Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Операционные среды и системное программирование

К защите допустить:
Заведующий кафедрой информатики
_____С.И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ПРОГРАММА ДЛЯ СОЗДАНИЯ И ЗАПУСКА МАКРОСОВ

БГУИР КП 1-40 04 01 007

Студент В.П. Бычко
Руководитель Н.Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Двадцать первый век бросает людям новый вызов – промышленная революция и достижения научно технического прогресса увеличили ценность интеллектуального труда и уменьшали количество рутинных действий, выполняемых на разных работах. С новыми открытиями в сфере информационных технологий появились новые работы, а вместе с тем и новая рутина – набор повторяющегося текста или выполнение одних и тех же действий. На помощь в автоматизации и упрощении таких задач приходит макрос – последовательность нажатий на клавиши или движений мышью, которые преобразуются в более длинные последовательности нажатий клавиш или движений мышью.

Целью данной курсовой работы является написание программы, позволяющей записывать и воспроизводить макросы в операционной системе windows. Для этого будет произведен анализ типов макросов, возможностей операционной системы windows по эмуляции ввода с клавиатуры и способов взаимодействия платформы .NET с ними.

Данную цель можно разложить на следующие задачи – изучение теории по созданию макросов, запись и имитация ввода клавиатуры и мыши средствами языка C#, создание программы с пользовательским интерфейсом, позволяющей считывать и воспроизводить ввод клавиатуры и мыши.

Для решения первой задачи необходимо осуществить поиск и изучение специализированной литературы, связанной с данной проблемой.

Для решения второй задачи можно использовать механизм Windows Hooks. Он позволяет перехватывать события мыши и клавиатуры. В C# с данным механизмом работает библиотека UserActivityMonitor. Для имитации нажатий на клавиатуру и действий мышью будут использованы функции keybd_event и mouse_event.

Для решения третьей задачи предполагается использовать WinForms.

В результате выполнения данной курсовой работы будет получена программа с графическим интерфейсом, позволяющая осуществлять ввод определенных макросов и их запуск.

1 МАКРОСЫ. ИСТОРИЯ, ВИДЫ

В данной части курсовой работы будет дано определение понятию макрос. Будут рассмотрены их основные виды, история и причины их возникновения.

1.1 Макросы

Макрос (что означает «макроинструкция») – это программируемый шаблон, который преобразует определенную последовательность ввода в заданную последовательность вывода. Макросы позволяют сделать задачи менее повторяющимися, представляя сложную последовательность нажатий клавиш, движений мыши, команд или других типов ввода. [1]

Применение макроса к вводу известно, как раскрытие макроса. Ввод и вывод могут представлять собой последовательность лексических токенов, символов, или синтаксическое дерево. В приложениях поддерживаются символьные макросы, чтобы упростить вызов команд. Макросы, состоящие из последовательности токенов или синтаксического дерева, поддерживаются в некоторых языках программирования для повторного использования кода или расширения возможностей языка.

Синтаксическое дерево — это дерево, в котором каждый листовой узел представляет операнд, а каждый внутренний узел представляет оператор. Дерево разбора сокращенно обозначается как синтаксическое дерево. Синтаксическое дерево обычно используется при представлении программы в древовидной структуре. [2]

Макросы используются для предоставления программисту последовательности вычислительных инструкций в виде одного оператора программы, что делает задачу программирования менее утомительной и уменьшает количество ошибок, возникающих в работе. Макросы, с греческого переводится как большой, получили именно такое название, потому что "большой" блок кода может быть раскрыт из "маленькой" последовательности символов. Макросы часто позволяют использовать позиционные или ключевые параметры, которые определяют, что генерирует препроцессор, и могут быть использованы для создания целых программ в зависимости от операционной системы, платформы или других факторов. Термин происходит от "макроинструкции", и такие расширения изначально использовались при генерации кода на языке ассемблера.

Различают такие виды макросов: макросы клавиатуры и мыши, макросы замены текста, процедурные и синтаксические макросы.

1.2 Макросы клавиатуры и мыши

Макросы клавиатуры и мыши позволяют преобразовать короткие последовательности нажатий клавиш и действий мыши в другие, обычно более трудоемкие, последовательности нажатий клавиш и действий мыши. Таким образом, часто используемые или повторяющиеся последовательности нажатий клавиш и движений мыши могут быть автоматизированы. Отдельные программы для создания этих макросов называются считывателями макросов.

В 1980-х годах макропрограммы – первоначально SmartKey, затем SuperKey, KeyWorks, Prokey – были очень популярны, сначала как средство автоматического форматирования сценариев, затем для различных задач пользовательского ввода. Эти программы были основаны на режиме работы terminate-and-stay-resident и применялись ко всему вводу с клавиатуры, независимо от того, в каком контексте это происходило. Они начали устаревать после появления пользовательских интерфейсов, управляемых мышью. Более широкое распространение макросов клавиатуры и мыши в текстовых редакторах и электронных таблицах, с возможностью создания для конкретных приложений, еще больше ускорило устаревание макропрограмм с режимом работы terminate-and-stay-resident.

Prokey – ранний процессор макросов клавиатуры для DOS и Windows от CE Software, который позволял пользователям исключить повторяющийся ввод текста, настраивая появление текста или серии команд в виде макроса. [3]

Terminate and Stay Resident (TSR) – это программа, которая находится в памяти компьютера до тех пор, пока ее не вызовут для выполнения. Программы TSR изначально были разработаны для однозадачных дисковых операционных систем (DOS). Компьютеры, работающие под управлением DOS, могут одновременно выполнять только одну программу? но с помощью TSR они могут позволить программам переключаться между задачами без необходимости перезагрузки всей программы. При установке TSR на компьютер ему присваивается управляющий символ, иначе называемый горячей клавишей. Как только пользователь компьютера открывает программу, он может нажать горячую клавишу, чтобы немедленно запустить программу TSR. [4]

Макросы клавиатуры и мыши получили широкое распространение в играх жанра MMORPG, в которых часто встречаются повторяющиеся задачи.

Однако, поскольку это делается без участия человека и это может сильно влиять на экономику игры, использование макросов является нарушением условий пользования или лицензионного соглашения большинства MMORPG, и их администраторы тратят значительные усилия на подавление использования программ такого рода. Однако макросы могут быть использованы для безобидных вещей.

Макросы клавиатуры и мыши, созданные с использованием встроенных функций приложения, иногда называются макросами приложения. Они создаются путем однократного выполнения последовательности действий и предоставления приложению возможности записывать их. Также может существовать базовый язык макропрограммирования, чаще всего язык скриптов, с прямым доступом к функциям приложения.

Текстовый редактор для программистов Emacs возводит эту идею в Абсолют. По сути, большая часть редактора состоит из макросов. Изначально Emacs был разработан как набор макросов на языке редактирования TECO; позже он был портирован на диалекты Lisp.

Lisp представляет собой один из самых старых языков программирования. Кроме того, он является первым функциональным, получившим большую популярность. Создание его ядра приходится на 60-е годы предыдущего столетия – разработчиком стал ученый Дж. Маккарти. Основополагающая структура данных языка Lisp – список, откуда и пошло наименование языка. Широкую известность Лисп получил в 70-80-е годы 20 века. В то время он использовался в качестве базового языка для научной деятельности в сфере искусственного интеллекта. [5]

Другой текстовый редактор для программистов, Vim (потомок vi), также имеет реализацию макросов клавиатуры. Он может записывать в регистр (макрос) то, что пользователь набирает на клавиатуре, и это можно воспроизводить или редактировать точно так же, как VBA макросах для Microsoft Office. В Vim также есть язык сценариев под названием Vimscript для создания макросов. Основными особенностями Vim считаются: постоянное многоуровневое дерево отмены, обширная система плагинов, поддержка сотен языков программирования и форматов файлов, мощный поиск и замена, интегрируемость со многими инструментами.

Visual Basic для приложений – это язык программирования, разработанный и принадлежащий Microsoft. VBA используется для написания программ для операционной системы Windows и работает как внутренний язык программирования в приложениях Microsoft Office (MS Office, Office), таких как Access, Excel, PowerPoint, Publisher, Word и Visio. VBA позволяет

пользователям выполнять настройку, выходящую за рамки того, что обычно доступно в хост-приложениях MS Office. Необходимо обратить внимание, что его функции развились из макроязыков, которые первоначально были включены в некоторые из этих приложений, и заменила их.

1.3 Макросы замены текста

Макросы замены текста. Языки, такие как C и некоторые языки ассемблера, имеют рудиментарные макросистемы, реализованные в виде препроцессоров для компилятора или ассемблера. Макросы препроцессора C работают путем простой текстовой замены на уровне токена, а не на уровне символов. Однако макросредства более сложных ассемблеров, например, IBM High Level Assembler (HLASM), не могут быть реализованы с помощью препроцессора; код для сборки инструкций и данных перемежается с кодом для сборки вызовов макросов.

Классическое использование макросов – в системе компьютерной верстки TeX и ее производных, где большая часть функциональности основана на макросах.

TeX, язык компьютерного программирования с описанием страниц, разработанный в 1977-86 годах Дональдом Кнутом, профессором Стэнфордского университета, для улучшения качества математических обозначений в его книгах. Системы форматирования текста, в отличие от текстовых процессоров WYSIWYG («Что видишь, то и получаешь»), встраивают в документ команды форматирования простого текста, которые затем интерпретируются языковым процессором для создания отформатированного документа для отображения или печати. TeX помечает текст, выделенный курсивом, например, как {\it это выделено курсивом}, который затем отображается как выделенный курсивом.

1.4 Процедурные макросы

PL/I – разработанный в 1964 году язык программирования, созданный для научных, инженерных и бизнес-ориентированных вычислений. Содержит столь широкий (особенно на время создания) набор синтаксических конструкций и встроенных функций, что в IBM даже не сразу появился компилятор, поддерживающего все возможности языка. PL/I поддерживает рекурсию и структурное программирование, широко применяется в обработке данных. [9]

Макросы на языке PL/I написаны в подмножестве самого PL/I: компилятор выполняет "инструкции препроцессора" во время компиляции, и выходные данные этого выполнения являются частью компилируемого кода. Возможность использовать знакомый процедурный язык в качестве макроязыка дает гораздо больше возможностей, чем у макросов подстановки текста. Макросы в PL/I, как и во многих ассемблерах, могут иметь дополнительные функции, например, устанавливать переменные, к которым могут получить доступ другие макросы.

