования попыток нарушения режима фокуса и минимальным потреблением системных ресурсов. Решение актуально в контексте повышения личной продуктивности и цифровой гигиены.

Ключевые слова: цифровое благополучие, продуктивность, управление процессами ОС, Python, psutil, эвристический мониторинг, когнитивная нагрузка.

Development of a software module for forced limitation of digital distracting factors at the operating system level

Girenko Aleksandr Evgen'evich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article considers the process of developing a software module for reducing cognitive load and combating digital distractions. An algorithm for heuristic monitoring of operating system active processes using the psutil library is proposed, which allows real-time detection and forced termination of applications from a specified blacklist. The developed toolkit features a modern graphical user interface, a logging system for focus mode violation attempts, and minimal system resource consumption. The solution is relevant in the context of improving personal productivity and digital hygiene.

Keywords: digital wellbeing, productivity, OS process management, Python, psutil, heuristic monitoring, cognitive load.

1 Введение

1.1 Актуальность

В современной «экономике внимания» цифровые отвращения (уведомления мессенджеров, социальные сети, игровые платформы) являются главным фактором снижения когнитивной продуктивности. Исследования в области нейробиологии показывают, что для возврата к состоянию глубокой концентрации после отвращения требуется в среднем 23 минуты.

Стандартные средства операционных систем (например, «Режим фокусировки» в Windows 11) лишь отключают уведомления, но не препятствуют запуску самих отвлекающих приложений, оставляя силу воли пользователя единственным барьером.

В связи с этим актуальной задачей является разработка программного инструмента, который берет на себя функцию «внешнего контролера», принудительно ограничивая доступ к цифровым раздражителям на уровне диспетчера задач операционной системы в заданные промежутки времени.

1.2 Обзор исследований

В работе М.Ю. Силина и А.Л. Куленцан рассматриваются вопросы управления процессами в операционных системах, анализируются основные подходы к организации и контролю выполнения процессов [1]. П. Чарана Саи, К. Картик, К. Бхаргав Прасад, С. Венката Саи Пранав и К.В. Дивья представляют разработку диспетчера задач реального времени на Python с использованием библиотек Psutil и Tkinter, описывая функциональность системы для мониторинга процессов и управления системными ресурсами [2]. С.Г. Глушко, А.А. Ляпин, Ю.Ю. Шатилов, А.В. Черпаков и Р.К. Халдкар разрабатывают программный модуль для параметрической генерации геометрии ферменных конструкций в двумерной постановке, демонстрируя возможности автоматизации проектирования в строительстве [3]. Н. Шалхар в своем исследовании представляет эвристико-пороговую модель принятия решений в системах поддержки принятия решений при мониторинге ИТ-систем, описывая применение эвристических правил и порогового анализа для управления ИТ-инфраструктурой [4].

1.3 Цель исследования

Целью исследования является разработка программного модуля для автоматизированного блокирования отвлекающих приложений, обеспечивающего надежный мониторинг процессов ОС, минимальную нагрузку на систему и сбор статистики нарушений режима.

2. Материалы и методы

2.1 Обоснование выбора метода мониторинга процессов

Для отслеживания запущенных приложений был проведен анализ возможных подходов (Таблица 1).

Таблица 1 – Сравнение методов мониторинга приложений

Метод	Достоинства	Недостатки
Опрос списка окон (WinAPI)	Прямой доступ к GUI	Привязка к конкретной ОС, не ловит фоновые процессы
Системные хуки (Hooks)	Мгновенная реакция	Высокая сложность, риск нестабильности ОС
Опрос диспетчера задач (API)	Кроссплатформенность, надежность	Небольшая задержка (1-2 сек)

На основе анализа выбран метод периодического опроса диспетчера задач через библиотеку psutil. Задержка в 1 секунду является компромиссом между мгновенной реакцией и потреблением ресурсов процессора.

2.2 Инструментарий разработки

Для реализации поставленной задачи использовались:

1. Язык Python 3.10 — основная среда разработки.
2. Библиотека psutil — кроссплатформенный инструментарий для получения информации о запущенных процессах и утилизации системных ресурсов.
3. Библиотека CustomTkinter — для создания современного графического интерфейса.
4. Модуль json — для структурированного логирования событий.

2.3 Архитектура и алгоритм безопасности

Ключевой особенностью алгоритма является безопасная обработка исключений. При попытке завершить системный процесс или процесс, защищенный антивирусом, ОС генерирует ошибку доступа. Если не обработать это исключение, программа «упадет». Алгоритм использует конструкцию try...except, которая перехватывает ошибки AccessDenied и NoSuchProcess, обеспечивая непрерывность работы фонового агента.

