СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc10234253)

[1 ОБЗОР ИСТОЧНИКОВ 6](#_Toc10234254)

[1.1 Теоретические сведения 6](#_Toc10234255)

[1.2 Обзор аналогов 6](#_Toc10234256)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ](#_Toc10234257) 7

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 8](#_Toc10234258)8

[3.1 Класс MainView](#_Toc10234259) 8

[3.2 Класс FSCrawler](#_Toc10234260) 9

[4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 10](#_Toc10234263)0

[4.1 Фоновое обновление строки состояния 10](#_Toc10234264)0

[4.2 Алгоритм открытия новой директории 11](#_Toc10234265)1

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 13](#_Toc10234266)3

[ЗАКЛЮЧЕНИЕ](#_Toc10234267) 17

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ](#_Toc10234268) 18

[ПРИЛОЖЕНИЕ А](#_Toc10234269) 19

[ПРИЛОЖЕНИЕ Б](#_Toc10234270) 20

[ПРИЛОЖЕНИЕ В](#_Toc10234271) 21

[ПРИЛОЖЕНИЕ Г](#_Toc10234272) 22

[ПРИЛОЖЕНИЕ Д](#_Toc10234273) 23

# **ВВЕДЕНИЕ**

Электронная книга — это устройство, предназначенное для отображения текстовой информации в электронном виде, например электронных книг. Эти устройства отличаются от планшетных компьютеров ограниченной функциональностью, но зато существенно большим временем автономной работы. Последнее достигается за счет использования особой технологии изготовления дисплея, так называемой «электронной бумаги» или «электронных чернил» (E-Ink). Такой дисплей имитирует бумажный лист, отображает обычно до 16 оттенков серого цвета, но при этом потребляет энергию только при перелистывании страницы. Основным преимуществом данных экранов является их бережное отношение к зрительной системе читающего, поскольку экран работает в отращенном свете нагрузка на глаза существенно ниже.  
 Однако экраны, созданные по технологии E-Ink значительно дороже обычных TFT панелей, за счет этого даже самые простые электронные книги имеют высокую цену, что препятствует их массовому распространению.   
 Большинство современных электронных книг имеют сенсорное управление, повреждение которого лишает возможности пользоваться устройством и вынуждает пользователя приобретать новое. Целью данного курсового проекта стало создание нового встроенного ПО электронной книги позволяющее заменить сенсорное управление на управление при помощи встроенных клавиш для придания второй жизни поврежденным экземплярам.

# **1 ОБЗОР ИСТОЧНИКОВ**

# **1.1 Теоретические сведения**

Аппаратную часть электронной книги PocketBook составляют:

* + - Дисплей E-Ink выполненный по технологии Pearl, сенсорный, 16 градаций серого.
    - 4 аппаратные клавиши для взаимодействия с пользовательским интерфейсом программ-читалок, однако никак не используемых в главном меню.
    - 32-битный ARM процессор FreeScale со встроенным специализированным драйвером E-Ink экрана.
    - 128МБ ОЗУ.
    - До 2ГБ ПЗ. Возможно использование большего объёма памяти при подключении micro-cd карточек.
    - USART интерфейс для тестирования и отладки.

Электронная книга работает под управлением встраиваемого ядра Linux (embeded linux), поэтому все особенности разработки ПО, встречающиеся при разработке для ОС Linux, актуальны в данном проекте.  
 Для написания новых программ для данной электронной книги используется фирменный SDK (Software Development Kit). Для отладки используется переходник USART to USB подключенный к соответствующему порту электронной книги.

## **1.2 Обзор аналогов**

На данный момент не существует аналогов разрабатываемого продукта, таким образом он может стать актуален среди пользователей, имеющих электронные книги с поврежденным сенсорным управлением, но рабочим экраном.

# **2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

Данное приложение состоит из нескольких модулей, главным из которых является класс Bookshelf, который настраивает работу всех остальных, путем установки обработчика нажатий клавиш, установка обработчика сигнала завершения работы SIGTERM, а так же запуском дополнительного потока.

За отображение пользовательского интерфейса отвечает класс MainView. Пользовательский интерфейс включает в себя список для отображения файлов и папок в текущем каталоге, строку состояния, в которой располагается уровень заряда батареи, текущие время, а так же индикатор подключения зарядного устройства, внизу экрана располагаются надписи, позволяющие узнать функционал кнопок управления. Кроме того данный модуль занимается отрисовкой наименований файлов, а так же осуществляет логику переключения выбранного для открытия файла.

