МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт Факультет информационных технологий и электроники	Кафедра «Информационная безопасность систем и технологий»
Специальность 10.05.03–	Информационная безопасность автоматизированных систем
Специализация –	Защищённые автоматизированные системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

СИСТЕМА УПРАВЛЕНИЯ СПЕЦИАЛЬНЫМ МОБИЛЬНЫМ ТЕРМИНАЛОМ

Студент _____ Молодык Сергей Антонович

Руководитель начальник НИГ-113 АО «ПНИЭИ» _____ Назаренко И.О.

Нормоконтролёр _____ Иванов А.П.

Рецензент заместитель генерального директора по специальным вопросам АО "ПНИЭИ" _____ Сомкин С.А.

Работа допущена к защите (протокол заседания кафедры от 16.06.23 № 10)

Заведующий кафедрой _____ С.Л. Зефиров

Работа защищена с отметкой отлично (протокол заседания ГЭК от 27.06.23 № 5)

Секретарь ГЭК _____ А.П. Иванов

Пенза 2023

Обозначения и сокращения

СМТ – специальный мобильный терминал

УБИ – угроза безопасности информации

ПО – программное обеспечение

ОЗУ – оперативное запоминающее устройство

ПЗУ – постоянное запоминающее устройство

ПК – персональный компьютер

УУ – устройство управления

Оглавление

Введение.....5
1 Анализ выдвигаемых специальных требований к программному обеспечению системы управления мобильным терминалом.....6
1.1 Угроза активации встроенных в микроконтроллер аппаратных закладок.....6
1.2 Угроза чтения информации ограниченного доступа из оперативной памяти.....8
2 Разработка алгоритмов обеспечения специальных свойств программного обеспечения системы управления специальным мобильным терминалом.....10
2.1 Разрежение опасной информации.....10
2.2 Стирание информации из освобождённой оперативной памяти.....18
3 Разработка программного обеспечения системы управления специальным мобильным терминалом.....21
3.1 Разработка системы хранения журналов.....21
3.2 Разработка графического пользовательского интерфейса для просмотра журналов.....32
4 Разработка тестов программного обеспечения системы управления специального мобильного терминала.....43
5 Вредные психофизиологические факторы, влияющие на сердечно-сосудистую систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие.....54
Заключение.....60
Список использованных источников.....61
Приложение А. Обязательное. Графические изображения алгоритмов работы с разреженными строками.....62
Приложение Б. Обязательное. Содержимое файла sstring.h.....64
Приложение В. Обязательное. Содержимое файла sstring.c.....67

Приложение Г. Обязательное. Графическое изображение алгоритма заполнения освобождённой области ОЗУ нулями.....79
Приложение Д. Обязательное. Код функции заполнения освобождённой области ОЗУ нулями.....80
Приложение Е. Обязательное. Графические изображения алгоритмов функций системы хранения журналов.....81
Приложение Ж. Обязательное. Содержимое файла journal_storage.h.....94
Приложение И. Обязательное. Содержимое файла journal_storage.c.....96
Приложение К. Обязательное. Графические изображения алгоритмов функций графического интерфейса просмотрщика журналов.....116
Приложение Л. Обязательное. Содержимое файла journal_view.h.....123
Приложение М. Обязательное. Содержимое файла journal_view.h.....125

Введение

На данный момент рынок мобильных средств безопасной передачи информации в Российской Федерации испытывает дефицит, в то время как спрос на подобные аппараты возрастает. Это приводит к необходимости разработки специальных мобильных терминалов, отвечающих специальным требованиям, которые выдвигают государственные структуры, занимающиеся проблемами информационной безопасности.

Важной частью создания подобного устройства является разработка системы управления специальным мобильным терминалом. Целью данной работы является разработка программного обеспечения подсистемы журналирования системы управления специального мобильного терминала.

Журналы позволяют отслеживать необычную активность в системе, такую как неудачные попытки входа, попытки доступа к защищенным ресурсам и т.д. Журналы помогают быстро обнаруживать и предотвращать потенциальные угрозы безопасности.

Целью ВКР является разработка модуля программного обеспечения для работы с журналами системы управления специальным мобильным терминалом.

Для достижения данной цели необходимо выполнить следующие задачи:

– анализ выдвигаемых специальных требований к программному обеспечению системы управления специальным мобильным терминалом;

– выбор и реализация защитных мер, направленных на нейтрализацию угроз;

– разработка программного обеспечения системы управления специального мобильного терминала;

– разработка тестов программного обеспечения системы управления специальным мобильным терминалом.

1 Анализ выдвигаемых специальных требований к программному обеспечению системы управления мобильным терминалом

1.1 Угроза активации встроенных в микроконтроллер аппаратных закладок

Были рассмотрены следующие УБИ[1]: угроза активации встроенных в микроконтроллер аппаратных закладок, угроза чтения информации ограниченного доступа из оперативной памяти.

Угроза активации встроенных в микроконтроллер аппаратных закладок заключается в том, что злоумышленник может использовать специально созданные функции или механизмы, которые были заложены в процессор на этапе его производства. Эти функции могут предоставлять злоумышленнику возможность получить удаленный доступ к системе или контролировать ее работу без ведома владельца.

Такие закладки могут быть использованы для проведения кражи конфиденциальной информации, управления системой или для ее уничтожения. Такие закладки трудно обнаружить и устранить, так как они могут быть встроены в процессор на физическом уровне.

Поскольку производством процессоров занимается иностранная компания, сам процессор не может являться доверенной средой. Фактических доказательств наличия какой-либо встроенной уязвимости нет, а единственным способом проверить является осмотр всех транзисторов, количество которых в современных процессорах составляет миллиарды единиц, с помощью электронного микроскопа. Осуществить такого рода проверку практически невозможно. Из соображений безопасности делается предположение, что встроенные в процессор уязвимости существуют, а значит, требуется реализовать меры защиты, которые снизили бы риск реализации угрозы информационной безопасности с помощью этой уязвимости.

Существует два метода активации аппаратных уязвимостей: внешний и внутренний.

В первом случае для запуска закладки используется внешний сигнал, который принимается антенной или датчиком. Сигналом от датчика может быть результат какого-либо измерения: температуры, высоты, давления, напряжения и т.д.

Для внутренней активации не требуется взаимодействие с внешним миром. В этом случае закладка или работает всегда или запускается при определенном условии, заложенном при её разработке. Условием для внутренней активации может быть, например, определенная комбинация внутренних сигналов, определенная последовательность выполнения операций, определённый пройденный интервал времени.

Наибольший интерес представляет внутренняя активация, а именно активация последовательностью сигналов. Это условие для активации уязвимости является наиболее гибким и простым для реализации: для него не требуются дополнительные датчики, а активация связана непосредственно с обрабатываемой информацией. Кроме того, в контексте работы с СМТ такая активация может быть рассмотрена в качестве внешней, так как данное оборудование включает в себя средства связи, которые могут передать для обработки активирующую последовательность. Также преимуществом этого способа является относительная защищённость от случайной активации уязвимости: вероятность случайной активации тем ниже, чем больше в активирующей последовательности сигналов.

Выбранная мера, направленная на нейтрализацию данной угрозы – маскирование информации.

Маскирование информации — метод, заключающийся в «наложении» последовательности, состоящей из случайных чисел, на открытый текст. Последовательность случайных чисел называется маской.

Разрежение – частный случай маскирования, представляющий собой наложение нулевой маски на каждый байт и хранение маски в соседнем байте. Таким образом обеспечивается исключение возможности хранения в оперативной памяти и обработки активирующей последовательности в исходном виде. Вместо этого информация обрабатывается малыми фрагментами.

Разрежение опасной информации используется с целью предотвращения обработки микроконтроллером последовательности, которая могла бы активировать потенциальную встроенную уязвимость, заложенную производителем для нарушения свойств информационной безопасности информации ограниченного доступа. Опасной информацией является любая информация, поступающая в СМТ извне.

1.2 Угроза чтения информации ограниченного доступа из оперативной памяти

Угроза чтения информации ограниченного доступа из оперативной памяти заключается в том, что злоумышленник может получить доступ к конфиденциальным данным, которые хранятся в оперативной памяти ПК или другого устройства. Эти данные могут быть паролями, ключами шифрования, личной информацией и другой конфиденциальной информацией.

Злоумышленник может использовать различные методы для чтения информации из оперативной памяти, такие как программное обеспечение для взлома, аппаратные устройства для перехвата данных или физический доступ к устройству. Также в результате работы устройства освобождённая область оперативной памяти может быть выделена для выполнения других функций, которая будет иметь доступ к хранимым в этой области данным.

Выбранная мера, направленная на нейтрализацию данной угрозы – стирание информации из оперативной памяти.

Стирание информации из оперативной памяти – это процесс замены содержимого оперативной памяти нулями при использовании алгоритмов шифрования или в случае, когда область памяти содержит критическую или конфиденциальную информацию, которую необходимо уничтожить.

Опасность невыполнения данного требования заключается в том, что освобождённая оперативная память, содержащая в себе информацию ограниченного доступа, предоставляется в пользование операционной системой для другого процесса, что может привести к раскрытию данной информации.

2 Разработка алгоритмов обеспечения специальных свойств программного обеспечения системы управления специальным мобильным терминалом

2.1 Разрежение опасной информации

Была построена схема разрежения опасной информации согласно определению, данному выше. Результат представлен на рисунке 1.

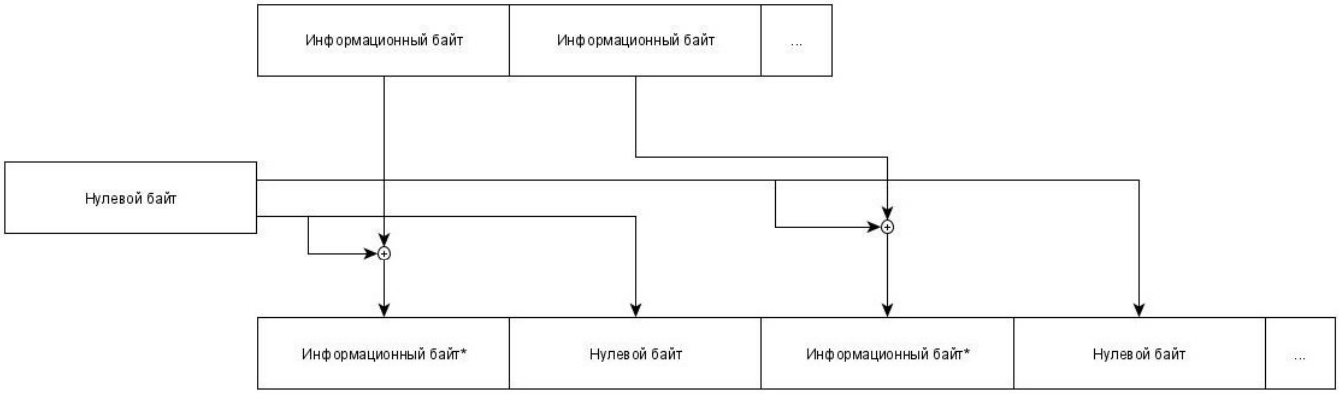

Рисунок 1 – Схема разрежения

На схеме видно, что в результате сложения по модулю два нулевого байта и информационного байта получается «Информационный байт*». Однако исходя из свойства сложения по модулю два «$x \oplus 0 = x$» можно сделать вывод, что «Информационный байт» равен «Информационный байт*». Это значит, что сложение по модулю два можно не выполнять.

В первую очередь, разрежение информации должно быть применено к информации, поступающей в СМТ через устройства ввода. Для этого, во-первых, необходимо реализовать базовые функции работы с разреженными строками (массивами символов). Кроме того, должны быть модифицированы функции работы с текстом и функции работы с надписями внутри самой библиотеки LittlevGL.

Алгоритмы разрежения информации и работы с разреженной информацией были реализованы на языке программирования Си [3]. При

написании кода была использована среда разработки STM32CubeIDE, которая является интегрированной средой разработки для микроконтроллеров STM32 от компании STMicroelectronics[4]. Она предоставляет инструменты для создания и отладки программного обеспечения на языке C или C++ для микроконтроллеров STM32.

Также при разработке использовалась система контроля версий Mercurial[5].

Была реализована функция `void smem2memcpy(void *dst, const void *src, uint16_t size_in_words)`, предназначенная для копирования разреженного буфера в обычный. Входными параметрами являются:

– `void *dst` – указатель на выходной буфер;

– `const void *src` – указатель на начало разреженной строки;

– `uint16_t size_in_words` – длина разреженной строки.

Выходное значение для этой функции отсутствует. Графическое изображение алгоритма копирования разреженного буфера в обычный представлено в приложении А на рисунке А.1.

Была реализована функция `void memcpysmem2(void *dst, const void *src, uint16_t size_in_bytes)`, предназначенная для копирования обычного буфера в разреженный. Входными параметрами являются:

– `void *dst` – указатель на выходной буфер;

– `const void *src` – указатель на начало обычной строки;

– `uint16_t size_in_bytes` – длина обычной строки.

Выходное значение для этой функции отсутствует. Графическое изображение алгоритма копирования обычного буфера в разреженный представлено в приложении А на рисунке А.2.

Была реализована функция `char* sstrcpy(char* dest, const char* src)`, предназначенная для копирования разреженной строки. Входными параметрами являются:

– `char* dest` – указатель на новое расположение строки;

– `const char* src` – указатель на копируемую строку.

Выходным значением является указатель на массив символов, в котором теперь хранится скопированная строка.

Была реализована функция `char* sstrncpy(char* dest, const char* src, size_t n)`, предназначенная для копирования разреженной строки не длиннее определённого количества символов. Входными параметрами являются:

– `char* dest` – указатель на новое расположение строки;

– `const char* src` – указатель на копируемую строку;

– `size_t n` – максимальная длина конечной строки в байтах.

Выходным значением является указатель на массив символов, в котором теперь хранится скопированная строка.

Была реализована функция `char* sstrcat(char* dest, const char* src)`, предназначенная для конкатенации разреженных строк. Входными параметрами являются:

– `char* dest` – первая часть разреженной строки;

– `char* src` – вторая часть разреженной строки.

Выходным значением является указатель на массив символов, содержащий в себе результат конкатенации строк.

Была реализована функция `char* sstrncat(char *dest, const char* src, size_t n)`, предназначенная для конкатенации разреженных строк, длина выходной строки ограничена n байт. Входными параметрами являются:

– `char* dest` – первая часть разреженной строки;

– `char* src` – вторая часть разреженной строки;

– `size_t n` – максимальная длина конечной строки в байтах.

Выходным значением является указатель на массив символов, содержащий в себе результат конкатенации строк, результирующая длина полученной строки не превышает n.

Была реализована функция `char* sstrchr(const char* src, char c)`, предназначенная для поиска первого вхождения символа в строку. Входными параметрами являются:

– `const char* src` – строка, в которой осуществляется поиск;

– `char c` – символ, позицию которого необходимо определить.

Выходным значением является указатель на элемент массива, содержащий первое вхождение символа в строку.

Была реализована функция `char* sstrchr_rev(const char* src, char c)`, предназначенная для поиска первого вхождения символа в строку. Входными параметрами являются:

– `const char* src` – строка, в которой осуществляется поиск;

– `char c` – символ, позицию которого необходимо определить.

Выходным значением является указатель на элемент массива, содержащий последнее вхождение символа в строку.

Была реализована функция `size_t sstrlen(const char* src)`, предназначенная для определения длины строки. Входным параметром является `const char* src` – строка, длину которой необходимо определить. Выходным значением является размер строки.

Была реализована функция `size_t sstrnlen(const char* src, size_t n)`, предназначенная для определения длины строки. Входным параметром является:

– `const char* src` – строка, длину которой необходимо определить;

– `size_t n` – предел длины строки.

Выходным значением является длина строки, если она не превышает n, либо n в противном случае.

Была реализована функция `long sstrtol(const char *nptr, char **endptr, register int base)`, предназначенная для преобразования массива символов в целое число. Входными параметрами являются:

– `const char *nptr` – указатель на начало строки;

– `char **endptr` – ссылка на объект типа char *, значение которой изменяется функцией после первого вызова. Этот параметр также может быть нулевым указателем, в этом случае он не используется;

– `register int base` – основание системы счисления.

Выходным значением является целое число. После завершения выполнения функции `char **endptr` будет содержать указатель на начало строки, если строка была целиком преобразована, или указатель на последний преобразованный символ, если не вся строка была преобразована в число из-за переполнения. Графическое изображение алгоритма преобразования разреженной строки в число представлено в приложении А на рисунке А.3.

Была реализована функция `int sstrcmp(const char *s1, const char *s2)`, предназначенная для сравнения двух разреженных строк. Входными параметрами являются:

– `const char *s1` – первая разреженная строка для сравнения;

– `const char *s2` – вторая разреженная строка для сравнения.

Выходным значением является разница между числовыми кодами первых различающихся символов, если строки не равны, или 0, если строки одинаковые.

Была реализована функция `int sstrncmp(const char *s1, const char *s2, size_t n)`, предназначенная для сравнения n символов двух разреженных строк. Входными параметрами являются:

– `const char *s1` – первая разреженная строка для сравнения;

– `const char *s2` – вторая разреженная строка для сравнения;

– `size_t n` – количество сравниваемых символов в строках.

Выходным значением является разница между числовыми кодами первых различающихся символов, если n символов двух разреженных строк не равны, или 0, если n символов двух разреженных строк одинаковые.

Была реализована функция `int sstrncmp2(const char *s1, const char *s2, size_t n)`, предназначенная для сравнения n символов разреженной и обычной строки. Входными параметрами являются:

– `const char *s1` – первая разреженная строка для сравнения;

– `const char *s2` – вторая обычная строка для сравнения;

– `size_t n` – количество сравниваемых символов в строках.

Выходным значением является разница между числовыми кодами первых различающихся символов, если n символов разреженной и обычной строк не равны, или нулевой указатель, если n символов разреженной и обычной строк одинаковые.

Была реализована функция `const char* sstrstr(const char* sstring, const char* ssubstring)`, предназначенная для поиска разреженной подстроки в разреженной строке. Входными параметрами являются:

– `const char* sstring` – разреженная строка, в которой выполняется поиск;

– `const char* ssubstring` – разреженная подстрока, поиск которой выполняется.

Выходным значением является указатель на начало подстроки в разреженной строке, если она была найдена, или нулевой указатель, если подстрока не была найдена.

Была реализована функция `const char* sstrstr2(const char* sstring, const char* substring)`, предназначенная для поиска обычной подстроки в разреженной строке. Входными параметрами являются:

– `const char* sstring` – разреженная строка, в которой выполняется поиск;

– `const char* ssubstring` – обычная подстрока, поиск которой выполняется.

Выходным значением является указатель на начало подстроки в разреженной строке, если она была найдена, или нулевой указатель, если подстрока не была найдена.

Была реализована функция `const char* sstrstr3(const char* string, const char* ssubstring)`, предназначенная для поиска разреженной подстроки в обычной строке. Входными параметрами являются:

– `const char* sstring` – обычная строка, в которой выполняется поиск;

– `const char* ssubstring` – разреженная подстрока, поиск которой выполняется.

Выходным значением является указатель на начало подстроки в строке, если она была найдена, или нулевой указатель, если подстрока не была найдена.

Была реализована функция `uint8_t sdatacopy(uint16_t* dest, uint16_t* src, size_t maxlen)`, предназначенная для копирования разреженной строки, если её длина не превышает n. Входными параметрами являются:

– `uint16_t* dest` – указатель на новое расположение строки;

– `uint16_t* src` – указатель на исходную строку;

– `size_t maxlen` – максимальная длина копируемой строки.

Выходным значением является длина скопированной строки, если она была скопирована, или 0, если длина копируемой строки превышает максимальную длину.

Была реализована функция `void sstrrej(char* src, char ch)`, предназначенная для удаления символа из разреженной строки. Входными параметрами являются:

– `char* src` – разреженная строка, из которой необходимо удалить символ;

– `char ch` – символ для удаления.

Выходное значение для этой функции отсутствует.

Была модифицирована библиотека LVGL таким образом, чтобы надписи (lv_label) поддерживали работу с разреженными строками.

В структуру `lv_label_ext_t` было добавлено поле `uint8_t sparse`, являющееся флагом разреженности строки.

Была добавлена функция `void lv_label_set_sparse(lv_obj_t * label, bool en)`, предназначенная для изменения значения флага разреженности строки. Входными параметрами являются:

– `obj_t * label` – указатель на надпись;

– `bool en` – значение, которое примет флаг после установки.

Выходное значение для этой функции отсутствует.

Была добавлена функция `bool lv_label_get_sparse(const lv_obj_t * label)`, предназначенная для получения состояния флага разреженности строки. Входным параметром для этой функции является `const lv_obj_t * label` – указатель на надпись. Выходным значением является логическая переменная, принимающая значение «истина», если надпись разреженная, и «ложь», если строка не является разреженной.

Была добавлена функция `void lv_label_append_text_fmt(lv_obj_t * label, const char * fmt, ...)`, предназначенная для добавления форматированного текста.

– `obj_t * label` – указатель на надпись;

– `const char * fmt` – форматированная строка (`printf`-подобный формат);

– ... – спецификаторы преобразования.

Выходное значение для этой функции отсутствует.

Разработанные функции представлены в приложении Б и приложении В.

2.2 Стирание информации из освобождённой оперативной памяти

Поскольку буфер для хранения информации (в том числе и информации о пикселях, подготовленных к отрисовке) в библиотеке LittleVGL находится в ОЗУ, необходимо осуществлять стирание информации из ОЗУ, освобождённой в ходе работы с графическим интерфейсом. Для этого необходимо определить, каким образом происходит выделение и освобождение памяти.

В LittlevGL выделение и освобождение памяти происходит через функции `lv_mem_alloc` и `lv_mem_free`.

Функция `lv_mem_alloc` должна выделить блок памяти размером `size` и вернуть указатель на него. Если память выделить не удалось, функция должна вернуть `NULL`.

Функция `lv_mem_free` должна освободить блок памяти, на который указывает `p`. Если `p` равно `NULL`, функция бездействует.

Если необходимо выделить память под объект, вызывается функция `lv_mem_alloc`, которая проверяет, достаточно ли свободной памяти для выделения. Если свободной памяти недостаточно, то происходит сборка мусора, в результате которой освобождаются неиспользуемые блоки памяти. Если после

сборки мусора свободной памяти все еще недостаточно, то вызывается функция `lv_mem_defrag`, которая дефрагментирует свободную память. После этого происходит повторная попытка выделения памяти.

Если объект больше не нужен, вызывается функция `lv_mem_free`, которая освобождает занятую объектом память и добавляет ее в список свободной памяти.

Для управления памятью LittlevGL использует связанный список записей типа `lv_mem_ent_t`.

Структура `lv_mem_ent_t` в версии библиотеки LittlevGL 6.1 используется для контроля записей в таблице управления памятью. Она содержит два поля:

– `lv_mem_header_t header` – заголовок блока памяти, содержащий информацию о его размере и состоянии (выделен/освобожден).

– `uint8_t first_data` – указатель на первый байт данных в выделенном блоке памяти.

Каждая запись в таблице соответствует одному выделенному блоку памяти. Записи хранятся в порядке возрастания адресов блоков памяти. При освобождении блока памяти LittlevGL ищет запись в таблице, соответствующую этому блоку, и удаляет ее из списка. Если запись не найдена, это означает, что блок памяти не был выделен с помощью стандартных функций и LittlevGL не может его освободить.

Тип `lv_mem_header_t` является объединением с полями:

– Структура:

– `MEM_UNIT used : 1` – флаг, указывающий на то, используется ли память;

– `MEM_UNIT d_size : 31` – размер используемой памяти;

– `MEM_UNIT header` – заголовок блока памяти целиком.

`MEM_UNIT` – тип переменной, определённый директивой препроцессора как целочисленное значение размером 32 бита или 64 бита (в зависимости от конфигурации).

Указатель на заголовок не хранится в явном виде, но, поскольку он находится прямо перед данными, для которых он является заголовком, можно хранить только указатель на данные и, при необходимости, использовать указатель со сдвигом назад (величина сдвига будет соответствовать размеру заголовка в байтах), чтобы обратиться к заголовку.

Таким образом, обладая длиной фрагмента памяти, готового к освобождению, и указателем на начало этого фрагмента, полученным из структуры `` `lv_mem_ent_t` ``, можно стереть информацию из освобождаемой памяти путём её заполнения нулями с помощью функции `` `memset` `` стандартной библиотеки `<memory.h>`. Функцию затирания следует поместить в функцию `` `lv_mem_free` ``, так как она вызывается каждый раз при освобождении памяти. Графическое изображение алгоритма заполнения освобождённой области ОЗУ представлено в приложении Г на рисунке Г.1. Код функции, выполняющей заполнение области ОЗУ нулями, приведён в приложении Д.

3 Разработка программного обеспечения системы управления специальным мобильным терминалом

3.1 Разработка системы хранения журналов

SPI Flash S25FL256 – это флэш-память, которая используется для хранения данных в электронных устройствах и имеет объем памяти 256 Мбайт[6]. SPI NAND - это тип флэш-памяти, который использует интерфейс Serial Peripheral Interface (SPI) для связи с микроконтроллером или другим устройством. Она является одним из самых популярных типов флэш-памяти, которые используются в настоящее время.

Организация памяти в S25FL256 основана на блоках, которые в свою очередь делятся на страницы. Всего в памяти S25FL256 содержится 65 536 блоков размером 4 Кбайт каждый и 8192 страницы по 256 байт.

Чтобы удалить данные из SPI NAND, необходимо удалить целый блок, который обычно состоит из нескольких страниц. Размер блока может быть от 16 Кбайт до 512 Кбайт.

Функциональные возможности SPI Flash S25FL256 включают в себя высокую скорость передачи данных, защиту от записи и защиту от чтения. Она также поддерживает различные режимы работы, такие как режим чтения, записи, стирания и блокировки. Также стоит отметить, что S25FL256 имеет низкий уровень энергопотребления.

Общая структурная схема СМТ представлена на рисунке 2.

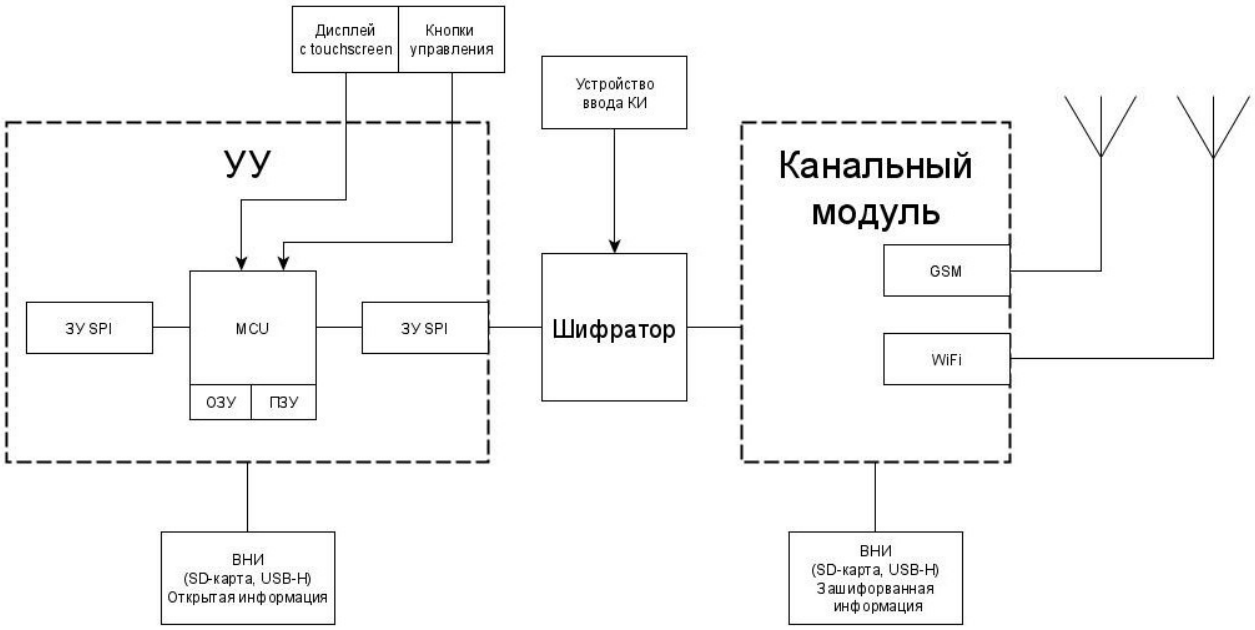

Рисунок 2 – Общая структурная схема СМТ

Система хранения журналов является частью системы управления СМТ. Система хранения журналов предназначена для сохранения журналов, которые содержат важную информацию о работе системы и приложений. Она позволяет быстро находить и анализировать данные журналов для выявления проблем и ошибок, улучшения производительности и безопасности системы. Система хранения журналов используется для соблюдения требований законодательства и нормативных актов, которые обязывают хранить записи журналов в течение определенного периода времени. Структурная схема системы управления СМТ представлена на рисунке 3.