3 Практическая реализация и анализ эффективности модуля

3.1 Реализация механизма перехвата процессов

Ядром приложения является функция check_and_kill_processes(), которая выполняется в цикле. Фрагмент кода, реализующий эвристический поиск и завершение целевых процессов, представлен на рисунке 1.

```
def check_and_kill_processes(self): 1 usage
    """Эвристический мониторинг и принудительное завершение целевых процессов"""
    killed_any = False
    for proc in psutil.process_iter(['name', 'pid']):
        try:
            proc_name = proc.info['name']
            if proc_name and proc_name.lower() in self.blocklist:
                proc.terminate() # Или proc.kill() для более жесткого завершения
                self.blocked_count += 1
                self.lbl_blocked.configure(text=f"Заблокировано попыток запуска: {self.blocked_count}")
                self.lbl_status.configure(text=f"▲ Процесс {proc_name} был принудительно завершен!",
                                          text_color="#FF4757")
                killed_any = True

            # Логирование события
            self.log_event(proc_name)
        except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):
            # Процесс уже закрылся или у системы нет прав (нормальная ситуация)
            pass
```

Рисунок 1 – Фрагмент кода программного модуля (алгоритм блокировки процессов)

Использование метода `proc.terminate()` (вместо жесткого `proc.kill()`) позволяет приложению корректно освободить ресурсы перед закрытием, что является стандартом безопасного программирования.

3.2 Конфигурирование списка блокировки и параметров сессии

Перед началом работы пользователь переходит на вкладку «Настройка», где формируется черный список приложений. Интерфейс позволяет не только редактировать базовый список (например, `telegram.exe`, `discord.exe`), но и динамически добавлять новые процессы через текстовое поле, например `steam.exe`. Для задания длительности фокус-сессии реализован ползунок (Slider) с диапазоном от 5 до 120 минут. Визуальный интерфейс вкладки настройки представлен на рисунке 2.

