Для взаємодії з апаратною частиною системи необхідно спеціальне програмне забезпечення — драйвер. Який візьме на себе функції передачі сигналів керування та отримання результатів вимірювань.

Драйвери взаємодіють с досить примітивними пристроями, що керуються мікроконтролерами, в яких може бути досить складна організація пам’яті та порядку байтів при передачі даних. Іноді вони навіть мають можливість працювати з окремими бітами. Отже для написання драйверів слід використовувати мову досить низького рівня, обов’язково з можливістю прямої взаємодії з пам’яттю. Саме такою мовою є C++.

З огляду не те, що операційна система Windows є найбільш популярною серед користувачів персональних комп’ютерів, а написання програм драйверів потребує взаємодії з операційною системою на низькому рівні інші операційні системи, для керування приладом, не розглядались. А отже для розробки драйвера керування приладом доречно використати середовище від того ж виробника що і операційна система — Microsoft Visual Studio.

Для забезпечення принципу модульності драйвер приладу буде реалізований у вигляді окремої бібліотеки, яку можна буде підключити до програми для виводу тривимірної графіки, написаної на мові більш високого рівня. Таким чином драйвер та апаратна частина стануть лише джерелом даних для системи відображення. А саму систему відображення можна використовувати окремо, для виводу збережених раніше даних.

Принцип модульності зобов’язує організовувати програми як сукупність невеликих незалежних блоків, які називаються модулями, структура і поведінка яких підкоряються певним правилам. Використання модульного програмування дозволяє спростити тестування програми і виявлення помилок. Апаратно-залежні частини можуть бути жорстко відділені від інших частин, що покращує мобільність створюваних програм.

Модуль — функціонально закінчений фрагмент програми. У багатьох мовах (але далеко не обов'язково) оформляється у вигляді окремого файлу з вихідним кодом або пойменованої безперервної її частини. Деякі мови передбачають об'єднання модулів в пакети.[XXXXXX Module]

Створення бібліотеки динамічного зв’язування (Dynamic Link Library, DLL) необхідно виконати деякі дії.

В середовищі Microsoft Visual Studio необхідно створити новий проект типу Win32. У вікні створення проекту вибрати «Пустий проект». Після компіляції буде отримано DLL, бібліотеку імпорту (.lib) та бібліотеку експорту (.exp).

Перш за все необхідно створити заготовочний файл з сигнатурами методів, який може бути використаний при підключенні DLL з бібліотекою імпорту. В цьому файлі повинні бути описані усі сигнатури функцій, що мають бути викликані зовнішніми програмами.

Всі ці функції мають бути помічені спеціальними ключовими словами. Наприклад, якщо функцію з сигнатурою «*void Start();*» необхідно викликати з іншої програми то записати так «*extern "C" \_\_declspec(dllexport) void Start();*». Так само необхідно записати функцію і в фалі з реалізаціє.

Перший варіант використовувати DLL разом з бібліотекою імпорту (.lib), яка виходить при компіляції проекту бібліотеки. Цей метод дуже простий, так як в такому випадку потрібно просто включити заголовки бібліотеки та саму бібліотеку в проект та просто викликати необхідні функції.

Це буде чудово працювати, якщо є заголовки і бібліотека імпорту знаходиться в каталозі, прописаному в бібліотечних шляхах. Перед запуском програми, потрібно переконатись, що DLL знаходиться в каталозі, прописаному в системній змінній PATH або в тому ж каталозі, що і виконуваний файл, інакше отримаєте повідомлення про помилку. Однак якщо декілька програм використовують цю DLL, потрібна всього одна її копія, що лежить, наприклад, в каталозі Windows\System.

Другий варіант — завантажити DLL "на льоту". Це потрібно в разі, якщо DLL створено сторонніми розробниками і немає заголовків і бібліотеки імпорту.[XXXXXXXXXX DLL]

На відміну від першого, де бібліотека підключається ще на етапі компіляції, тобто якщо відповідні файли не знайдено буде видано помилку компіляції і проект не буде побудований. Тут бібліотека та всі її функції завантажуються динамічно на етапі виконання програми.

Другий метод є більш гнучким, оскільки він дозволяє завантажувати бібліотеки вибірково, або при виконанні деяких умов. Таким чином можна організувати динамічне завантаження лише тих частин програми які необхідні в даний момент користувачу, і вивантажувати коли вони вже не потрібні. В той же час як при першому варіанті всі бібліотеки гарантовано будуть завантажені перед початком виконання.