Для получения информации о файловой системе (ФС) используется модуль FSCrawler, который получает список файлов, а так же определяет их тип (файл, папка, ссылка на каталог назад). Его работа основана на использовании системной библиотеки dirent. К этому модулю обращается MainView при перерисовке отображаемых на экране наименований файлов, а так же для определения и соответствующего отображения каталогов.

Для выведения актуального времени и заряда батареи строка состояния обновляется в фоновом режиме благодаря использованию дополнительного потока. Реализация многопоточности производится с использованием системной библиотеки pthread. Получения состояния заряда батареи производится путем чтения системного файла capacity по пути "/sys/class/power\_supply/battery/capacity", а статус подключения зарядного устройства определяется чтением из файла status по пути "/sys/class/power\_supply/battery/status".

Обработчик сигнала SIGTERM используется для установки заставки после выключения книги. В связи с особенностями технологии E-Ink эта заставка будет постоянно отображаться на экране.

Функции для работы с аппаратной частью электронной книги находятся в библиотеке inkview, распространяемой вместе с официальным SDK.

Для настройки электронной книги и для установки вновь разработанного приложения в систему использовался переходник USB to USART выполненный на базе микроконтроллера arduino и схемы согласования уровней.

Структурная схема представлена в приложении А.

# **3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

В данном разделе функционального проектирования рассматриваются основные функциональные составляющие проектируемого программного обеспечения.

Диаграмма классов продемонстрирована в приложении Б.

## **3.1 Класс MainView**

Основное предназначение – это отображения списка файлов и папок, отображения строки состояния и подписей кнопок.

Поля класса MainView:

* + - FSCrawler fscrawler – экземпляр класса FSCrawler, который в свою очередь имеет методы для открытия директорий и получения списка файлов;
    - ifont \*font\_small – шрифт, используемый для отрисовки малых текстовых строк интерфейса;
    - ifont \*font\_big – шрифт, используемый для отрисовки больших текстовых строк интерфейса;
    - int currentPosition – индекс текущего выбранного файла на экране;
    - int currentBase – базовый индекс текущего выбранного файла на экране в полном списке файлов предоставляемых классом FSCrawler отладки;

Методы класса MainWindow:

* + - void drawUi () – метод производящий начальную инициализацию экрана и отрисовку базового интерфейса;
    - void updateBase() – метод, обновляющий список файлов исходя из текущего индекса currentBase;
    - void updatePosition() – метод, обновляющий выделенный файл исходя из текущего индекса currentPosition;
    - void updateTray() – метод, обновляющая информацию в строке состояния;
    - void update() – метод, производящий полное обновление данных на экране;
    - void forceNext() – метод, используемый обработчиком нажатий клавиатуры для выбора следующего файла;
    - void forcePrev() – метод, используемый обработчиком нажатий клавиатуры для выбора предыдущего файла;
    - void open() – метод, используемый обработчиком нажатий клавиатуры для открытия выбранного файла либо папки;
    - void showLogo () – метод, используемый обработчиком сигнала для отрисовки заставки;

## **3.2 Класс FSCrawler**

Основное предназначение – это работа с файловой системой, получения списка файлов и каталогов, а так же получения необходимой для работы информации о них.

Поля класса FSCrawler:

* + - DIR\* dir – представляет собой текущий открытый каталог;
    - string currentPath – хранит путь текущего каталога;
    - dirent\* backBath – хранит путь предыдущего каталога;
    - dirent\* emptyPath – хранит пустой вспомогательный путь, упрощающий отображение;
    - vector<dirent\*> files – содержит списко файлов открытого каталога;

Методы класса MainWindow:

* + - bool isDirectory (int index) – метод, проверяющий является ли файл по указанному индексу внутри списка файлов каталогом;
    - bool isBack(int index) – метод, проверяющий является ли файл по указанному индексу внутри списка ссылкой на предыдущий каталог;
    - int filesCount() – метод, получающий количество файлов в данном каталоге вместе с ссылками на вспомогательный путь и ссылкой на предыдущий каталог;
    - void update() – метод, обновляющая информацию в строке состояния;
    - void update() – метод, производящий обновление данных о текущей открытой директории;
    - void open(int index) – метод, открывабщий новую директорию по пути, являющимся индексов списка файлов предыдущей открытой директории;
    - char\* getName(int index) – метод, получающий имя файла по его индексу в списке файлов;
    - char\* getPath (int index) – метод, получающий путь к файлу по его индексу в списке файлов;
    - char\* operator[] (int index)– перегруженный оператор индексирования используемый внутри класса;

# **4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

В данном разделе будут рассмотрены алгоритмы, используемые в проекте.