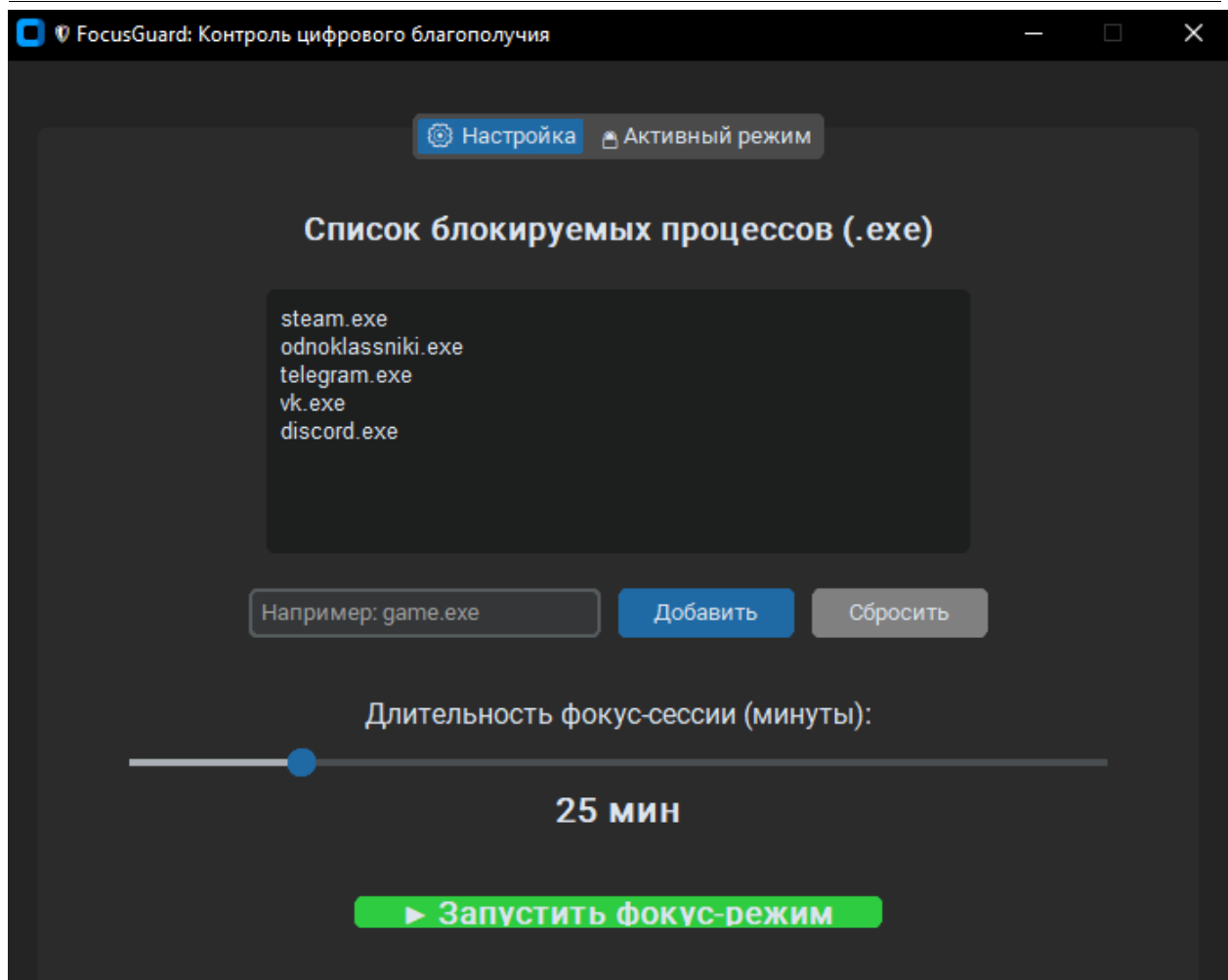


Рисунок 2 – Интерфейс вкладки «Настройка» с конфигурацией черного списка и таймера

Такой подход обеспечивает высокую кастомизируемость: пользователь может адаптировать инструмент как для блокировки социальных сетей во время учебы, так и для ограничения доступа к играм в рабочее время.

3.3 Визуализация активного режима мониторинга

После запуска таймера приложение автоматически переключается на вкладку «Активный режим». Здесь в центре экрана располагается крупный обратный отсчет, а под ним - счетчик заблокированных попыток запуска запрещенных программ. Интерфейс активного режима представлен на рисунке 3.