Макросы frame technology имеют свой собственный синтаксис команд, но также могут содержать текст на любом языке. Frame содержит последовательность команд для frame процессора и текстовые элементы, содержащие framed component, семантика кадрированного компонента не влияет на поведение процессора кадров; он просто руководствуется frame командами, с которыми сталкивается. Каждый фрейм является как общим компонентом в иерархии вложенных подсборок, так и процедурой интеграции со своими подсборными фреймами (рекурсивный процесс, который разрешает конфликты интеграции в пользу подсборок более высокого уровня). Выходные данные представляют собой пользовательские документы, обычно компилируемые исходные модули. Фреймовая технология позволяет избежать распространения похожих, но слегка отличающихся компонентов – проблемы, которая преследует разработчиков программного обеспечения с момента изобретения макросов и подпрограмм. [10]

Большинство языков ассемблера имеют менее мощные процедурные макросредства, например, позволяющие повторять блок кода N раз для развертывания цикла; но они имеют синтаксис, совершенно отличный от реального языка ассемблера.

1.5 Синтаксические макросы

Макросистемы, такие как описанный ранее препроцессор C, которые работают на уровне лексических токенов, не могут надежно сохранять лексическую структуру. Синтаксические макросистемы вместо этого работают на уровне абстрактных синтаксических деревьев и сохраняют лексическую структуру исходной программы. Наиболее широко используемые реализации синтаксических макросистем находятся в языках, подобных Lisp. Эти языки особенно подходят для этого стиля макросов из-за их единообразного синтаксиса. В частности, именно он упрощает определение вызовов макросов. Макросы Lisp преобразуют саму структуру программы, при

этом для выражения таких преобразований доступен полный язык. Хотя синтаксические макросы часто встречаются в языках, подобных Lisp, они также доступны на других языках, таких как Prolog, Erlang, Dylan, Scala, Nemerle, Rust. Они также доступны в качестве сторонних расширений для JavaScript и C#.

1.5.1 Макросы раннего Lisp

До того, как в Lisp появились макросы, в нем были так называемые FEXPRs, функционально-подобные операторы, входными данными которых были не значения, вычисляемые с помощью аргументов, а скорее синтаксические формы аргументов, и выходными данными которых были значения, которые будут использоваться при вычислении. Другими словами, FEXPRs были реализованы на том же уровне, что и EVAL, и предоставляли окно на уровень метаоценки.

В 1963 году Тимоти Харт предложил добавить макросы в Lisp 1.5 в AI Memo 57: MACRO Definitions for LISP.

1.5.2 Анафорические макросы

Анафорический макрос – это тип программного макроса, который намеренно фиксирует некоторую форму, предоставляемую макросу, на которую может ссылаться анафора (выражение, ссылающееся на другое). Т.е. анафорический макрос – это макрос, который намеренно фиксирует переменную из форм, предоставленных макросу. Эти переменные позволяют контролировать расширение макросов. Через эти окна возможно манипулировать расширением посредством комбинаций. Анафорические макросы впервые появились в книге Пола Грэхема "On Lisp", и их название является отсылкой к лингвистической анафоре – использованию слов в качестве замены предшествующих слов. [11]

1.5.3 Гигиенические макросы

В середине восьмидесятых в ряде статей было введено понятие расширения гигиенических макросов (синтаксические правила), система на основе шаблонов, в которой синтаксические среды определения макроса и использования макроса различаются, что позволяет разработчикам макросов и пользователям не беспокоиться о случайном захвате переменных. Гигиенические макросы стандартизированы для Scheme в стандартах R5RS, R6RS и R7RS. Существует ряд конкурирующих реализаций гигиенических макросов, таких как синтаксические правила, синтаксический регистры, явное

переименование и синтаксические замыкания. Как синтаксические правила, так и синтаксические регистры стандартизированы в стандартах Scheme.

Racket объединил понятия гигиенических макросов с "башней вычислителей", так что время синтаксического расширения одной макросистемы является обычным временем выполнения другого блока кода, и показал, как применять чередующееся расширение и синтаксический анализ в языке без скобок.

1.6 Макросы для машинно-независимого программного обеспечения

Макросы обычно используются для сопоставления короткой строки (вызова макроса) с более длинной последовательностью инструкций. Другое, менее распространенное использование макросов заключается в обратном: для сопоставления последовательности инструкций со строкой макроса. Именно такого подхода придерживалась система мобильного программирования STAGE2, которая использовала элементарный компилятор макросов (называемый SIMCMP) для отображения конкретного набора команд данного компьютера в машинно-независимые макросы. Приложения (в частности, компиляторы), написанные с помощью этих машинно-независимых макросов, затем могут быть запущены без изменений на любом компьютере, оснащенном элементарным компилятором макросов. Первым приложением, запускаемым в таком контексте, является более сложный и мощный компилятор макросов, написанный на машинно-независимом макроязыке. Этот компилятор макросов применяется к самому себе в режиме начальной загрузки для создания скомпилированной и гораздо более эффективной версии самого себя. Преимущество этого подхода заключается в том, что сложные приложения могут быть перенесены с одного компьютера на совершенно другой компьютер с очень небольшими усилиями (для каждой архитектуры целевой машины достаточно написать элементарный компилятор макросов). Появление современных языков программирования, в частности C, компиляторы для которых доступны практически на всех компьютерах, сделало такой подход излишним. Однако это был один из первых примеров (если не первый) начальной загрузки компилятора.

В данной части курсовой работы было дано определение понятию макрос. Рассмотрены их основные виды, история и причины их возникновения.

2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В данной части курсовой работы будет произведен краткий обзор инструментов и платформы для разработки, которые будут использованы в данной курсовой работе.

2.1 Структура и архитектура платформы

Архитектура Windows – это совокупность программных и аппаратных компонентов, которые определяют, как операционная система управляет ресурсами компьютера, обеспечивает взаимодействие между приложениями и устройствами, решает проблемы и защищает данные от постороннего доступа. Архитектура Windows может быть разделена на два основных уровня: ядро (kernel) и пользовательский режим (user mode). Ядро – это низкоуровневая часть операционной системы, которая работает в специальном режиме процессора и имеет непосредственный доступ к аппаратным ресурсам. Пользовательский режим – это высокоуровневая часть операционной системы, которая работает в обычном режиме процессора и содержит разнообразные приложения и службы. Ядро и пользовательский режим общаются друг с другом через специальные механизмы, такие как системные вызовы (system calls), сообщения (messages) и объекты ядра (kernel objects). [12]

Windows поддерживает различные архитектуры процессоров, такие как x86, x64, ARM и ARM64. Каждая архитектура имеет свои особенности, такие как размер адресного пространства, набор инструкций, регистры и т.д. Windows адаптируется к разным архитектурам с помощью специального слоя абстракции аппаратного обеспечения (hardware abstraction layer, HAL), который предоставляет единый интерфейс для доступа к аппаратным ресурсам.

Windows отличается от других операционных систем своей гибкостью и совместимостью. Windows может запускать приложения, написанные для разных API (application programming interface), таких как Win32, POSIX, WinRT и т.д., благодаря подсистемам пользовательского режима, которые реализуют эти API. Windows также может работать с разными типами устройств, таких как клавиатуры, мыши, диски, принтеры и т.д., благодаря драйверам устройств, которые обеспечивают связь между ядром и устройствами.

Ядро – это низкоуровневая часть операционной системы, которая работает в специальном режиме процессора и имеет непосредственный доступ к аппаратным ресурсам, таким как память, диски, процессоры и т.д. Ядро отвечает за обработку запросов от приложений и устройств, планирование потоков исполнения, синхронизацию данных, обработку прерываний и исключений, защиту памяти и т.д.

Ядро Windows имеет гибридную структуру, то есть оно состоит из нескольких компонентов, которые работают на разных уровнях абстракции. Основные компоненты ядра Windows – это.

Ядро (kernel) – это самая низкоуровневая часть ядра, которая содержит базовые функции для работы с процессором, памятью и прерываниями. Ядро также содержит диспетчер объектов (object manager), который управляет объектами ядра, такими как потоки (threads), процессы (processes), события (events), семафоры (semaphores) и т.д., и предоставляет механизмы для синхронизации и коммуникации между ними.

Слой абстракции аппаратного обеспечения (hardware abstraction layer, HAL) – это программный компонент, который действует как интерфейс между аппаратным обеспечением и операционной системой. Он обеспечивает согласованный и единообразный способ взаимодействия программ с различными аппаратными устройствами без необходимости знания конкретных деталей каждого устройства. HAL позволяет разработчикам писать код, не зависящий от аппаратного обеспечения, что упрощает перенос программного обеспечения на разные платформы.

HAL предоставляет набор стандартизированных функций и протоколов, которые абстрагируют низкоуровневые детали аппаратных устройств. Эта абстракция позволяет программам взаимодействовать с аппаратными устройствами, используя высокоуровневый интерфейс прикладного программирования (API), вместо того, чтобы разбираться в тонкостях драйверов и протоколов, специфичных для конкретного устройства.

HAL упрощает разработку программного обеспечения, предоставляя согласованный и четко определенный интерфейс для взаимодействия с аппаратными устройствами. Это означает, что разработчикам не нужно изучать конкретные детали каждого устройства, с которым они работают, что экономит время и усилия. Также, HAL улучшает переносимость, поскольку программное обеспечение, написанное с использованием HAL, может быть легко перенесено на различные платформы с минимальными изменениями. Наконец, HAL повышает удобство обслуживания, отделяя программный код,

зависящий от оборудования, от остального программного обеспечения, что упрощает обновление и исправление ошибок.

HAL поддерживает широкий спектр аппаратных устройств, включая принтеры, клавиатуры, мыши, сетевые адаптеры, мониторы и устройства хранения данных. Однако важно отметить, что доступность HAL для конкретного устройства зависит от используемой операционной системы или платформы. Такие операционные системы, как Windows, Linux и другие, предоставляют HAL для широкого спектра устройств, в то время как поддержка других может быть более ограниченной. [13]

Executive – это часть ядра, которая содержит высокоуровневые службы для управления системными ресурсами, такими как память (memory manager), процессы (process manager), потоки (thread manager), ввод-вывод (I/O manager), безопасность (security reference monitor), конфигурация (configuration manager) и т.д. Executive также содержит подсистему Plug and Play (PnP), которая обнаруживает, устанавливает и управляет устройствами в системе.