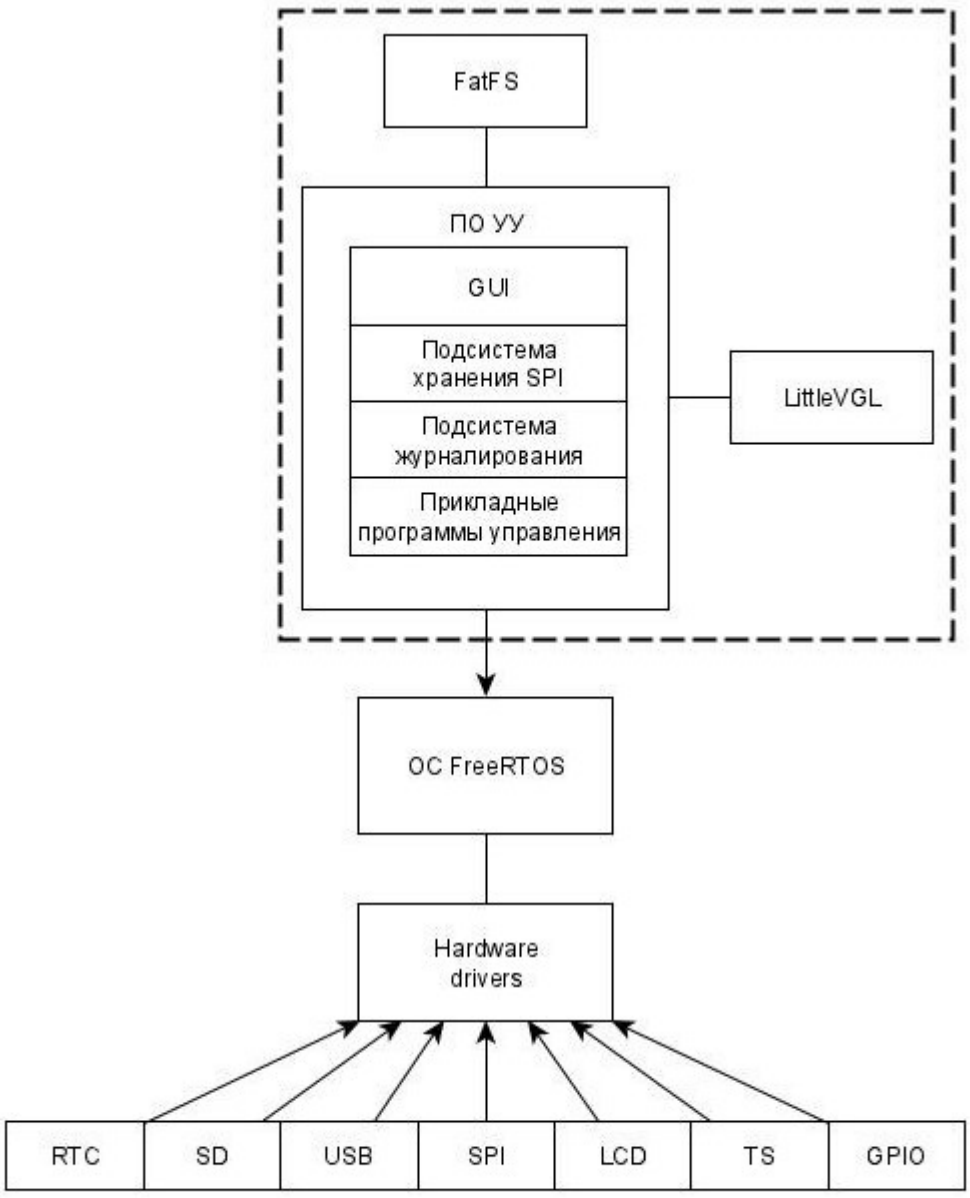

Рисунок 3 – Структурная схема системы управления СМТ

Существует два вида журналов: журнал безопасности и журнал событий, хранящиеся в разных зонах flash. В начале каждого блока хранится служебная информация о блоке:

– 2 байта – последовательность, сигнализирующая о начале блока;

– 2 байта – порядковый номер блока.

Для работы с разными журналами было определено перечисление типов журналов `Journal_type`:

– `JEVENT = 0` – журнал событий;

– JSECURITY – журнал безопасности;

– JCNT – общее количество журналов.

В коде программы были введены следующие макроопределения:

– ADR_JOURNAL_EVENT 0xA80000 – начало пространства адресов во внешней флеш, выделенного для хранения записей журнала событий;

– ADR_JOURNAL_SECUR 0xAB0000 – начало пространства адресов во внешней флеш, выделенного для хранения записей журнала безопасности;

– FL_PAGE_SIZE 0x10000 – длина страницы во внешней флеш.

Журналы пишутся циклически, перезапись организуется стиранием блока, содержащего самые старые записи.

Было определено перечисление:

– BLOCK_USE = 0xBEAF – последовательность, обозначающая начало используемого блока;

– BLOCK_FREE = 0xFFFF – последовательность, обозначающая начало неиспользуемого блока.

Было определено перечисление:

– RECORD_USE = 0xA525 – последовательность, обозначающая начало новой непустой записи;

– RECORD_FREE = 0xFFFF – последовательность, обозначающая начало пустой записи;

– RECORD_ENDBLOCK = 0x0000 – последовательность для заполнения свободной части блока, если размер новой записи больше, чем размер свободной части блока.

Хранение записи во flash организовано в двоичном виде. Это позволяет не хранить в памяти лишний объём информации, обходясь лишь кодом события. Кроме того, запись содержит в себе параметр с опциональными текстовыми атрибутами.

Записи в журналах обладают переменной длиной. Записи формируются из следующих параметров:

– 2 байта – последовательность, обозначающая начало новой записи и флаг, символизирующий, используется запись или нет;

– 2 байта – длина записи в байтах. Длина записи равна сумме длины данных, длины даты и времени, длины CRC-последовательности и длины этого параметра;

– данные журнала:

  – 4 байта – дата/время в формате FatFS;

  – 2 байта – код события;

  – m байт – атрибуты;

– 2 байт длина – длина записи в байтах;

– 2 байта – CRC-последовательность.

Параметр длины записи присутствует дважды для реализации возможности поиска предыдущей записи.

Минимальный размер записи журнала составляет 14 байт. Средний размер записи журнала – 64 байта.

CRC-последовательность – это последовательность бит, которая используется для проверки целостности данных в сетевых протоколах и устройствах хранения данных. Она создается путем применения алгоритма циклического избыточного кодирования (CRC) и генерирует уникальный код на основе исходных данных. Она используется для определения, были ли данные повреждены в процессе передачи или хранения.

CRC-последовательность формируется для проверки целостности записи журнала и формируется из следующих параметров записи: стартовая последовательность, длина записи, данные, длина записи.

В блоке умещается в среднем около 1000 записей. Длина журнала событий равна трём блокам. Длина журнала безопасности также равна трём блокам.

Если в блоке осталось свободное место, но свободного места недостаточно для того, чтобы поместить в него новую запись, то свободное место в блоке после последней записи в блоке заполняется нулями.

Для работы с записями журналов была определена структура `Journal_record`:

– `is_bad_crc` – флаг проверки целостности записи;

– `datetime` – дата/время в формате FATFS;

– `event_code` – код события;

– `ext_len` – размер дополнительной области;

– `ext_data[MAX_EXTDATA_LEN]` – указатель на буфер для дополнительной области;

– `storage_addr` – адрес записи на носителе;

– `parcedstringbuf` – буфер для строки текстового представления записи журнала (для функции с парсингом);

– `parcedbuf_len` – размер буфера `parcedstringbuf` (для функции с парсингом);

– `parcedbuf_offset` – смещение в буфере `parcedstringbuf` (для добавления текста о нарушении целостности).

Для удобства работы с функциями поиска блоков и записей было определено перечисление `Seek_type`:

– `FIRST=0` – поиск первого блока (записи);

– `LAST` – поиск последнего блока (записи);

– `NEXT` – поиск следующего блока (записи);

– `PREV` – поиск предыдущего блока (записи);

– `NEW` – поиск места для нового блока (только для работы с блоками).

Было определено перечисление возможных значений, возвращаемых функциями работы с записями журналов `Jretcode`:

– JERRORPARCE = -3 – ошибка парсинга записи журнала;

– JERROR – ошибка работы с записью журнала;

– JBADCRC – ошибка, связанная с проверкой целостности записи;

– JSUCCESS – успешное выполнение функции;

– JEOF – конец журнала.

Служебная информация о блоке содержит в себе его порядковый номер. Значение каждого нового блока увеличивается каждый раз при его записи в память. Так как количество памяти, выделенное для хранения порядкового номера, ограничено, блок может обладать только номером от 0 до 65535. Эти числа можно рассматривать как конечное поле GF(65536), так как для них определены операции сложения, взятия противоположного значения, умножения и деления. В общем случае, операции "больше" и "меньше" не могут быть определены. Однако при хранении в памяти нескольких последовательно идущих элементов поля возможно определить, когда стоящие рядом два элемента из разных циклов записи. Циклами записи здесь называется последовательное записывание элементов от 0 до 65535. Если принять во внимание то, что запись новых блоков и перезапись старых блоков новыми идёт от меньших адресов к большим адресам, можно сделать вывод, что большим элементом будет:

– большее число, если элементы из одного цикла;

– меньшее число, если элементы из разных циклов.

Это важно для определения блока с самыми свежими и с самыми старыми записями. Для этого также необходимо, чтобы количество записанных блоков не превышало размерность поля-1, иначе записанные порядковые номера начнут повторяться, что приведёт к невозможности определения наибольшего и наименьшего номера блока.

Была реализована вспомогательная статическая функция `static bool is_from_different_cycle(uint16_t curr_num, uint16_t num,`

uint16_t total_block), предназначенная для определения того, являются ли два номера блока элементами из разных циклов записи. Входными параметрами для этой функции являются:

– uint16_t curr_num – текущий сохранённый номер блока;

– uint16_t num – номер рассматриваемого блока;

– uint16_t total_block – общее количество блоков.

Выходным значением является переменная логического типа, возвращающая единицу, если номера блоков из разных циклов, и 0 в обратном случае.

В отличие от глобальных функций в Си, доступ к статическим функциям ограничен файлом, в котором они объявлены. Причиной для придания статичности этой функции является возможность повторного использования того же имени функции в других файлах.

Была реализована функция uint32_t get_block_addr(Journal_type jtype, Seek_type type, uint32_t current_block_addr), предназначенная для получения адреса необходимого блока. Входными параметрами для этой функции являются:

– Journal_type jtype – тип журнала, среди адресов блоков которого выполняется поиск. Типы возможных журналов описаны в перечислении Journal_type;

– Seek_type type – тип необходимого блока. Возможные для поиска блоки определены в перечислении Seek_type;

– uint32_t current_block_addr – адрес текущего блока, относительно которого выполняется поиск. Данный параметр должен принимать значение, не равное NULL, в случае, если входной параметр Seek_type принимает значение NEXT или PREV.

Выходным значением является целое число размером 32 байта, являющееся адресом необходимого блока, или 0, если в процессе поиска

произошла ошибка или такой блок не был найден. Графическое изображение алгоритма поиска адреса блока представлено в приложении Е на рисунках от Е.1 до Е.6.

Была реализована функция `uint32_t get_next_free_space(Journal_type jtype, uint32_t start_address, uint16_t need_space)`, предназначенная для получения свободного места для сохранения записи журнала. Входными параметрами для этой функции являются:

– `Journal_type jtype` – тип журнала, запись которого необходимо сохранить. Типы возможных журналов описаны в перечислении `Journal_type`;

– `uint32_t start_address` – адрес, начиная с которого осуществляется поиск свободного места для сохранения записи журнала. Если `start_address` равен нулю, тогда поиск свободного места осуществляется с начала блока с самыми свежими записями;

– `uint16_t need_space` – размер сохраняемой записи в байтах.

Выходным значением является целое число размером 32 байта, являющееся адресом свободного пространства размером не менее `need_space`, или 0, если в процессе выполнения поиска произошла ошибка. Графическое изображение алгоритма поиска адреса свободного пространства представлено в приложении Е на рисунке Е.7.

Была реализована функция `int read_record_header(uint32_t addr, uint16_t *start, uint16_t *lenght)`, предназначенная для чтения заголовка записи журнала. Входными параметрами для этой функции являются:

– `uint32_t addr` – адрес записи, заголовок которой необходимо получить;

– uint16_t *start – указатель на переменную для сохранения последовательности, обозначающей начало записи журнала, и флага использования записи;

– uint16_t *lenght – указатель на переменную для сохранения длины записи журнала.

Выходным значением является: 0, если чтение заголовка произошло успешно, и -1, если в процессе выполнения функции произошла ошибка.

Была реализована функция uint32_t find_last_record_in_block(uint32_t start_addr), предназначенная для поиска адреса самой свежей записи в блоке. Входными параметрами для этой функции является uint32_t start_addr – адрес начала блока. Выходным значением является адрес самой свежей записи в блоке или 0, если в процессе выполнения функции произошла ошибка. Графическое изображение алгоритма поиска адреса последней записи в блоке представлено в приложении Е на рисунке Е.8.

Была реализована функция uint32_t get_record_addr(Journal_type jtype, Seek_type type, uint32_t start_address), предназначенная для получения адреса записи журнала. Входными параметрами для этой функции являются:

– Journal_type jtype – тип журнала, в котором осуществляется поиск записи. Типы возможных журналов описаны в перечислении Journal_type;

– Seek_type type – направление поиска записи. Возможные направления поиска представлены в перечислении Seek_type;

– uint32_t start_address – адрес, относительно которого определяется направление поиска записи журнала. Данный параметр должен принимать значение, не равное NULL, в случае, если входной параметр Seek_type принимает значение NEXT или PREV.

Выходным значением является адрес необходимой записи или 0, если в процессе выполнения функции произошла ошибка. Графическое изображение алгоритма поиска адреса необходимой записи представлено в приложении Е на рисунке Е.9.

Была реализована функция `JRetcode add_record(Journal_type jtype, uint16_t event_code, uint8_t *data, uint16_t datalen)`, предназначенная для сохранения записи журнала. Входными параметрами для этой функции являются:

– `Journal_type jtype` – тип журнала, в котором должна быть сохранена запись. Типы возможных журналов описаны в перечислении `Journal_type`;

– `uint16_t event_code` – код события;

– `uint8_t *data` – указатель на буфер с расширенными данными;

– `uint16_t datalen` – длина буфера `data` в байтах.

Выходным значением является код завершения функции. Возможные коды завершения функции представлены в перечислении `Jretcode`. Графическое изображение алгоритма добавления новой записи представлено в приложении Е на рисунке Е.10.

Была реализована функция `JRetcode get_record(Journal_type jtype, Seek_type stype, uint32_t prev_addr, Journal_record *record)`, предназначенная для получения записи журнала. Входными параметрами для этой функции являются:

– `Journal_type jtype` – тип журнала, из которого осуществляется извлечение записи. Типы возможных журналов описаны в перечислении `Journal_type`;

– `Seek_type stype` – направление поиска записи. Возможные направления поиска представлены в перечислении `Seek_type`;

– `uint32_t prev_addr` – адрес записи журнала, относительно которого осуществляется поиск записи;

– `Journal_record *record` – указатель на переменную для хранения найденной записи журнала.

Выходным значением является код завершения функции. Возможные коды завершения функции представлены в перечислении `Jretcode`. Графическое изображение алгоритма получения необходимой записи в блоке представлено в приложении Е на рисунке Е.11.

Была реализована функция `void journal_storage_init()`, предназначенная для первичной инициализации журналов. Функция не принимает параметров и не возвращает значений.

Разработанные функции системы хранения журналов представлены в приложении Ж и приложении И.

3.2 Разработка графического пользовательского интерфейса для просмотра журналов

LittlevGL (Light and Versatile Graphics Library) - это библиотека графического интерфейса пользователя (GUI), которая может быть использована для разработки приложений с графическим интерфейсом на микроконтроллерах и других устройствах с ограниченными ресурсами[7].

Основные характеристики LittlevGL:

– поддержка различных типов экранов, включая монохромные, цветные, с возможностью сенсорного ввода данных и без;

– возможность создавать собственные виджеты;

– эффективное использование памяти и процессора, что позволяет использовать библиотеку на микроконтроллерах с ограниченными ресурсами;

– поддержка многозадачности и анимации, включая анимацию переходов, изменение прозрачности элементов интерфейса и т. д.;

– наличие документации, кода примеров и сообщества разработчиков.

LittlevGL доступна как Open Source и имеет активное сообщество, которое поддерживает различные форумы обмена знаниями и опытом, а также обновления и патчи для правок ошибок.

Преимуществами перед другими графическими библиотеками для встроенных систем является открытый исходный код и наличие MIT лицензии.

На рисунке 4 представлена схема взаимодействия между драйверами и приложением, написанным пользователем.

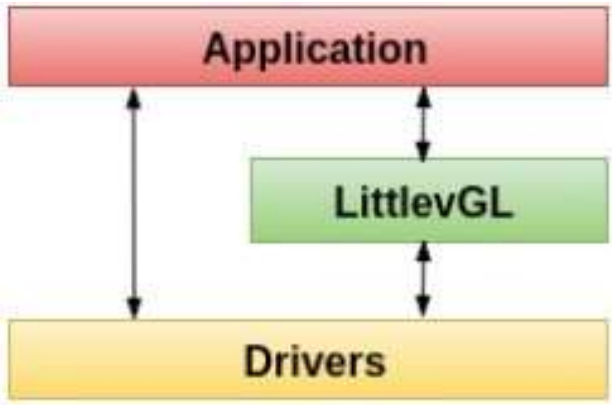

Рисунок 4 – Схема взаимодействия между драйверами и приложением

На верхнем уровне расположено приложение, которое создает графический интерфейс и выполняет определенные задачи.

Приложение взаимодействует с библиотекой для создания графического интерфейса. Библиотека содержит интерфейс HAL (уровень аппаратной абстракции) для регистрации драйверов дисплея и устройств ввода.

Драйвер содержит функции для управления дисплеем, опционально для графического процессора и для чтения входных данных с тачпада или кнопок.

Существуют два основных вида аппаратных конфигураций. Один со встроенным периферийным драйвером LCD/TFT, а другой без него. В обоих случаях потребуется кадровый буфер для хранения текущего изображения:

Микроконтроллер с драйвером TFT/LCD

Если микроконтроллер имеет драйвер для TFT/LCD, то есть возможность подключить дисплей напрямую через интерфейс RGB. При этом буфер кадра

может находиться во внутренней оперативной памяти (если у микроконтроллера достаточно оперативной памяти) или во внешней оперативной памяти (если у микроконтроллера есть интерфейс памяти).

Контроллер внешнего дисплея

Если микроконтроллер не имеет драйвера для TFT/LCD, тогда необходимо использовать внешний контроллер дисплея. В этом случае микроконтроллер может обмениваться данными с контроллером дисплея через параллельный порт, SPI или I^2C. Обычно в контроллере дисплея находится буфер кадра, что экономит много оперативной памяти для микроконтроллера.

Библиотека имеет возможность настраивать количество внутренних буферов:

– один буфер, когда LittlevGL рисует содержимое экрана в буфер и отправляет его на дисплей;

– два буфера с неполным размером экрана, пока идет отрисовка в одном буфере, содержимое другого буфера отправляется для отображения в фоновом режиме;

– два буфера размером с целый экран.

Разработка системы управления специального мобильного устройства ведётся на базе микроконтроллера из семейства STM32, которые являются очень популярными для использования в различных электронных устройствах.

LittlevGL поддерживает микроконтроллеры, способные управлять дисплеем через параллельный порт, SPI, интерфейс RGB, включая такие микроконтроллеры, как STM32F, STM32H, NXP Kinetis, LPC, iMX, dsPIC33, PIC32.

Минимальные требования:

– 16, 32 или 64-разрядный микроконтроллер или процессор;

– флэш-память/ПЗУ: размер более 64 Кб;

– статическое использование оперативной памяти: примерно от 8 до 16 КБ;

– стек: более 2 КБ;

– динамические данные (heap): более 4 КБ;

– буфер отображения: больше, чем горизонтальное разрешение в пикселях.

При использовании сторонних библиотек во время разработки программного обеспечения для специального мобильного устройства стоит учитывать возможность просмотра исходного кода. Это однако даёт возможность проверки библиотеки на наличие вредоносного кода.

Таким образом, использовать библиотеку LittlevGL для разработки графического пользовательского интерфейса для специального мобильного устройство возможно и целесообразно.

Для использования графической библиотеки необходимо ее инициализировать и настроить необходимые компоненты. Порядок инициализации следующий:

– Вызвать `lv_init()`.

– Инициализировать драйверы устройств отображения и ввода.

– Зарегистрируйте драйверы устройств отображения и ввода в LittlevGL.

– Настроить вызов `lv_tick_inc(x)` каждые x миллисекунд в прерывании, чтобы сообщить об истекшем времени в LittlevGL.

– Настроить вызов `lv_timer_handler()` каждые несколько миллисекунд для обработки задач, связанных с LittlevGL.

Каждый элемент интерфейса является экземпляром структуры `lv_obj_t`. Структура поддерживает возможность сохранения пользовательской информации.

Для создания элемента используются функции `lv_«название_элемента»_create(«родительский_объект»)`.

Размещение на экране производится с помощью функции `lv_obj_align(«объект», «расположение_на_родительском_объекте», «сдвиг_по_x», «сдвиг_по_y»)`.

Задать размер объекта можно с помощью функции `lv_obj_set_size(«объект», «ширина», «высота»)`.

Реагирование на события закрепляются за объектами с помощью функции `lv_obj_add_event_cb(«объект», «пользовательская_функция», «тип_события», «пользовательская_информация»)`, где «пользовательская_функция» - функция типа `static void` от одной переменной типа `lv_event_t*`.

Для изменения внешнего вида объекта следует использовать стили. Экземпляр стиля содержит такие свойства, как цвет фона, ширина границы, шрифт и т. д. Стили представлены переменными типа `lv_style_t`. В объектах сохраняется только их указатель, поэтому они должны быть определены как статические или глобальные. Перед использованием стиля его необходимо инициализировать с помощью `lv_style_unit(&«название_стиля»)`. После этого можно добавить свойства для настройки стиля.

Был разработан графический интерфейс для просмотра журнала безопасности и журнала событий.

Описание графического интерфейса:

– экран выбора режима журнала;

– экран просмотра журнала.

Было определено перечисление `read_direction_t`, предназначенное для упрощения передачи аргументов в функции:

– `READ_DIRECTION_FORWARD` – направление чтения журнала вперёд;

– `READ_DIRECTION_BACKWARD` – направление чтения журнала назад.

Была определена статическая переменная `static uint32_t journal_page_num = 1`, предназначенная для хранения номера текущей страницы журнала.

Была определена статическая переменная `static char journal_page_num_text[5]`, предназначенная для постоянного хранения номера страницы журнала в текстовом виде.

Была определена статическая переменная `static Journal_type active_journal_type = JEVENT`, предназначенная для хранения типа просматриваемого журнала.

Была реализована функция `void show_journal(Journal_type jtype)`, предназначенная для отрисовки первой страницы журнала и создания и загрузки двух соседних экранов (страниц). Входным параметром является `Journal_type jtype` – тип просматриваемого журнала. Выходное значение для этой функции отсутствует.

Альтернативой загрузке фрагмента журнала с возможностью смены страниц является загрузка всего журнала на одной странице с возможностью перелистывать записи. Преимуществом выбранного способа является отсутствие необходимости хранить все отрисованные записи в видеопамяти. Это является важным условием при работе с мобильными устройствами, поскольку их ресурсы ограничены.

Экран выбора режима журнала представляет собой список (lv_list) с двумя кнопками. К кнопкам привязаны две функции обратного вызова (callback), вызывающие одну и ту же функцию отрисовки экрана просмотра журнала, но с разными аргументами: `jtype` является аргументом типа `journal_type`.

В действительности в результате работы функции `'show_journal_screen'` создаётся 3 экрана: активный, предыдущий и

следующий. В поле 'user_data' каждого экрана находится структура типа 'screen_neighbors_t'. Эта структура содержит следующие поля:

– lv_obj_t * prev – указатель на предыдущий экран;

– lv_obj_t * next – указатель на следующий экран.

Эти поля предназначены для хранения указателей на предыдущий и следующий экран соответственно. Хранение указателей необходимо для смены активного экрана.

Экран просмотра журналов представляет собой страницу (lv_page), на которой расположены таблица с записями, номер страницы и кнопки «Пред.» и «След.», вызывающие функцию смены активного экрана. Эта функция также вызывает функцию загрузки более новых или более старых записей в зависимости от направления чтения. Кнопки «Пред.» и «След.» не отображаются, если отсутствуют соответствующие записи для отображения.

Была определена структура page_children_t, предназначенная для сохранения указателей на дочерние объекты страницы и удобного доступа к ним:

– lv_obj_t * table – указатель на таблицу с записями;

– lv_obj_t * label – указатель на номер страницы;

– lv_obj_t * prev_pg_btn – указатель на кнопку перехода на предыдущую страницу;

– lv_obj_t * next_pg_btn – указатель на кнопку перехода на следующую страницу.

Была реализована статическая функция static lv_obj_t * create_journal_screen_new(), предназначенная для создания экрана для просмотра журнала. Входные параметры для этой функции отсутствуют. Выходным значением является указатель на экземпляр созданного экрана. Графическое изображение алгоритма создания нового экрана для просмотра журналов представлено в приложении К на рисунке К.1.

Была реализована статическая функция `static void create_new_table(lv_obj_t * page)`, предназначенная для создания таблицы с записями журналов. Входным параметром является `v_obj_t * page` – указатель на страницу-родителя. Выходное значение для этой функции отсутствует. Графическое изображение алгоритма создания таблицы для просмотра журналов представлено в приложении К на рисунке К.2.

В поле `'user_data'` таблицы записывается структура типа `'edge_records_t'`. Структура содержит следующие поля:

– `uint32_t first` – адрес первой записи в таблице;

– `uint32_t last` – адрес последней записи в таблице.

Эти поля нужны для хранения адресов в памяти на SPI-flash записей журналов, расположенных в начале и в конце этой таблицы. Это требуется для поиска и загрузки записей в зависимости от направления чтения журнала: загрузка предыдущих записей для последней, загрузка следующих записей для первой. Это обусловлено чтением журнала от свежих записей к старым и тем, что в рамках одного блока более новые записи имеют номер ячейки памяти больше, чем более старые.

Была реализована статическая функция `static lv_obj_t* get_table(lv_obj_t* screen)`, предназначенная для получения таблицы с записями. Входным параметром для этой функции является `lv_obj_t* screen` – указатель на экран, на котором расположена таблица. Выходным значением является указатель на таблицу, расположенную на экране.

Была реализована статическая функция `static void align_buttons_to_table(lv_obj_t * page)`, предназначенная для выравнивания кнопок относительно таблицы. Входным параметром является `lv_obj_t * page` – указатель на страницу-родителя. Выходные значения для этой функции отсутствуют.