## **4.1 Фоновое обновление строки состояния**

Алгоритм реализован в классе MainView. Все необходимые данные для обновления информации получаются либо при помощи встроенных функций языка, либо читаются из системных файлов.

Шаг 1. Установка параметров шрифта.

SetFont(font\_small, BLACK);

SetTextStrength(1);

Шаг 2. Получение данных о текущем времени.

time\_t t = time(NULL);

struct tm \*t\_m;

t\_m=localtime(&t);

char time[11];

sprintf(time, "%2d:%02d\n",t\_m->tm\_hour, t\_m->tm\_min);

Шаг 3. Создание необходимых для работы переменных.

char capacity[5];

char status[12];

Шаг 4. Открытие системного файла, содержащего заряд батареи в процентах.

ifstream fbat("/sys/class/power\_supply/battery/capacity");

if (!fbat.is\_open())

cout << "battery capacity error" << endl;

Шаг 5. В случае успешного открытия добавляем знак «%» в конец строки.

else

{

fbat.getline(capacity, 5);

if (capacity[2] == NULL) capacity[1] = '%';

else if (capacity[3] == NULL) capacity[2] = '%';

else if (capacity[4] == NULL) capacity[3] = '%';

fbat.close();

}

Шаг 6. Открытие системного файла, содержащего статус заряда батареи.

ifstream fstat("/sys/class/power\_supply/battery/capacity");

if (!fstat.is\_open())

cout << "battery status error" << endl;

Шаг 7. В случае успешного открытия определяем статус.

else

{

fstat.getline(status, 12)

fstat.close();

}

Шаг 8. Очищаем область экрана, на которой располагается строка состояния и проводим разделяющую линию.

FillArea(0 , 0 , 600, 50, WHITE);

DrawLine(0, 50, 600, 50, BLACK);

Шаг 9. Отображаем обновленную информацию.

DrawTextRect(500, 0, 100, 50, capacity, ALIGN\_CENTER | VALIGN\_MIDDLE);

if (status[0] == 'C')

DrawBitmap(450, 0, &charge);

DrawTextRect(0, 0, 600, 50, time, ALIGN\_CENTER | VALIGN\_MIDDLE);

Шаг 10. Далее при помощи библиотечной функции из SDK производим обновление экрана без полной перерисовки.

SoftUpdate();

## **4.2 Алгоритм открытия новой директории**

Алгоритм реализован в классе FSCrawler. Все необходимые данные для открытия новой директории передаются в метод в качестве аргумента, а так же хранятся в полях класса.

Шаг 1. Проверяем валидность переданного аргумента.

if(index < 0 || index >= files.size()|| !isDirectory(index))

return;

Шаг 2. Проверяем, не является ли открываемая директория ссылкой на предыдущую.

else if(files[index]->d\_ino == BACK)

Шаг 3. Если это так, то изменяем текущий путь на предыдущий.

int pos = currentPath.find\_last\_of('/');

currentPath = currentPath.substr(0, currentPath.length() - pos - 1);

Шаг 4 Если нет, то изменяем текущий путь на новый.

currentPath += '/';

currentPath += string(files[index]->d\_name);

Шаг 5. Вызываем метод update() в котором пытаемся открыть директорию по только что созданному пути.

if(dir) closedir(dir);

dir = opendir(currentPath.c\_str());

if(!dir)

cout << "Error opening directory" << endl;

Шаг 6. В случае успеха читаем из директории файлы, пока они есть.

else

{

dirent\* item;

while((item = readdir(dir)) != NULL)

{

Шаг 7. Добавляем файлы в список исключая системные.

if(!strcmp(item->d\_name,".")||!strcmp(item->d\_name, ".."))

continue;

files.push\_back(item);

}

Шаг 8. Добавляем вспомогательный путь.

files.push\_back(emptyPath);

# **5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Минимальные системные требования:

* Электронная книга PocketBook 602 и новее.
* Переходник USB to UART

Для загрузки программы:

1. Первоначально необходимо получить готовый проект и записать его на micro-cd карточку.
2. Вставить micro-cd карточку в соответствующий слот электронной книги.
3. Снять заднюю крышку книги.
4. Подготовить переходник (см. рис. 5.1).
5. Найти на плате соответствующий разъем и подключить переходник(см. рис. 5.1).