Драйверы – это часть ядра, которая содержит программы для работы с различными типами устройств, такими как диски, клавиатуры, мыши, принтеры и т.д. Драйверы обеспечивают связь между ядром и устройствами через стек ввода-вывода (I/O stack) и модель драйверов Windows (WDM).

Ядро Windows работает в соответствии с моделью клиент-сервер. Клиентами являются приложения и подсистемы пользовательского режима, которые отправляют запросы к ядру через системные вызовы (system calls). Серверами являются компоненты ядра, которые обрабатывают эти запросы и возвращают результаты клиентам. Системные вызовы – это специальные функции API, которые переводят процессор из обычного режима в специальный режим и передают управление ядру. Системные вызовы обычно имеют префикс Nt или Zw, например, NtCreateFile или ZwQuerySystemInformation.

Ядро Windows также работает в соответствии с моделью прерываний. Прерывания – это сигналы от аппаратных устройств или программ, которые требуют срочного внимания от ядра. Прерывания могут быть синхронными или асинхронными. Синхронные прерывания – это прерывания, которые возникают в результате действий текущего потока, например, деления на ноль или обращения к неверному адресу памяти. Асинхронные прерывания – это прерывания, которые возникают в результате действий внешних устройств, например, нажатия клавиши или прихода пакета по сети. Прерывания обрабатываются специальными функциями, называемыми обработчиками

прерываний (interrupt handlers), которые выполняют необходимые действия и возвращают управление ядру.

Драйвер физического устройства, получающего прерывания, регистрирует одну или несколько процедур обслуживания прерываний (ISR) для обслуживания прерываний. Система вызывает ISR каждый раз, когда получает это прерывание.

Устройства для портов и автобусов до PCI 2.2 создают прерывания на основе линий. Устройство создает прерывание, отправляя электрический сигнал на выделенный контакт, известный как линия прерывания. Версии Microsoft Windows до Windows Vista поддерживают только прерывания на основе строк.

Начиная с PCI 2.2, устройства PCI могут создавать прерывания, сигнализируют о сообщениях. Устройство создает прерывание с сигналом сообщения, записывая значение данных на определенный адрес. Операционные системы Windows Vista и более поздних версий поддерживают прерывания как на основе строк, так и на основе сообщений.

Система поддерживает два разных типа ISR.

Драйвер может зарегистрировать подпрограмму InterruptService для обработки прерываний на основе строк или сообщений. (Это единственный тип, доступный до Windows Vista.) Система передает предоставленное драйвером значение контекста.

Драйвер может зарегистрировать подпрограмму InterruptMessageService для обработки прерываний, сигналируемых сообщениями. Система передает как предоставленное драйвером значение контекста, так и идентификатор сообщения прерывания. [14]

Ядро Windows обеспечивает высокую производительность, надежность и безопасность системы за счет своей гибридной структуры, модели клиент-сервер, модели прерываний и механизмов защиты памяти и доступа к ресурсам. Ядро Windows также поддерживает многозадачность (multitasking), многопоточность (multithreading) и многопроцессорность (multiprocessing), что позволяет эффективно использовать ресурсы компьютера и выполнять несколько задач одновременно.

Многопоточность позволяет выполняться нескольким потокам одновременно. Поток – это путь выполнения программы. Это также наименьшая единица выполнения, которая планирует Win32. Поток состоит из стека, состояния регистров ЦП и записи в списке выполнения системного планировщика. Каждый поток делится всеми ресурсами процесса.

Процесс состоит из одного или нескольких потоков и кода, данных и других ресурсов программы в памяти. Типичными ресурсами программы являются открытые файлы, семафоры и динамически выделенная память. Программа выполняется, когда системный планировщик предоставляет один из своих потоков управления выполнением. Планировщик определяет, какие потоки должны выполняться и когда они должны выполняться. Потоки более низкого приоритета могут ждать, пока потоки с более высоким приоритетом выполняют свои задачи. На компьютерах с несколькими обработчиками планировщик может перемещать отдельные потоки на разные процессоры, чтобы сбалансировать нагрузку ЦП.

Каждый поток в процессе работает независимо. Если они не отображаются друг другу, потоки выполняются по отдельности и не знают о других потоках в процессе. Однако потоки, совместно использующие общие ресурсы, должны координировать свою работу с помощью семафоров или другого метода межпроцессного взаимодействия. [15]

Подсистемы – это часть операционной системы, которая работает в пользовательском режиме и предоставляет различные API (application programming interface) для приложений. API – это набор функций, констант, типов данных и структур, которые определяют, как приложение может взаимодействовать с операционной системой и использовать ее ресурсы. Подсистемы реализуют разные API для разных типов приложений, таких как Win32, POSIX, WinRT и т.д.

Win32 – это набор API (интерфейс разработки приложений) Microsoft Windows, используемый для разработки 32-разрядных приложений. Другие его названия это: Windows API, WinAPI.

Win32 отвечает за функции в таких категориях:

– администрирование и управление: установка, настройка и обслуживание приложений или систем;

– диагностика: устранение неполадок приложений, системных проблем и мониторинг производительности;

– графика и мультимедиа: форматирование текста, графика, аудио и видео;

– сеть: сетевое взаимодействие, общение приложений на разных компьютерах в сети;

– системные службы: предоставление доступа к компьютерным ресурсам (памяти, файловой системе, устройствам, процессам и потокам);

– пользовательский интерфейс Windows: создание и управление пользовательским интерфейсом (вывод на дисплей, запросы на ввод пользователя и другая поддержка взаимодействия с пользователем).

В каждой своей версии Windows менялся, также изменялось и дорабатывалось API. Поэтому Win32 неправильно отражает корни в 16-разрядных Windows (там Win16), и не поддерживает 64 битные Windows. Для них есть Win64, но в целом это все Windows API. [16]

POSIX – это API для приложений, написанных для UNIX-подобных операционных систем, который поддерживает базовые функции для работы с файлами, процессами, сигналами и т.д. POSIX позволяет запускать на Windows приложения, которые были разработаны для других платформ.

WinRT (Windows Runtime) - общее наименование целого набора базовых функций, являющихся основой для разработки Metro приложений для Windows 8. WinRT является API на основе технологии COM и поэтому может быть использовано в различных языках программирования, таких как C++, JavaScript, C#, VB.NET.

WinRT это новое API операционной системы Windows, своего рода аналог старого, доброго Win32 API. Для создания богатого пользовательского интерфейса в Metro стиле применяется HTML/CSS в случае использования JavaScript. При программировании на языках C++, C#, VB.NET пользовательский интерфейс описывается с помощью XAML. Для взаимодействия с ядром операционной системы Windows теперь используется WinRT, к которой можно легко обращаться из различных языков программирования. Функции API WinRT кажутся родными для каждого языка, поскольку, WinRT предоставляет определенную версию библиотеки для каждого из них.

WinRT позволяет:

– использовать более простую модель программирования для разработки пользовательского интерфейса для Windows;

– использовать модель программирования приложений WPF, Silverlight и XAML в качестве языка разметки пользовательского интерфейса;

– легко обращаться к нему из различных языков программирования;

– обеспечивать связь приложений с операционной системой;

– обеспечивать доступ к устройствам и сетям, подключённым к персональному компьютеру под управлением ОС Windows.

Подсистемы пользовательского режима общаются с ядром через системные вызовы (system calls), которые передают запросы от приложений к ядру и возвращают результаты от ядра к приложениям. Подсистемы также

общаются друг с другом через механизмы IPC (interprocess communication), такие как каналы (pipes), очереди сообщений (message queues), разделяемая память (shared memory) и т.д.

Подсистемы пользовательского режима работают под управлением интегральной подсистемы (integral subsystem), которая выполняет системные функции от имени подсистем среды. Интегральная подсистема состоит из двух компонентов: службы локального процедурного вызова (local procedure call service, LPCS) и службы сервера процессов (process server service, PSS). LPCS отвечает за передачу сообщений между подсистемами пользовательского режима и ядром через порты LPC (local procedure call ports). PSS отвечает за создание и завершение процессов подсистем пользовательского режима.

Подсистемы пользовательского режима обеспечивают гибкость и совместимость операционной системы за счет поддержки разных API для разных типов приложений. Подсистемы также обеспечивают изоляцию и безопасность операционной системы за счет работы в обычном режиме процессора и ограничения доступа к ресурсам ядра. Подсистемы также поддерживают многозадачность (multitasking) и многопоточность (multithreading), что позволяет выполнять несколько приложений и потоков одновременно.

Драйвер – это программный компонент, который позволяет операционной системе и устройству взаимодействовать. Например, когда приложению требуется считывать данные с устройства, оно вызывает функцию, реализованную операционной системой. Затем операционная система вызывает функцию, реализованную драйвером. Драйвер, обычно разрабатываемый производителем устройства, знает, как взаимодействовать с аппаратным обеспечением устройства для получения данных. Как только драйвер получает данные, он передает их обратно в операционную систему, которая затем передает их обратно в приложение. Драйверы устройств обеспечивают связь между ядром и устройствами через стек ввода-вывода (I/O stack) и модель драйверов Windows (WDM).

Стек ввода-вывода – это набор структур данных и функций, которые определяют, как данные передаются от приложений к устройствам и обратно. Стек ввода-вывода состоит из нескольких слоев, каждый из которых содержит один или несколько драйверов устройств. Слои стека ввода-вывода – это.

Высший слой (upper layer) – это слой, который содержит драйверы устройств, которые реализуют логические функции для работы с данными, например, шифрование, сжатие, фильтрация и т.д.

Промежуточный слой (intermediate layer) – это слой, который содержит драйверы устройств, которые реализуют протоколы для работы с данными, например, SCSI, USB, TCP/IP и т.д.

Низший слой (lower layer) – это слой, который содержит драйверы устройств, которые реализуют физические функции для работы с данными, например, чтение и запись на диск, передача и прием по сети и т.д.

Диспетчер ввода-вывода в режиме ядра Windows управляет взаимодействием между приложениями и интерфейсами, предоставляемыми драйверами устройств. Поскольку устройства работают на скоростях, которые могут не соответствовать операционной системе, взаимодействие между операционной системой и драйверами устройств в основном осуществляется с помощью пакетов запросов ввода-вывода (IRP). Эти пакеты аналогичны сетевым пакетам или пакетам сообщений Windows. Они передаются из операционной системы в определенные драйверы и от одного драйвера к другому.