Была реализована статическая функция `static void load_recent_journal_records()`, предназначенная для загрузки в таблицу самых свежих записей выбранного журнала. Для удобства просмотра журналов первыми показываются самые новые записи. Входные параметры для этой функции отсутствуют. Выходные значения для этой функции отсутствуют. Графическое изображение алгоритма выведения на экран самых свежих записей представлено в приложении К на рисунке К.3.

Была реализована статическая функция `static void load_next_journal_records(lv_obj_t * screen, lv_obj_t * prev_screen)`, предназначенная для загрузки более старых записей выбранного журнала. Входными параметрами являются:

– `lv_obj_t * screen` – указатель на активный экран;

– `lv_obj_t * prev_screen` – указатель на предыдущий экран.

Выходные значения для этой функции отсутствуют. Графическое изображение алгоритма выведения на экран следующих записей представлено в приложении К на рисунке К.4.

Была реализована статическая функция `static void load_previous_journal_records(lv_obj_t * screen, lv_obj_t * next_screen)`, предназначенная для загрузки более новых записей выбранного журнала. Входными параметрами являются:

– `lv_obj_t * screen` – указатель на активный экран;

– `lv_obj_t * next_screen` – указатель на следующий экран.

Выходные значения для этой функции отсутствуют. Графическое изображение алгоритма выведения на экран предыдущих записей представлено в приложении К на рисунке К.5.

Была реализована статическая функция `static void change_journal_screen(lv_obj_t * screen, read_direction_t`

direction), предназначенная для смены активного экрана. Входными параметрами являются:

– lv_obj_t * screen – указатель на активный экран;

– read_direction_t direction – направление чтения журнала. Возможные значения представлены в перечислении read_direction_t.

Выходные значения для этой функции отсутствуют. Графическое изображение алгоритма смены активного экрана представлено в приложении К на рисунке К.6.

Была реализована статическая функция static void close_journal_screen(), предназначенная для закрытия просмотрщика журналов и освобождения памяти. Входные параметры для этой функции отсутствуют. Выходные значения для этой функции отсутствуют. Графическое изображение алгоритма закрытия просмотрщика журналов представлено в приложении К на рисунке К.7.

Была реализована статическая функция обратного вызова static void event_next_btn_cb(lv_obj_t * btn, lv_event_t event), предназначенная для вызова функции смены активного экрана на экран с более старыми записями. Входными параметрами являются:

– lv_obj_t * btn – кнопка (объект), которая вызвала срабатывание функции обратного вызова;

– lv_event_t event – событие, которое вызвало срабатывание функции обратного вызова.

Была реализована статическая функция обратного вызова static void event_prev_btn_cb(lv_obj_t * btn, lv_event_t event), предназначенная для вызова функции смены активного экрана на экран с более свежими записями. Входными параметрами являются:

– lv_obj_t * btn – кнопка (объект), которая вызвала срабатывание функции обратного вызова;

– `lv_event_t event` – событие, которое вызвало срабатывание функции обратного вызова.

Была реализована статическая функция обратного вызова `static void next_records_task_cb(lv_task_t * t)`, предназначенная для вызова функции загрузки более старых записей журнала. Входными параметрами являются:

– `lv_obj_t * btn` – кнопка (объект), которая вызвала срабатывание функции обратного вызова;

– `lv_event_t event` – событие, которое вызвало срабатывание функции обратного вызова.

Была реализована статическая функция обратного вызова `static void previous_records_task_cb(lv_task_t * t)`, предназначенная для вызова функции загрузки более свежих записей журнала. Входными параметрами являются:

– `lv_obj_t * btn` – кнопка (объект), которая вызвала срабатывание функции обратного вызова;

– `lv_event_t event` – событие, которое вызвало срабатывание функции обратного вызова.

Была реализована статическая функция обратного вызова `static bool close_journal_cb(uint32_t key)`, предназначенная для вызова функции закрытия журнала. Входными параметрами являются:

– `lv_obj_t * btn` – кнопка (объект), которая вызвала срабатывание функции обратного вызова;

– `lv_event_t event` – событие, которое вызвало срабатывание функции обратного вызова.

Разработанные функции вывода на экран графического интерфейса и работы с ним представлены в приложении Л и приложении М.

4 Разработка тестов программного обеспечения системы управления специального мобильного терминала

Для проверки работоспособности и правильного функционирования разработанного ПО были разработаны тестовые значения. Тестирование выполнялось с целью проверки правильности работы ПО и выявления возможных ошибок. Тестирование проводилось в симуляции на ПК.

Было выполнено тестирование функции `uint32_t get_block_addr(Journal_type jtype, Seek_type type, uint32_t current_block_addr)`. Перед выполнением тестирования были очищены журналы, после чего было выполнено искусственное наполнение журнала на 1500 записей. Входные тестовые значения, ожидаемые и полученные выходные данные представлены в таблице 1.

Таблица 1 – Результат тестирования функции `uint32_t get_block_addr(Journal_type jtype, Seek_type type, uint32_t current_block_addr)`

Аргумент	Входные данные	Ожидаемое выходное значение	Реальное выходное значение
Расположение блоков и порядок совпадают (1→ 2)			
FIRST			
jtype	JSECURITY	0xab0000	0xab0000
type	FIRST		
current_block_addr	0		
LAST			
jtype	JSECURITY	0xac0000	0xac0000
type	LAST		
current_block_addr	0		
NEXT			
jtype	JSECURITY	0xac0000	0xac0000
type	NEXT		
current_block_addr	0xab0000		

Продолжение таблицы 1.

Аргумент	Входные данные	Ожидаемое выходное значение	Реальное выходное значение
PREV			
jtype	JSECURITY	0xab0000	0xab0000
type	PREV		
current_block_addr	0xac0000		
Расположение блоков и порядок не совпадают (3 → 1 → 2)			
FIRST			
jtype	JSECURITY	0xac0000	0xac0000
type	FIRST		
current_block_addr	0		
LAST			
jtype	JSECURITY	0xab0000	0xab0000
type	LAST		
current_block_addr	0		
NEXT			
jtype	JSECURITY	0xab0000	0xab0000
type	NEXT		
current_block_addr	0xad0000		
PREV			
jtype	JSECURITY	0xad0000	0xad0000
type	PREV		
current_block_addr	0xab0000		

Экран тестового режима представлен на рисунке 5.

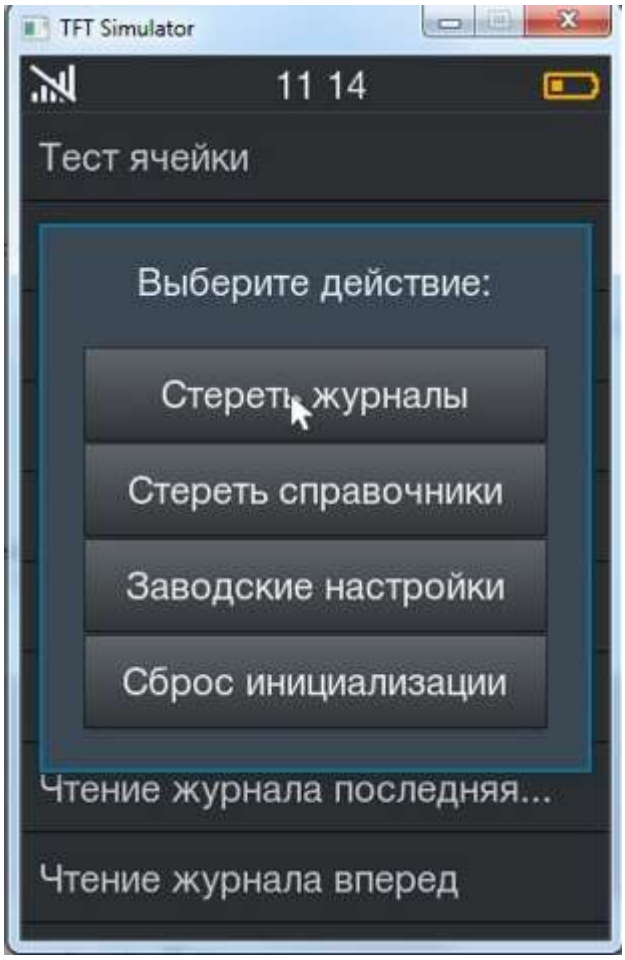

Рисунок 5 – Тестовый экран

Было выполнено тестирование функции `uint32_t get_next_free_space(Journal_type jtype, uint32_t start_address, uint16_t need_space)`. Входные тестовые значения, ожидаемые и полученные выходные данные представлены в таблице 2.

Таблица 2 – Результат тестирования функции `uint32_t get_next_free_space(Journal_type jtype, uint32_t start_address, uint16_t need_space)`

Аргумент	Входные данные	Ожидаемое выходное значение	Реальное выходное значение
jtype	JSECURITY	0xad3d12	0xad3d12
start_address	0x0		
need_space	78		

Было выполнено тестирование функции `JRetcode add_record(Journal_type jtype, uint16_t event_code,`

uint8_t *data, uint16_t datalen). Входные тестовые значения, ожидаемые и полученные выходные данные представлены в таблице 3.

Таблица 3 – Результат тестирования функции Retcode add_record(Journal_type jtype, uint16_t event_code, uint8_t *data, uint16_t datalen)

Аргумент	Входные данные	Ожидаемое выходное значение	Реальное выходное значение
jtype	JSECURITY	JSUCCESS	JSUCCESS
event_code	255		
data	«Включение аппаратуры»		
datalen	0x40		

По результатам выполнения функций uint32_t get_next_free_space(Journal_type jtype, uint32_t start_address, uint16_t need_space) и JRetcode add_record(Journal_type jtype, uint16_t event_code, uint8_t *data, uint16_t datalen) была выполнена проверка добавления записи. Результат представлен на рисунке 6. Красным цветом выделена добавленная запись.

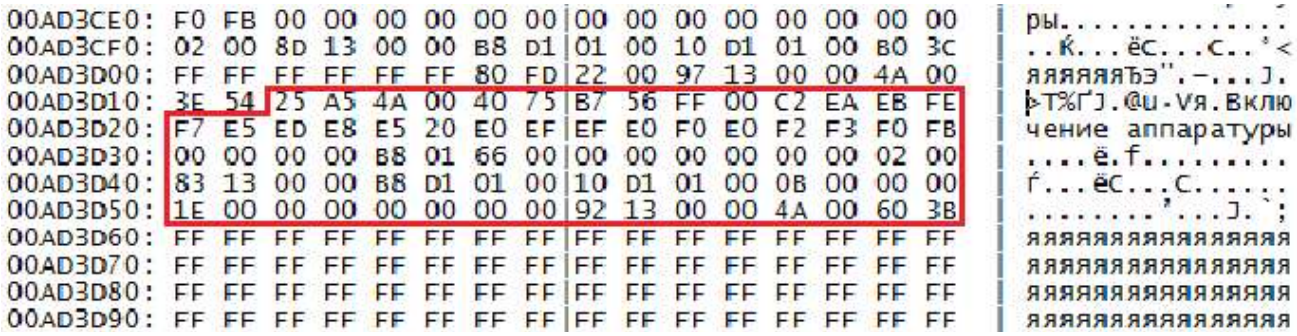

Рисунок 6 – Содержимое файла spiflash.bin

Было выполнено тестирование функции uint32_t find_last_record_in_block(uint32_t start_addr). Входные тестовые значения, ожидаемые и полученные выходные данные представлены в таблице 4.

Таблица 4 – Результат тестирования функции uint32_t find_last_record_in_block(uint32_t start_addr)

Аргумент	Входные данные	Ожидаемое выходное значение	Реальное выходное значение
start_addr	0xad0004	0xad3d60	0xad3d60

Было выполнено тестирование функции `JRetcode get_record(Journal_type jtype, Seek_type stype, uint32_t prev_addr, Journal_record *record)`.

Была выполнена проверка использования видеопамяти графическим интерфейсом с помощью встроенного инструмента мониторинга памяти. Результат представлен на рисунке 7.

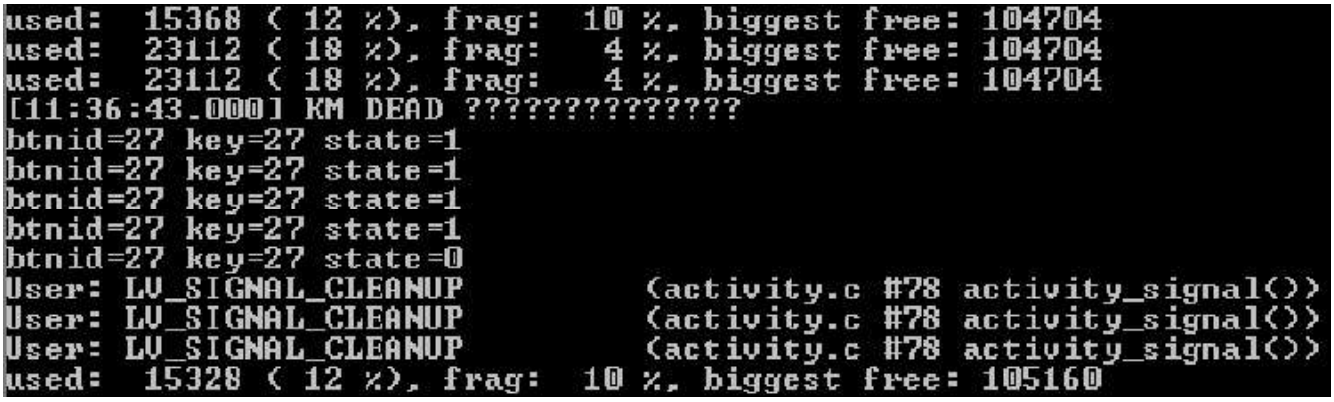

Рисунок 7 – Монитор ресурсов видеопамяти в окне консоли

Из содержимого монитора ресурсов видеопамяти видно, что количество свободной видеопамяти до открытия инструмента просмотра журналов равно количеству свободной видеопамяти после закрытия инструмента просмотра журналов.

Было выполнено открытие пустого журнала. Результат представлен на рисунке 8. Кнопки на экране отсутствуют, что соответствует ожиданиям, так как отсутствуют записи для чтения вперёд и назад.

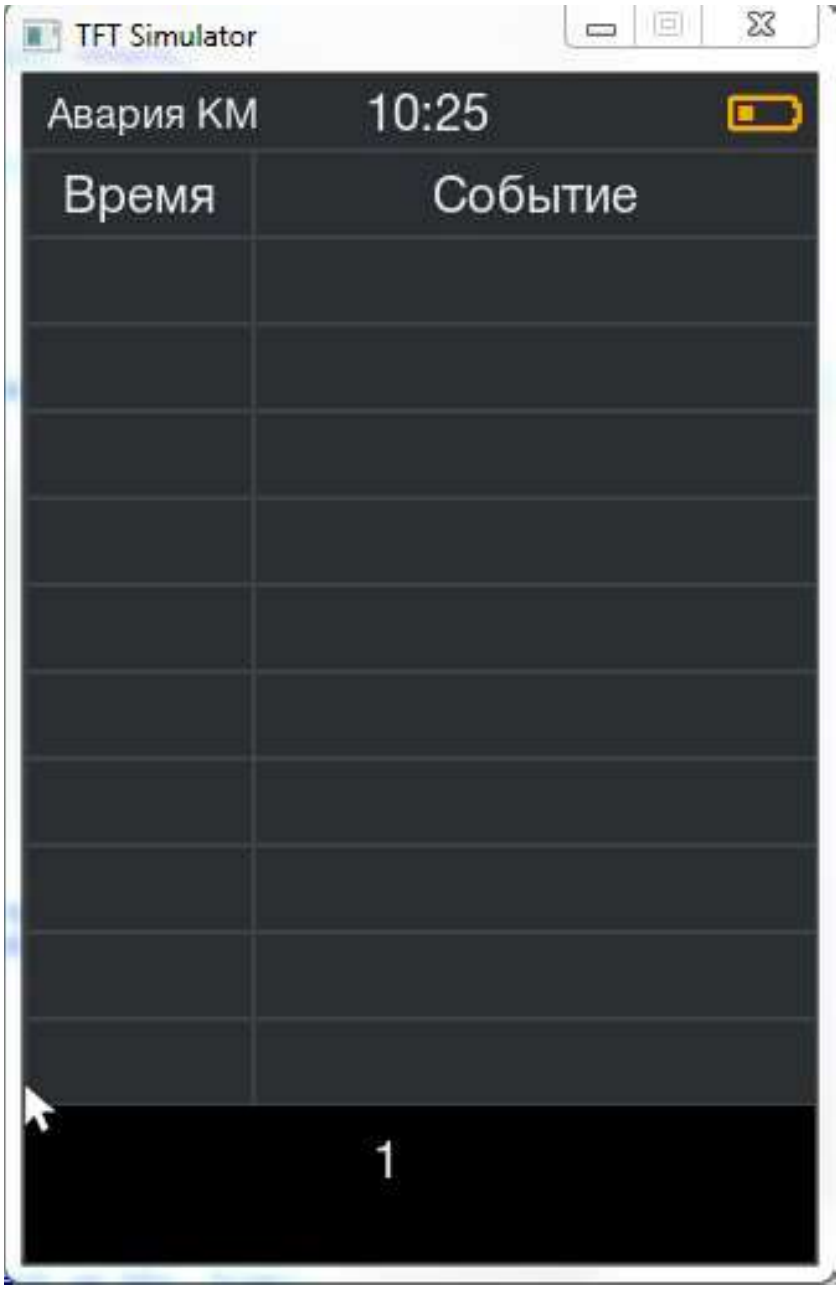

Рисунок 8 – Интерфейс просмотра журналов пустого журнала

Было выполнено наполнение журнала тестовыми значениями. Было выполнено открытие журнала. Первая страница представлена на рисунке 9. Теперь кнопка «След.» присутствует, так как количество записей в журнале больше, чем возможно отобразить на одной странице интерфейса просмотра журнала.

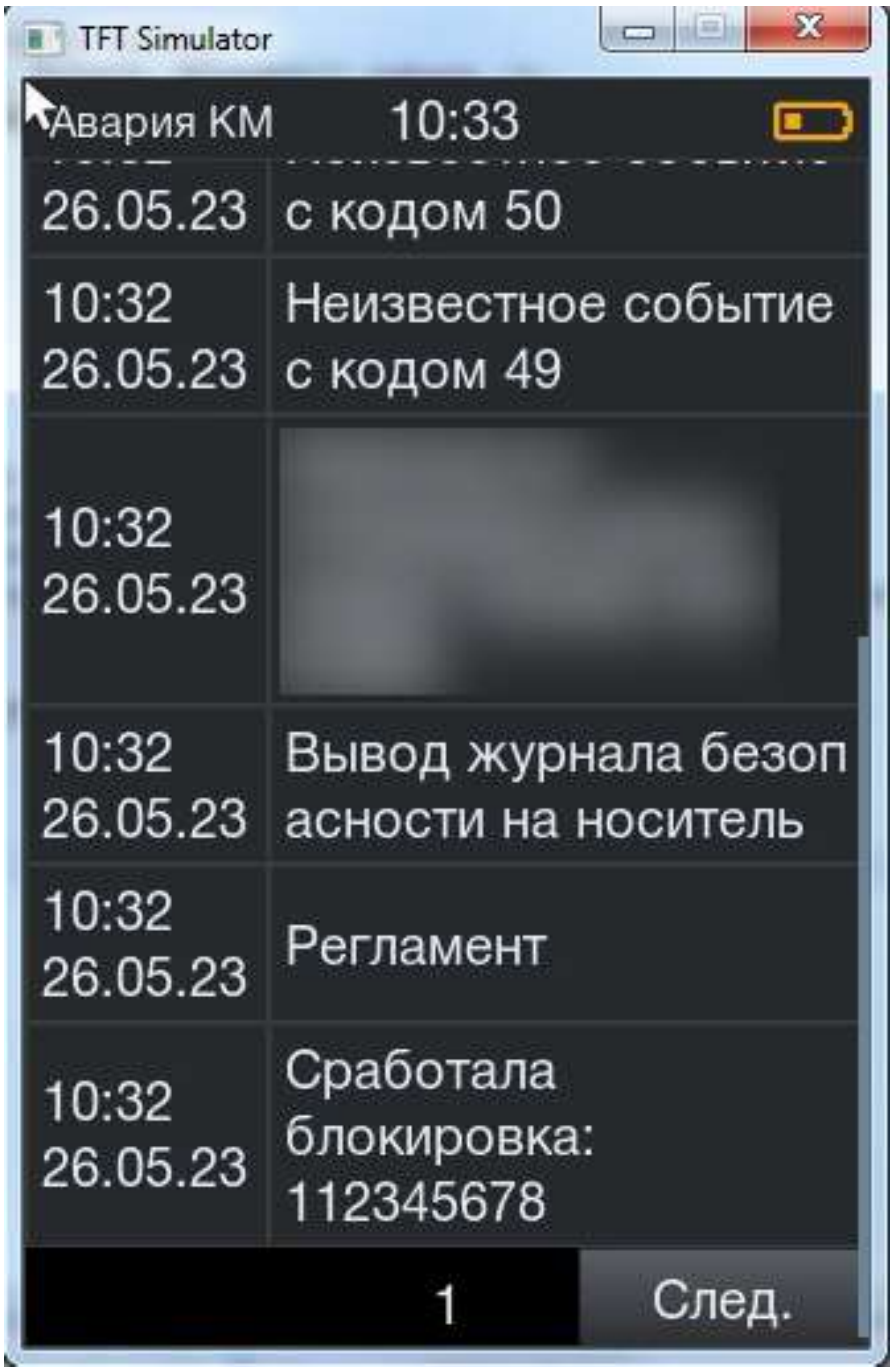

Рисунок 9 – Интерфейс просмотра журналов с наполнением на первой странице

Последняя отрисованная запись на последней странице обладает адресом 0xab0614. Был выполнен поиск последней записи журнала в spiflash.bin. Результат представлен на рисунке 10.

```
00AB0610: 3C 00 05 A2 25 A5 3C 00|3C 58 BA 56 FF 00 52 65  | <..ў%Ґ<.<XєVя.Re
00AB0620: 63 6F 72 64 20 6E 75 6D|62 65 72 20 32 37 00 FC  | cord number 27.ь
00AB0630: 22 00 00 00 00 00 3A 72|45 00 00 00 00 00 78 4E  | ".....:rE.....xN
00AB0640: 66 00 00 00 00 00 00 09|4B 00 00 00 00 00 C8 02  | f.......K.....И.
00AB0650: 3C 00 4E 56 FF FF FF FF|FF FF FF FF FF FF FF FF  | <.NVяяяяяяяяяяяя
```

Рисунок 10 – Последняя запись журнала в spiflash.bin

На рисунке 8 видно, что по адресу 0xab0614 расположено начало последовательности, обозначающей начало непустой записи. Это последняя последовательность начала непустой записи, находящаяся перед неиспользуемой областью блока. Из этого следует, что запись, расположенная по адресу 0xab0614 действительно является последней.

Был выполнен просмотр второй страницы интерфейса просмотра журналов. Результат представлен на рисунке 11. Было выполнено отображение кнопки «Пред.».

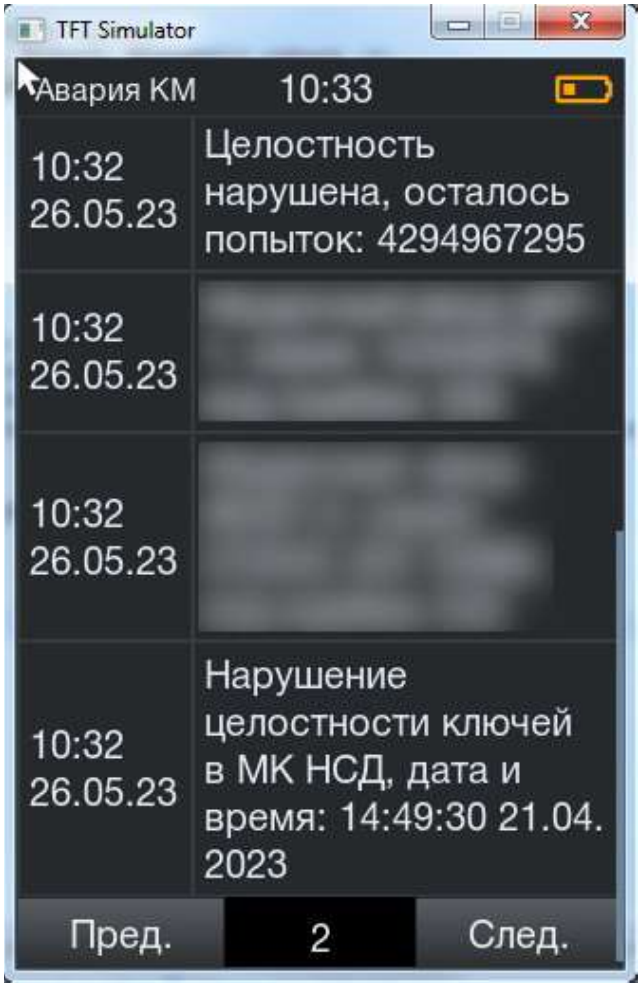

Рисунок 11 – Интерфейс просмотра журналов с наполнением на второй странице

Был выполнен просмотр последней (третьей) страницы интерфейса просмотра журналов. Результат представлен на рисунке 12. Было прекращено отображение кнопки «След.».

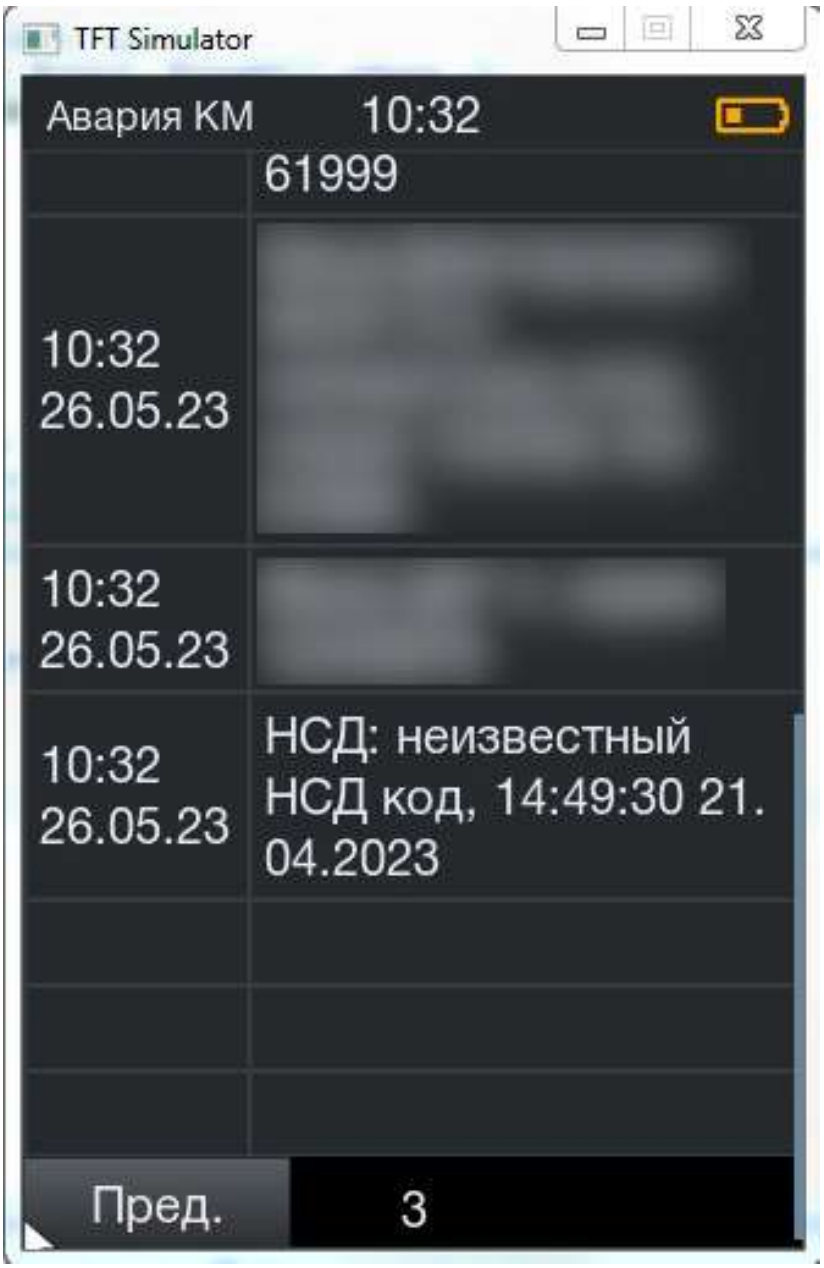

Рисунок 12 – Интерфейс просмотра журналов с наполнением на последней странице

На рисунках 9, 11, 12 видно, что графический интерфейс просмотра журналов работает правильно, время и событие отображается корректно.

