4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

4.1 Функции программного обеспечения

Основной функцией разработанного в рамках курсового проекта обеспечения является тестирование носителей флеш-памяти по показателям скорости записи и скорости чтения данных. Однако, типы тестирования, которые можно производить с помощью разработанного программного обеспечения, кардинально различаются.

Среди чётко отделённых функций программного обеспечения необходимо выделить выбор носителя флеш-памяти для тестирования, тест последовательной записи, тест последовательного чтения, тест случайной записи и тест случайного чтения.

При запуске программы пользователь выбирает носитель флеш-памяти, который должен быть протестирован (Приложение Г, Рисунок 1).

После выбора носителя, необходимо выбрать вид тестирования записи, и, как следствие, после тестирования одного из двух видов записи становится возможным тестирование соответствующего вида чтения, т.к. файлы (их названия и размеры), генерируемые конкретным видом теста записи, отличаются от соответствующих файлов другого вида теста записи (Приложение Б).

4.2 Выбор тестируемого носителя флеш-памяти

Выбор носителя флеш-памяти осуществляется путём нажатия на кнопку «Выбор флеш-носителя», после чего открывается меню выбора директорий, содержащее в себе все возможные нескрытые директории и поддиректории системы (Приложение Г, Рисунок 2). Данный функционал реализуется с помощью функции *SHBrowseForFolder*, которая принимает переданную по ссылке структуру типа *BROWSEINFO*. Функция *SHBrowseForFolder* (*shlobj_core.h*) отображает диалоговое окно, позволяющее пользователю выбрать папку оболочки [12]. Структура *BROWSEINFO* (*shlobj_core.h*) содержит параметры для функции *SHBrowseForFolder* и получает сведения о папке, выбранной пользователем [13]. Результат вызова функции *SHBrowseForFolder* помещается в экземпляр структуры типа *LPITEMIDLIST*. Структура *LPITEMIDLIST* (*shtypes.h*) содержит список идентификаторов элементов [14]. В разработанном программном обеспечении в экземпляре структуры *LPITEMIDLIST* будет храниться уникальный идентификатор выбранной директории. Далее вызывается функция *SHGetPathFromIDList*, в которую передается ранее инициализированный экземпляр структуры *LPITEMIDLIST* и строка *path*, в которую будет помещён путь до выбранной

директории. Функция *SHGetPathFromIDList* (*shlobj_core.h*) преобразует список идентификаторов элементов в путь файловой системы [15].

В получившейся строке с путём к выбранной директории выделяются первых три символа, то есть путь к диску, на котором находится выбранная директория, после чего эти три символа передаются в качестве строки в функцию *GetDriveType*. Функция *GetDriveType* (*fileapi.h*) определяет, является ли диск съемным, фиксированным, компакт-диском, диском ОЗУ или сетевым диском [16]. Возвращаемое значение проверяется на равенство *DRIVE_REMOVABLE*. Благодаря такой реализации есть возможность проверять не только носители флэш-памяти с интерфейсом подключения *USB*, но и все остальные съемные носители памяти, тем самым увеличивая диапазон применения разработанного программного продукта. Если проверка пройдена успешно, и диск является съёмным, то путь к директории, в которой будут создаваться тестовые файлы сохраняется в глобальную переменную *currentPath* типа *wstring*. В последующем почти все строковые переменные в разработанном программном продукте будут иметь тип *wstring*.

Wstring – это тип данных в языке программирования *C*, который представляет собой последовательность символов в формате *Unicode*. Он используется для работы с текстовыми данными, такими как строки, и обеспечивает поддержку различных языков и символов [17].

После того, как пользователь успешно выбрал директорию для тестовых файлов, кнопки «Начать тест последовательной записи» и «Начать тест случайной записи» становятся активными.

4.3 Тестирование записи

В разработанном программном продукте присутствует два вида тестирования записи: последовательная запись и случайная запись.

При последовательной записи блоки данных записываются на диск последовательно, то есть друг за другом. С точки зрения реального применения носителей флеш-памяти такой тип записи является идеализированным случаем, так как из-за размеров файлов и их расположения в памяти носителя часто возникает необходимость записывать файл блоками, расположенными друг от друга достаточно далеко. Для контроллера памяти скорость доступа к следующему блоку за текущим блоком быстрее, чем скорость доступа к случайному, и, как следствие, скорость последовательной записи значительно выше, чем скорость случайной записи.

За последовательную запись в разработанном программном обеспечении отвечает функция *WriteTestSequential*.