Система ввода-вывода Windows предоставляет многоуровневую модель драйверов, называемую стеками. Обычно протоколы IRP передаются от одного драйвера к другому в одном стеке для облегчения взаимодействия. Например, драйвер джойстика должен был бы подключаться к USB-концентратору, который, в свою очередь, должен был бы подключаться к хост-контроллеру USB, который затем должен был бы подключаться через шину PCI к остальному компьютерному оборудованию. Стек состоит из драйвера джойстика, USB-концентратора, USB-хост-контроллера и шины PCI. Эта связь координируется тем, что каждый драйвер в стеке отправляет и получает IRP-сообщения. [19]

Модель драйверов Windows (WDM) — это платформа, разработанная Microsoft, которая позволяет создавать программное обеспечение драйверов, совместимое с операционной системой Windows. Он позволяет оборудованию и устройствам взаимодействовать с операционной системой через стандартизированный интерфейс. Следовательно, он обеспечивает плавное взаимодействие между системой и всеми видами оборудования, а также повышает стабильность и производительность системы. [20]

WDM состоит из трех частей: WDM базовый (WDM basic), WDM расширенный (WDM extended) и WDM потоковый (WDM stream). WDM базовый определяет общие функции для всех драйверов устройств, такие как загрузка и выгрузка драйверов, обработка запросов ввода-вывода (I/O requests), обработка прерываний и т.д. WDM расширенный определяет специфические функции для драйверов устройств Plug and Play (PnP) и

энергосбережения (power management), такие как обнаружение и удаление устройств, переход в разные состояния питания и т.д. WDM потоковый определяет специфические функции для драйверов устройств аудио-видео потоков (audio-video streams), такие как форматирование и обработка потоковых данных, синхронизация потоков и т.д.

Драйверы устройств обеспечивают поддержку разных типов устройств в операционной системе за счет реализации специфических функций для работы с данными на разных уровнях абстракции. Драйверы устройств также обеспечивают гибкость и совместимость операционной системы за счет поддержки механизма Plug and Play (PnP), который позволяет автоматически обнаруживать, устанавливать и управлять устройствами в системе.

2.2 История, версии и достоинства

Windows 1.0. Операционна система Windows изначально создавалась как графический интерфейс для MS DOS. Первая версия была выпущена 20 ноября 1985 года и называлась Windows 1.0. Минимальные системные требования заключались в наличии 2 дискет или жесткого диска, графического адаптера и 256К оперативной памяти. Несмотря на то, что Windows 1.0 не имела такого успеха, как аналогичная система Macintosh компании Apple, но несмотря на это Microsoft осуществляла поддержку аж до 31 декабря 2001 года.

Windows 2.0. В ноябре 1987 года была выпущена новая версия – 2.0, которая включала в себя массу нововведений и улучшений. Для новой операционной системы требовался более мощный процессор Intel 286, благодаря которому была значительно улучшена многозадачность и графика. Появилась возможность передвигать и переключать окна программ, а также была реализована система перекрывания окон. Появились кнопки минимизации и максимизации окон. Появилась поддержка комбинаций клавиш, при помощи которых пользователи могли осуществлять системные операции. К тому же, программы получили возможность обмениваться данными друг с другом при помощи системы “Dynamic Data Exchange”, разработанной Microsoft. Когда появился процессор Intel 386, Windows 2.0 был обновлен, чтобы предоставлять преимущества памяти для различных программ.

Windows 3.0. 22 мая 1990 года выходит версия 3.0 чья популярность стремительно растет. Она получила новые цветные иконки и значительно улучшенный интерфейс. Так же Microsoft полностью изменила среду

разработки приложений. Именно благодаря новому программному обеспечению “Software Development Kit”, разработчики устремили свое внимание к Windows. Ведь теперь они могли полностью сосредоточится на создании приложений, а не заниматься написанием драйверов для устройств.

Еще одним нововведением версии 3.0 был пакет программ Microsoft Office. На тот момент он состоял из MS Word, MS Excel и PowerPoint. И именно в этой версии впервые появился знаменитый пасьянс “Косынка”.

Windows NT 3.1. 27 июля 1993 года была представлена Windows NT 3.1, которая была уже 32-разрядной операционной системой. Данная версия была специально предназначена для сетей и бизнес приложений. Это была первая серверная Windows, которая так же могла использоваться на рабочих станциях. Была включена поддержка сетевых протоколов TCP/IP, NetBIOS Frames и DLC.

Эта система уже использовала файловую систему NTFS, когда предыдущие версии были на FAT.

Windows 95. 24 августа 1995 года мир увидел Windows 95. Теперь windows – это отдельная операционная система и уже не является графическим интерфейсом для MS-DOS, но он все еще остается важным компонентом. Это был успех. За первые 4 дня, после выпуска, было продано 1 млн. копий. В течении года цифра достигла отметки 7 млн. Доля рынка основного конкурента, компании Apple, начала значительно снижаться.

В Windows 95 появились такие привычные элементы, как значки на рабочем столе, панель задач с кнопкой “Пуск”. Была реализована функция Plug&Play, которая автоматически устанавливала драйвера во время подключения устройства. Она была простой и понятной, что делало её идеальной системой для домашнего использования.

Пользователи, установившие новую ОС от Microsoft впервые увидели программу, которая открыла им путь в новый мир, мир Интернет. Программу Internet Explorer.

Windows 98. По сути являлась обновленной Windows 95 и была выпущена 25 июня 1998 года. В ней были исправлены недочеты предыдущей версии, доработаны драйвера USB и AGP, появилась поддержка работы с несколькими мониторами, улучшен Internet Explorer. Добавлена возможность поиска информации в интернете и на компьютере. Что касается мультимедиа, появилась поддержка DVD дисков. В обновленной ОС появился Updater, позволяющий самостоятельно устанавливать свежие обновления.

Windows 2000. Дата выхода – 17 февраля 2000 года. Предназначена для бизнеса и выпускалась в 4-х изданиях: Professional (издание для рабочих

станций и опытных пользователей), Server, Advanced Server и Datacenter Server (для применения на серверах). В новую версию включена поддержка беспроводных сетевых устройств, инфракрасных устройств, различных игровых устройств, FireWire(IEEE 1394) и многое другое. Microsoft утверждала, что это самая безопасная операционная система, хотя на деле оказалось, что это далеко не так.

Windows ME (Melennium Edition). Операционная система, ориентированная на домашних пользователей. Является последней системой, основанной на коде Windows 95. В ней появилась функция восстановления системы, благодаря которой можно создавать резервные копии и в случае серьезных сбоев можно было легко вернуть работоспособность компьютера. Она была первой системой, которая включала в себя инструменты для обработки видео: Windows Media Player и Windows Movie Maker. По своей сути ME – это аналог Win.2000, только для другого круга пользователей.

Windows XP. 25 октября 2001 года Windows XP быстро пришла на замену Win.2000 и Win.ME. Она была ориентирована сразу и для бизнеса, и для установки на домашнем ПК. С 2003 до 2011 год была самой распространенной ОС в мире. В 2007 году доля рынка достигла максимума и составляла 76.1%. В отличии от предыдущих систем, XP была исключительно клиентской. Серверным аналогом была Windows Server 2003. Среди нововведений были: более округлый графический интерфейс, поддержка метода сглаживания текста ClearType, которая применялась для улучшения отображения на ЖК-дисплеях, возможность работы нескольких пользователей, улучшены функции управления системой при помощи командной строки, улучшение совместимости приложений со старыми версиями Windows. В XP появилась возможность записи дисков прямо из проводника, не устанавливая дополнительного ПО.

Windows Vista. Считается самым неудачным обновлением семейства Windows. Была представлена 30 января 2007 года. Основные жалобы пользователей были направлены на излишнюю секретность системы и низкую производительность. Из-за этого большинство компаний предпочли остаться на проверенной, надежной и производительной Windows XP. Несмотря на это, компании Microsoft все же удалось продать более 100 млн. лицензий Windows Vista. Среди нововведений переработанный интерфейс, усовершенствованную работу с сетью, аудио и печатью, гаджеты рабочего стола.

Windows 7. Самая распространенная ОС в мире с 2011 года. Дата выпуска 22 октября 2009 года. Всего в первый год было продано 240

миллионов лицензий, что сделало её самой продаваемой операционной системой в истории. На январь 2013 года доля Windows 7 на рынке операционных систем, используемых для доступа в интернет, составила 55.2%. Новая Windows обзавелась поддержкой мультитач управления, большое количество тем и визуальных эффектов, тесная интеграция с производителями драйверов. Более 90% устройств устанавливаются автоматически. Улучшена поддержка приложений, разработанных для старых версий Windows. Windows Media Player 12, вошедший в “семерку” стал самостоятельно читать практически все форматы в то время, как раньше было необходимо устанавливать большое количество дополнительных кодеков. Добавлены функции интерфейса Aero, такие как Shake, Peek и Snap. Они позволяют эффективно управлять окнами при помощи мыши и сочетаний клавиш.

Windows 8. 26 октября 2012 года вышла Windows 8. Использует новый пользовательский интерфейс Modern. Он имеет плиточную структуру и функционально схож с привычным рабочим столом. Количество плиток на экране определяется автоматически в зависимости от разрешения. В этой версии отсутствует кнопка “Пуск”, а вместо неё используется активный угол. В этой версии появилась возможность синхронизации параметров нескольких устройств при помощи учетной записи Microsoft. Появился магазин приложений. Обновлен Internet Explorer, новый Диспетчер задач, добавлена поддержка USB 3.0, Bluetooth 4.0 DirectX 11.1 и NET.Framework 4.5. Особенностью Windows 8 является ориентирование как на ПК, так и на устройства с сенсорным управлением.

2.3 Обоснование выбора платформы

Выбор операционной системы Windows в качестве платформы для разработки программы создания и запуска макросов обоснован следующими ключевыми факторами.

Распространенность: Windows является одной из наиболее широко используемых операционных систем в мире, особенно в корпоративной среде и среди пользователей ПК. Это значит, что созданная в рамках данной курсовой работы программа будет иметь большое прикладное значение для широкого круга пользователей.

Удобство разработки: Microsoft предоставляет мощные инструменты для разработки под Windows, такие как Visual Studio, .NET Framework и последующие платформы .NET Core. При помощи данных инструментов

возможно создать программу для записи и воспроизведения макросов, с наилучшим сочетанием показателей затраченных сил и качества программного продукта.