Последняя обработанная запись в интерфейсе просмотра журналов обладает адресом 0xab0004, что соответствует адресу первой сделанной записи.

Была выполнена проверка последовательного отображения следующих записей на разных страницах. Адреса последней записи на первой странице и первой записи на второй странице представлены в таблице 5.

Таблица 5 – Адреса записей, расположенных на границе разных страниц при чтении вперёд

последняя запись на первой странице	Длина атрибута расширения	первая запись на второй странице	Длина атрибута расширения	Разница между значениями адресов
0xab019c	10	0xab0184	10	0x18

Был выполнен поиск этих записей в файле spiflash.bin. Результат представлен на рисунке 13.

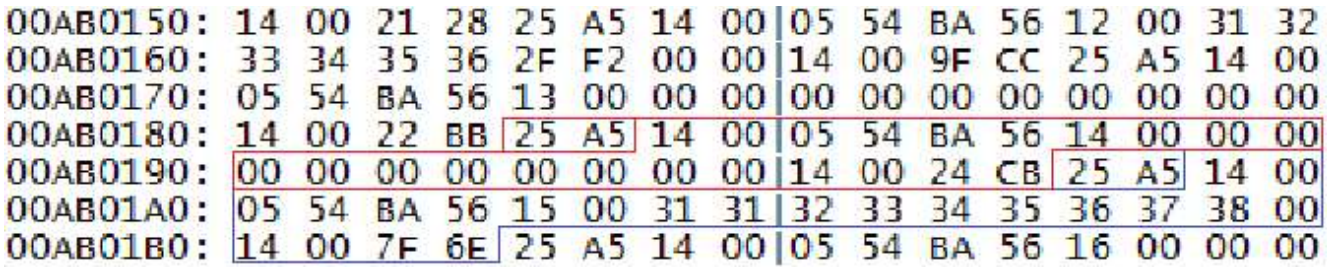

Рисунок 13 – Записи, расположенные по адресам 0xab019c и 0xab0184 в файле spiflash.bin

Синим цветом выделена запись, расположенная по адресу 0xab019c. Красным цветом выделена запись, расположенная по адресу 0xab0184. На рисунке 13 видно, что между этими двумя записями отсутствуют другие последовательности начала непустой записи, длины записей равны суммам длин хранимых атрибутов. Из этого можно сделать вывод, что чтение и отображение записей, расположенных на границе разных страниц, при чтении вперёд происходит корректно.

Была выполнена проверка отображения предыдущих записей на разных страницах. Адреса первой записи на последней странице и последней записи на предпоследней странице представлены в таблице 6.

Таблица 6 – Адреса записей, расположенных на границе разных страниц при чтении назад

первая запись на последней странице	последняя запись на предпоследней странице	Разница между значениями адресов
0xab04ac	0xab04c4	0x18

Был выполнен поиск этих записей в файле spiflash.bin. Результат представлен на рисунке 14.

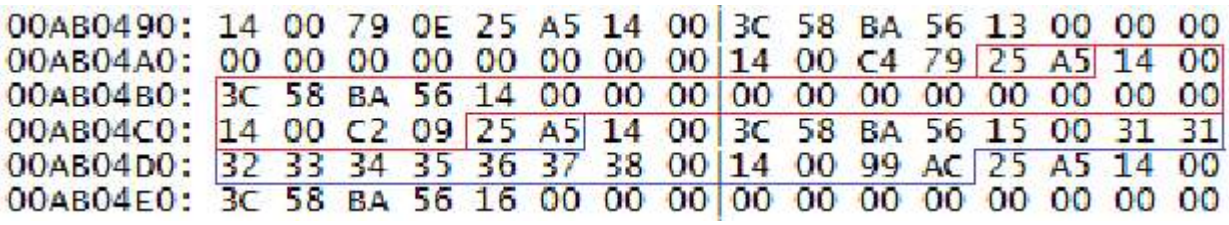

Рисунок 14 – Записи, расположенные по адресам 0xab04ac и 0xab04c4 в файле spiflash.bin

Красным цветом выделена запись, расположенная по адресу 0xab04ac. Синим цветом выделена запись, расположенная по адресу 0xab04c4. На рисунке 14 видно, что между этими двумя записями отсутствуют другие последовательности начала непустой записи, длины записей равны суммам длин хранимых атрибутов. Из этого можно сделать вывод, что чтение и отображение записей, расположенных на границе разных страниц, при чтении назад происходит корректно.

5 Вредные психофизиологические факторы, влияющие на сердечно-сосудистую систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

Специалист по защите информации работает в условиях постоянного стресса и напряжения. Его работа связана с защитой конфиденциальной информации, что требует от него высокой концентрации внимания, скрупулезности и ответственности. Такая работа может оказывать негативное воздействие на здоровье сердечно-сосудистой системы.

Исследования свидетельствуют, что люди с профессиями, требующими высокого напряжения, чаще страдают от высокого давления и левой вентикулярной гипертрофии, чем люди менее напряженных профессий[8]. В 2003 году академиком Е.И. Чазовым были опубликованы данные, доказавшие, что к росту смертности от сердечно-сосудистых заболеваний не имеют прямого отношения ни ожирение, ни курение, ни повышение холестерина и что причину сердечно-сосудистых заболеваний надо искать в последствиях стрессов.

Долгосрочные последствия стресса обычно приводят к широкому спектру сердечно-сосудистых проблем. Под действием гормонов стресса (адреналина, норадреналина и кортизола) кровеносные сосуды сужаются, чтобы направлять больше кислорода и энергии к мышцам. Но это также повышает кровяное давление. Кроме того, продолжающийся острый стресс может способствовать воспалению в коронарных артериях и сосудах.

В условиях стресса происходят изменения эндокринного статуса, метаболизма, а также развитие острофазной реакции. Биологический смысл этих явлений на ранних стадиях развития стрессовой реакции направлен на поддержание функций жизненоважных органов и систем путем повышения доступности энергоресурсов, регуляции регионального кровотока, активации ферментов клеточного метаболизма и других факторов биологической адаптации. Ключевым механизмом в реализации этих событий является

внутриклеточное накопление кальция. В кардиомиоцитах (клетках сердечной мышцы) кальций накапливается уже в первые минуты развития стрессовой реакции. С действием этого фактора связаны такие проявления со стороны сердечно-сосудистой системы, как тахикардия и гипертензия, усиливается сокращение мышечных волокон, синтез белков, а соответственно – обновление и рост клеточных структур. Однако избыточное накопление кальция может вызывать клеточную гибель. Применительно к сердцу избыток кальция вызывает кардиотоксический эффект. Механизмы повреждения сердца стрессовой реакцией идентичны ишемическим повреждениям миокарда.

Также стресс и тревожное состояние нередко провоцируют появление вредных привычек, которые негативно влияют на состояние сердечно-сосудистой системы и на организм человека вцелом. Алкоголь и курение нарушают тонус сосудов, увеличивает число сердечных сокращений. Это может привести к развитию многих серьезных заболеваний, например, атеросклероза, эндартериита, гипертонии, ишемии. В некоторых случаях может развиться инфаркт или инсульт. У человека при употреблении спиртного повышается пульс, нарушается метаболизм и питание мышцы сердца. Табак и этанол сгущают кровь, в результате этого сердце работает в усиленном режиме.

Работа специалиста по защите информации связана с длительной работой с ПК, что предполагает длительное нахождение в сидячем состоянии и пониженную физическую активность. Для того чтобы поддерживать сердечную мышцу и кровеносные сосуды в тонусе, необходимо соблюдать хотя бы минимальный режим активности. Гиподинамия становится предпосылкой отклонений в работе сердца и кровеносных сосудов. При малоподвижном образе слабеет сердечная мышца, ослабевают сосуды, снижается частота сердечных сокращений. Функциональные изменения сердечно-сосудистой системы проявляются одышкой и резким учащением сердечных сокращений при быстрой ходьбе, возникают боли в области сердца при физической работе,

отрицательных эмоциях. Длительная гиподинамия способствует развитию атеросклероза, гипертонической болезни и варикозного расширения вен.

Также негативным влиянием на здоровье специалиста по информационной безопасности влияет длительное пребывание в закрытом пространстве. Концентрация углекислого газа повышается за счёт пребывания в офисном помещении большого количества человек. При концентрации углекислого газа выше 0,1% возникает ощущение духоты: общий дискомфорт, слабость, головная боль, снижение концентрации внимания. Также увеличивается частота и глубина дыхания, происходит сужение бронхов, а при концентрации выше 15% - спазм голосовой щели. При длительном нахождении в помещениях с избыточным количеством углекислого газа происходят изменения в кровеносной, центральной нервной, дыхательной системах, при умственной деятельности нарушается, восприятие, оперативная память, распределение внимания.

Существуют различные способы борьбы с повышенным содержанием CO_2 в помещении, включая проветривание, использование специальных систем вентиляции и установку растений, которые способны поглощать CO_2. Также важно следить за уровнем физической активности и правильно организовывать рабочее место, чтобы минимизировать воздействие негативных факторов на здоровье.

Профилактикой сердечно-сосудистых заболеваний являются физические упражнения. Оптимальная физическая нагрузка делает мышцы сильным, укрепляет кости (профилактика остеопороза), позволяет сохранить подвижность суставов, улучшает состояние их хрящей, и даже способствует снижению уровня холестерина и сахара крови.

У тренированных людей значительно улучшается сократительная способность миокарда, усиливается центральное и периферическое кровообращение, повышается коэффициент полезного действия, уменьшается частота сердечных сокращений не только в состоянии покоя, но и при любых

нагрузках, вплоть до максимальной (это состояние называется тренировочной брадикардией), повышается систолический, или ударный, объем крови. Благодаря увеличению ударного объема крови сердечно-сосудистая система тренированного человека гораздо легче, чем нетренированного, справляется с возрастающими физическими нагрузками, полностью обеспечивая кровью все мышцы тела, принимающие участие в нагрузке с большим напряжением.

Давно установлено, что люди, занимающиеся систематическим физическим трудом и физкультурой, имеют более широкие сосуды сердца. Коронарный кровоток у них при необходимости может быть увеличен в значительно большей степени, чем у физически неактивных людей. Но, что самое важное, благодаря экономной работе сердца тренированные люди на одну и ту же работу затрачивают меньше крови для работы сердца, чем нетренированные.

Таким образом, систематическая физическая тренировка оказывает чрезвычайно благотворное влияние на сердечно-сосудистую систему человека и в целом на весь его организм.

Ходьба — ценное и важное средство лечебной и оздоровительной физической культуры. Это естественная и привычная форма мышечной деятельности. Во время ходьбы в работу включаются многие мышечные группы туловища, ног и рук. При самостоятельных занятиях ходьба — наиболее доступная, легко дозируемая форма занятий.

Одним из эффективнейших способов профилактики сердечно-сосудистых заболеваний является физические занятия, польза которых несомненна: во-первых, при активных занятиях, особенно на свежем воздухе, клетки и ткани организма насыщаются кислородом, и, во-вторых, во время физической активности происходит усиление кровообращения и укрепление сердечной мышцы.

Физические нагрузки, необходимые для профилактики, выбираются в зависимости от возраста, функционального состояния человека, а также от того,

имеются ли у него уже какие-либо заболевания сердечно-сосудистой системы. При первичной профилактике сердечно-сосудистых заболеваний (когда человек здоров) выбирать вид и интенсивность занятий можно самостоятельно. При вторичной профилактике (когда человек болен и стремится не допустить прогрессии заболевания) режим активности должен планироваться после обследования и в согласовании с рекомендациями врача.

Предпочтительными являются аэробные занятия, при которых увеличивается частота сердечных сокращений – бег, ходьба, в том числе скандинавская, прогулки на лыжах, езда на велосипеде, плавание.

В ходе медицинских исследований было обнаружено, что минимальная динамическая нагрузка составляет 3 раза в неделю по 30 минут в комфортном темпе.

Помимо аэробных нагрузок, полезной для организма является гимнастика, которая способствует укреплению сердечных мышц и стенок сосудов, нормализует давление.

Например, можно выполнять следующую систему упражнений:

– вращение сжатыми в кулак кистями, а также ступнями, выставленными на носок (20 раз в одну и в другую стороны);

– подняться на носочки и ходить с очень высоким поднятием коленей;

– ноги находятся на уровне ширины плеч. Руки поднимаются кверху, сцепив при этом руки в замок. При наклоне корпуса вправо – правая нога отводится вправо. Наклон стремиться выполнять как можно глубже. Повторить те же движения в левую сторону (8-9 раз).

– не поднимаясь со стула, прямые руки вытянуть в стороны, одновременно поднять согнутую в колене ногу к груди. Руки при этом сводят перед собой, помогая придерживать поднятую ногу. Выполнять поочередно правой и левой ногой (6-8 раз);

– выпады вперед поочередно каждой ногой (10-20 раз);

– лежа на спине, нужно поднять согнутые в коленях ноги под прямым углом и имитировать движения езды на велосипеде.

Заключение

В результате выполнения задания был разработан модуль системы управления специальным мобильным терминалом, выполнены отладка и тестирование разработанного ПО.

В первом разделе выпускной квалификационной работы были рассмотрены следующие УБИ: угроза активации встроенных в микроконтроллер аппаратных закладок, угроза чтения информации ограниченного доступа из оперативной памяти. Были определены меры, направленные на нейтрализацию этих угроз.

Во втором разделе выпускной квалификационной работы были разработаны алгоритмы реализации мер, направленных на нейтрализацию этих угроз.

В третьем разделе выпускной квалификационной работы была выполнена разработка программного обеспечения системы управления СМТ.

В четвертом разделе выпускной квалификационной работы было проведено тестирование и отладка разработанного ПО.

В пятом разделе выпускной квалификационной работы были определены вредные психофизиологические факторы, влияющие на сердечно-сосудистую систему специалиста по защите информации, а также описаны методы и средства физической культуры, снижающие их воздействие.

Таким образом, поставленные задачи выполнены в полном объеме.

Список использованных источников

1 ГОСТ Р50922-2006 «Защита информации. Основные термины и определения» от 01.02.2008 [Электронный ресурс] — https://docs.cntd.ru (дата обращения: 13.04.2023)

2 Земор, Ж. Курс криптографии / Ж. Земор. - М.: Регулярная и хаотическая динамика, Институт компьютерных исследований, 2006. - 256 с.

3 Брайан Керниган, Деннис Ритчи. Язык программирования С. — М.: Вильямс, 2015. — 304 с. — ISBN 978-5-8459-1975-5.

4 STMicroelectronics, STM32CubeIDE user guide, версия UM2609, 2023

5 Брайан О'салливан, Mercurial: Полное руководство, 2018

6 Infineon Technologies, Infineon S25FL256S V_SPI_Flash_Memory-DataSheet, версия 18.00, 2022

7 SquareLine Studio, Light and Versatile Graphics Library, версия 9.0, 2023

8 Кацилович Ю. В. «Стресс. Влияние на сердечно-сосудистую систему» [Электронный ресурс] — https://www.10gkb.by/informatsiya/stati/stress-vliyanie-na-serdechno-sosudistuyu-sistemu (дата обращения 10.05.2023)

Приложение А

(обязательное)

Графические изображения алгоритмов работы с разреженными строками

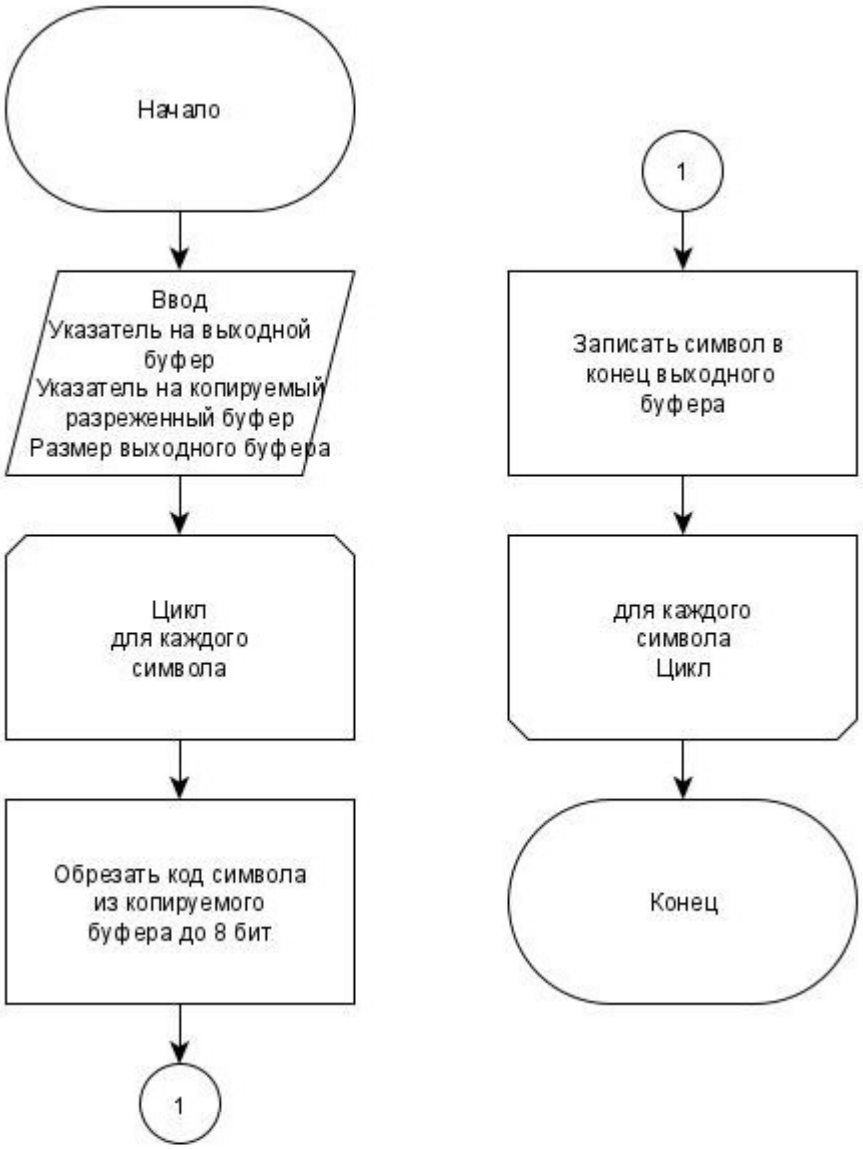

Рисунок А.1. – Графическое изображение алгоритма копирования разреженного буфера в обычный

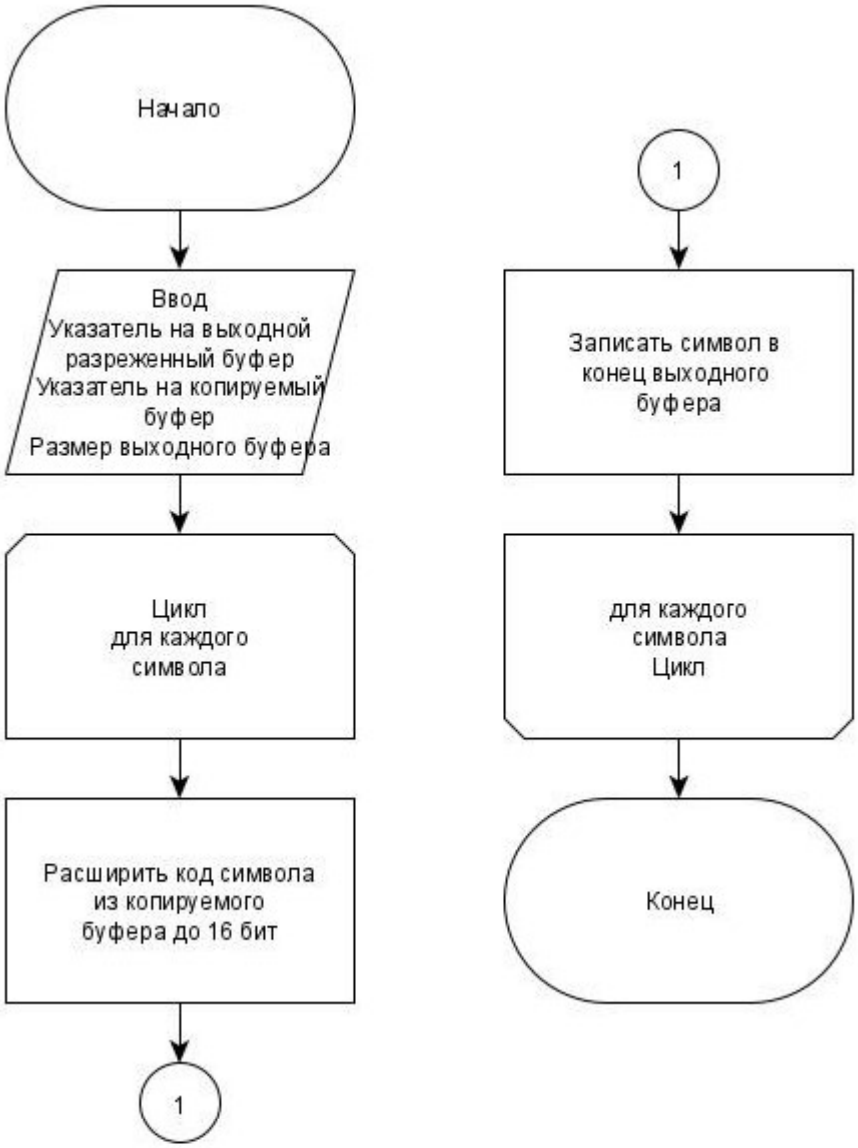

Рисунок А.2 – Графическое изображение алгоритма копирования обычного буфера в разреженный

Приложение Б

(обязательное)

Содержимое файла sstring.h

```
#ifndef UU_CLR_SSTRING_H
#define UU_CLR_SSTRING_H

#include <stddef.h>

/* Copy SRC to DEST.  */
char* sstrcpy(char* dest, const char* src);

/* Copy no more than N characters of SRC to DEST.  */
char* sstrncpy(char* dest, const char* src, size_t n);

/* Append SRC onto DEST.  */
char* sstrcat(char* dest, const char* src);

/* Append no more than N characters from SRC onto DEST.  */
char* sstrncat(char *dest, const char* src, size_t n);

/* Find the first occurrence of C in S.  */
char* sstrchr(const char* src, char c);

/* Find the last occurrence of C in S.  */
char* sstrchr_rev(const char* src, char c);

/* Return the length of S.  */
size_t sstrlen(const char* src);

/* Find the length of S, but scan at most N characters.  */
size_t sstrnlen(const char* src, size_t n);

/*
```

```
 * Convert a sparse string to a long integer.
 *
 * Ignores `locale' stuff. Assumes that the upper and lower case
 * alphabets and digits are each contiguous.
 */
long sstrtol(const char *nptr, char **endptr, register int base);

/* Compare S1 and S2.  */
int sstrcmp(const char *s1, const char *s2);

/* Compare N characters of S1 and S2.
 * S1 and S2 are sparse strings.  */
int sstrncmp(const char *s1, const char *s2, size_t n);

/* Compare N characters of S1 and S2.
 * S1 is sparse, S2 is simple string.  */
int sstrncmp2(const char *s1, const char *s2, size_t n);

/* Substring search. Case-sensitive.
 * string is sparse, substring is sparse.   */
const char* sstrstr(const char* sstring, const char* ssubstring);

/* Substring search. Case-sensitive.
 * string is sparse, substring is simple string.   */
const char* sstrstr2(const char* sstring, const char* substring);

/* Substring search. Case-sensitive.
 * string is simple string, substring is sparse.   */
const char* sstrstr3(const char* string, const char* ssubstring);

uint8_t sdatacopy(uint16_t* dest, uint16_t* src, size_t maxlen);
void sstrrej(char* src, char ch);
```

```
void smem2memcpy(void *dst,const void *src,uint16_t size_in_words);
void memcpysmem2(void *dst,const void *src,uint16_t size_in_bytes);

void iptostring(uint32_t ip,char *ipstring);
uint8_t stringtoip(const char *ipstring,uint32_t *ip);

#endif //UU_CLR_SSTRING_H
```

Приложение В

(обязательное)

Содержимое файла sstring.c

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <stdint-gcc.h>
#include "lvgl/lib/src/lv_misc/lv_log.h"
#include "sstring.h"
#include "ctype.h"

//Копирование разреженной строки в dest из src
char* sstrcpy(char* dest, const char* src)
{
	const size_t src_len = sstrlen(src);
	memcpy(dest, src, src_len);
	memset(dest + src_len, 0, 2);
	return dest;
}

//Копирование разреженной строки в dest из src с ограничением по
результирующей длине dest не более n
char* sstrncpy(char* dest, const char* src, size_t n)
{
	const size_t src_len = sstrnlen(src, n);
	memcpy(dest, src, src_len);

	if(n-src_len>1) memset(dest + src_len, 0, 2);

	if(dest[src_len]!=0 || dest[src_len+1]!=0) LV_LOG_WARN("strncpy:
non zero-ended string");
```

```
	return dest;
}

//Добавление разреженной строки dest разреженной строкой src
char* sstrcat(char* dest, const char* src)
{
	sstrcpy(dest + sstrlen(dest), src);
	return dest;
}

//Добавление разреженной строки dest разреженной строкой src с
ограничением по результирующей длине dest не более n
char* sstrncat(char *dest, const char* src, size_t n)
{
	int dest_len = sstrlen(dest);
	int src_len = sstrlen(src);

	if(n-src_len-dest_len-2 < 0) src_len=n-dest_len-2;
	if(src_len>0)
	{
		memcpy(dest + dest_len, src, src_len);
		memset(dest + dest_len + src_len, 0, 2);
	}

	return dest;
}

char* sstrchr(const char* src, char c)
{
	do
	{
		if (*src == c) return (char*)src;
		src++;
```

```
	} while (*++src);
	return (0);
}

char* sstrchr_rev(const char* src, char c)
{
	char *rtnval = 0;

	do {
		if (*src == c) rtnval = (char*) src;
		src++;
	} while (*++src);
	return (rtnval);
}

//Возвращает длину разреженной строки в байтах
size_t sstrlen(const char* src)
{
	const char *s;
	for (s = src; *s; s += 2);
	return (size_t)(s - src);
}

//Возвращает длину разреженной строки в байтах, но не более n
size_t sstrnlen(const char* src, size_t n)
{
	const char* s;
	for (s = src; *s && n--; s += 2);
	return (size_t)(s - src);
}

long sstrtol(const char *nptr, char **endptr, register int base)
{
```

```
register const uint16_t *s = (const uint16_t *) nptr;
register unsigned long acc;
register int c;
register unsigned long cutoff;
register int neg = 0, any, cutlim;

/*
 * Skip white space and pick up leading +/- sign if any.
 * If base is 0, allow 0x for hex and 0 for octal, else
 * assume decimal; if base is already 16, allow 0x.
 */
do {
     c = *s++;
} while (isspace(c));
if (c == '-') {
     neg = 1;
     c = *s++;
} else if (c == '+')
     c = *s++;
if ((base == 0 || base == 16) &&
    c == '0' && (*s == 'x' || *s == 'X')) {
     c = s[1];
     s += 2;
     base = 16;
}
if (base == 0)
     base = c == '0' ? 8 : 10;

cutoff = neg ? -(unsigned long)LONG_MIN : LONG_MAX;
cutlim = cutoff % (unsigned long)base;
cutoff /= (unsigned long)base;
for (acc = 0, any = 0;; c = *s++) {
     if (isdigit(c))
```