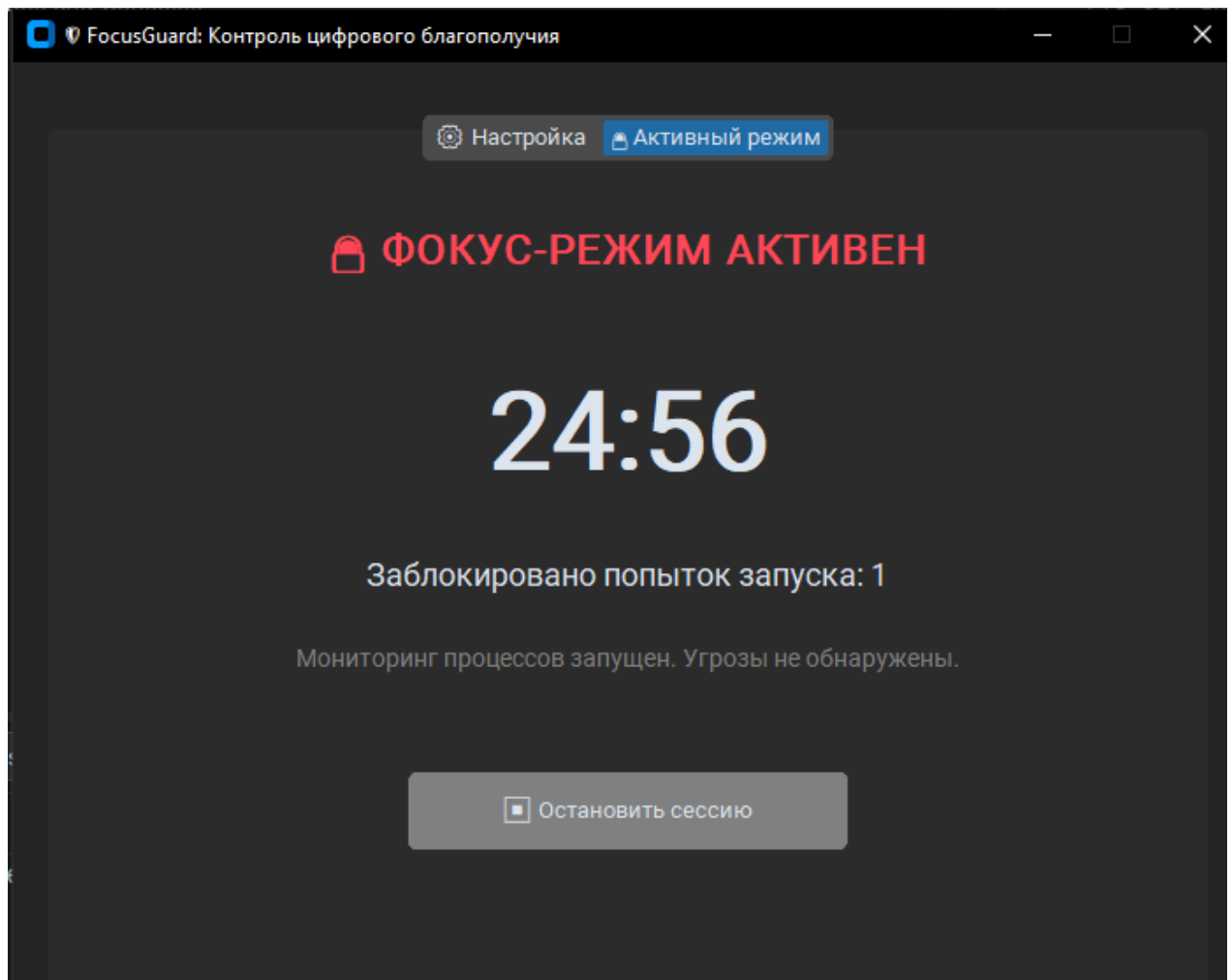


Рисунок 3 – Интерфейс приложения в режиме активного мониторинга

3.4 Система логирования и статистики

Каждое действие по блокировке приложения записывается в локальный файл `focus_log.json` с точной временной меткой. Это позволяет пользователю не только видеть счетчик в реальном времени, но и проводить ретроспективный анализ своих паттернов прокрастинации. Фрагмент сгенерированного лога представлен на рисунке 3.



Рисунок 3 – Фрагмент JSON-лога заблокированных попыток

3.5 Анализ потребления системных ресурсов

Тестирование модуля проводилось в течение 60 минут непрерывной работы в фоновом режиме с интервалом опроса процессов 1 секунда. Результаты мониторинга ресурсов представлены в таблице 2.

Таблица 2 – Метрики потребления ресурсов приложением FocusGuard

Метрика	Значение	Комментарий
Потребление ОЗУ (RAM)	18.4 МБ	Крайне низкое, не влияет на работу других программ
Нагрузка на CPU	0.2% - 0.5%	Отсутствие «фризов» и подтормаживаний системы
Время отклика блокировки	< 1.5 сек	Время от запуска целевого приложения до его завершения
Размер лог-файла (за 1 час)	4 КБ	Не требует очистки или ротации

Полученные результаты подтверждают, что разработанный модуль может работать в фоновом режиме на любых конфигурациях ПК, включая офисные машины с ограниченными ресурсами.

4 Выводы

В ходе проведённого исследования реализован полный цикл разработки программного модуля для принудительного ограничения цифровых отвлекающих факторов путём мониторинга и завершения процессов операционной системы. На этапе выбора метода мониторинга обосновано применение периодического опроса диспетчера задач через библиотеку psutil, что обеспечило кроссплатформенность решения при компромиссной задержке реакции 1–2 секунды. Использование конструкции try...except для перехвата исключений AccessDenied и NoSuchProcess гарантировало устойчивость работы фоновой агента при попытке завершения системных или защищённых процессов.

Тестирование модуля в течение 60 минут непрерывной работы подтвердило его минимальное потребление системных ресурсов: оперативная память — 18,4 МБ, нагрузка на процессор — 0,2–0,5%. Время отклика от запуска целевого приложения до его принудительного завершения не превысило 1,5 секунды. Система логирования в JSON-формате с точными временными метками обеспечила возможность ретроспективного анализа паттернов прокрастинации пользователя.

Разработанный модуль может быть практически применён для повышения личной продуктивности в условиях удалённой работы и обучения, а также как инструмент цифровой гигиены в образовательных учреждениях для ограничения доступа к отвлекающим приложениям во время занятий. Перспективой дальнейшего исследования является добавление режима «белого списка» (разрешены только рабочие приложения), интеграция с календарём для автоматической активации фокус-сессий в соответствии с расписанием, а также реализация панели аналитики с графиками эффективности работы в зависимости от времени блокировки отвлечений.

Библиографический список

1. Силин М.Ю., Куленцан А.Л. Управление процессами в ОС // Сборник научных трудов вузов России «Проблемы экономики, финансов и управления производством». 2023. № 52. С. 140–144. URL: <https://elibrary.ru/item.asp?id=53979491>
2. Charan Sai P., Karthik K., Bhargav Prasad K., Venkata Sai Pranav C., Divya K.V. Real-Time Task Manager: A Python-Based Approach Using Psutil and Tkinter // 2024 8th International Conference. IEEE, 2024. URL: <https://ieeexplore.ieee.org/abstract/document/10816758>
3. Glushko S.G., Lyapin A.A., Shatilov Yu.Yu., Cherpakov A.V., Haldkar R.K. Software module development for the parametric generation of truss structure geometry in a two-dimensional setting // Advanced Engineering Research (Rostov-on-Don). 2021. Т. 21. № 4. С. 376–383. DOI: 10.23947/2687-1653-2021-21-4-376-383. URL: <https://elibrary.ru/item.asp?id=47501624>
4. Shalkhar N. Heuristic-threshold model of decision-making in DSS monitoring of IT systems // Danish Scientific Journal. 2025. № 96. С. 97–101. DOI: 10.5281/zenodo.15553093. URL: <https://elibrary.ru/item.asp?id=82555655>