API и документация: Windows имеет обширный набор API (программных интерфейсов приложений), который позволяет взаимодействовать с операционной системой и её компонентами. Также существует обширная документация что положительно скажется на процессе разработки.

2.4 Анализ операционной системы и программного обеспечения для написания программы

Для создания программы позволяющей записывать и воспроизводить макросы была выбрана операционная система windows. Причины, лежащие у основания данного выбора озвучены в подпункте выше. Используемый язык программирования – C#. Он был выбран в силу свой совместимости с windows, простоты, наличия готовых библиотек для получения событий клавиатуры и мыши.

Для считывания нажатий на клавиши будет использована библиотека UserActivityMonitor для языка программирования C#. Она использует Windows Hooks для перехвата событий клавиатуры и мыши. Была выбрана именно эта библиотека потому что она предоставляет наиболее удобный доступ для работы с используемыми функциями win32 api.

Записанные макросы необходимо как-то воспроизводить. Для этого будут использованы функции такие функции win32 api как keybd_event и mouse_event.

Графический интерфейс пользователя будет создан при помощи WinForms.

В данной части курсовой работы была рассмотрена используемая платформа разработки, ее история. Был произведен анализ программного обеспечения и операционной системы, для создания приложения, а также был обоснован выбор платформы для разработки

3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

В данной части курсовой работы будет обоснована необходимость разработки данного приложения, а также будет проведен краткий экскурс в используемые технологии.

3.1 Обоснование необходимости разработки

В современное время при выполнении различных задач на компьютере люди сталкиваются с необходимостью выполнять повторяющиеся действия. Для экономии времени их можно записывать и воспроизводить по требованию. В этом и заключается основная суть макросов.

Будет произведено создание программы для записи макросов клавиатуры и нажатий кнопок мыши, потому что именно с этими повторяющимися действиями сталкивается большинство людей.

В целях обеспечения удобства использования данного программного продукта будет создан пользовательский интерфейс, предоставляющий возможность удобно осуществлять запись и управление макросами. Вызов же макросов будет происходить по нажатии определенных сочетаний клавиш.

3.2 Технологии программирования, используемые для решения поставленных задач

Для решение поставленной задачи будет использована платформа .NET и фреймворк WinForms. Ниже будет рассмотрены конкретные детали данных технологий.

3.2.1 Язык программирования и платформа

C# – современный объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в .NET

Вот лишь несколько функций языка C#, которые позволяют создавать надежные и устойчивые приложения. Сборка мусора автоматически освобождает память, занятую недостижимыми неиспользуемыми объектами. Типы, допускающие значение null, обеспечивают защиту от переменных, которые не ссылаются на выделенные объекты. Обработка исключений предоставляет структурированный и расширяемый подход к обнаружению

ошибок и восстановлению после них. Лямбда-выражения поддерживают приемы функционального программирования. Синтаксис LINQ создает общий шаблон для работы с данными из любого источника. Поддержка языков для асинхронных операций предоставляет синтаксис для создания распределенных систем.

.NET – это бесплатная кроссплатформенная платформа разработчика с открытым исходным кодом для создания различных типов приложений. Платформа .NET создана на основе высокопроизводительной среды выполнения, которая используется в рабочей среде многими высокомасштабными приложениями.

Можно выделить следующие ее основные черты:

1 Поддержка нескольких языков. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), благодаря чему .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. При компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language) – своего рода ассемблер платформы .NET. Поэтому при определенных условиях можно сделать отдельные модули одного приложения на отдельных языках.

2 Кроссплатформенность. .NET является переносимой платформой (с некоторыми ограничениями). Например, последняя версия платформы на данный момент – .NET 7 поддерживается на большинстве современных операционных системах Windows, MacOS, Linux. Используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ - Windows, MacOS, Linux, Android, iOS, Tizen.

3 Мощная библиотека классов. .NET представляет единую для всех поддерживаемых языков библиотеку классов. И какое бы приложение не было написано на C# – текстовый редактор, чат или сложный веб-сайт – так или иначе задействуется библиотеку классов .NET.

4 Разнообразие технологий. Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология ADO.NET и Entity Framework Core. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF и WinUI, для создания более простых графических приложений – Windows Forms. Для разработки кроссплатформенных мобильных и

десктопных приложений – Xamarin/MAUI. Для создания веб-сайтов и веб-приложений – ASP.NET и т.д.

5 Производительность. Согласно ряду тестов веб-приложения на .NET 7 в ряде категорий сильно опережают веб-приложения, построенные с помощью других технологий. Приложения на .NET 7 в принципе отличаются высокой производительностью.

Стоит отметить, что .NET долгое время развивался преимущественно как платформа для Windows под названием .NET Framework. В 2019 вышла последняя версия этой платформы – .NET Framework 4.8. Она больше не развивается.

С 2014 Microsoft стал развивать альтернативную платформу – .NET Core, которая уже предназначалась для разных платформ и должна была вобрать в себя все возможности устаревшего .NET Framework и добавить новую функциональность. Затем Microsoft последовательно выпустил ряд версий этой платформы: .NET Core 1, .NET Core 2, .NET Core 3, .NET 5. И текущей версией является платформа .NET 7.

3.2.2 Технология WinForms

Windows Forms – интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код – классы, реализующие API для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, C++, так и на VB.Net, J# и др.

С одной стороны, Windows Forms рассматривается как замена более старой и сложной библиотеке MFC, изначально написанной на языке C++. С другой стороны, WF не предлагает парадигмы, сравнимой с MVC. Для исправления этой ситуации и реализации данной функциональности в WF существуют сторонние библиотеки. Одной из наиболее используемых подобных библиотек является User Interface Process Application Block, выпущенная специальной группой Microsoft, занимающейся примерами и рекомендациями, для бесплатного скачивания. Эта библиотека также содержит исходный код и обучающие примеры для ускорения обучения.

Внутри .NET Framework Windows Forms реализуется в рамках пространства имён System.Windows.Forms.

Можно выделить следующие основные преимущества Windows Forms:

1 Богатый набор контролов и легкая кастомизация. WinForms включает в себя все стандартные элементы управления, которые могут потребоваться для разработки настольных приложений. Элементы управления могут быть легко настроены и расширены для соответствия специфическим требованиям.

2 Совместимость с .NET Framework. WinForms может использоваться со многими версиями .NET Framework, что обеспечивает совместимость с большим количеством кода и библиотек.

3 Событийно-ориентированный подход. Разработка интерфейса основана на обработке событий, что делает логику приложения понятной и организованной.

4 Производительность. Для стандартных приложений WinForms обеспечивает достаточную производительность без значительных затрат ресурсов системы.

5 Глубокая интеграция с Windows. Программы на WinForms тесно интегрированы с Windows, что позволяет легко использовать различные возможности операционной системы.

6 Поддержка Visual Studio. WinForms полностью интегрирован с Visual Studio, предоставляя разработчикам мощные инструменты для проектирования, разработки и отладки приложений.

3.2.3 Библиотека UserActivityMonitor

UserActivityMonitor – это библиотека, предназначенная для отслеживания действий пользователя в операционной системе Windows, таких как нажатия клавиш и движения мыши. В качестве ее основных преимуществ можно выделить:

1 Простота интеграции. Библиотека может быть легко интегрирована в любое .NET приложение, предоставляя простой интерфейс для мониторинга действий пользователя.

2 Событийно-ориентированный подход. UserActivityMonitor работает на основе событий. Это означает, что можно подписаться на события, такие как нажатие клавиши или движение мыши, и выполнять определенные действия в ответ на эти события.

3 Низкоуровневый доступ. Данная библиотека предоставляет низкоуровневый доступ к системным событиям, что позволяет отслеживать активность пользователя более детально, чем это возможно с использованием стандартных средств .NET.

4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

В рамках данной курсовой работы разрабатывается программа для создания и воспроизведения макросов. В данной части будут описаны функции данной программы.

4.1 Обоснование и описание функций программного обеспечения

Есть 2 основные функции, которые должна предоставлять данная программа – создание макроса и его воспроизведение. Рассмотрим их подробнее.

4.1.1 Создание макроса

В своей сущности, макрос – это некая последовательность инструкций, которые будут выполняться при определённых условиях. По своей функциональности макросы бывают разные, выделим те макросы, которые наиболее полезны при работе с компьютером:

1 Макрос клавиатуры и мыши.

2 Макрос замены текста.

При определении основных инструкций, которые будут использованы в макросах важно сконцентрироваться на тех, которые подходят под цели данной курсовой работы. В соответствии с этим выдели следующе основные команды:

1 Нажатие на клавишу – просто нажатие клавиши на клавиатуре.

2 Зажатие клавиши – клавишу на клавиатуре нажали и не отпустили.

3 Отпускание клавиши – зажатую клавишу отпустили.

4 Нажатие на кнопки – просто нажатие кнопки на мыши.

5 Зажатие кнопки – кнопки на мыши нажали и не отпустили.

6 Отпускание кнопки – зажатую кнопки отпустили.

7 Ожидание – ожидать определенное количество времени перед следующей инструкцией.

8 Запустить программу.

9 Закрыть программу.

10 Экстренно завершить программу.

В качестве инструкций макроса были выбраны именно эти команды, потому что они представляют собой наиболее частые команды, которые могут быть использованы в автоматизации рутинных действий.

Надо заметить, что инструкции для работы с мышью должны принимать координаты, на которых будет произведено определенное действие

4.1.2 Воспроизведение макроса

Макросы могут быть воспроизведены в различных условиях:

1 Нажатие клавиши активации и клавиш или кнопки, вызывающей макрос.

2 Нажатие клавиш или кнопки, вызывающей макрос.

3 Открытие окна или приложения, удовлетворяющего определённым критериям.

4 Запуск данной программы.

5 Бездействие компьютера в течении определенного промежутка времени.

В качестве условий воспроизведения макросы были выбраны условия, представленные выше, потому что они позволяют гибко оптимизировать большинство рутинных задач, встречающихся при работе с компьютером.

4.1.3 Пользовательский интерфейс

Пользовательский интерфейс должен предоставлять следующие возможности:

1 Создание макросов.

2 Ввод команд для макросов.

3 Удаление макросов.

4 Изменение макросов.

5 Активацию и деактивацию макросов.

Были выбраны именно такие функции поскольку они являются основными при работе с макросами. Также, в целях оптимизации пользовательского интерфейса и удобства работы с ним, при запуске приложения, программа будет помещаться на панели задач.