```
                c -= '0';
            else if (isalpha(c))
                c -= isupper(c) ? 'A' - 10 : 'a' - 10;
            else
                break;
            if (c >= base)
                break;
            if (any < 0 || acc > cutoff || (acc == cutoff && c >
cutlim))
                any = -1;
            else {
                any = 1;
                acc *= base;
                acc += c;
            }
        }
        if (any < 0) {
            acc = neg ? LONG_MIN : LONG_MAX;
        } else if (neg)
            acc = -acc;
        if (endptr != 0)
        {
            if(any) *endptr = (char *)(s - 1);
            else *endptr = (char *) nptr;
        }
        return (acc);
}

int sstrcmp(const char *s1, const char *s2) {
        uint16_t *ss1, *ss2;
        ss1 = (uint16_t *) s1;
        ss2 = (uint16_t *) s2;
```

```
      while (*ss1 != '\0' && *ss2 != '\0' && *ss1 == *ss2) {
            ss1++;
            ss2++;
      }
      return *ss1 - *ss2;
}

/* Compare N characters of S1 and S2.
 * S1 and S2 are sparse strings.  */
// size in "words" (bytes / 2)
int sstrncmp(const char *s1, const char *s2, size_t n) {
      uint16_t *ss1, *ss2;
      ss1 = (uint16_t *) s1;
      ss2 = (uint16_t *) s2;

      while (n != 0 && *ss1 != '\0' && *ss2 != '\0'  && *ss1 == *ss2) {
            ss1++;
            ss2++;
            n--;
      }
      return n == 0 ? 0 : (*ss1 - *ss2);
}

/* Compare N characters of S1 and S2.
 * S1 is sparse, S2 is simple string.  */
// size in "words" (bytes / 2)
int sstrncmp2(const char *s1, const char *s2, size_t n) {
      uint16_t* ss1 = (uint16_t*) s1;
      uint8_t* ss2 = (uint8_t*) s2;

      while (n != 0 && *ss1 != '\0' && *ss2 != '\0'  && *ss1 == *ss2) {
            ss1++;
            ss2++;
```

```
            n--;
        }
        return n == 0 ? 0 : (*ss1 - *ss2);
}

const char* sstrstr(const char* sstring, const char* subsstring)
{
    const char *a, *b;

    b = subsstring;

    if (*b == 0) return sstring;

    for ( ; *sstring != 0; sstring += 2)
    {
        if (*sstring != *b) continue;

        a = sstring;
        while (1)
        {
            if (*b == 0) return sstring;
            if (*a++ != *b++) break;
            a++;
            b++;
        }
        b = subsstring;
    }
    return NULL;
}

const char* sstrstr2(const char* sstring, const char* substring)
{
    const char *a, *b;
```

```
    b = substring;
    if (*b == 0) return sstring;

    for ( ; *sstring != 0; sstring += 2)
    {
        if (*sstring != *b) continue;

        a = sstring;
        while (1)
        {
            if (*b == 0) return sstring;
            if (*a++ != *b++) break;
            a++;
        }
        b = substring;
    }
    return NULL;
}

const char* sstrstr3(const char* string, const char* subsstring)
{
    const char *a, *b;

    b = subsstring;
    if (*b == 0) return string;

    for ( ; *string != 0; string++)
    {
        if (*string != *b) continue;

        a = string;
        while (1)
```

```
        {
            if (*b == 0) return string;
            if (*a++ != *b++) break;
            b++;
        }
        b = subsstring;
    }
    return NULL;
}

//копирование разреженной строки целиком в случае если ее длина не
превышает maxlen
uint8_t sdatacopy(uint16_t* dest, uint16_t* src, size_t maxlen)
{
      size_t len;
      uint8_t lenw;

      len=sstrlen((const char *)src)+2;
      lenw=len/sizeof(uint16_t);

      if(lenw>maxlen) return 0;
      memcpy(dest, src, len);
      return lenw;
}

//удаление символа из разреженной строки
void sstrrej(char* src, char ch)
{
      char *dst;
      for (dst = src; *src; src+=2)
      {
            if(*src!=ch)
            {
```

```
                *dst=*src;
                dst+=2;
            }
        }
        *dst=0;
}

//копирование разреженного буфера в обычный
void smem2memcpy(void *dst,const void *src,uint16_t size_in_words)
{
        uint16_t i;
        uint16_t *in = (uint16_t *) src;
        uint8_t *out = (uint8_t *) dst;

        for(i=0;i<size_in_words;i++)
        {
            *out=(uint8_t) *in;
            out++;
            in++;
        }
}

//копирование обычного буфера в разряженный
void memcpysmem2(void *dst,const void *src,uint16_t size_in_bytes)
{
        uint16_t i;
        uint8_t *in = (uint8_t *) src;
        uint16_t *out = (uint16_t *) dst;

        for(i=0;i<size_in_bytes;i++)
        {
            // *out=(uint16_t)(((uint16_t)(*in))|(((uint16_t)(0)<<8)));
            *out=(uint16_t) *in;
```

```
            out++;
            in++;
      }
}

void iptostring(uint32_t ip,char *ipstring)
{
      sprintf(ipstring,"%d.%d.%d.%d",(uint8_t)((ip>>24)&0xFF),(uint8_t)
((ip>>16)&0xFF),(uint8_t)((ip>>8)&0xFF),(uint8_t)((ip)&0xFF));
}

uint8_t stringtoip(const char *ipstring,uint32_t *ip)
{
      uint8_t i;
      uint32_t newip=0;
      long oct=0;
      char *end;
      const char *start;

      if(strlen(ipstring)>15) return 1;
      if(strlen(ipstring)<7) return 1;

      start=ipstring;

      for(i=0;i<4;i++)
      {
            oct=strtol(start,&end,10);
            start=end+1;
            if(oct<=255 && oct>=0)     newip=newip+(((uint8_t)oct)<<((3-
i)*8));
            else return 2;
      }
```

```
        *ip=newip;

        return 0;
    }
```

Приложение Г

(обязательное)

Графическое изображение алгоритма заполнения освобождённой области ОЗУ нулями

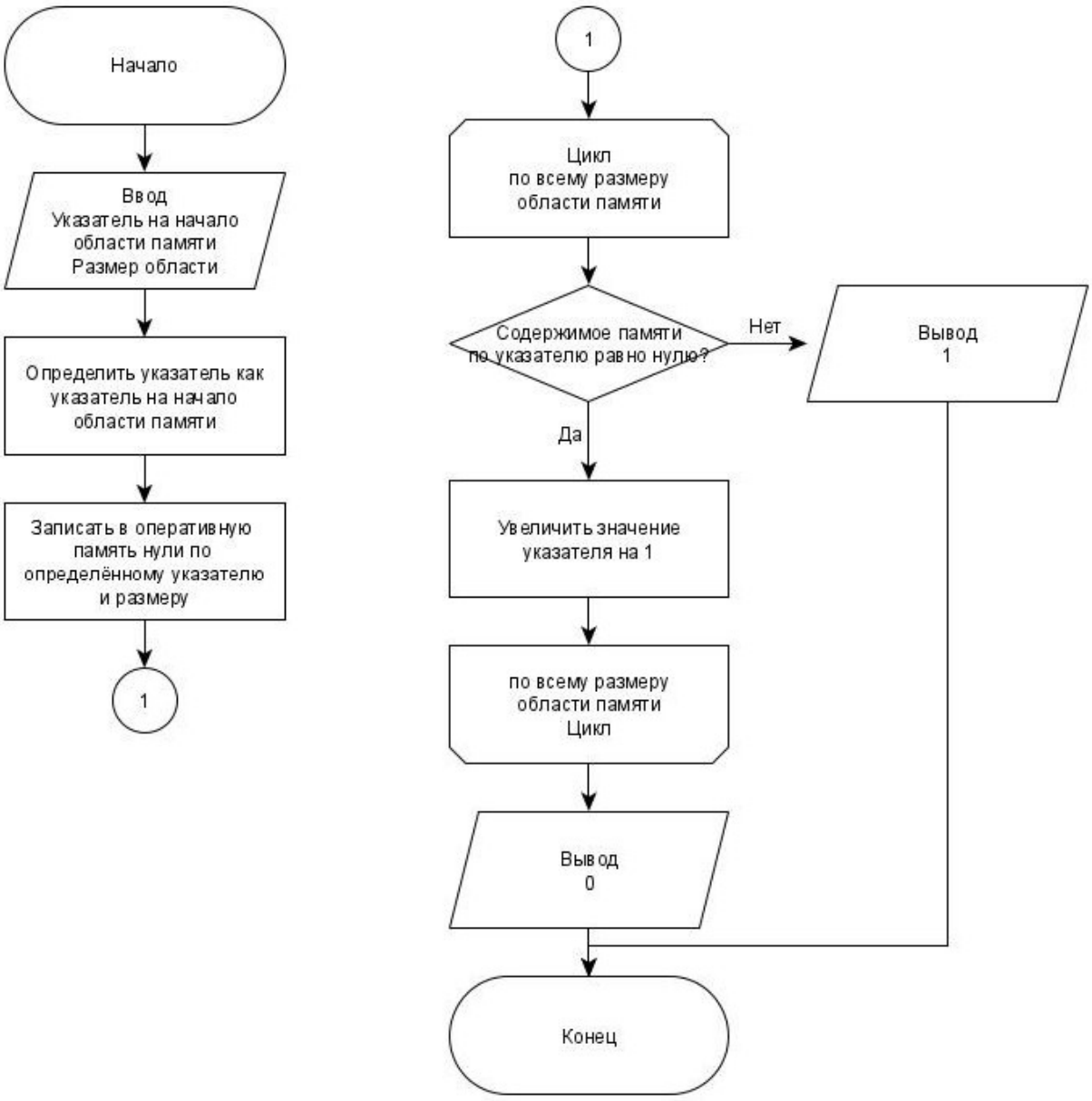

Рисунок Г.1 – Графическое изображение алгоритма заполнения освобождённой области ОЗУ нулями

Приложение Д

(обязательное)

Код функции заполнения освобождённой области ОЗУ нулями

```
static uint8_t erase_mem(void *memptr, uint16_t size)
{
	volatile uint8_t *ptr;
	uint16_t i;
	ptr = memptr;