Сообщая пользователю в текстовом элементе *hStatus*, что операция тестирования последовательной записи началась, функция *WriteTestSequential* отключает весь функционал приложения на время тестирования. Далее формируются строковые переменные, содержащие абсолютные пути к трём

тестовым файлам (*testfile*25*MBSequential.bin*, *testfile*250*MBSequential.bin*, *testfile*2500*MBSequential.bin*). Из названий файлов соответствует и размер, записываемых в ходе теста файлов. Такие размеры являются оптимальными с точки зрения продолжительности тестов и относительно размеров реальных файлов. Далее создаются целочисленные переменные, отвечающие за количество байт в тестовых файлах, то есть за их размер.

Алгоритм тестирования последовательной записи представляет собой формирование строки с таким количеством символов, которое соответствует записываемому размеру файла, после чего фиксируется время начала записи. Для записи в файл используется поток открытия файла *ofstream*, который либо создаёт файл и пишет в него данные, либо перезаписывает уже существующий. При ошибке записи приложение уведомляет пользователя об этом, приостанавливает запись и указывает в каком конкретно тесте произошла ошибка с помощью функции *MessageBox*. Функция *MessageBox* (*winuser.h*) отображает модальное диалоговое окно, содержащее системный значок, набор кнопок и краткое сообщение для конкретного приложения, например сведения о состоянии или ошибке. Окно сообщения возвращает целочисленное значение, указывающее, какую кнопку нажал пользователь [18]. После успешной записи файла фиксируется время окончания операции, рассчитывается время продолжительности записи и средняя скорость записи конкретного файла. Если запись всех трёх файлов прошла успешно, то приложения выводит результаты тестирования (длительность записи в секундах, средняя скорость записи в МБ/с) для каждого из тестов (Приложение Г, Рисунок 3). Тот факт, что запись будет последовательной обеспечивает ещё одно важное условие тестирования: необходимо форматировать накопитель флеш-памяти перед тестированием. Об этом приложение уведомляет перед выбором директории, в которой буду располагаться файлы тестирования. Также в этом уведомлении содержится информация о минимальном требуемом размере накопителя (4 ГБ). Если тестирование последовательной записи прошло без сбоев, то пользователю открывается функционал тестирования последовательного чтения.

Для реализации тестирования случайной записи в разработанном программном продукте используется функция *WriteTestRandom*.

Как и в тесте последовательной записи изначально формируются абсолютные пути трёх файлов (*testfile*250*KBRandom.bin*, *testfile*2500*KBRandom.bin*, *testfile*25000*KBRandom.bin*), которые будут соответствовать тесту случайной записи и содержать информацию о их размерах. Такие размеры выбраны не случайно: время тестирования при таких размерах является оптимальным. Если тестирование последовательной записи предлагало идеальный случай записи файла, то при тестировании случайной записи моделируется худший случай: каждый новый блок данных (4096 байт) пишется в случайное место на накопителе. В реальных же условиях подходы последовательной и случайной записи комбинируются. Это означает что, при привычном использовании носителей контроллеру памяти с точки зрения

длительности операции будет выгодно записать часть файла последовательным образом в некоторую область памяти на носителе, после чего записать ещё одну часть файла последовательно, начиная с другого места в памяти носителя и так далее. Аналогично тестированию последовательной записи создаются целочисленные переменные, хранящие в себе количество байт, которые будут записаны в соответствующие файлы теста.

В общем случае тест начинается с формирования строки с количеством символов, соответствующих размеру записываемого в рамках теста файла. Далее в зависимости от размера файла формируется структура данных типа *vector*, которая будет хранить в себе информацию о уже записанных позициях файла. После этого фиксируется время начала операции случайной записи и открывается файловый поток аналогично открытию потока в тесте последовательной записи.

Под позицией файла в рамках реализации теста понимается число, умножаемое на размер блока. Эта позиция выбирается случайным образом с учётом уже занятых позиций (благодаря учёту уже занятых позиций возможность возникновения ошибка перезаписи данных в одно и то же место файла исключается), после чего она умножается на размер блока и, как следствие, получается номер конкретного байта файла, с которого будет начинаться запись блока (абсолютная позиция записи в файл). Далее решается задача вычисления количества байт, ведь если размер файла в байтах не кратен размеру блока (4096 байт), то в одну из итераций цикла придётся записать количество байт, меньшее чем размер блока. Для решения этой задачи сравниваются два значения: размер блока и разность общего размера файла и текущей выбранной случайным образом абсолютной позиции записи в файл. Если размер блока больше вышеупомянутой разности, то тогда количество байт, которые необходимо записать в текущей итерации, равно разности общего размера файла и текущей выбранной случайным образом абсолютной позиции записи в файл. В других случаях количество байт для записи в файл в текущей итерации равно размеру блока. Данные соответствующего размера записываются в файл, начиная с соответствующей позиции в файле, после чего позиция помечается как записанная. Последующие итерации будут выполняться таким же образом, как и итерация, описанная выше в этом абзаце, пока не будет записано количество байт, которое должно быть записано в контексте конкретного тестового файла.