В данной части курсовой работы было произведено проектирование и обоснование функциональных возможностей программы. Это было сделано исходя из цели данной курсовой работы, а также исходя из проблем, с которыми люди сталкиваются при работе с компьютерными программами.

5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

В данном подпункте будет описана архитектура разрабатываемой программы, а также алгоритм работы замены макросов.

5.1 Структура и архитектура разрабатываемого приложения

Все приложение разбито на классы, каждый из которых отвечает за свои функции, такие как: хранение целонго макроса, хранение инструкций макроса, отрисовка пользовательского интерфейса и другие.

5.1.1 Класс Macro

Основным классом программы является класс Macro. Он представляет собой один конкретный макрос. Он имеет следующие поля

– triggerKeys – данное поле представляет собой список клавиш, которые вызывают срабатывание макроса. В случае использования только одной клавиши для активации макроса это поле не инициализируется;

– type – данное поле отвечает за тип макроса. Возможные типы макросов определены в подпункте 4 данной курсовой работы;

– idletime – данное поле хранит в себе время, которое компьютер должен простаивать, чтобы макрос начала свое выполнение. Если данная функция не используется, это поле инициализируется 0;

– triggerKey – данное поле хранит в себе клавишу, которая вызывает активацию макроса. В случае использование более чем одной клавиши – данное поле не инициализируется;

– targetapp – данное поле отвечает за приложения, которое вызовет срабатывание макроса. Если данная функциональность не используется, это поле не инициализируется;

– override – данное поле отвечает за вызов макроса в случае ввода определенной последовательности символов. Если данная функциональность не используется, это поле не инициализируется;

– script – данное поле содержит в себе список команд, которые будут выполнены при срабатывании макроса.

5.1.2 Класс MacroAction

За команды макроса отвечает абстрактный класс Macro Action и классы, наследуемые от него. Он содержит в себе метод Do, вызов которого для класса потомка и осуществит выполнение конкретной команды макроса.

5.1.3 Класс MacroFiles

За загрузку макросов из файла отвечает класс MacroFiles. Он загружает все файлы из директории Macros с расширением .macro. Результатом его работы является класс MacroSet.

5.1.4 Класс MacroSet

Данный класс представляет собой список макросов. Он необходим для хранения макросов или получения их по таким критериям как: тип макроса и клавиша его активирующая

5.1.5 MacroType

MacroType представляет собой перечисление. Оно описывает возможные типы макросов. Тип макроса обозначает при каких условиях вызывается тот или иной макрос.

5.1.6 Класс FormMain

Данный класс содержит всю логику, отвечающую за считывание пользовательского ввода, отслеживание условий запуска макросов и запуск макросов.

В конструкторе устанавливаются класса, позволяющие отслеживать действия пользователя, обработчики событий на действия пользователя, устанавливается таймер с тиком равным одной секунде. Рассмотрим обработчики событий немного подробнее. Полное описание алгоритма работы программы будет дано в пункте 5.2

Обработчик открытия окна вызывает функцию обработки макросов для макросов открытия окна.

Обработчик взаимодействия с мышью сбрасывает время неактивности пользователя.

Обработчик тика таймера увеличивает переменную, в которой хранится время неактивности пользователя, и проверяет должен ли быть вызван какой-либо макрос.

Обработчик нажатия на мышь переводит нажатие кнопки в понимаемое программой значение и проверяет, должен ли быть вызван макрос.

Обработчик зажатия клавиши клавиатуры переводит действие в понимаемое программой значение, управляет нажатием специальных комбинаций и обрабатывает макрос замены текста

Обработчик отпущенной клавиши клавиатуры сбрасывает таймер ожидания, если необходимо, изменяет ключ активации макроса, обновляет

состояние клавиши Shift, удаляет отпущенную клавишу из списка нажатых клавиш и проверяет, следует ли выполнять какие-либо макросы на основе отпущенной клавиши.

5.2 Подробное описание алгоритма работы приложения

В данном подпункте курсовой работы пойдет речь про алгоритм работы приложения. Будет рассмотрен класс FormMain, поскольку в нем находится вся основная логика работы программы и логика отрисовки пользовательского интерфейса.

5.2.1 Конструктор

В конструкторе класса FormMain производятся начальные настройки программы. Сперва отрисовывается окно пользовательского интерфейса. Далее устанавливаются переменные, отвечающие за ввод с клавиатуры и мыши. Впоследствии, устанавливаются обработчики событий клавиатуры и мыши. Потом загружаются макросы, обрабатывается макрос запуска приложения, запускается таймер с тиком в одну секунду и устанавливается обработчик тика таймера. На этом выполнение конструктора заканчивается.

5.2.2 Обработчики событий

Обработчик открытия окна. Данный обработчик вызывает метод обработки макросов, передав в качестве параметра в данный метод тип макроса – открытие окна.

Обработчик нажатия клавиши на клавиатуре. Сперва данный обработчик проверяет является ли нажатая клавиша, клавишей активации макроса. Данная проверка необходима, поскольку некоторые макросы активируются только при нажатии специальной клавиши. Далее проверяется является ли нажатая клавиша одной из клавиш Shift. Если да, то устанавливается соответствующее поле. Далее нажатая клавиша добавляется в список нажатых клавиш, при том, проверяется, чтобы больше таких же нажатых клавиш не было. Далее вызывается метод обработки макросов замены текста, и обработчик события завершает свою работу.

Обработчик отпускания клавиши на клавиатуре. Сперва данный метод проверяет, происходит ли изменение клавиши активации макросов. Если да, то он записывает и сохраняет изменения. Далее происходит проверка на отпускание клавиши Shift. Если это так, то поле, установленное в обработчике выше, сбрасывается. Потом, если отпущенная клавиша – это клавиша

активации, сбрасывается соответствующее поле. Впоследствии, отпущенная клавиша убирается из списка нажатых клавиш и происходит вызов метода обработки макросов с соответствующими параметрами. Здесь обработчик события завершает свою работу.

5.2.3 Методы работы с макросами

Обработка макроса замены текста. Сперва извлекаются все макросы для замены текста. Далее в цикле перебираются все макросы замены текста. В цикле, первым действием, проверяется, что макрос замены текста содержит в себе последнюю нажатую клавишу. Если нет – обработка завершается. Потом идет проверка что все нажатые клавиши – это клавиши, вызывающие макрос, а все клавиши, вызывающие макрос – это нажатые клавиши. Если все проверки пройдены, наступает последняя проверка – проверка целевого приложения. Если для макроса нет целевого приложения, то макрос запускается сразу. Если приложение есть, то проверяется соответствие активного окна целевому приложению. Если это соответствие удовлетворено, то макрос запускается. На этом метод заканчивает свою работу.

Обработка других макросов. Сперва в методе получаются макросы переданного типа и активируемые переданной кнопкой или клавишей. Далее происходит итерация по полученным макросам. Если нажата клавиша активации макроса, или макрос не требует нажатия клавиши активации, то метод проверяет есть ли целевая программа или окно. Если нет – запускает макрос сразу, если есть – проверяет активно ли необходимое окно, и, по результатам этой проверки, запускает необходимый макрос.

5.2.4 Запуск макросов

Для запуска макросов используется абстрактный класс MacroAction и его метод Do(). Классы реальных действий, по типу нажатия на клавишу клавиатуры или кнопку мыши, переопределяют данный метод. Детали его реализации зависят от выполняемого действия. Например, для отправки нажатий на клавиатуру или кнопку мыши используется соответствующий метод класса KeySender. Он вызывает соответствующие методы библиотеки user32.dll. Для завершения процесса, используется класс Process из библиотеки System.Diagnostics.

Подводя итог можно сказать, что в этой части курсовой работы был подробно описан алгоритм работы приложения. Были разобраны отдельные классы и методы выполняющие необходимые действия для работы программы.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была успешно разработана и реализована программа для записи и воспроизведения макросов на операционной системе Windows. Данная программа помогает в автоматизации таких рутинных действий как: набор одного и того же текста, осуществление замены текста, имитации повторяющихся действий мышью.

Была выполнена основная цель данной работы – создание программы для записи и воспроизведения макросов. Был произведен анализ типов макросов, возможностей операционной системы windows по эмуляции ввода с клавиатуры и способов взаимодействия платформы .NET с ними.

Для достижения данной цели были выделены следующие задачи: изучение теории по созданию макросов, запись и имитация ввода клавиатуры и мыши средствами языка C#, создание программы с пользовательским интерфейсом, позволяющей считывать и воспроизводить ввод клавиатуры и мыши.

Для решения первой задачи была изучена литература, указанная в списке литературных источников.

Для решения второй задачи был использован механизм Windows Hooks, который позволяет перехватывать события мыши и клавиатуры. Для использования данного механизма в языке C# была выбрана библиотека UserActivityMonitor. Для имитации нажатий на клавиатуру и действий мышью были использованы функции keybd_event и mouse_event.

Для решения третьей задачи была использована технология WinForms.