	memset((void *)ptr,0,size);
	for ( i = 0; i < size; i++ )    if ( *ptr++ != 0 ) return 1;
	return 0;
}
```

Приложение Е

(обязательное)

Графические изображения алгоритмов функций системы хранения журналов

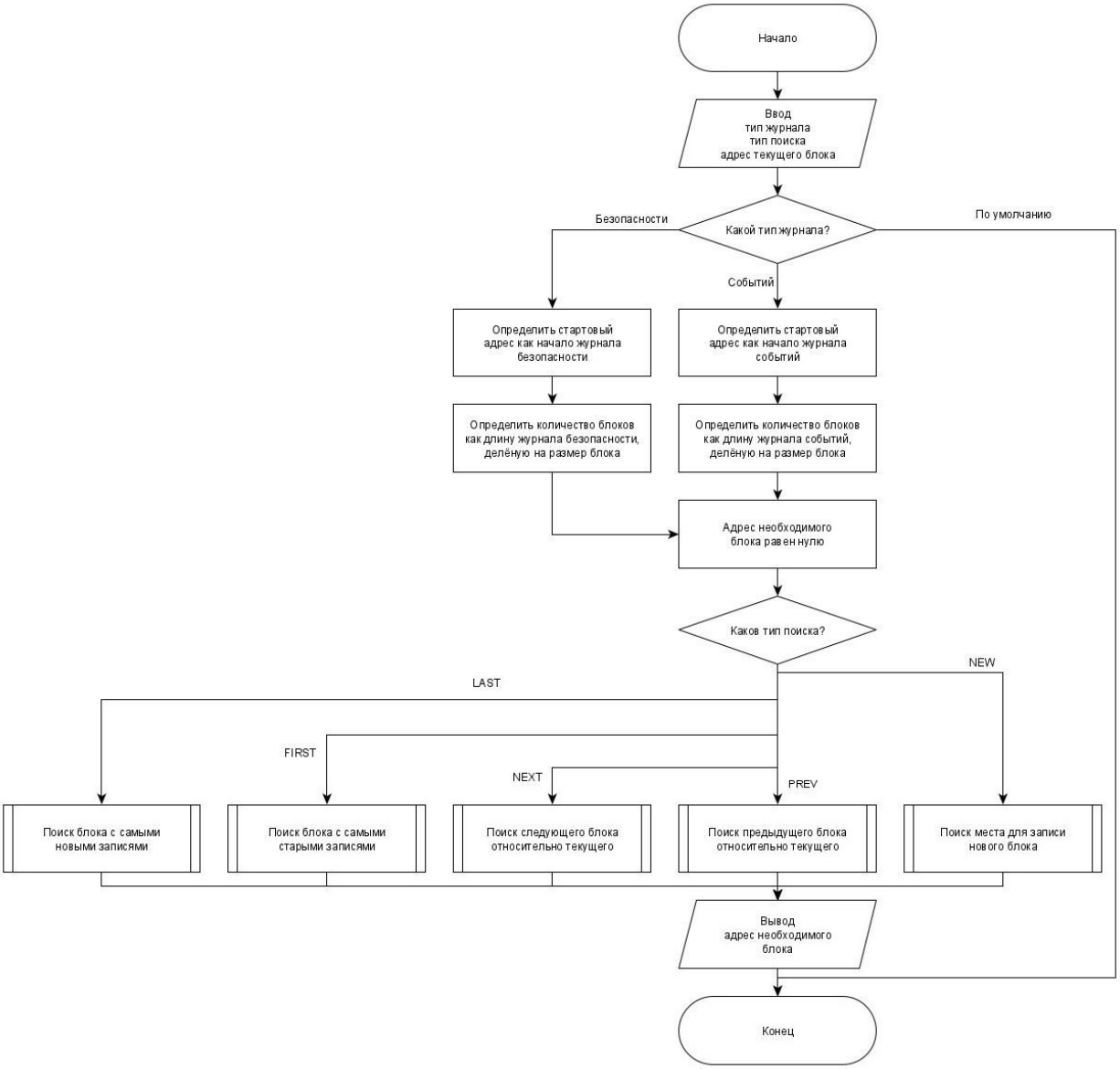

Рисунок Е.1 – Графическое изображение алгоритма поиска адреса блока

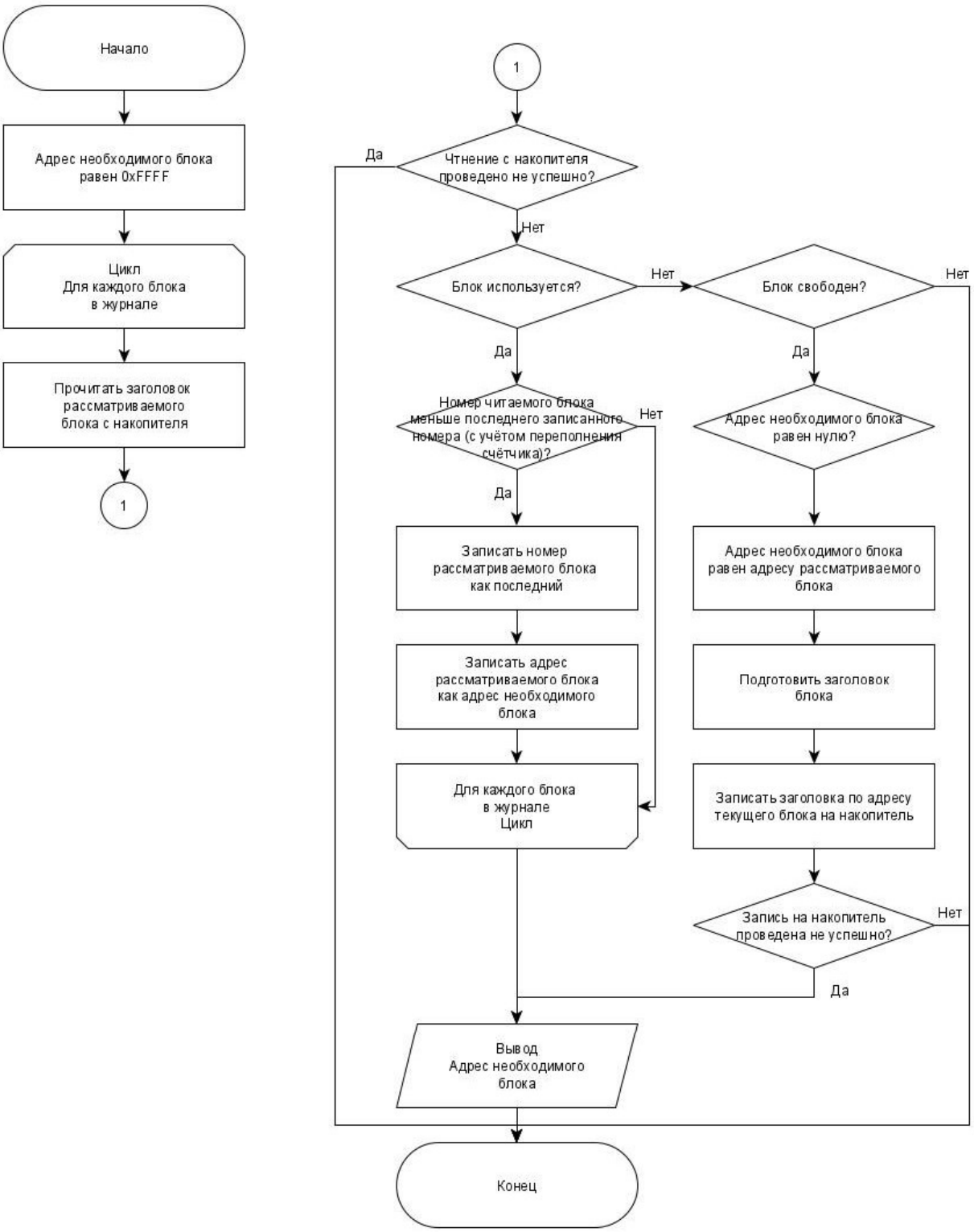

Рисунок Е.2 – Предопределённый процесс «Поиск блока с самыми новыми записями»

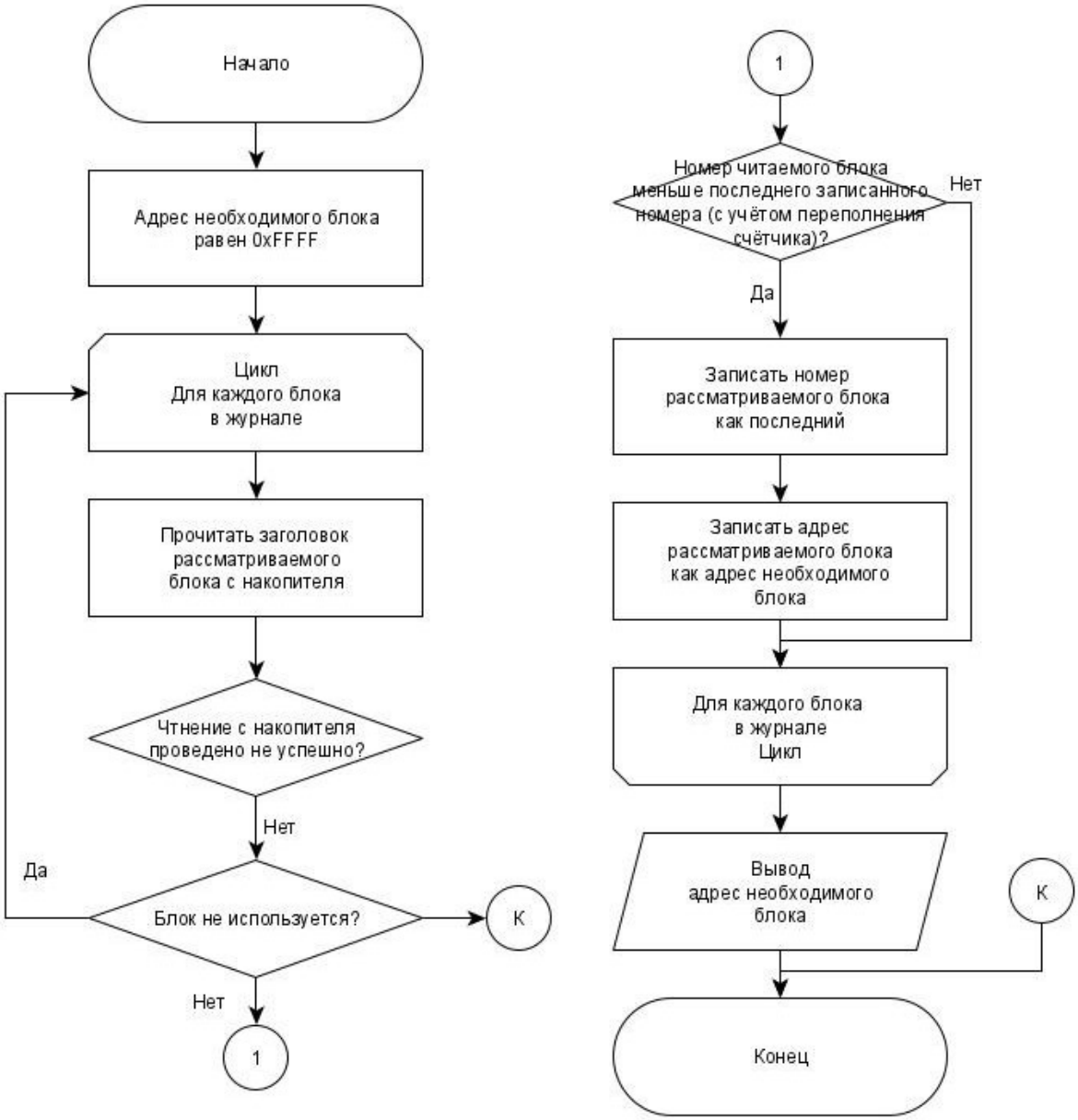

Рисунок Е.3 – Предопределённый процесс «Поиск блока с самыми старыми записями»

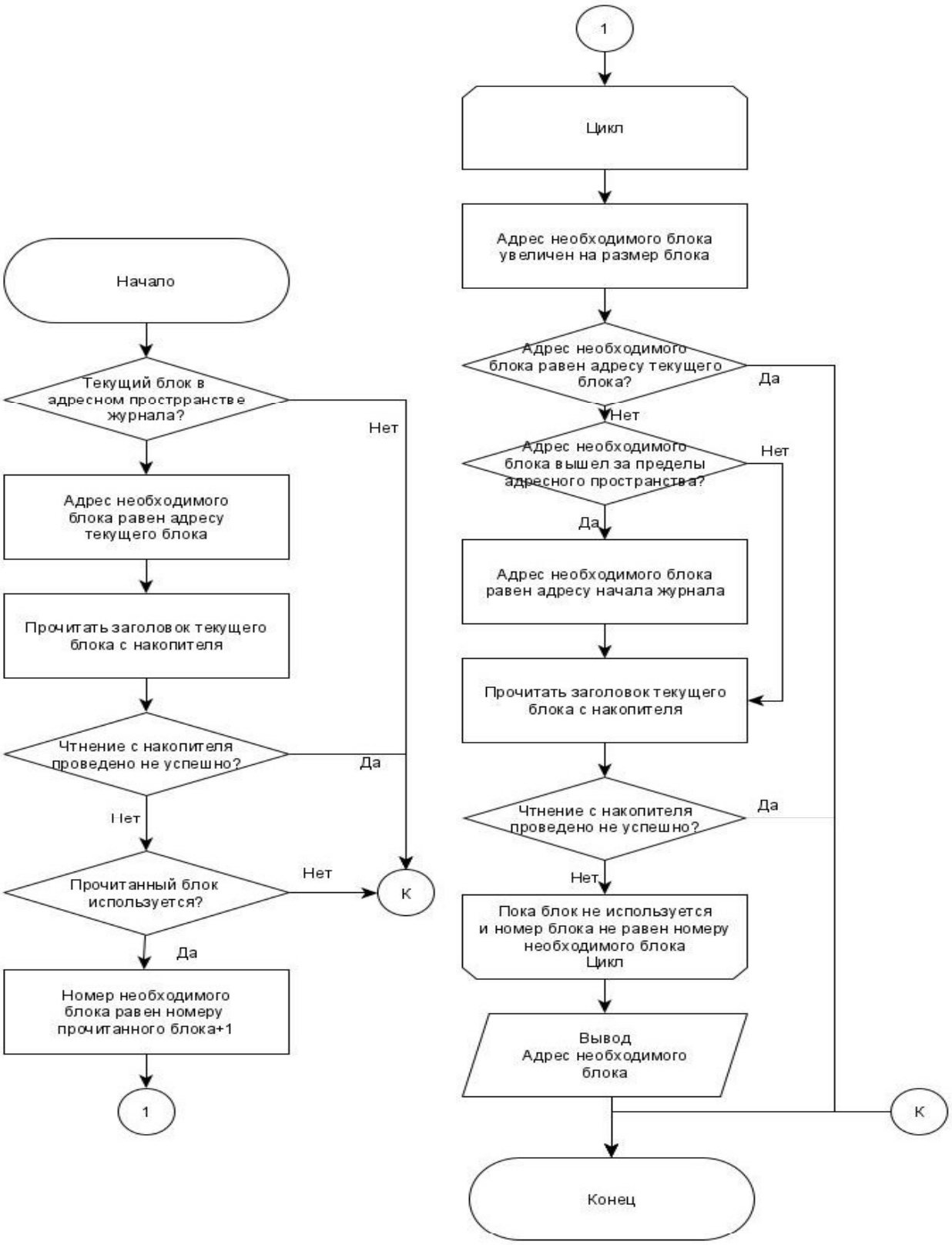

Рисунок Е.4 – Предопределённый процесс «Поиск следующего блока относительно текущего»

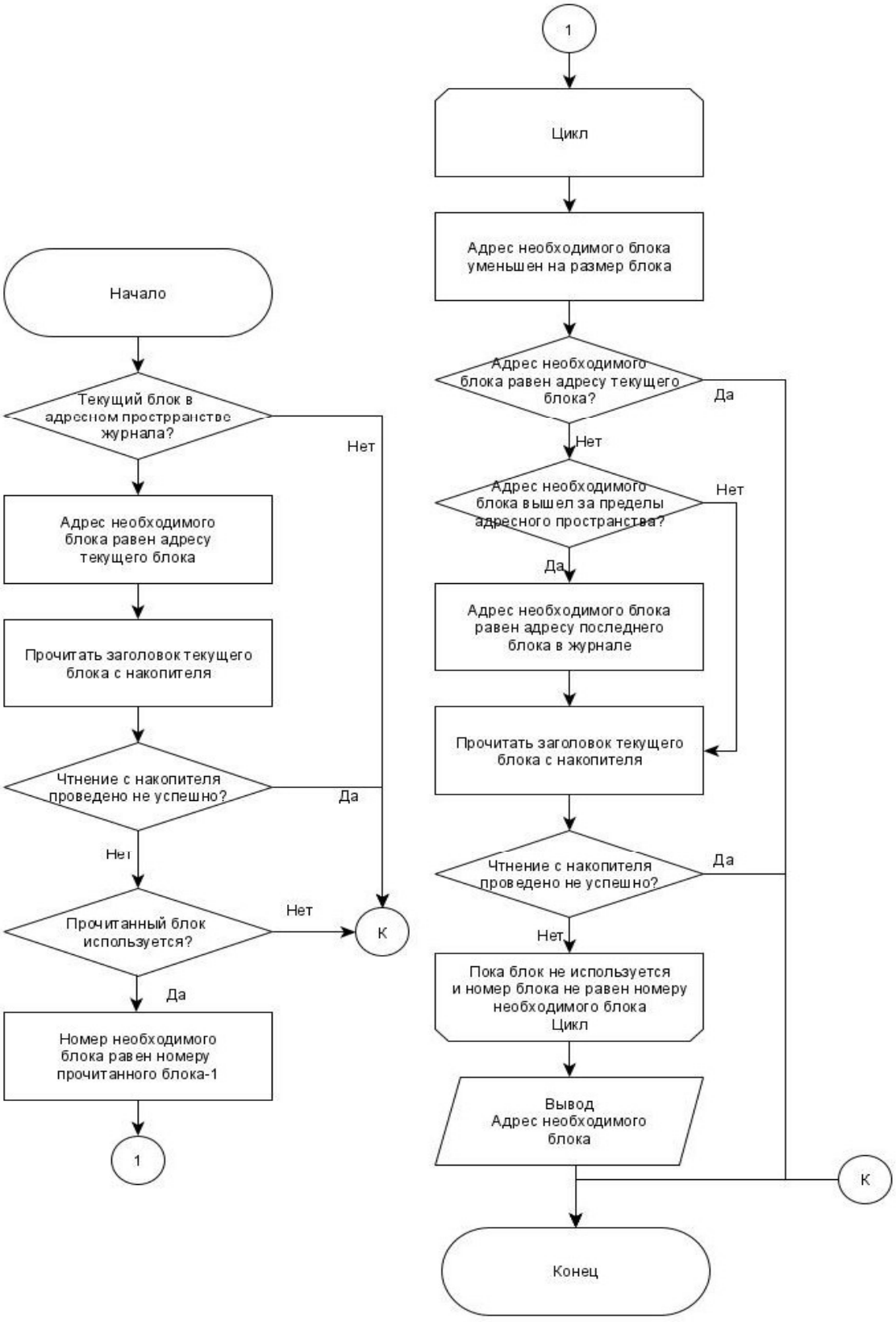

Рисунок Е.5 – Предопределённый процесс «Поиск блока предыдущего блока относительно текущего»

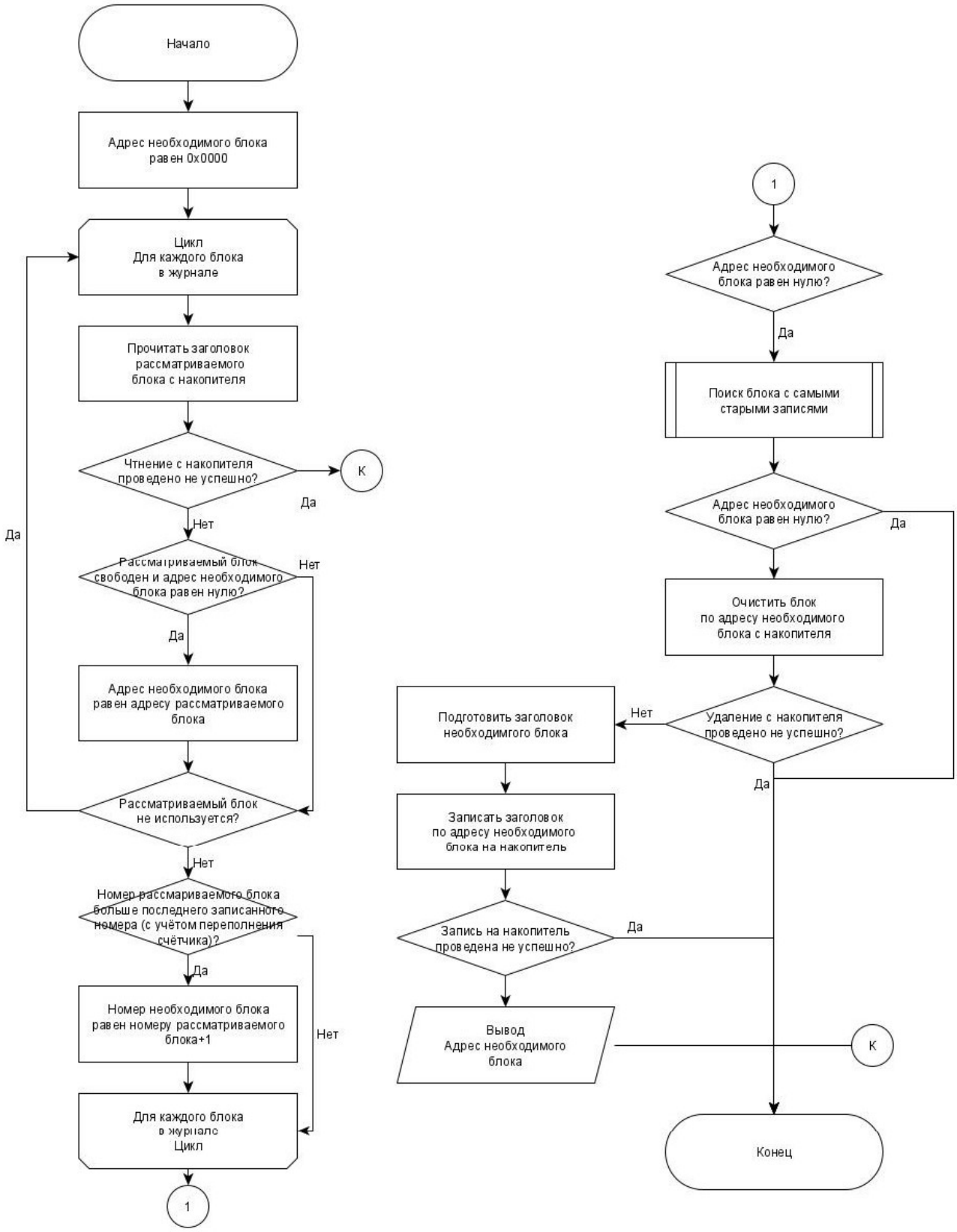

Рисунок Е.6 – Предопределённый процесс «Поиск места для записи нового блока»

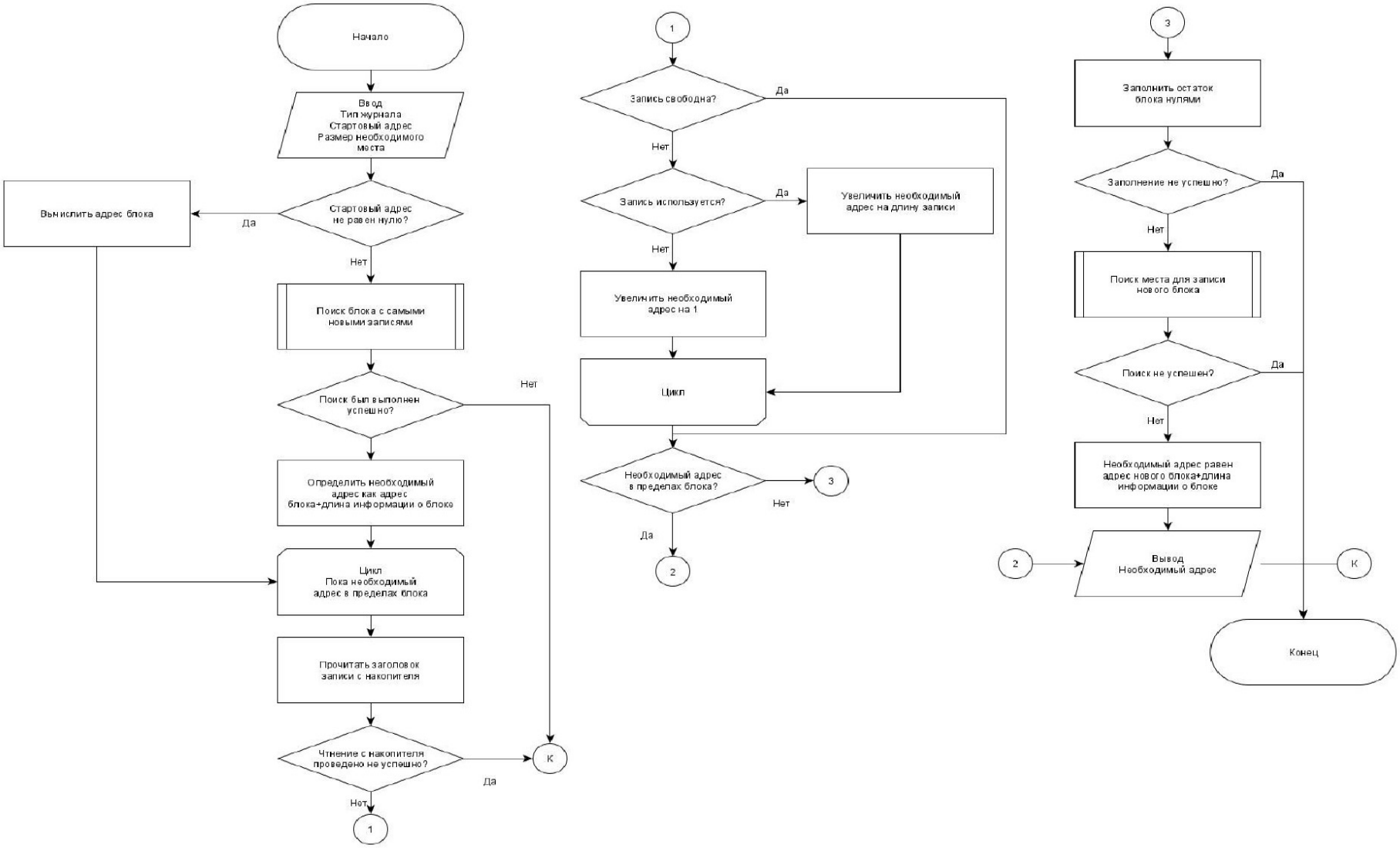

Рисунок Е.7 – Графическое изображение алгоритма поиска адреса свободного пространства

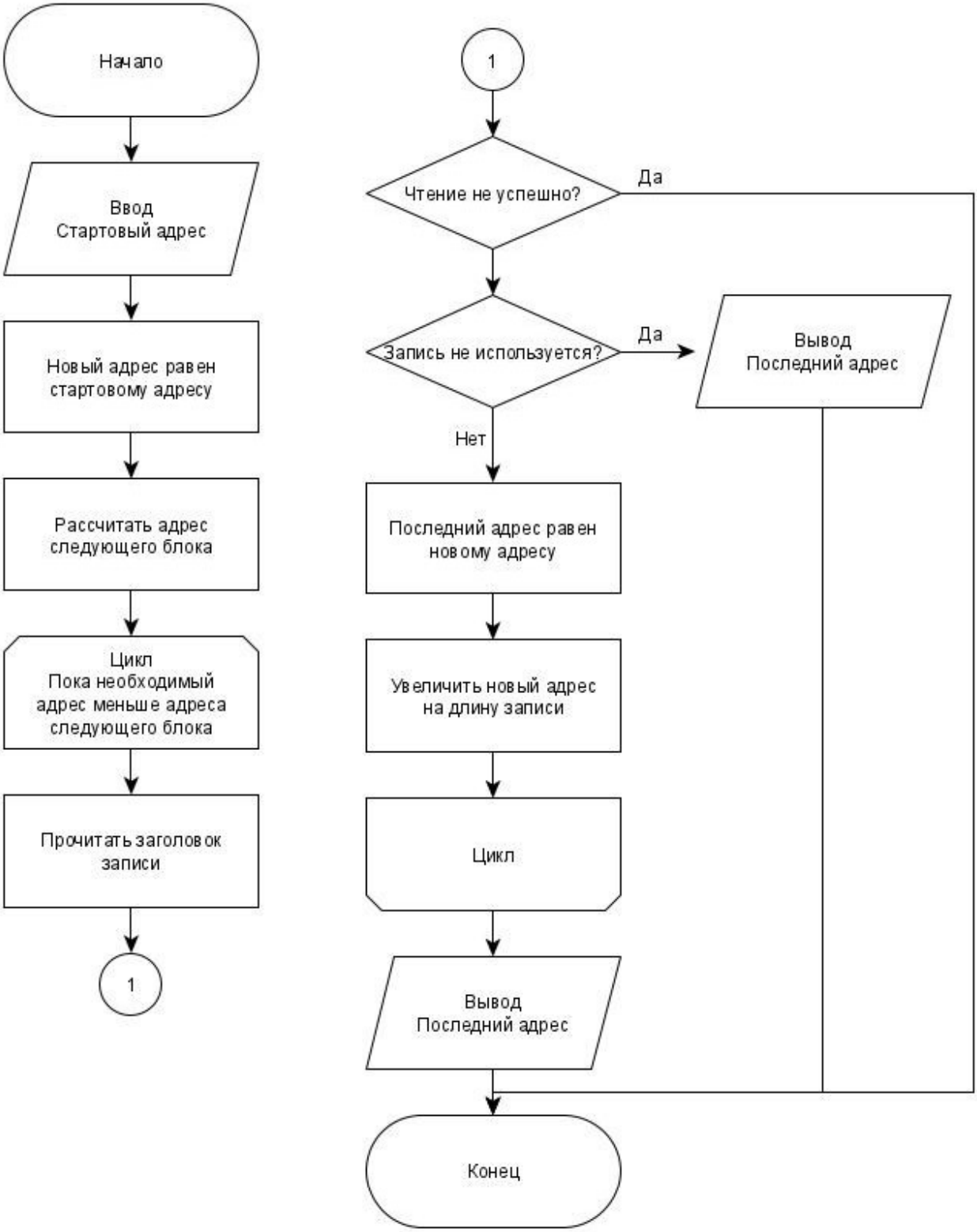

Рисунок Е.8 – Графическое изображение алгоритма поиска адреса последней записи в блоке

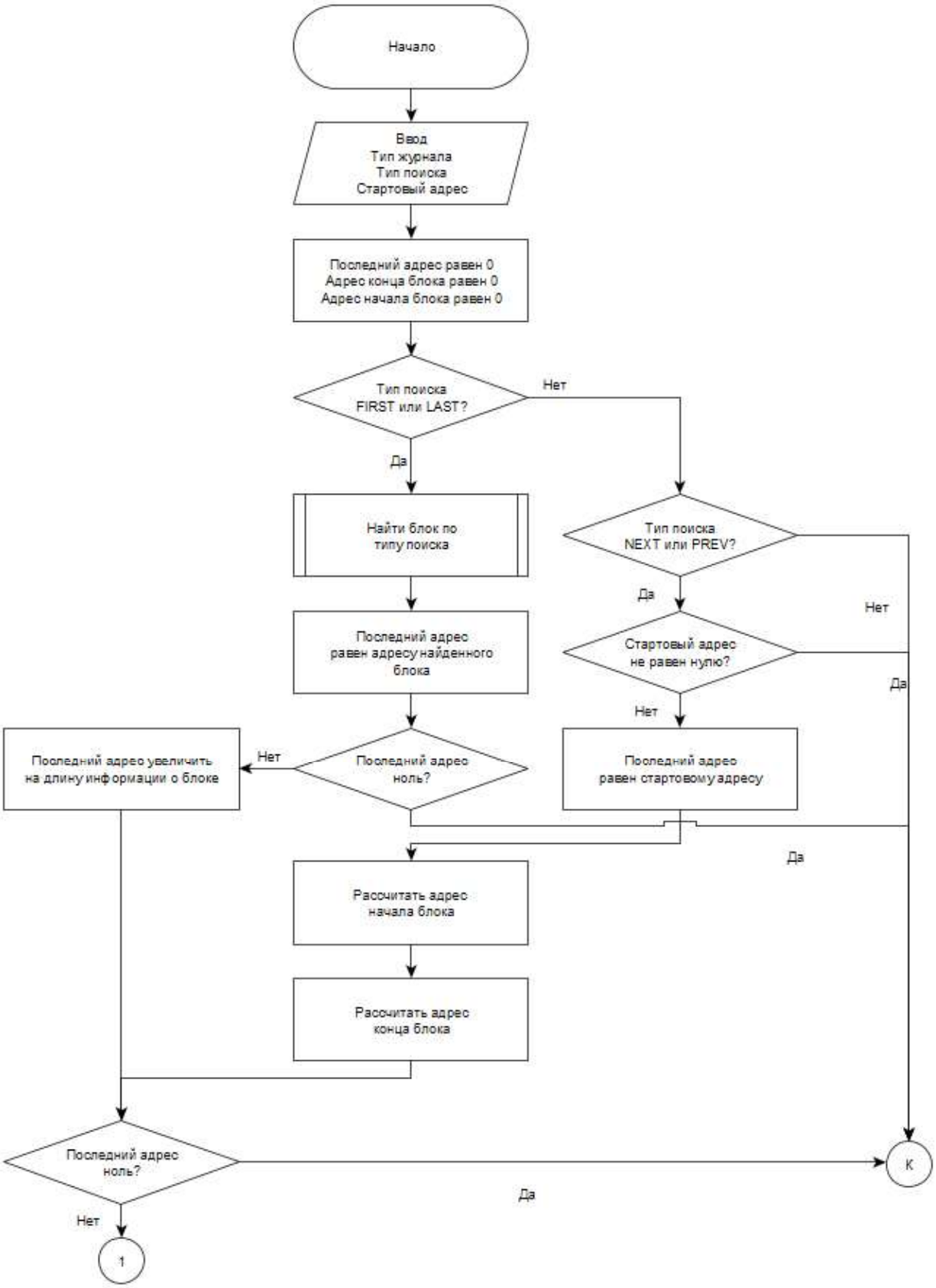

Рисунок Е.9 – Графическое изображение алгоритма поиска адреса необходимой записи. Лист 1

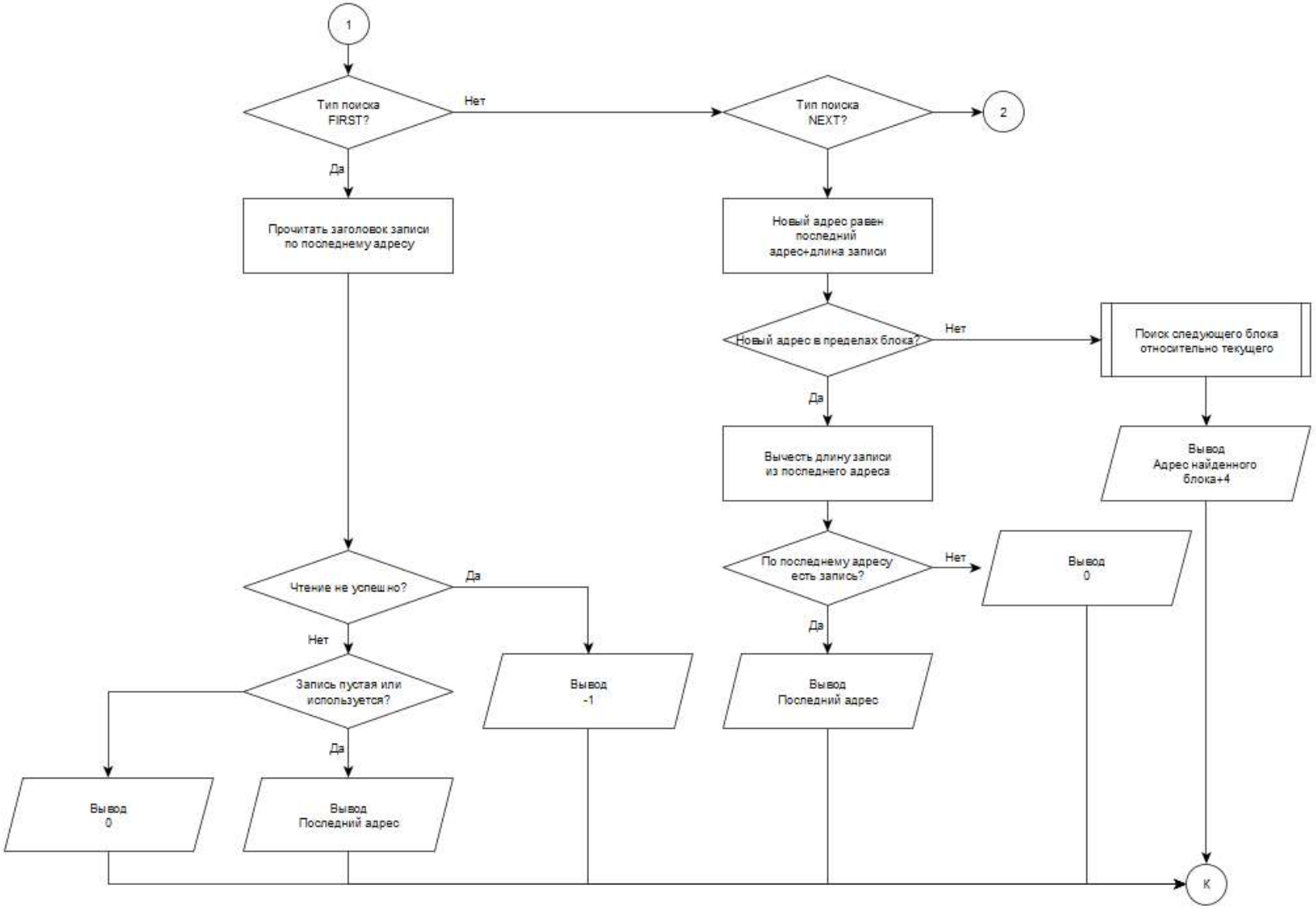

Рисунок Е.9 – Графическое изображение алгоритма поиска адреса необходимой записи. Лист 2

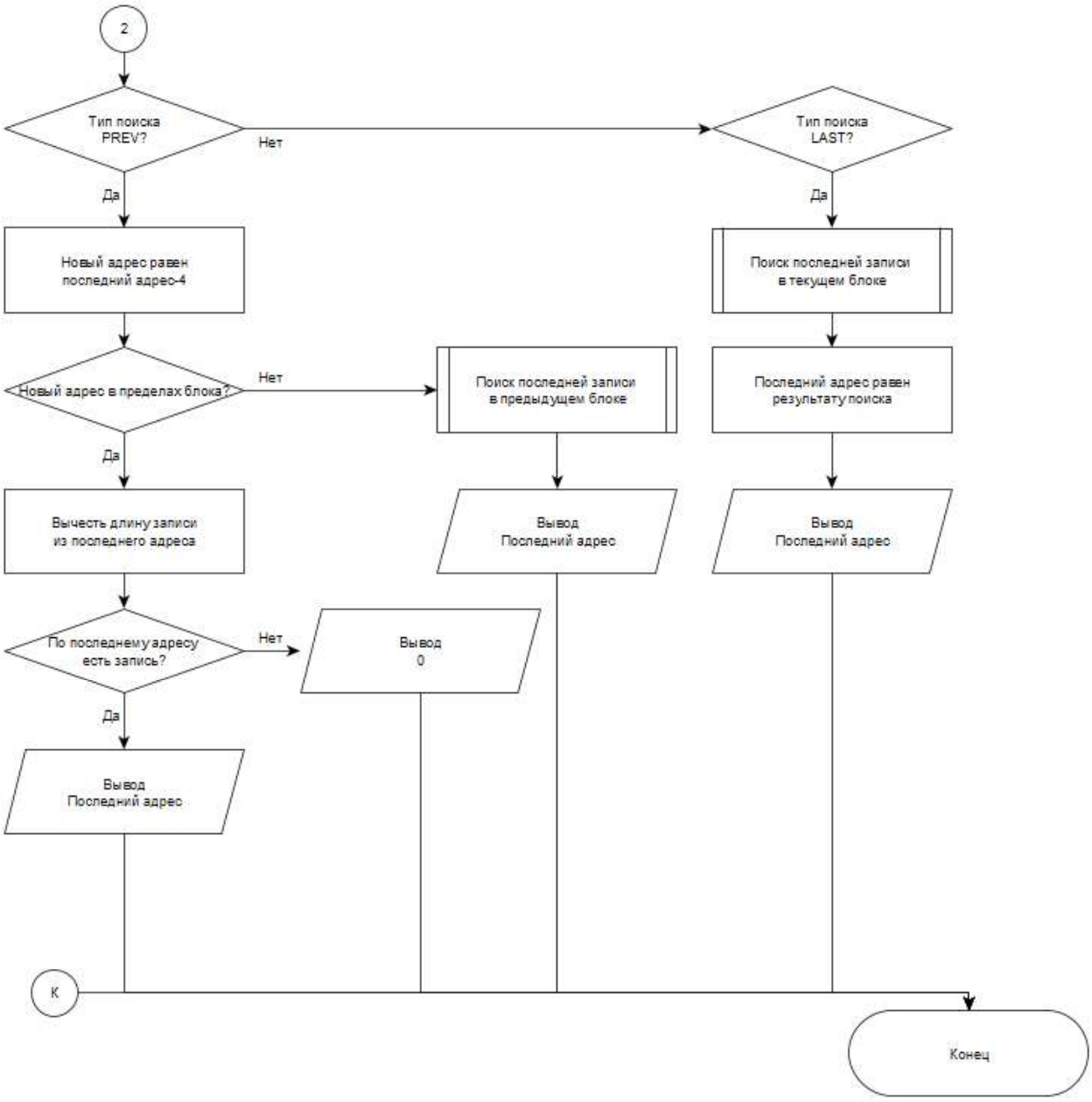

Рисунок Е.9 – Графическое изображение алгоритма поиска адреса необходимой записи. Лист 3

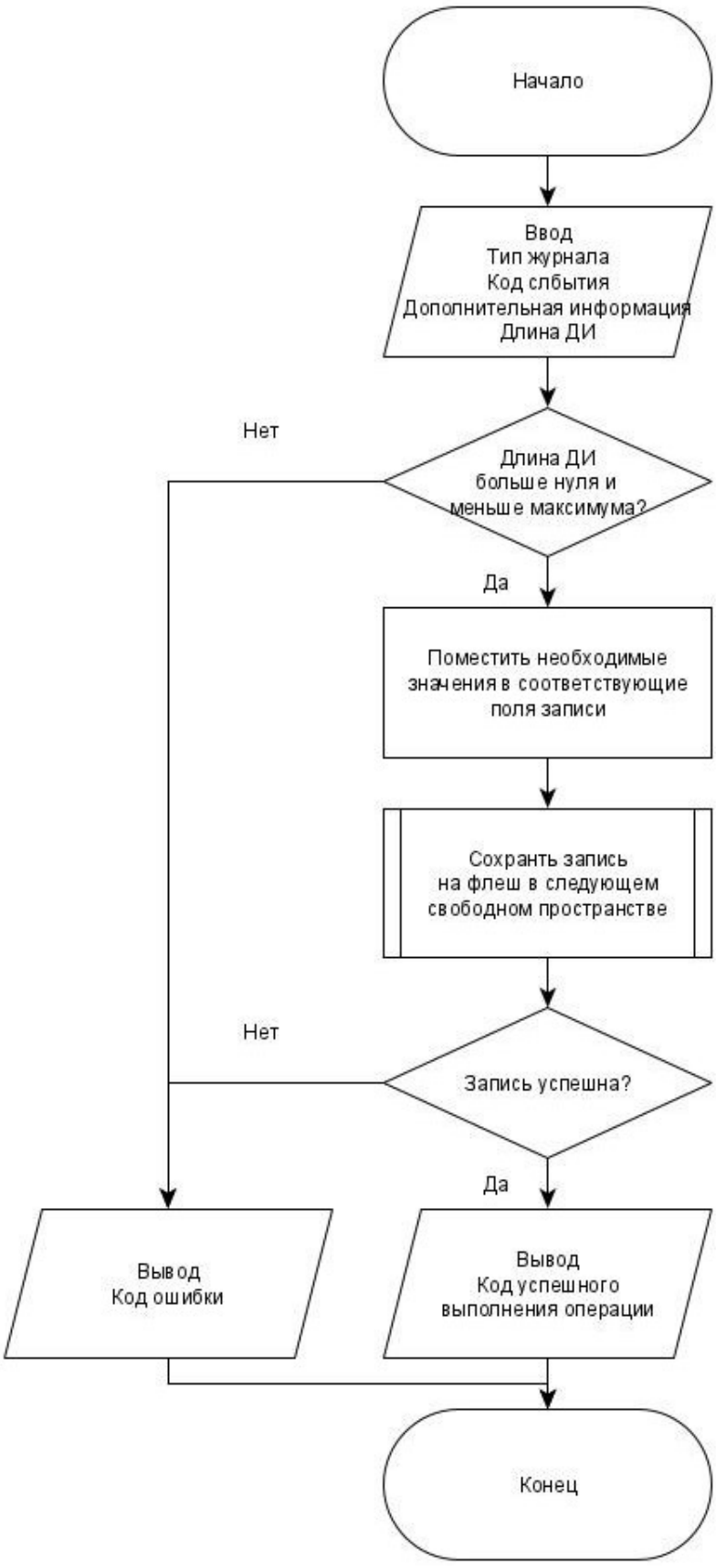

Рисунок Е.10 – Графическое изображение алгоритма добавления новой записи

Рисунок Е.11 – Графическое изображение алгоритма получения необходимой записи в блоке

Приложение Ж

(обязательное)

Содержимое файла journal_storage.h