Далі описано функції, що необхідно викликати та кроки, що необхідно заробити щоб завантажити DLL-бібліотеку. Всі описані функції є функціями прикладного інтерфейс Windows — WinApi, тому для їх використання необхідно включити заготовочний файл windows.h.

Для завантаження бібліотеки потрібно викликати функцію LoadLibrary. Вона приймає єдиний аргумент — строку зі шляхом до бібліотеки, яку необхідно завантажити. Шлях може бути абсолютний, відносний або містити тільки назву файлу. В останньому випадку пошук буде проводитись спочатку в каталозі з виконуваним файлом, а потім у всіх каталогах що опсані в системній змінній PATH. Виклик функції поверне дескриптор бібліотеки типу HMODULE або значення NULL, якщо бібліотеку не знайдено.

Тепер бібліотека є завантаженою в пам’ять і доступна для використання. Далі потрібно отримати адреси функцій, які будуть викликатись з неї. Для цього використовують функцію GetProcAddress, яка отримує на вхід дескриптор бібліотеки та ім’я функції яку необхідно завантажити. Функція GetProcAddress повертає значення типу FARPROC, це і є вказівник на потрібну функцію. Отримане значення за допомогою приведення типів необхідно привести до правильної сигнатури. Найпростіший варіант це визначити новий тип вказівника на функцію з необхідною кількістю та типами параметрів та типом значення, що вона повертає, визначити змінну цього типи та виконати приведення до нього. Тепер з отриманою змінною вказівника можна поводитись як зі звичайною функцією — викликати з необхідними їй списком параметрів та отримувати результат її роботи.

Якщо функцію з заданим ім’ям не знайдено в бібліотеці то GetProcAddress поверне значення NULL. Це свідчить про те, що сталась помилка і подальше використання бібліотеки може привести до невизначеної поведінки всієї програми, оскільки з великою імовірністю завантажено неправильну бібліотеку. Тому необхідно контролювати отримане значення вказівника і вразі помилки переривати роботу програми.

Системна функція FreeLibrary вивантажує бібліотеку коли вона вже непотрібна. В якості параметру вона отримує дескриптор бібліотеки, що необхідно вивантажити, та повертає логічне значення, що показує чи успішно була вивантажена бібліотека.

Технічно функція FreeLibrary не вивантажує бібліотеку, а лише зменшує кількість вказівників на неї. Власне вивантаження з пам’яті буде виконано лише тоді коли лічильник вказівників досягне нуля. Це так званий механізм підрахунку посилань. Він дозволяє економити ресурси системи коли одну бібліотеку використовують декілька додатків. Перший виклик LoadLibrary завантажує бібліотеку в пам’ять, а всі інші лише збільшують лічильник посилань.

Тому дуже важливо викликати FreeLibrary кожен раз при завершенні програми, навіть при аварійному, лічильник посилань ніколи не дійде до нуля і бібліотека не буде вивантажена, аж до перезавантаження комп’ютера. В цей час її файл неможливо пермітити, перейменувати або перекомпілювати проект, оскілки файл вважається зайнятим.

З DLL бібліотеки можна викликати лише окремі функції. А при розробці програмного коду бібліотеки в об’єктно-орієнтованому стилі весь функціонал розділений між об’єктами, які досить часто повинні бути в пам’яті на протязі всієї роботи програми. Виникає ситуація при якій об’єкт повинен бути створений перед викликом функції та видалятись після кількох викликів різних функцій. Навіть якщо ствоювати об’єкт «ліниво», тобто при кожному виклику кожної функції перевіряти чи сторений об’єкт, якщо ні то створити його, залишається невирішеною проблема видалення.

Найбільш простий варіант це в бібліотеці створити функції, що необхідно вилки до початку роботи та після її завершення. Але може ускладнити використання бібліотеки, оскільки розробники часто забувають викликати такі функції, особливо ті що пов’язані з очисткою ресурсів, адже їх влив не роботу програми не завжди очевидний.