Полученная программа позволяет осуществлять ввод определенных макросов, их сохранение и запуск. Таким образом, люди, сталкивающиеся с большим количеством рутинных задач, смогут освободить больше времени на выполнение действительно важных задач.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Macro [Электронный ресурс]. – Режим доступа: https://www.computerhope.com/jargon/m/macro.htm – Дата доступа: 24.03.2024

[2] Syntax-tree [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/compiler-design-variants-of-syntax-tree/ – Дата доступа: 25.04.2024

[3] Prokey [Электронный ресурс]. – Режим доступа: https://www.pcmag.com/encyclopedia/term/prokey – Дата доступа: 25.04.2024

[4] TSR [Электронный ресурс]. – Режим доступа: https://www.capterra.com/glossary/terminate-and-stay-resident-tsr/ – Дата доступа: 25.04.2024

[5] Lisp [Электронный ресурс]. – Режим доступа: https://gb.ru/blog/lisp/ – Дата доступа: 25.04.2024

[6] Vim [Электронный ресурс]. – Режим доступа: https://www.vim.org – Дата доступа: 25.04.2024

[7] VBA [Электронный ресурс]. – Режим доступа: https://www.investopedia.com/terms/v/visual-basic-for-applications-vba.asp#:~:text=Visual%20Basic%20for%20Applications%20is,not%20a%20stand-alone%20product – Дата доступа: 25.04.2024

[8] Tex [Электронный ресурс]. – Режим доступа: https://www.britannica.com/technology/TeX – Дата доступа: 25.04.2024

[9] PL/I [Электронный ресурс]. – Режим доступа: https://pl1.su – Дата доступа: 25.04.2024

[10] Frame Technology [Электронный ресурс]. – Режим доступа: https://www.researchgate.net/publication/2876781_A_Review_of_Frame_Technology – Дата доступа: 25.04.2024

[11] Anaphoric Macros [Электронный ресурс]. – Режим доступа: https://letoverlambda.com/index.cl/guest/chap6.html – Дата доступа: 25.04.2024

[12] Как устроена Windows: разбираемся в основных компонентах ее архитектуры [Электронный ресурс]. – Режим доступа: https://fileenergy.com/windows/kak-ustroena-windows-razbiraemsya-v-osnovnykh-komponentakh-ee-arkhitektury – Дата доступа: 25.04.2024

[13] HAL [Электронный ресурс]. – Режим доступа: https://www.lenovo.com/us/en/glossary/hardware-abstraction-layer/?orgRef=https%253A%252F%252Fwww.google.com%252F – Дата доступа: 25.04.2024

[14] Общие сведения о процедурах прерывания обслуживания [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows-hardware/drivers/kernel/introduction-to-interrupt-service-routines – Дата доступа: 25.04.2024

[15] Multithreading [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/cpp/parallel/multithreading-with-c-and-win32?view=msvc-170 – Дата доступа: 25.04.2024

[16] Win32 [Электронный ресурс]. – Режим доступа: https://linchakin.com/словарь/w/win32/ – Дата доступа: 25.04.2024

[17] WinRT [Электронный ресурс]. – Режим доступа: https://pvs-studio.ru/ru/blog/terms/0092/– Дата доступа: 25.04.2024

[18] Driver [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/what-is-a-driver-– Дата доступа: 25.04.2024

[19] Windows Kernel-Mode I/O Manager [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/windows-kernel-mode-i-o-manager– Дата доступа: 25.04.2024

[20] Windows Driver Model [Электронный ресурс]. – Режим доступа: https://www.devx.com/terms/windows-driver-model/ – Дата доступа: 25.04.2024

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

ActionClose.cs

```
using System;
using System.Diagnostics;
using EasyMacros.Macros;

namespace EasyMacros.Actions
{
    public class ActionClose : MacroAction
    {
        public string toclose;
        public ActionClose(string windowtoclose)
        {
            toclose = windowtoclose;
        }
        public override void Do()
        {
            Process[] procs = Process.GetProcesses();
            for (int i = 0; i < procs.Length; i++)
            {
                if (procs[i].ProcessName.ToLower() + ".exe" ==
toclose.ToLower())
                {
                    procs[i].CloseMainWindow();
                }
                else if
(procs[i].ProcessName.ToLower().Contains(toclose.ToLower()))
                {
                    procs[i].CloseMainWindow();
                }
                else if (procs[i].MainWindowHandle != IntPtr.Zero &&
procs[i].MainWindowTitle.ToLower().Contains(toclose.ToLower()))
                {
                    procs[i].CloseMainWindow();
                }
            }
        }
    }
}
```

ActionKeyPress.cs

```
using EasyMacros.Macros;
using EasyMacros.Utilities;

namespace EasyMacros.Actions
{
    public class ActionKeyPress : MacroAction
    {
        public byte Key;

        public ActionKeyPress(byte K)
        {
```

```
            Key = K;
        }

        public override void Do()
        {
            KeySender.KeyPress(Key);
        }
    }
}
```

ActionKeyRelease.cs

```
using EasyMacros.Macros;
using EasyMacros.Utilities;

namespace EasyMacros.Actions
{
    public class ActionKeyRelease : MacroAction
    {
        public byte Key;

        public ActionKeyRelease(byte K)
        {
            Key = K;
        }

        public override void Do()
        {
            KeySender.KeyRelease(Key);
        }
    }
}
```

ActionKeyStroke.cs

```
using EasyMacros.Macros;
using EasyMacros.Utilities;

namespace EasyMacros.Actions
{
    public class ActionKeyStroke : MacroAction
    {
        public byte Key;

        public ActionKeyStroke(byte K)
        {
            Key = K;
        }

        public override void Do()
        {
            KeySender.KeyStroke(Key);
        }
    }
}
```

ActionKill.cs

```
using System.Diagnostics;
using EasyMacros.Macros;

namespace EasyMacros.Actions
{
    public class ActionKill : MacroAction
    {
        public string process;

        public ActionKill(string name)
        {
            process = name;
        }

        public override void Do()
        {
            ProcessStartInfo P = new ProcessStartInfo("taskkill", "/f /im " +
process);
            P.WindowStyle = ProcessWindowStyle.Hidden;
            Process.Start(P).WaitForExit();
        }
    }
}
```

ActionMouseClick.cs

```
using System.Drawing;
using EasyMacros.Utilities;
using EasyMacros.Macros;

namespace EasyMacros.Actions
{
    public class ActionMouseClick : MacroAction
    {
        public KeySender.MouseButton Button;
        public Point ClickPos = Point.Empty;
        public bool CustomPos = false;

        public ActionMouseClick(KeySender.MouseButton B, Point Coordinates)
        {
            Button = B;
            CustomPos = true;
            ClickPos = Coordinates;
        }

        public ActionMouseClick(KeySender.MouseButton B)
        {
            Button = B;
            CustomPos = false;
        }

        public override void Do()
        {
            if (CustomPos)
            {
                KeySender.MouseClick(Button, ClickPos);
            }
            else KeySender.MouseClick(Button);
```

```
        }
    }
}
```

ActionMousePress.cs

```
using System.Drawing;
using EasyMacros.Utilities;
using EasyMacros.Macros;

namespace EasyMacros.Actions
{
    public class ActionMousePress : MacroAction
    {
        public KeySender.MouseButton Button;
        public Point ClickPos = Point.Empty;
        public bool CustomPos = false;

        public ActionMousePress(KeySender.MouseButton B, Point Coordinates)
        {
            Button = B;
            CustomPos = true;
            ClickPos = Coordinates;
        }

        public ActionMousePress(KeySender.MouseButton B)
        {
            Button = B;
            CustomPos = false;
        }

        public override void Do()
        {
            if (CustomPos)
            {
                KeySender.MousePress(Button, ClickPos);
            }
            else KeySender.MousePress(Button);
        }
    }
}
```

ActionMouseRelease.cs

```
using System.Drawing;
using EasyMacros.Utilities;
using EasyMacros.Macros;

namespace EasyMacros.Actions
{
    public class ActionMouseRelease : MacroAction
    {
        public KeySender.MouseButton Button;
        public Point ClickPos = Point.Empty;
        public bool CustomPos = false;

        public ActionMouseRelease(KeySender.MouseButton B, Point Coordinates)
        {
```

```
            Button = B;
            CustomPos = true;
            ClickPos = Coordinates;
        }

        public ActionMouseRelease(KeySender.MouseButton B)
        {
            Button = B;
            CustomPos = false;
        }

        public override void Do()
        {
            if (CustomPos)
            {
                KeySender.MouseRelease(Button, ClickPos);
            }
            else KeySender.MouseRelease(Button);
        }
    }
}
```

ActionPaste.cs

```
using System.Windows.Forms;
using EasyMacros.Macros;
using EasyMacros.Utilities;

namespace EasyMacros.Actions
{
    public class ActionPaste : MacroAction
    {
        public string topaste;
        public ActionPaste(string textToPaste)
        {
            topaste = textToPaste;
        }
        public override void Do()
        {
            string previous_clipboard = Clipboard.GetText();
            Clipboard.SetText(topaste);
            System.Threading.Thread.Sleep(50);

KeySender.KeyPress(KeyConverter.Name2Key(KeyConverter.WinFormKey2Name(Keys.LC
ontrolKey)));

KeySender.KeyStroke(KeyConverter.Name2Key(KeyConverter.WinFormKey2Name(Keys.V
)));

KeySender.KeyRelease(KeyConverter.Name2Key(KeyConverter.WinFormKey2Name(Keys.
LControlKey)));
            System.Threading.Thread.Sleep(50);
            try
            {
                Clipboard.SetText(previous_clipboard);
            }
            catch
            {
                try
                {
```

```
                    Clipboard.SetText("");
                }
                catch { }
            }
        }
    }
}
```

ActionRun.cs

```
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using EasyMacros.Macros;

namespace EasyMacros.Actions
{
    public class ActionRun : MacroAction
    {
        [DllImport("shell32.dll", SetLastError = true)]
        private static extern IntPtr
CommandLineToArgvW([MarshalAs(UnmanagedType.LPWStr)] string lpCmdLine, out
int pNumArgs);

        private static string[] CommandLineToArgs(string commandLine)
        {
            int argc;
            var argv = CommandLineToArgvW(commandLine, out argc);
            if (argv == IntPtr.Zero)
                throw new System.ComponentModel.Win32Exception();
            try
            {
                var args = new string[argc];
                for (var i = 0; i < args.Length; i++)
                {
                    var p = Marshal.ReadIntPtr(argv, i * IntPtr.Size);
                    args[i] = Marshal.PtrToStringUni(p);
                }

                return args;
            }
            finally
            {
                Marshal.FreeHGlobal(argv);
            }
        }

        public string command;

        public ActionRun(string command)
        {
            this.command = command;
        }

        public override void Do()
        {
            string[] args = CommandLineToArgs(command);
            string program = args[0];
            string arguments = "";
            for (int i = 1; i < args.Length; i++)
                arguments += '"' + args[i] + '"' + ' ';
```

```
            Process.Start(program, arguments);
        }
    }
}
```

ActionSleep.cs

```
using System.Threading;
using EasyMacros.Macros;

namespace EasyMacros.Actions
{
    public class ActionSleep : MacroAction
    {
        public int duration;
        public ActionSleep(int dur)
        {
            duration = dur;
        }

        public override void Do()
        {
            Thread.Sleep(duration);
        }
    }
}
```