```
#ifndef TASKS_GUI_JOURNAL_JOURNAL_STORAGE_H_
#define TASKS_GUI_JOURNAL_JOURNAL_STORAGE_H_

#define MAX_EXTDATA_LEN         242 //Максимальная длина области
аттрибутов записи в журнале
#define MAX_JRECORD_LEN         (MAX_EXTDATA_LEN + 6)
#define MAX_FULLRECORD_LEN (MAX_JRECORD_LEN + 8) //=256. Максимальная
полная длина одной записи в журнале (от старта до контрольной суммы)

void journal_storage_init();

typedef struct
{
     uint8_t is_bad_crc;                        // повреждена ли запись?
     uint32_t datetime;                         // дата время в формате
FATFS
     uint16_t event_code;                  // код события
     uint16_t ext_len;                          // размер дополнительной
области
     uint8_t ext_data[MAX_EXTDATA_LEN];    // указатель на буфер для
дополнительной области
     uint32_t storage_addr;                     // адрес записи на
носителе
     char *parcedstringbuf;                     // буфер для строки
текстового представления записи журнала  (для функции с парсингом)
     uint16_t parcedbuf_len;                    // размер буфера
parcedstringbuf (для функции с парсингом)
     uint16_t parcedbuf_offset;            // смещение в буфере
parcedstringbuf (для добавления текста о нарушении целостности)
} Journal_record;
```

```
typedef enum
{
	JEVENT=0, //Журнал событий
	JSECURITY, //Журнал безопасности
	JCNT
} Journal_type;

typedef enum
{
	FIRST=0,
	LAST,
	NEXT,
	PREV,
	NEW
} Seek_type;

typedef enum
{
	JERRORPARCE	= -3,
	JERROR,
	JBADCRC,
	JSUCCESS,
	JEOF
} JRetcode;

#endif /* TASKS_GUI_JOURNAL_JOURNAL_STORAGE_H_ */
```

Приложение И

(обязательное)

Содержимое файла journal_storage.c

```
#include "config.h"
#if !E81_VERSION
#include "../flash.h"
#else
#include "../flash_e81.h"
#endif
#include "utils/time_utils.h"
#include "journal_storage.h"
#include "utils/crc16.h"
#include "SpiFlash.h"
#include "lvgl/lib/lvgl.h"
#include "sgs_store.h"

#define BLOCK_SIZE FL_PAGE_SIZE //Размер блока
#define JOURNAL_EVENT_BLOCK_0         ADR_JOURNAL_EVENT
#define JOURNAL_EVENT_LEN        3*BLOCK_SIZE
#define JOURNAL_SECURITY_BLOCK_0      ADR_JOURNAL_SECUR
#define JOURNAL_SECURITY_LEN          3*BLOCK_SIZE

enum
{
      RECORD_USE = 0xA525,
      RECORD_FREE = 0xFFFF,
      RECORD_ENDBLOCK = 0x0000,
};

typedef struct
{
      uint16_t start;
      uint16_t number;
```

```
} Block_info;

static uint32_t _next_record_addr[JCNT]={0,0};

static bool is_from_different_cycle(uint16_t curr_num, uint16_t num,
uint16_t total_block);

/*
 * Функция получения начального адреса блока:
 Journal_type jtype - тип журнала
 Seek_type type - позиционирование запрашиваемого блока:
      FIRST - первый (самый старый) блок
      LAST - последний (самый новый) блок
      NEXT - последующий блок
      PREV - предыдущий блок
      NEW - новый блок (следующий неиспользованный или стертый самый
старый)

      uint32_t current_block_addr - стартовый адрес текущего блока

      Возвращает адрес начала блока или 0 в случае ошибки

      В начале каждого блока 64K хранится служебная информация Block_info
о блоке:
    2 байта старт - 0xBEAF
      2 байта - порядковый номер блока, монотонно увеличивающийся при
назначении новых блоков
 * */

enum
{
      BLOCK_USE = 0xBEAF,
      BLOCK_FREE = 0xFFFF,
```

```
};

uint32_t get_block_addr(Journal_type jtype, Seek_type type, uint32_t
current_block_addr)
{
      static uint32_t block_start_addr;
      static uint32_t req_block_addr;
      uint16_t buff[2];
      uint16_t req_block_num;
      uint16_t total_block;

      switch(jtype)
      {
      case JEVENT:
            block_start_addr=JOURNAL_EVENT_BLOCK_0;
            total_block=JOURNAL_EVENT_LEN/BLOCK_SIZE;
            break;
      case JSECURITY:
            block_start_addr=JOURNAL_SECURITY_BLOCK_0;
            total_block=JOURNAL_SECURITY_LEN/BLOCK_SIZE;
            break;
      default:
            return 0;
            break;
      }
      req_block_addr = 0;

      switch(type)
      {
            //Первый блок, с самыми старыми записями
            case FIRST:
                  req_block_num = 0xFFFF;
```

```
                for(uint16_t i = 0; i < total_block; i++){//каждый блок
по всей длине журнала
                      if(SpiFlash_Iface_ReadData(block_start_addr +
BLOCK_SIZE * i, (uint8_t*)buff, sizeof(buff))) return 0;
                      if(buff[0] != BLOCK_USE)
                            continue;
                      if // поиск самого маленького номера блока с учётом
номеров до перехода 0xFFFF -> 0x0000
                      (
                            ( (buff[1] <= req_block_num) && !
is_from_different_cycle(buff[1], req_block_num, total_block) )
                      ||  ( (buff[1] >= req_block_num) &&
is_from_different_cycle(buff[1], req_block_num, total_block) )
                      ||  ( i == 0 )
                      ){
                            req_block_num = buff[1];
                            req_block_addr = block_start_addr + BLOCK_SIZE
* i;
                      }
                }
                return req_block_addr; // ноль, если не был найден самый
старый блок
          //Последний блок, с самыми свежими записями
          case LAST:
                req_block_num = 0x0000;
                for(uint16_t i = 0; i < total_block; i++){//каждый блок
по всей длине журнала
                      if(SpiFlash_Iface_ReadData(block_start_addr +
BLOCK_SIZE * i, (uint8_t*)buff, sizeof(buff))) return 0;
                      if(buff[0] == BLOCK_USE)
                      {
                            if // поиск самого большого номера блока без
учёта номеров до перехода 0xFFFF -> 0x0000
```

```
                        (
                                ( (buff[1] >= req_block_num) && !
is_from_different_cycle(buff[1], req_block_num, total_block) )
                        ||   ( (buff[1] <= req_block_num) &&
is_from_different_cycle(buff[1], req_block_num, total_block) )
                        ||  ( i == 0 )
                        ){
                            req_block_num = buff[1];
                            req_block_addr = block_start_addr +
BLOCK_SIZE * i;
                        }
                    }
                    else if(buff[0] == BLOCK_FREE)
                    {
                        if(!req_block_addr)
                        {
                            req_block_addr = block_start_addr +
BLOCK_SIZE * i;
                            buff[0] = BLOCK_USE;
                            buff[1] = req_block_num;

    if(SpiFlash_Iface_WriteData(req_block_addr, (uint8_t*)buff,
sizeof(buff))) return 0;
                        }
                        return req_block_addr;
                    }
                    else return 0;
                }
                return req_block_addr; // ноль, если не был найден самый
свежий блок
            //Следующий используемый блок
            case NEXT:
                if(current_block_addr <  block_start_addr
```

```
                || current_block_addr >= block_start_addr +
BLOCK_SIZE*total_block) {
                      return 0;
                }
                req_block_addr = current_block_addr;
                if(SpiFlash_Iface_ReadData(current_block_addr,
(uint8_t*)buff, sizeof(buff))) return 0;
                if(buff[0] != BLOCK_USE) return 0;
                req_block_num = (uint16_t)(buff[1] + 1);

                do{
                      req_block_addr += BLOCK_SIZE;
                      if (req_block_addr == current_block_addr){
                            return 0;
                      }

                      if(req_block_addr == block_start_addr +
BLOCK_SIZE*total_block){
                            req_block_addr = block_start_addr;
                      }
                      if(SpiFlash_Iface_ReadData(req_block_addr,
(uint8_t*)buff, sizeof(buff))) return 0;
                }while(buff[0] != BLOCK_USE || buff[1] !=
req_block_num);

                return req_block_addr;
          //Предыдущий используемый блок
          case PREV:
                if(current_block_addr <  block_start_addr
                || current_block_addr >= block_start_addr +
BLOCK_SIZE*total_block) {
                      return 0;
                }
```

```
                req_block_addr = current_block_addr;
                if(SpiFlash_Iface_ReadData(current_block_addr,
(uint8_t*)buff, sizeof(buff))) return 0;
                if(buff[0] != BLOCK_USE) return 0;
                req_block_num = (uint16_t)(buff[1] - 1);

                do{
                      req_block_addr -= BLOCK_SIZE;
                      if(req_block_addr == current_block_addr){
                            return 0;
                      }

                      if(req_block_addr == block_start_addr - BLOCK_SIZE)
{
                            req_block_addr = block_start_addr +
(BLOCK_SIZE*(total_block - 1));
                      }
                      if(SpiFlash_Iface_ReadData(req_block_addr,
(uint8_t*)buff, sizeof(buff))) return 0;
                }while(buff[0] != BLOCK_USE || buff[1] !=
req_block_num);

                return req_block_addr;
          //Следующий новый блок: следующий пустой или первый из занятых
          case NEW:
          {
                req_block_num = 0x0000;
                for(uint16_t i = 0;  i < total_block; i++){
                      if(SpiFlash_Iface_ReadData(block_start_addr +
BLOCK_SIZE * i, (uint8_t*)buff, sizeof(buff))) return 0;
                      if(buff[0] == BLOCK_FREE && !req_block_addr) { //
поиск пустого блока
```

```
                        req_block_addr = block_start_addr + BLOCK_SIZE
* i;
                    }

                    if(buff[0] != BLOCK_USE) // поиск самого большого
номера блока без учёта номеров до перехода 0xFFFF -> 0x0000 и запись в
block_num номера следующего блока
                        continue;
                    if (
                            ( (buff[1] >= (uint16_t)(req_block_num -
1)) && !is_from_different_cycle(buff[1], (uint16_t)(req_block_num - 1),
total_block) )
                        ||   ( (buff[1] <= (uint16_t)(req_block_num -
1)) &&  is_from_different_cycle(buff[1], (uint16_t)(req_block_num - 1),
total_block) )
                        ||  ( i == 0 )
                    ){
                        req_block_num = buff[1] + 1;
                    }
                }

                if(req_block_addr == 0){ // поиск самого старого блока с
целью его затирания (если пустой не найден)
                    req_block_addr = get_block_addr(jtype, FIRST, 0);
                    if (req_block_addr == 0){
                        return 0;
                    }
                }
                if(SpiFlash_Iface_EraseSector(req_block_addr)) return 0;

                buff[0] = BLOCK_USE;
                buff[1] = req_block_num;
```

```
                if(SpiFlash_Iface_WriteData(req_block_addr,
(uint8_t*)buff, sizeof(buff))) return 0;
                return req_block_addr;
           }
      }

      return 0;
}

static bool is_from_different_cycle(uint16_t curr_num, uint16_t num,
uint16_t total_block){
      return curr_num > num ? (curr_num - num) > total_block : (num -
curr_num) > total_block;
}

//Получение адреса следующего свободного пространства (в конце журнала)
размером need_space байт
//для журнала указанного в Journal_type jtype
//Если задан start_address!=0 (последний свободный адрес в последнем
блоке) поиск начинается с него.
//Возвращает адрес или 0 в случае ошибки
uint32_t get_next_free_space(Journal_type jtype, uint32_t start_address,
uint16_t need_space)
{
      uint32_t block_start_addr=0;
      uint32_t current_addr=0;
      uint16_t buffer[2];
      uint16_t block_num;
      uint8_t bzero[MAX_FULLRECORD_LEN];

      if(start_address)
      {
            current_addr=start_address;
```

```
            block_num=(uint16_t) (current_addr>>16);
            block_start_addr=(block_num<<16);
      }
      else
      {
            block_start_addr = get_block_addr(jtype,LAST,0);
            if(!block_start_addr) return 0;
            current_addr = block_start_addr + sizeof(Block_info);
      }

      //Поиск места в текущей странице
      while(current_addr < block_start_addr + BLOCK_SIZE)
      {
            if(SpiFlash_Iface_ReadData(current_addr, (uint8_t *) buffer,
4)) return 0;
            if(buffer[0]==RECORD_FREE) break;
            else if(buffer[0]==RECORD_USE)
            {
                  current_addr += 4;
                  current_addr += buffer[1];
            }
            else current_addr++;
      }

      //Проверка выхода за пределы данного блока
      if((current_addr + need_space) > (block_start_addr + BLOCK_SIZE))
      {
            //Места нет - затереть остаток, пометить блок занятым
            memset(bzero, 0, sizeof(bzero));
            if(SpiFlash_Iface_WriteData(current_addr, bzero,
block_start_addr + BLOCK_SIZE - current_addr)) return 0;

            //Выделение и стирание нового блока
```

```
            block_start_addr = get_block_addr(jtype,NEW,0);
            if(!block_start_addr) return 0;
            current_addr = block_start_addr + sizeof(Block_info);
      }
      return current_addr;
}


//Функция чтения заголовка записи по адресу addr
// start - указатель на переменную под (старт+флаг использования)
// lenght - указатель на переменную под длину (данные + crc + длина)
//Возвращает -1 в случае ошибки и 0 в случае успеха
int read_record_header(uint32_t addr, uint16_t *start, uint16_t *lenght)
{
      uint16_t buf[2];
      if(SpiFlash_Iface_ReadData(addr, (uint8_t *) buf, sizeof(buf)))
return -1;
      if(start) *start=buf[0];
      if(lenght) *lenght=buf[1];
      return 0;
}

//Функция поиска последней записи в текущем блоке начиная с записи с
адресом start_address
//Возвращает адрес последней записи или 0 в случае ошибки
uint32_t find_last_record_in_block(uint32_t start_addr)
{
      uint32_t new_addr;
      uint32_t last_addr=0;
      uint32_t end_block_addr;
      uint16_t current_block_num;
      uint16_t start;
      uint16_t lenght;
```

```
	new_addr=start_addr;

	current_block_num=(uint16_t)(start_addr>>16);
	end_block_addr=((current_block_num+1)<<16);

	while(new_addr < end_block_addr)
	{
		//Проверка наличия записи по новому адресу
		if(read_record_header(new_addr, &start, &lenght)) return -1;
		if(start!=RECORD_USE) return last_addr;
		last_addr=new_addr;
		new_addr+=lenght+4;
	}

	return last_addr;
}

//Получение адреса записи журнала типа RB_type для журнала указанного в
Journal_type
//Адрес start_address используется только в случае получения предыдущей /
следующей записи
//Возвращает адрес или 0 в случае ошибки
uint32_t get_record_addr(Journal_type jtype,  Seek_type type, uint32_t
start_address)
{
	uint32_t last_addr=0;
	uint32_t new_addr;
	uint32_t end_block_addr=0;
	uint32_t start_block_addr=0;
	uint16_t current_block_num;
	//uint16_t buf[2];
	uint16_t start;
```

```
uint16_t lenght;

if(!get_init_flag()) return 0;

if(type==FIRST || type==LAST)
{
     last_addr=get_block_addr(jtype, type,0);
     if(!last_addr) return 0;
     last_addr += sizeof(Block_info);
}
else if(type==NEXT || type==PREV)
{
     if(!start_address) return 0;
     last_addr=start_address;

     current_block_num=(uint16_t)(last_addr>>16);
     start_block_addr=(current_block_num<<16);
     end_block_addr=((current_block_num+1)<<16);

}
else return 0;

if(!last_addr) return 0;

if(type==FIRST)
{
     if(read_record_header(last_addr, &start, &lenght)) return -1;
     if(start==RECORD_USE || start==RECORD_FREE) return last_addr;
     else return 0;
}
else if(type==NEXT)
{
     if(read_record_header(last_addr, &start, &lenght)) return -1;
```

```
            if(start==RECORD_USE) new_addr=last_addr+lenght+4;
            else new_addr=last_addr+1;

            if(new_addr>=end_block_addr)
            {
                  new_addr=get_block_addr(jtype, NEXT, start_block_addr);
                  if(!new_addr) return 0;
                  new_addr+=sizeof(Block_info);
            }
            //Проверка наличия записи по новому адресу
            if(read_record_header(new_addr, &start, &lenght)) return -1;
            if(start==RECORD_USE || start==RECORD_FREE) return new_addr;

            if(start==RECORD_ENDBLOCK)
            {
                  new_addr=get_block_addr(jtype, NEXT, start_block_addr);
                  if(!new_addr) return 0;
                  new_addr+=sizeof(Block_info);
                  if(read_record_header(new_addr, &start, &lenght)) return
-1;
                  if(start==RECORD_USE || start==RECORD_FREE) return
new_addr;
            }
      }
      else if(type==PREV)
      {
            new_addr=last_addr-4;
            if(new_addr<(start_block_addr+sizeof(Block_info)))
            {
                  //Переход на предыдущий блок
                  new_addr=get_block_addr(jtype, PREV, start_block_addr);
                  if(!new_addr) return 0;
                  new_addr+=sizeof(Block_info);
```

```
                //Поиск последней записи
                new_addr=find_last_record_in_block(new_addr);
                if(new_addr) return new_addr;
                return last_addr;
          }
          else
          {
                //Переход на предыдущую запись
                if(read_record_header(new_addr-2, &start, &lenght))
return -1;
                new_addr-=lenght;
                if(new_addr<(start_block_addr+sizeof(Block_info)))
return -1;

                //Проверка наличия записи по новому адресу
                if(read_record_header(new_addr, &start, &lenght)) return
-1;
                if(start==RECORD_USE) return new_addr;
                else return -1;
          }
     }
     else if(type==LAST)
     {
          //Поиск последней записи в последнем блоке
          return find_last_record_in_block(last_addr);
     }

     return 0;
}

void journal_storage_init()
{
```

```
      for(int i=0; i<JCNT; i++) _next_record_addr[i]=0;
}

//Добавить запись в журнал
//Возвращает 0 в случае успешного выполнения или код ошибки
//Journal_type jtype - тип журнала:   JEVENT=0 - Журнал событий,
JSECURITY=1 - Журнал безопасности
//uint16_t event_code - код события из journal.h
//uint8_t *data - указатель на буфер с расширенными данными (до
MAX_EXTDATA_LEN байт)
//uint16_t datalen - длина буфера data в байтах (до MAX_EXTDATA_LEN байт)
//Возвращает 0 в случае успешного выполнения или код ошибки
JRetcode add_record(Journal_type jtype, uint16_t event_code, uint8_t
*data, uint16_t datalen)
{
      uint8_t recordbuf[MAX_FULLRECORD_LEN];
      uint32_t datetime;
      uint16_t crc=0;
      uint16_t wrptr=0;
      uint16_t header=RECORD_USE;

      if(!get_init_flag()) return JERROR;

      if(datalen>0 && !data) return JERROR;
      if(datalen>MAX_EXTDATA_LEN) return JERROR;

      //Подготовка места для записи

_next_record_addr[jtype]=get_next_free_space(jtype,_next_record_addr[jtyp
e],datalen+14);
      if(!_next_record_addr[jtype]) return JERROR;

       /** Запись переменной длины
```

```
         * 2 байт (старт+флаг использования)
         * 2 байт длина (данные + crc + длина)
         * n байт данные журнала:
         *         4 байта - дата время в формате FatFS
                   2 байта - код события
                   datalen байт - атрибуты
         * 2 байт длина (данные + crc + длина)
         * 2 байт crc (старт + длина + данные + длина)*/

      //Подготовка буфера с записью
      *((uint16_t *)(&recordbuf[wrptr]))=header;
      wrptr+=sizeof(header);
      *((uint16_t *)(&recordbuf[wrptr]))=datalen+2+4+2+2;
      wrptr+=sizeof(datalen);
      datetime=get_current_systime_fatfs();
      *((uint32_t *)(&recordbuf[wrptr]))=datetime;
      wrptr+=sizeof(datetime);
      *((uint16_t *)(&recordbuf[wrptr]))=event_code;
      wrptr+=sizeof(event_code);
      if((wrptr+datalen+sizeof(datalen)+sizeof(crc))>sizeof(recordbuf))
return JERROR;
      if(datalen>0) memcpy(&recordbuf[wrptr], data, datalen);
      wrptr+=datalen;
      *((uint16_t *)(&recordbuf[wrptr]))=datalen+2+4+2+2;
      wrptr+=sizeof(datalen);

      crc = crc16_calc(crc, recordbuf, wrptr);

      *((uint16_t *)(&recordbuf[wrptr]))=crc;
      wrptr+=sizeof(crc);

      //Запись на flash
```

```
	if(SpiFlash_Iface_WriteData(_next_record_addr[jtype], recordbuf,
wrptr)) return JERROR;

	//LV_LOG_USER("Written record at addr 0x%x: time=%d event=%d data=
%s", _next_record_addr[jtype], datetime, event_code, data);

	_next_record_addr[jtype]+=wrptr;

	return JSUCCESS;
}

//Получение RB_type записи журнала из журнала Journal_type
//При навигации NEXT/PREV необходимо передать адрес предыдущей записи
(prev_addr) из структуры Journal_record->storage_addr
//Возвращает 0 в случае успешного выполнения или код ошибки
JRetcode get_record(Journal_type jtype, Seek_type stype, uint32_t
prev_addr, Journal_record *record)
{
	uint32_t curr_addr=0;
	uint8_t recordbuf[MAX_FULLRECORD_LEN];
	uint16_t len=0;
	uint16_t datalen=0;
	uint16_t crc_calc=0;
	uint16_t crc_store=0;
	uint16_t header;
	uint16_t rdptr=0;

	if(!get_init_flag()) return JERROR;

	curr_addr=get_record_addr(jtype, stype, prev_addr);
	if(!curr_addr)
	{
		if(stype==PREV) return JEOF;
```

```
            else return JERROR;
        }

        //Считывание записи
        if(SpiFlash_Iface_ReadData(curr_addr, recordbuf, 4)) return JERROR;
        header=*((uint16_t *)(&recordbuf[rdptr]));
        if(header==RECORD_FREE) return JEOF;
        if(header!=RECORD_USE) return JERROR;
        rdptr+=sizeof(header);
        len = *((uint16_t *)(&recordbuf[rdptr]));
        rdptr+=sizeof(len);

        if((len+4) > (uint16_t)sizeof(recordbuf)) return JERROR;
        if(SpiFlash_Iface_ReadData(curr_addr, recordbuf, len+4)) return
JERROR;

        //Проверка целостности
        crc_store = *((uint16_t *)(&recordbuf[len+2]));
        crc_calc = crc16_calc(crc_calc, recordbuf, len+2);

        datalen=len-10; //длина области атрибутов
        record->storage_addr=curr_addr;
        record->datetime=*((uint32_t *)(&recordbuf[rdptr]));
        rdptr+=sizeof(record->datetime);
        record->event_code=*((uint16_t *)(&recordbuf[rdptr]));
        rdptr+=sizeof(record->event_code);
        record->ext_len=datalen;
        if(datalen>sizeof(record->ext_data)) return JERROR;
        if(datalen && record->ext_data) memcpy(record-
>ext_data,&recordbuf[rdptr],datalen);

        record->is_bad_crc = 0;
        if(crc_store!=crc_calc){
```

```
            record->is_bad_crc = 1;
            return JBADCRC;
        }

        return JSUCCESS;