Для вирішення цієї проблеми можна додати спеціальну функцію DllMain, яка буде автоматично викликана при виникненні однієї з описаних нижче подій. Щоб визначити, що це за подія необхідно порівняти значення параметру fdwReason з однією з наступних констант:

* DLL\_PROCESS\_ATTACH — бібліотека була завантажена в віртуальний адресний простір поточного процесу, як результат запуску процесу або виклику LoadLibrary.
* DLL\_PROCESS\_DETACH - бібліотека вивантажена з віртуального адресного простору процесу, що викликав через невдалу спробу завантаження або тому, що лічильник посилань досяг нуля.
* DLL\_THREAD\_ATTACH — Поточний процес створює новий потік. Коли це відбувається, система викликає функцію точки входу всіх бібліотек DLL, що в даний час приєднані до цього процесу. Виклик виконується в контексті нового потоку.
* DLL\_THREAD\_DETACH — Потік завершився повністю. Якщо DLL зберігає вказівник на виділену пам'ять, потрібно використовувати цю можливість, щоб звільнити її. Система викликає функцію точки входу всіх завантажених в даний момент бібліотек DLL з цим значенням.[XXXXXX DLLMAIN]

Найбільш цікавими є перші дві події, тому, що саме при їх виникненні можна виконати ініціалізацію і очитку ресурсів в середині бібліотеки.

При додавані до порожнього проекту функції DllMain можна отримати помилку, що така функція уже визначена. Щоб позбавитись від неї необхідно просто іще раз перекомпілювати проект.

Для роботи з USB використано вбудований USB-модуль мікроконтролера PIC18F2550. Розробка драйвера «з нуля» потребує багато часу та ресурсів. Тому для полегшення розробки було викорстано вихідні коди універсального драйвера фірми Microchip для операційної системи Windows (Microchip General Purpose USB Windows).

Він включає бібліотечні файли, що містять вихідні коди програм для мікроконтролерів різних типів, та демонстраційні приклади, з яких можна скомпілювати програми прошивки для різних видів пристроїв. Головний приклад побудований таким чином, щоб можна було видалити файли, що не є частиною бібліотеки і використати його як базу для створення своїх програм.

Для спрощення розробки програм керування пристроями зі сторони персонального комп’ютера в комплект поставки входить DLL бібліотека для безпосередньої взаємодії з описаною вище частиною бібліотеки для мікроконтролера. В ній є описано функції для з’єднання з пристроєм, прийому та передачі даних на пристрій та закриття з’єднання.

Ця бібліотека не взаємодіє безпосередньо з контролером шини USB. Взаємодія відбувається через спеціальну частину драйвера, що також входить в комплект поставки. Щоб успішно встановити з’єднання з пристроєм при першому підключенні коли операційна система запитає драйвер необхідно вказати шлях саме до паки з цією частиною драйвера.

Разом з бібліотекою поставляються демонстраційні приклади, які показують особливості використання кожної її частини. Всі приклади легко перекомпілювати, що сильно спрощує використання бібліотеки навіть з урахуванням того, що документація є лише англійською мовою.

ООП це добре

Він слідкує за станом з’єднання, сам завантажує та очищає необхідні ресурси

USBDevice основний клас бібліотеки драйвера. Він є об’єктним відображенням підключеного пристрою з яким можна обмінюватись інформацією, тобто до якого можна підключитись, послати запит та отримати відповідь. Всі інші реалізують більш конкретний функціонал драйвера. Цей клас є найбільш низькорівненвим, він безпосередньо взаємодіє з універсальним драйвером, що розроблений фірмою Microchip, за допомогою засобів, що надає операційна система Windows.

Слід зауважити, що даний клас є абстрактним тобто не представляє конкретного пристрою і має лише функціонал доступний для будь-якого USB пристрою, що може бути розроблений за подібною схемою. Додавати реалізацію функціоналу для конкретного пристрою можна за допомогою механізму наслідування. Додавати реалізацію функціоналу означає додати можливість відправляти конкретні повідомлення, що можуть бути оброблені на стороні мікроконтролера.

Ініціалізація пристрою проходить в два етапи. На першому завантажується бібліотека з універсальним драйвером та проводиться пошук в ній необхідних для роботи функцій. На другому проводиться пошук підключених до USB-порту пристроїв. В разі якщо файл бібліотеки, одну або більше функцій з неї не знайдено або немає підключених пристроїв буде отримано повідомлення про помилку та викликане аварійне завершення роботи, а всі спроби послати повідомлення будуть завершуватись з помилкою. Для спрощення обидві ці дії об’єднані в одному методі Connect, однак для більшої гнучкості кожну с цих дій можна викликати окремо.

Всі помилки, що можуть повернути виклики методів USBDevice мають унікальний код, а отже однозначно вказують на причину виникнення помилки.