После того, как все байты записаны, фиксируется время окончания случайной записи файла. Обработка ошибок при выполнении теста случайной записи реализована аналогично обработке ошибок теста последовательной записи. Если же все три теста случайной записи пройдены без ошибок, то пользователь получает результаты тестов в виде, аналогичном виду результатов тестирования последовательной записи (длительность записи в секундах, средняя скорость записи в МБ/с) и ему становится доступен функционал тестирования случайного чтения (Приложение В, лист 1).

4.4 Тестирование чтения

В разработанном программном продукте присутствует два вида тестирования чтения: последовательное чтение и случайное чтение.

Как и в случае с записью, тестирование последовательного чтения является идеализированным случаем и не отражает реальную скорость чтения файлов при повседневном использовании носителя флэш-памяти. Как следствие, скорость последовательного чтения больше скорости случайного чтения, так как операция доступа к следующему за текущим блоком выполняется контроллером памяти быстрее операции доступа к случайному блоку памяти. В свою очередь тестирование случайного чтения является худшим случаем, так как обращение каждый раз происходит к случайному блоку файла. В реальности же одна часть файла может быть записана последовательно в памяти накопителя, и вторая часть будет записана последовательно, но начиная уже с другого места в памяти носителя.

Важным аспектом тестирования записи является требование извлечения накопителя перед тестированием, так как считываются те же самые файлы, что записываются при тестах записи, в ходе которых они сохраняются в оперативной памяти. Для того, чтобы избежать некорректных результатов тестирования (слишком большие скорости чтения), флэш-накопитель извлекается и соответствующие файлы удаляются из оперативной памяти.

За тестирование последовательного чтения в разработанном программном обеспечении отвечает функция *ReadTestSequential*.

Функция *ReadTestSequential* аналогично тесту последовательной записи формирует абсолютные пути файлов, которые будут считываться последовательно в ходе теста, и целочисленные переменные, хранящие в себе размер файлов, подлежащих к чтению. Под читаемый файл выделяется буфер, в который данные из файла будут считываться, после чего фиксируется время начала теста. Далее создаётся файловый поток *ifstream*, который используется для чтения файлов. С помощью функций этого потока происходит чтение данных из файла в буфер. Если происходит ошибка чтения, то приложение уведомляет об этом пользователя, указывая на тест, во время которого произошла ошибка, после чего останавливает выполнение теста. Если все три теста успешно пройдены, то пользователю выводятся результаты тестов (длительности чтения в секундах, средняя скорость последовательного чтения в МБ/с) с помощью функции *MessageBox*. Вне зависимости от того, прошёл тест успешно или же произошла ошибка чтения файлов, файлы, которые были записаны и прочитаны с помощью соответствующих тестов, удаляются.

Тестирование случайного чтения реализовано функцией *ReadTestRandom*.

Аналогично тестированию случайной записи в ходе работы кода функции *ReadTestRandom* изначально формируются абсолютные пути к файлам, которые будут читаться в ходе тестирования, и целочисленные

переменные, отвечающие за размеры файлов. В общем случае тестирования случайного чтения происходит всё то же самое, что и при тестировании случайной записи. Основное различие в том, что при случайном чтении используется файловый поток *ifstream* для открытия и чтения файла. Все остальные операции в рамках одной итерации цикла, идущего пока все байты соответствующего файла не будут считаны (выбор позиции указателя чтения в файле, нахождение абсолютной позиции указателя чтения в файле, решение задачи о нахождение размера считывания, пометка позиции как прочитанной и так далее), остаются по своей сути такими же, как и при тестировании случайной записи. Аналогично тестированию последовательного чтения выводятся результаты тестирования: длительность случайного чтения в секундах, средняя скорость (далее – скорость) случайного чтения в МБ/с. Если же происходит ошибка во время теста, то пользователю сообщается, что произошёл сбой при чтении файла, указывая при это на конкретный тест. Вне зависимости от того, прошло тестирование успешно или была ошибка, тестовые файлы, которые читаются в ходе теста случайного чтения, удаляются.

Как следствие удаления файлов после тестов чтения, перед повторным тестированием одного из двух видов чтения необходимо провести тестирование соответствующего вида записи для того, чтобы необходимые файлы для тестирования чтения существовали на носителе.

Из вышеописанных методов тестирования и реализованных в программном коде функций складывается функционал разработанного приложения, которое может быть использовано для предоставления данных о производительности носителей флэш-памяти, которые впоследствии могут быть использованы для сравнения конкретных моделей носителей (Приложение В, лист 2).