    }
```

Приложение К

(обязательное)

Графические изображения алгоритмов функций графического интерфейса просмотрщика журналов

Рисунок К.1 – Графическое изображение алгоритма создания нового экрана для просмотра журналов

Рисунок К.2 – Графическое изображение алгоритма создания таблицы для просмотра журналов

Рисунок К.3 – Графическое изображение алгоритма выведения на экран самых свежих записей

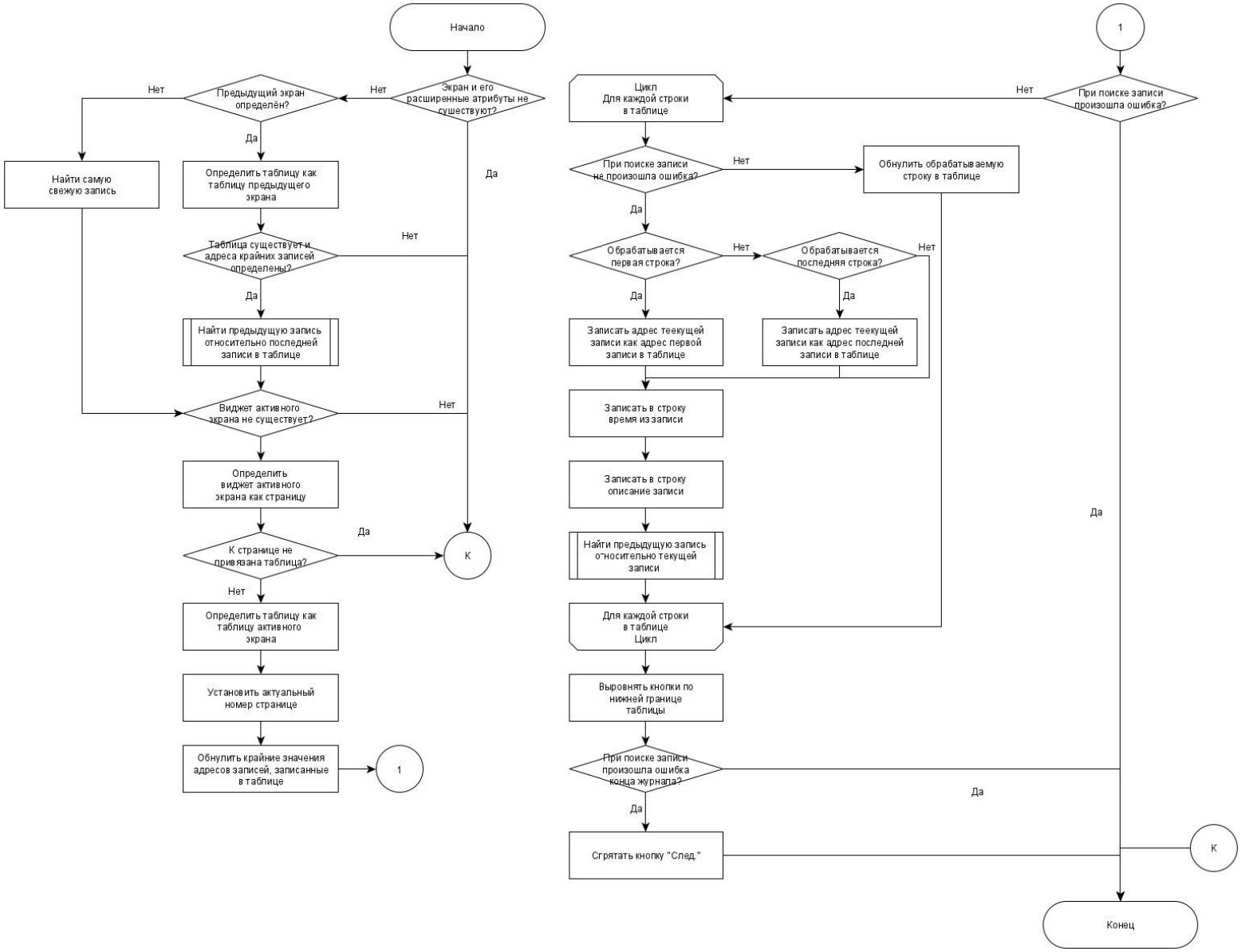

Рисунок К.4 – Графическое изображение алгоритма выведения на экран следующих записей

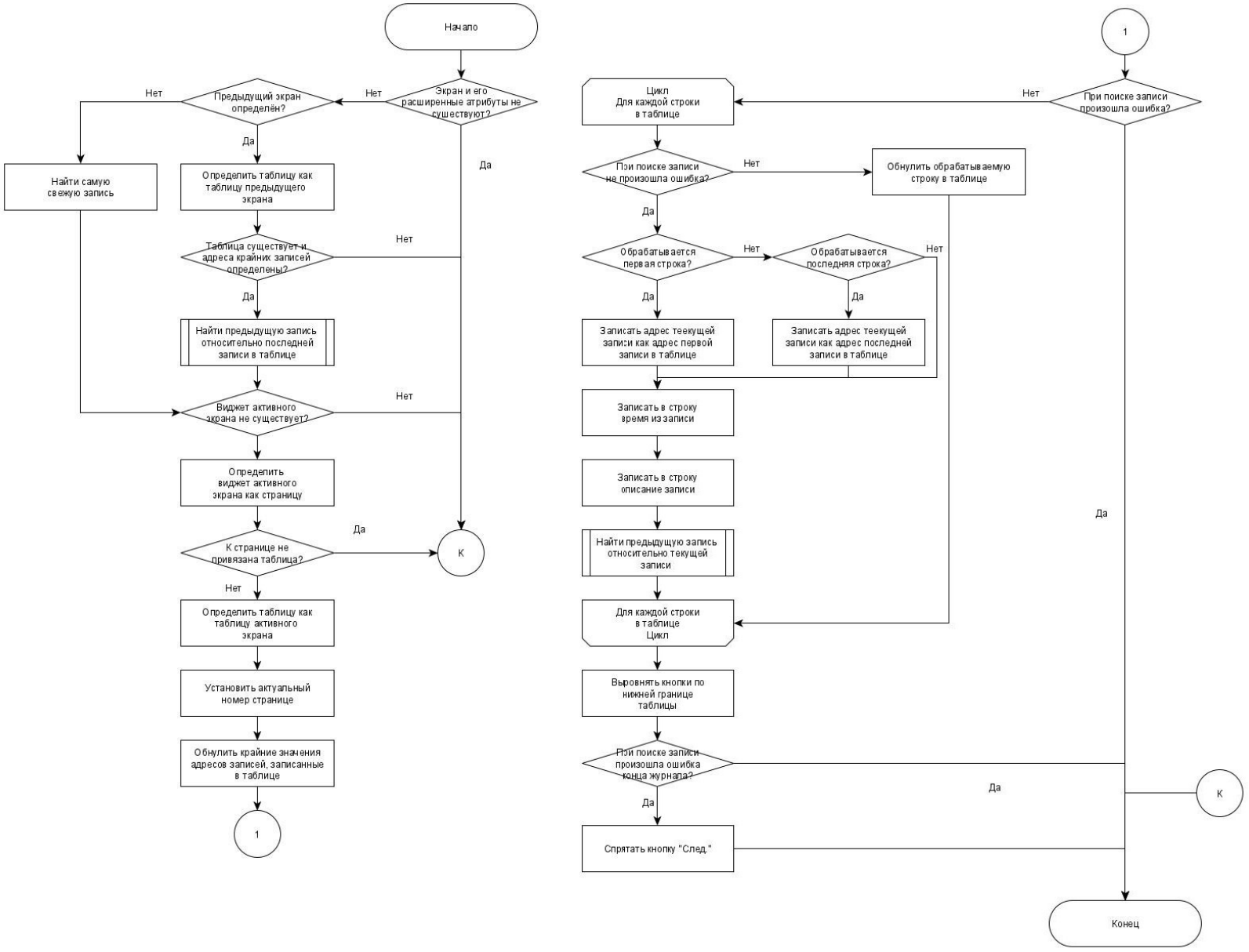

Рисунок К.5 – Графическое изображение алгоритма выведения на экран предыдущих записей

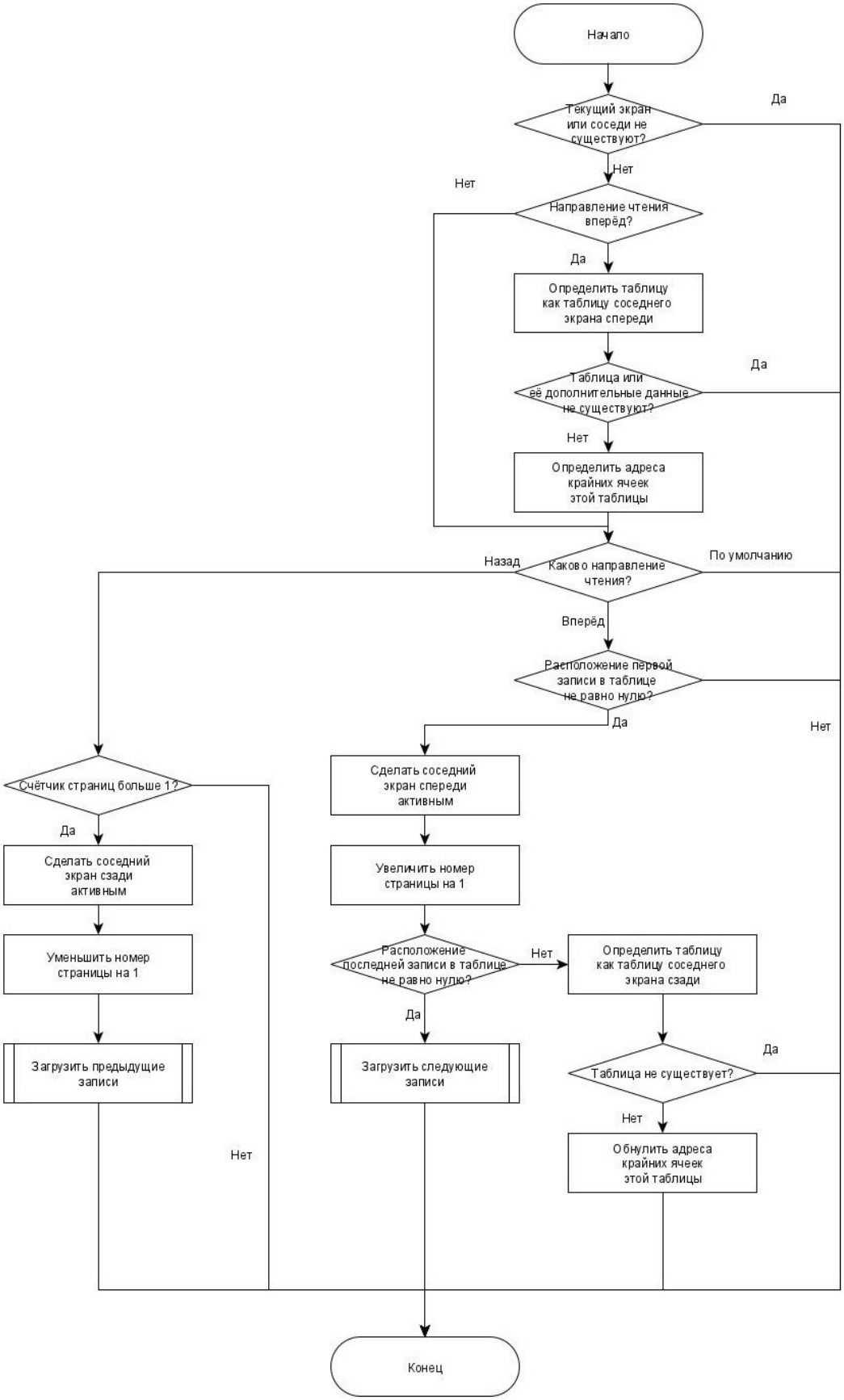

Рисунок К.6 – Графическое изображение алгоритма смены активного экрана

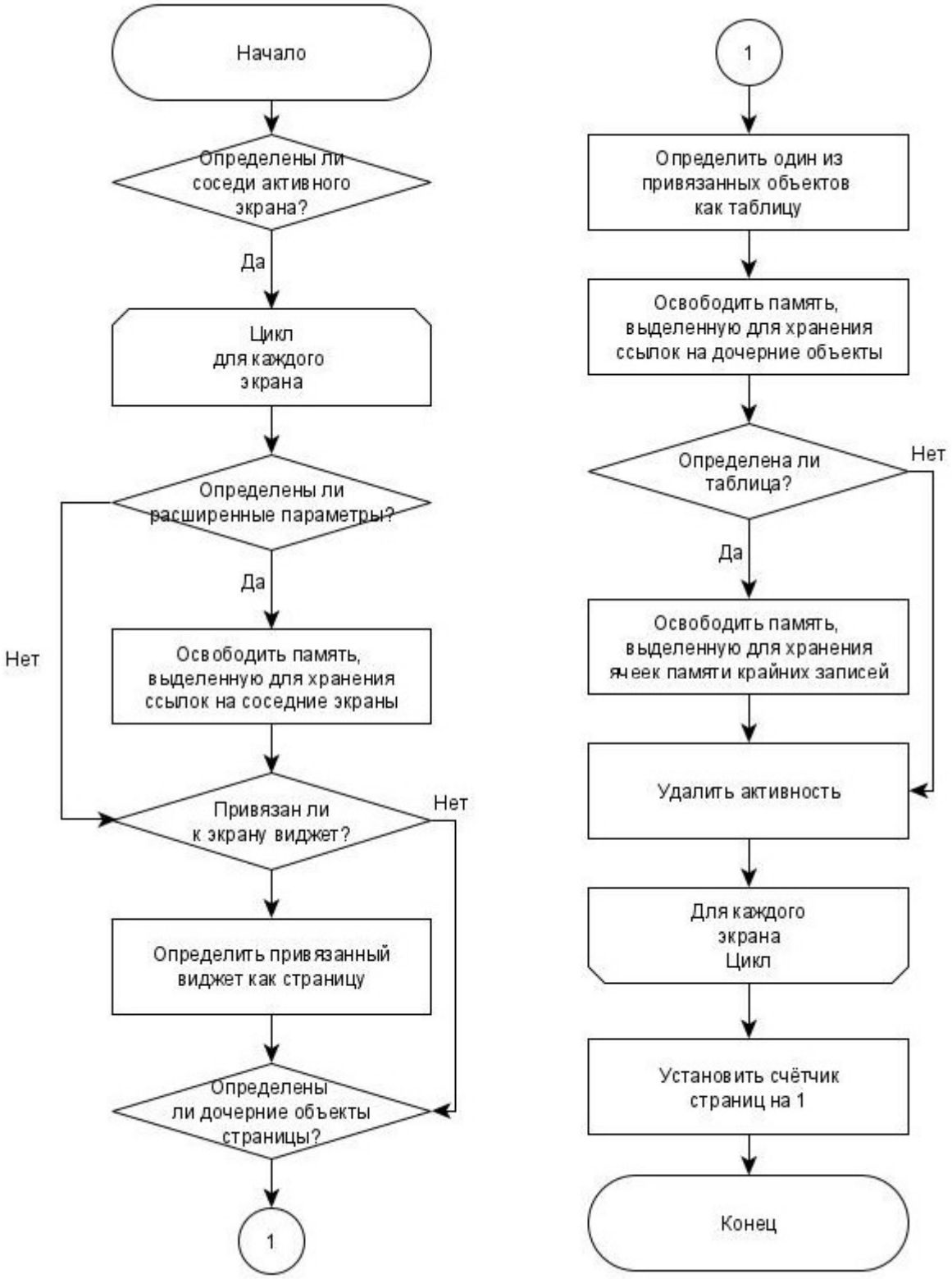

Рисунок К.7 – Графическое изображение алгоритма закрытия просмотрщика журналов

Приложение Л

(обязательное)

Содержимое файла journal_view.h

```
#ifndef UU_JOURNAL_VIEW_H_
#define UU_JOURNAL_VIEW_H_

#include "lvgl/lib/lvgl.h"
#include "widgets/activity.h"
#include "desktop.h"
#include "settings.h"
#include "storage.h"
#include "journal/journal_storage.h"

typedef enum {
      READ_DIRECTION_FORWARD = 0,
      READ_DIRECTION_BACKWARD
} read_direction_t;

typedef struct{
      lv_obj_t * prev;
      lv_obj_t * next;
} neighbors_screen_t;

typedef struct{
      lv_obj_t * table;
      lv_obj_t * label;
      lv_obj_t * prev_pg_btn;
      lv_obj_t * next_pg_btn;
      lv_obj_t * first_pg_btn;
} page_children_t;

typedef struct{
      uint32_t first;
```

```
      uint32_t last;
} edge_records_t;

void show_journal(Journal_type jtype);

#endif /* UU_JOURNAL_H_ */
```

Приложение М

(обязательное)

Содержимое файла journal_view.h

```
#include "journal.h"
#include "journal_view.h"
#include "stdlib.h"
#include "specmenu.h"
#include "phone/phone_utils.h"

static uint32_t journal_page_num = 1;
static char journal_page_num_text[5];
static Journal_type active_journal_type = JEVENT;

static lv_obj_t * create_journal_screen_new();
static void create_new_table(lv_obj_t * page);
static lv_obj_t* get_table(lv_obj_t* screen);
static void align_buttons_to_table(lv_obj_t * page);
static void load_recent_journal_records();
static void load_next_journal_records(lv_obj_t * screen, lv_obj_t *
prev_screen);
static void load_previous_journal_records(lv_obj_t * screen, lv_obj_t *
next_screen);
static void change_journal_screen(lv_obj_t * screen, read_direction_t
direction);
static void close_journal_screen();
static void event_next_btn_cb(lv_obj_t * btn, lv_event_t event);
static void event_prev_btn_cb(lv_obj_t * btn, lv_event_t event);
static void next_records_task_cb(lv_task_t * t);
static void previous_records_task_cb(lv_task_t * t);
static bool close_journal_cb(uint32_t key);


static lv_obj_t* create_journal_screen_new(){
```

```
    lv_obj_t * journal_screen = activity_create();
    lv_obj_t * page = activity_page_create(journal_screen, NULL);
    page_children_t * children = lv_mem_alloc(sizeof(page_children_t));
    lv_obj_set_user_data(page, children);

    create_new_table(journal_screen);

    lv_obj_t *prev_pg_btn = lv_btn_label_create("Пред.", page);
    ((page_children_t*)(page->user_data))->prev_pg_btn = prev_pg_btn;
    lv_obj_set_event_cb(prev_pg_btn, event_prev_btn_cb);
    lv_obj_set_size(prev_pg_btn, (LV_HOR_RES>>1) - 50, 38);

    lv_obj_t *next_pg_btn = lv_btn_label_create("След.", page);
    ((page_children_t*)(page->user_data))->next_pg_btn = next_pg_btn;
    lv_obj_set_event_cb(next_pg_btn, event_next_btn_cb);
    lv_obj_set_size(next_pg_btn, (LV_HOR_RES>>1) - 50, 38);

    lv_obj_t * label_pg_num = lv_label_create(page, NULL);
    ((page_children_t*)(page->user_data))->label = label_pg_num;

    align_buttons_to_table(page);

    return journal_screen;
}

void show_journal(Journal_type jtype){
    active_journal_type=jtype;

    lv_obj_t * journal_screen_1 = create_journal_screen_new();
    lv_obj_t * journal_screen_2 = create_journal_screen_new();
    lv_obj_t * journal_screen_3 = create_journal_screen_new();
```

```
     neighbors_screen_t * neighbors_1 =
lv_mem_alloc(sizeof(neighbors_screen_t));
     if (neighbors_1 == NULL) return;
     neighbors_1->prev = journal_screen_3; neighbors_1->next =
journal_screen_2;
     lv_obj_set_user_data(journal_screen_1, neighbors_1);
     neighbors_screen_t * neighbors_2 =
lv_mem_alloc(sizeof(neighbors_screen_t));
     if (neighbors_2 == NULL) return;
     neighbors_2->prev = journal_screen_1; neighbors_2->next =
journal_screen_3;
     lv_obj_set_user_data(journal_screen_2, neighbors_2);
     neighbors_screen_t * neighbors_3 =
lv_mem_alloc(sizeof(neighbors_screen_t));
     if (neighbors_3 == NULL) return;
     neighbors_3->prev = journal_screen_2; neighbors_3->next =
journal_screen_1;
     lv_obj_set_user_data(journal_screen_3, neighbors_3);

     lv_disp_load_scr(journal_screen_2);
     load_recent_journal_records();

    lv_indev_set_on_key_release_listener(close_journal_cb);
    lv_indev_set_on_key_longpress_listener(close_journal_cb);
}

static void change_journal_screen(lv_obj_t * screen, read_direction_t
direction){
     neighbors_screen_t * neighbors = lv_obj_get_user_data(screen);
     if (screen == NULL || neighbors == NULL) return;
     lv_obj_t * table;
     edge_records_t * edges;
     if (direction == READ_DIRECTION_FORWARD){
```

```
            table = get_table(neighbors->next);
            if(table == NULL || lv_obj_get_user_data(table) == NULL)
return;
            edges = (edge_records_t*)lv_obj_get_user_data(table);
      }
      switch (direction){
      case READ_DIRECTION_FORWARD:
            if(edges->first){
                  lv_disp_load_scr(neighbors->next);
                  journal_page_num++;
                  if(edges->last){
                        lv_task_create(next_records_task_cb, 0 ,
LV_TASK_PRIO_LOW, neighbors);
                  } else {
                        table = get_table(neighbors->prev);
                        if(table == NULL) return;
                        edges =
(edge_records_t*)lv_obj_get_user_data(table);
                        edges->first = 0;
                        edges->last = 0;
                  }
            }


            break;
      case READ_DIRECTION_BACKWARD:
            if(journal_page_num > 1){
                  lv_disp_load_scr(neighbors->prev);
                  journal_page_num--;
                  lv_task_create(previous_records_task_cb, 0 ,
LV_TASK_PRIO_LOW, neighbors);
            }
            break;
```

```
	default:
		return;
	}
}

static lv_obj_t* get_table(lv_obj_t* screen){
	activity_ext_t* ext;
	lv_obj_t* page;
	lv_obj_t* table;
	if(lv_obj_get_ext_attr(screen) == NULL) return NULL;
	ext = lv_obj_get_ext_attr(screen);
	if(ext->widget == NULL) return NULL;
	page = ext->widget;
	if(((page_children_t*)lv_obj_get_user_data(page))->table == NULL)
return NULL;
	table = ((page_children_t*)lv_obj_get_user_data(page))->table;
	if(lv_obj_get_user_data(table) == NULL) return NULL;
	return table;
}

static void create_new_table(lv_obj_t * screen){
	if (screen == NULL || screen->ext_attr == NULL) return;
	activity_ext_t* ext = screen->ext_attr;
	if(ext->widget == NULL) return;
	lv_obj_t * page = ext->widget;
	lv_obj_t * table;
	table = lv_table_create(page, NULL);
	page_children_t * children = lv_obj_get_user_data(page);
	children->table = table;
	edge_records_t* edges = lv_mem_alloc(sizeof(edge_records_t));
	edges->first = 0;
	edges->last = 0;
	lv_obj_set_user_data(table, edges);
```

```
	lv_table_set_col_cnt(table, 2);
	lv_table_set_row_cnt(table, 11);
	lv_obj_align(table, ext->widget, LV_ALIGN_IN_TOP_LEFT, 0, 0);
	lv_table_set_col_width(table, 1, (LV_HOR_RES*71/100));
	lv_table_set_col_width(table, 0, (LV_HOR_RES*29/100));
	lv_table_set_cell_value(table, 0, 0, "Время");
	lv_table_set_cell_type(table, 0, 0, 1);
	lv_table_set_cell_align(table, 0, 0, LV_LABEL_ALIGN_CENTER);
	lv_table_set_cell_value(table, 0, 1, "Событие");
	lv_table_set_cell_type(table, 0, 1, 1);
	lv_table_set_cell_align(table, 0, 1, LV_LABEL_ALIGN_CENTER);
}

static void load_next_journal_records(lv_obj_t* screen, lv_obj_t*
prev_screen){
	if(screen == NULL || screen->ext_attr == NULL) return;
	char buff[512];
	char *strtime;
	activity_ext_t * ext;
	lv_obj_t * page;
	lv_obj_t * table;
	Journal_record record = {.parcedstringbuf =
buff, .parcedbuf_len=sizeof(buff)};
	edge_records_t * edges;
	int ret;

	if(prev_screen != NULL){
		table = get_table(prev_screen);
		if(table != NULL && lv_obj_get_user_data(table) != NULL){
			ret = Journal_get_parced_record(active_journal_type,
PREV,((edge_records_t*)lv_obj_get_user_data(table))->last, &record);
		} else return;
```

```
    } else {
        ret = Journal_get_parced_record(active_journal_type, LAST, 0,
&record);
    }

    ext = screen->ext_attr;
    if (ext->widget == 0) return;
    page = ext->widget;
    if(((page_children_t*)lv_obj_get_user_data(page))->table == NULL)
return;
    table = ((page_children_t*)lv_obj_get_user_data(page))->table;
    itoa(journal_page_num + 1, journal_page_num_text, 10);
    lv_label_set_text(((page_children_t*)lv_obj_get_user_data(page))-
>label, journal_page_num_text);

    edges = lv_obj_get_user_data(table);
    edges->first = 0;
    edges->last = 0;
    if (ret == JEOF || ret == JERROR){
        return;
    }

    for(uint16_t i = 1; i < (lv_table_get_row_cnt(table)); i++){
        if(!ret){
            if(i == 1){
                edges->first = record.storage_addr;
            }else if (i == (lv_table_get_row_cnt(table)) - 1) {
                edges->last = record.storage_addr;
            }

            fatfs_time_to_string(record.datetime, &strtime);
            lv_table_set_cell_value(table, i, 0, strtime);
```

```
                lv_table_set_cell_value(table, i, 1,
record.parcedstringbuf);
                ret = Journal_get_parced_record(active_journal_type,
PREV, record.storage_addr, &record);
          } else {
                lv_table_set_cell_value(table, i, 0, "");
                lv_table_set_cell_value(table, i, 1, "");
          }
     }

     align_buttons_to_table(page);
     if(ret == JEOF){
          lv_obj_set_hidden(((page_children_t*)(page->user_data))-
>next_pg_btn, true);
     }
}

static void load_previous_journal_records(lv_obj_t * screen, lv_obj_t *
next_screen){
     if(screen == NULL || screen->ext_attr == NULL) return;
     char buff[512];
     char *strtime;
     activity_ext_t * ext;
     lv_obj_t * page;
     lv_obj_t * table;
     Journal_record record = {.parcedstringbuf =
buff, .parcedbuf_len=sizeof(buff)};
     edge_records_t * edges;
     int ret;

     if(next_screen == NULL) return;
     table = get_table(next_screen);
     if(lv_obj_get_user_data(table) != NULL){
```

```
          ret = Journal_get_parced_record(active_journal_type, NEXT,
((edge_records_t*)lv_obj_get_user_data(table))->first, &record);
     }
     if (ret == JEOF || ret == JERROR){
          return;
     }

     ext = screen->ext_attr;
     if (ext->widget == 0) return;
     page = ext->widget;
     if(((page_children_t*)lv_obj_get_user_data(page))->table == NULL)
return;
     table = ((page_children_t*)lv_obj_get_user_data(page))->table;
     itoa(journal_page_num - 1, journal_page_num_text, 10);
     lv_label_set_text(((page_children_t*)lv_obj_get_user_data(page))-
>label, journal_page_num_text);

     edges = lv_obj_get_user_data(table);
          for(uint16_t i = (lv_table_get_row_cnt(table)) - 1; i >= 1 ;
i--){
               if(!ret){
                    if(i == 1){
                         edges->first = record.storage_addr;
                    }else if (i == (lv_table_get_row_cnt(table)) - 1){
                         edges->last = record.storage_addr;
                    }

                    fatfs_time_to_string(record.datetime, &strtime);
                    lv_table_set_cell_value(table, i, 0, strtime);
                    lv_table_set_cell_value(table, i, 1,
record.parcedstringbuf);
```

```
                    ret =
Journal_get_parced_record(active_journal_type, NEXT, record.storage_addr,
&record);
               } else {//в действительности не должно быть такой
ситуации
                    lv_table_set_cell_value(table, i, 0, "");
                    lv_table_set_cell_value(table, i, 1, "");

               }
          }

          align_buttons_to_table(page);
          ret = Journal_get_record(active_journal_type, NEXT,
record.storage_addr, &record);
          if(ret == JEOF){
               lv_obj_set_hidden(((page_children_t*)(page->user_data))-
>prev_pg_btn, true);
          }
}

static void load_recent_journal_records(){
     load_next_journal_records(lv_scr_act(), NULL);
     activity_ext_t * ext;
     lv_obj_t * page;
     lv_obj_t * label;

     if(lv_obj_get_ext_attr(lv_scr_act())){
          ext = lv_obj_get_ext_attr(lv_scr_act());
          page = ext->widget;
          if(!page) return;
          label = ((page_children_t*)(page->user_data))->label;
          if(!label) return;
          lv_label_set_text(label, "1");
```

```
          if(!((page_children_t*)(page->user_data))->prev_pg_btn)
return;
          lv_obj_set_hidden(((page_children_t*)(page->user_data))-
>prev_pg_btn, true);
     }
     load_next_journal_records(((neighbors_screen_t*)
(lv_obj_get_user_data(lv_scr_act())))->next, lv_scr_act());
}

static void align_buttons_to_table(lv_obj_t * page){
     lv_obj_align(((page_children_t*)(page->user_data))->next_pg_btn,
((page_children_t*)(page->user_data))->table, LV_ALIGN_OUT_BOTTOM_RIGHT,
0, 0);
     lv_obj_set_hidden(((page_children_t*)(page->user_data))-
>next_pg_btn, false);
     lv_obj_align(((page_children_t*)(page->user_data))->prev_pg_btn,
((page_children_t*)(page->user_data))->table, LV_ALIGN_OUT_BOTTOM_LEFT,
0, 0);
     lv_obj_set_hidden(((page_children_t*)(page->user_data))-
>prev_pg_btn, false);
     lv_obj_align(((page_children_t*)(page->user_data))->label,
((page_children_t*)(page->user_data))->table, LV_ALIGN_OUT_BOTTOM_MID, 0,
10);
     return;
}

static void close_journal_screen(){
     neighbors_screen_t * neighbors =
lv_obj_get_user_data(lv_scr_act());
     if(neighbors == NULL){
          return;
     }
     lv_obj_t * journal_screen[3];
```

```
      journal_screen[0] = lv_scr_act();
      journal_screen[1] = neighbors->next;
      journal_screen[2] = neighbors->prev;
      activity_ext_t * ext;
      lv_obj_t * page;
      lv_obj_t * table;


      for(int i = 0; i < 3; i++){
            if(journal_screen[i]->ext_attr != NULL) {
                  ext = journal_screen[i]->ext_attr;


lv_mem_free((neighbors_screen_t*)lv_obj_get_user_data(journal_screen[i]))
;
            }
            if(ext->widget != NULL)
                  page = ext->widget;
            if(((page_children_t*)lv_obj_get_user_data(page))->table !=
NULL){
                  table = ((page_children_t*)lv_obj_get_user_data(page))-
>table;

      lv_mem_free((page_children_t*)lv_obj_get_user_data(page));
            }
            if(lv_obj_get_user_data(table) != NULL)

      lv_mem_free((edge_records_t*)lv_obj_get_user_data(table));
            activity_delete(journal_screen[i]);
      }

      journal_page_num = 1;
      }
```

```
static void event_next_btn_cb(lv_obj_t * btn, lv_event_t event){
      if(event != LV_EVENT_CLICKED) return;
      change_journal_screen(lv_obj_get_screen(btn),
READ_DIRECTION_FORWARD);
}

static void event_prev_btn_cb(lv_obj_t * btn, lv_event_t event){
      if(event != LV_EVENT_CLICKED) return;
      change_journal_screen(lv_obj_get_screen(btn),
READ_DIRECTION_BACKWARD);
}

static void next_records_task_cb(lv_task_t * t){
      neighbors_screen_t * neighbor = t->user_data;
      load_next_journal_records(neighbor->prev, neighbor->next);
      lv_task_del(t);
}

static void previous_records_task_cb(lv_task_t * t){
      neighbors_screen_t * neighbor = t->user_data;
            load_previous_journal_records(neighbor->next, neighbor->prev);
            lv_task_del(t);
}

static bool close_journal_cb(uint32_t key){
      if (key == KEY_HANG_UP){
            close_journal_screen();
            page_spec_journals_view();
            return false;
      }
      return true;
}
```