Коди помилок записані у вигляді констант у спеціальному файлі заголовку (USBDeviceError.h), який можна підключити при використанні бібліотеки драйвера характерографа із програми на мові С або С++. Однак можливі варіанти використання бібліотек з іншими мовами програмування в які неможливо підключити згаданий файл заголовку. В цьому випадку доведеться працювати безпосередньо з числовими значеннями кодів. В таблиці наведено константи можливих помилок, їх числові значення та короткий опис.

Таблиця

|  |  |  |
| --- | --- | --- |
| Константа | Код | Опис |
| STATE\_OK | 0 | Виклик функції завершися успішно. |
| NO\_DLL | 1 | Помилка при завантаженні бібліотеки — файл бібліотеки не знайдено. |
| NO\_FUNCTION | 2 | Помилка при завантаженні бібліотеки — неможливо отримати адресу однієї або більше функцій, що необхідно завантажити. Тобто в завантаженому файлі немає функції з відповідним ім’ям. |
| NO\_DEVICE\_CONNECTED | 3 | Пристрій не знайдено. Необхідно пересвідчитись, що підключено до USB порту комп’ютера. |
| NOT\_FULL\_DATA | 4 | Помилка при отриманні даних з мікроконтролера — прийнято менше ніж очікувалося. |
| INVALID\_PIPE\_HANDLE | 5 | Не вдалось відкрити з’єднання. |
| CALL\_GET\_LAST\_ERROR | 6 | Виклик однієї з системних функцій Windows завершився з помилкою. Щоб отримати більше інформації необхідно викликати функцію GetLastError(). |
| SESSION\_ALREADY\_OPEN | 7 | Спроба відкрити другу сесію, коли перша ще не закрита. Виникає при використанні кількох потоків, або внаслідок неуважного використання методів при ручному відкритті сесій. |
| COMMUNICATION\_ERROR | 8 | Помилка передачі. Код відповіді від мікроконтролера не співпадає з очікуваним. |

Коди помилок від 0 до 1000 є зарезервованими використання при подальшій розробці даного класу. Похідні класи можуть додавати нові коди помилок починаючи зі значення , що не входять в визначений діапазон.

Для передачі даних використовується метод SendReceive, який посилає дані вказаної довжини та приймає дані у відповідь. Якщо дані не були відправлені або прийняті за час, що вказується при виклику, то метод завершиться з помилкою.

Передача даних реалізована у вигляді, так званих, сесій. Тобто фактичне з’єднання з пристроєм відкривається безпосередньо перед відправленням даних (за допомогою виклику методу SendReceive) та закривається одразу після отримання відповіді, або в разі збою при передачі. Це дозоляє зменшити затрати ресурсів.

Власне передача даних відбувається за допомогою іменованих каналів Windows (Named Pipes). Це засіб дозволяє організувати передачу даних між локальними процесами, а також між процесами, запущеними на різних робочих станціях в мережі.

Канали типу Pipe найбільше схожі на файли, тому вони досить прості у використанні.

Через канал можна передавати дані тільки між двома процесами. Один з процесів створює канал, інший відкриває його. Після цього обидва процеси можуть передавати дані через канал в одно або двосторонньому режимі, використовуючи для цього добре знайомі функції, призначені для роботи з файлами, такі як ReadFile і WriteFile. Слід зауважити, що додатки можуть виконувати синхронні або асинхронні операції з каналами Pipe, аналогічно тому, як це можна робити з файлами.[XXXXXX 1]

В класі USBDevice є ще один метод призначений для передачі даних — SendRequest. На відміну від попереднього він сам відкриває та закриває сесії. Цей метод отримує параметром вказівник на об’єкт класу RequestToDevice або похідного від нього.

Клас RequestToDevice є об’єктним представленням запиту. Він є абстрактним та розроблений для спрощення рутинних операцій, таких як створення та видалення буфера, слідкування кількістю даних у буферах та убезпеченням від переповнення. Похідні класи можуть реалізовувати більш складну логіку для наповнення буфера даними для відправки та обробки прийнятої відповіді. Таким чином реалізовано логічне розділення обов’язків при якому кожен клас похідний від RequestToDevice є конкретною дією, що може бути виконана пристроєм.

Логер

Фабрика

Шляхи інклудів

Юнікод

Тестовий проект

SendReceive синхронна

Module

<https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5>

DLL

<http://www.xserver.ru/computer/sredaprogr/msvc/2/>

1

<http://www.frolov-lib.ru/books/bsp/v27/ch2_3.htm>

DLLMAIN

<https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms682583(v=vs.85).aspx>