Macro.cs

```
using System.Collections.Generic;
using System.Windows.Forms;

namespace EasyMacros.Macros
{
    /// <summary>
    /// Represents a single macro
    /// </summary>
    public class Macro
    {
        private static object MacroRunLock = new object();
        public List<Keys> TriggerKeys = new List<Keys>();
        public MacroType Type = MacroType.Keyboard;
        public int idletime = 0;
        public Keys TriggerKey;
        public string targetapp = "";
        public bool Override = false;
        public List<MacroAction> script;

        public Macro(Keys triggerkey)
        {
            TriggerKey = triggerkey;
            script = new List<MacroAction>();
        }

        public Macro(Keys triggerkey, List<MacroAction> actions)
        {
            TriggerKey = triggerkey;
            script = actions;
```

```
        }

        public void Add(MacroAction a)
        {
            script.Add(a);
        }

        public void Run()
        {
            lock (MacroRunLock)
            {
                foreach (MacroAction a in script)
                {
                    a.Do();
                }
            }
        }
    }
}
```

MacroAction.cs

```
namespace EasyMacros.Macros
{
    public abstract class MacroAction
    {
        public abstract void Do();
    }
}
```

MacroFiles.cs

```
using System;
using System.IO;
using System.Windows.Forms;
using System.Drawing;
using EasyMacros.Utilities;
using EasyMacros.Actions;

namespace EasyMacros.Macros
{
    /// <summary>
    /// Macro file save/load utility
    /// </summary>
    public static class MacroFiles
    {
        public const string MacrosDir = "Macros";
        public const string MacrosExt = ".macro";
        public const string DisablePrefix = "[OFF] ";

        public static MacroSet Load()
        {
            MacroSet macroSet = new MacroSet();

            if (Directory.Exists(MacrosDir))
            {
                string[] files = Directory.GetFiles(MacrosDir, "*" +
MacrosExt);
```

```
                for (int i = 0; i < files.Length; i++)
                {
                    if (files[i].Length < 13 || files[i].Substring((MacrosDir
+ Path.DirectorySeparatorChar).Length, DisablePrefix.Length) !=
DisablePrefix)
                    {
                        Macro macro =
LoadMacroFile(File.ReadAllLines(files[i]));
                        if (macro != null) { macroSet.Add(macro); }
                    }
                }
            }

            return macroSet;
        }

        public static Macro LoadMacroFile(string[] file)
        {
            if (file.Length > 0)
            {
                string[] args = file[0].Split(' ');

                if (args.Length <= 1) { args = new string[2] { args[0], "ÿ"
}; }
                if (args[0].ToLower().Contains("macro"))
                {
                    Macro macro = new
Macro(KeyConverter.Name2WinFormKey(args[1]));

                    if (args[0].ToLower().Contains("mouse"))
                    {
                        macro.Type = MacroType.Mouse;
                        macro.TriggerKey = KeyConverter.Name2WinFormKey("" +
args[1][0]);
                    }
                    else if (args[0].ToLower().Contains("winopen"))
                    {
                        macro.Type = MacroType.WinOpen;
                    }
                    else if (args[0].ToLower().Contains("startup"))
                    {
                        macro.Type = MacroType.Startup;
                    }
                    else if (args[0].ToLower().Contains("idle"))
                    {
                        macro.Type = MacroType.IDLE;
                        try { macro.idletime = Int32.Parse(args[1]); }
                        catch { }
                    }
                    else if (args[0].ToLower().Contains("rewrite"))
                    {
                        macro.Type = MacroType.RewriteKeyboard;
                        string[] tmp = args[1].Split("+".ToCharArray());
                        foreach (string k in tmp)
                        {
                            Keys result = KeyConverter.Name2WinFormKey(k);
                            if (result != Keys.None)
                            {
                                macro.TriggerKeys.Add(result);
                            }
                        }
                    }
```

```
                    if (args[0].ToLower().Contains("super"))
                    {
                        macro.Override = true;
                    }

                    if (args.Length >= 3)
                    {
                        macro.targetapp = args[2];
                    }

                    for (int i = 1; i < file.Length; i++)
                    {
                        args = file[i].Split(' ');
                        if (args.Length >= 2)
                        {
                            int time = 0;
                            byte code = 0;
                            Point P = Point.Empty;
                            KeySender.MouseButton button;
                            switch (args[0].ToLower())
                            {
                                case "key":
                                    code = KeyConverter.Name2Key(args[1]);
                                    if (code != 0) { macro.Add(new
ActionKeyStroke(code)); }
                                    break;

                                case "keydown":
                                    code = KeyConverter.Name2Key(args[1]);
                                    if (code != 0) { macro.Add(new
ActionKeyPress(code)); }
                                    break;

                                case "keyup":
                                    code = KeyConverter.Name2Key(args[1]);
                                    if (code != 0) { macro.Add(new
ActionKeyRelease(code)); }
                                    break;

                                case "mouse":
                                    button =
KeyConverter.Name2MouseKey(args[1]);
                                    if (args.Length >= 3 &&
ParsePoint(args[2], ref P))
                                    { macro.Add(new ActionMouseClick(button,
P)); }
                                    else macro.Add(new
ActionMouseClick(button));
                                    break;

                                case "mousedown":
                                    button =
KeyConverter.Name2MouseKey(args[1]);
                                    if (args.Length >= 3 &&
ParsePoint(args[2], ref P))
                                    { macro.Add(new ActionMousePress(button,
P)); }
                                    else macro.Add(new
ActionMousePress(button));
                                    break;
```

```
                                    case "mouseup":
                                        button =
KeyConverter.Name2MouseKey(args[1]);
                                        if (args.Length >= 3 &&
ParsePoint(args[2], ref P))
                                        { macro.Add(new
ActionMouseRelease(button, P)); }
                                        else macro.Add(new
ActionMouseRelease(button));
                                        break;

                                    case "sleep":
                                        try { time = Int32.Parse(args[1]); }
                                        catch { }
                                        if (time != 0) { macro.Add(new
ActionSleep(time)); }
                                        break;

                                    case "run":
                                        try { macro.Add(new
ActionRun(file[i].Substring(4))); }
                                        catch { }
                                        break;

                                    case "paste":
                                        try { macro.Add(new
ActionPaste(file[i].Substring(6))); }
                                        catch { }
                                        break;

                                    case "close":
                                        try { macro.Add(new
ActionClose(file[i].Substring(6))); }
                                        catch { }
                                        break;

                                    case "kill":
                                        macro.Add(new ActionKill(args[1]));
                                        break;
                                }
                            }
                        }

                        return macro;
                    }
                    else return null;
                }
                else return null;
            }

            private static bool ParsePoint(string toparse, ref Point P)
            {
                int posX; int posY;
                string[] coords = toparse.Split(',');
                if (coords.Length == 2 && Int32.TryParse(coords[0], out posX) &&
Int32.TryParse(coords[1], out posY))
                {
                    P.X = posX;
                    P.Y = posY;
                    return true;
                }
                else return false;
```

```
        }
    }
}
```

MacroSet.cs

```
using System.Collections.Generic;
using System.Windows.Forms;

namespace EasyMacros.Macros
{
    /// <summary>
    /// Represents a set of macros
    /// </summary>
    public class MacroSet
    {
        private List<Macro> MacroList = new List<Macro>();

        public List<Macro> getMacros(Keys key, MacroType type)
        {
            List<Macro> Result = new List<Macro>();
            foreach (Macro macro in MacroList)
            {
                if (macro.Type == MacroType.Keyboard || macro.Type ==
MacroType.Mouse)
                {
                    if (macro.Type == type && macro.TriggerKey == key)
                    {
                        Result.Add(macro);
                    }
                }
                else
                {
                    if (macro.Type == type)
                    {
                        Result.Add(macro);
                    }
                }
            }
            return Result;
        }

        public void Add(Macro m)
        {
            lock (MacroList)
            {
                MacroList.Add(m);
            }
        }
    }
}
```

MacroType.cs

```
namespace EasyMacros.Macros
{
    /// <summary>
    /// Describes a macro type
```

```
    /// </summary>
    public enum MacroType
    {
        /// <summary>
        /// Keyboard macro. Launched by pressing keyboard keys.
        /// </summary>
        Keyboard,

        /// <summary>
        /// Mouse macro. Launched by pressing mouse buttons.
        /// </summary>
        Mouse,

        /// <summary>
        /// Window Open macro. Launched when a window opens.
        /// </summary>
        WinOpen,

        /// <summary>
        /// Startup macro. Launched when this app is launched.
        /// </summary>
        Startup,

        /// <summary>
        /// IDLE macro. Launched after no mouse/keyboard input for a certain
amount of time.
        /// </summary>
        IDLE,

        /// <summary>
        /// Rewrite Keyboard macro. Launched by pressing keyboard keys, and
suppress triggering keypresses.
        /// </summary>
        RewriteKeyboard
    };
}
```

KeyHelper.cs

```
using System;
using System.Windows.Forms;

namespace EasyMacros
{
    public partial class KeyHelper : Form
    {
        public static string BoxContent = "";
        public static bool Form_Visible = false;
        public FormMain handler;
        Timer timer = new Timer();

        public KeyHelper()
        {
            InitializeComponent();

            timer.Interval = 150;
            timer.Tick += new EventHandler(timer_Tick);
            timer.Start();

            BoxContent = "";
```

```
            Form_Visible = true;
            Btn_CreateMacro.Enabled = false;
        }

        private void Btn_Vider_Click(object sender, EventArgs e)
        {
            BoxContent = "";
            Btn_CreateMacro.Enabled = false;
        }

        private void timer_Tick(object sender, EventArgs e)
        {
            Box_Scratchpad.Text = BoxContent;
            if (BoxContent != "")
            {
                Btn_CreateMacro.Enabled = true;
            }
        }

        private void Btn_Copier_Click(object sender, EventArgs e)
        {
            System.Windows.Clipboard.SetData(DataFormats.Text, BoxContent);
        }

        private void Btn_OK_Click(object sender, EventArgs e)
        {
            Form_Visible = false;
            Close();
        }

        private void Btn_CreateMacro_Click(object sender, EventArgs e)
        {
            if (handler.Nouvelle_Macro())
            {
                string shortcut = BoxContent.Replace("\n", "+").Trim('+');
                handler.SetMacroContent("RewriteMacro " + shortcut + "\n");
                Btn_OK_Click(sender, e);
            }
        }
    }
}

}
```

ПРИЛОЖЕНИЕ Б
(обязательное)
UML диаграмма исходного кода программного средства

ПРИЛОЖЕНИЕ В

(обязательное)

Блок схема алгоритма, реализующего программное средство

ПРИЛОЖЕНИЕ Г
(обязательное)
Графический интерфейс пользователя

ПРИЛОЖЕНИЕ Д
(обязательное)
Ведомость документов