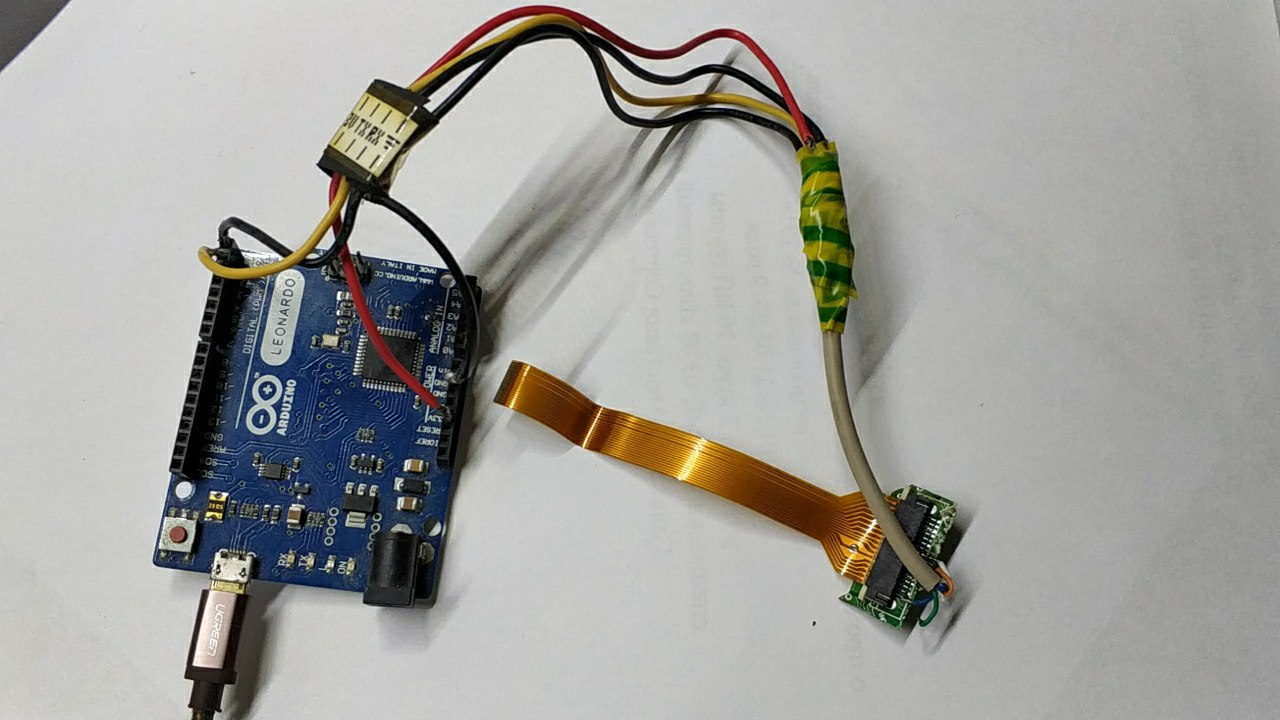


Рисунок 5.1 – Переходник USB to USART



Рисунок 5.2 – Разьем USART

1. Необходимо подключить переходник к ПК и открыть программу-терминал (например PyTTY)
2. После подключения переходника необходимо включить книгу путем нажатия кнопки питания.
3. В окне терминала отобразится процесс загрузки операционной системы книги. В процессе загрузки необходимо несколько раз нажать клавишу Enter на клавиатуре для активации режима командной строки. (см. рис. 5.3)
4. Далее необходимо завершить процесс, контролирующий оболочку ОС. Для этого в терминале вводится команда “top” и определяется pid процесса с именем “taskmgr.app”.
5. Затем путем ввода команды “kill 1234”, где 1234 – pid процесса из пункта 9, завершается этот процесс, что дает нам возможность изменить составные части оболочки.
6. Далее при помощи команды “mv /mnt/ext2/путь\_к\_проекту/bookshelf.app /ebrmain/bin/ bookshelf.app” производится установка программы
7. После этого производится перезагрузка книги кнопкой питания, при этом новая программа уже окажется загружена

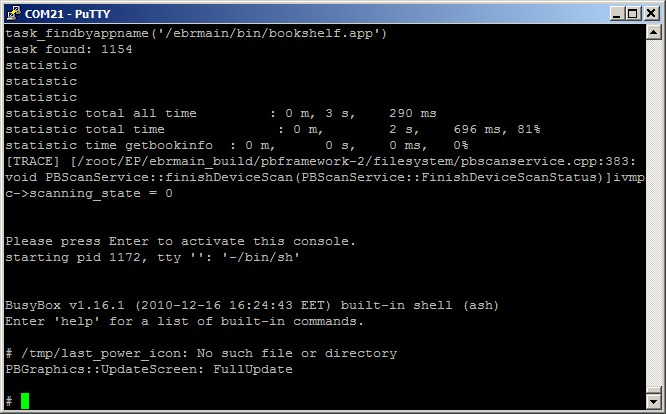


Рисунок 5.3 – Активация режима командной строки

Инструкция по использованию программы:

1. После загрузки программы перед пользователем предстает графический интерфейс (см. рис. 5.4)
2. Нажатием подэкранных клавиш происходит перемещение по меню и выбор книги для открытия
3. Аналогичным образом осуществляется открытие папок (см. рис. 5.5)
4. Нажатием клавиши Home из любого ридера можно вернуться в это меню.



Рисунок 5.4 – Интерфейс программы

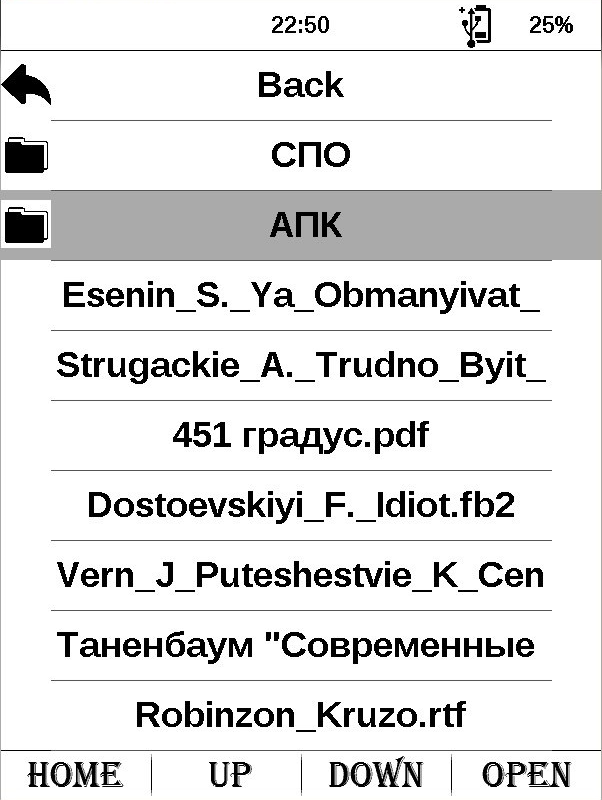


Рисунок 5.5 – Выбор папки

# **ЗАКЛЮЧЕНИЕ**

При конструировании данного программного средства в виде системного ПО для электронной книги на базе embedded Linux были реализованы спроектированные алгоритмы в программном коде, создано руководство пользователя для облегчения восприятия функционала программы.

Во время написания курсовой работы был получены ключевые навыки работы с терминалом операционной системы Linux, а так же особенности написания программ под данную ОС. Изучена иерархия системных каталогов ОС Linux и иерархия каталогов электронной книги в частности.

В результате разработки данного приложения было создано альтернативное меню для открытия книг на устройствах, по неосторожности лишенных данной возможности. Оно сможет найти свое применение как одно из дешевых средств ремонта электронной книги в ремонтных мастерских.

У данного проекта имеется огромный потенциал для развития: добавление изменения размера шрифтов в ридерах, изменение ориентации дисплея, использование встроенного в книгу wi-fi модуля для закачки новых книг.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1 Сайт github.com [Электронный ресурс]. – Режим доступа: https://github.com/pocketbook-free?tab=repositories

2 Сайт the-ebook.org [Электронный ресурс]. – Режим доступа: http://www.the-ebook.org/forum/viewtopic.php?t=18625

3 Сайт opennet.ru [Электронный ресурс]. – Режим доступа: https://www.opennet.ru/docs/RUS/bash\_scripting\_guide/

4 Сайт habr.com [Электронный ресурс]. – Режим доступа: https://habr.com

# 

# **ПРИЛОЖЕНИЕ А**

Схема структурная (чертёж)

# **ПРИЛОЖЕНИЕ Б**

Диаграмма классов (чертёж)

# **ПРИЛОЖЕНИЕ В**

Схема алгоритма

# **ПРИЛОЖЕНИЕ Г**

Схема алгоритма

# **ПРИЛОЖЕНИЕ Д**

Листинг программы

#include <iostream>

#include <pthread.h>

#include <signal.h>

#include "mainview.h"

using namespace std;

MainView\* mainview;

pthread\_t controller;

void test();

struct sigaction logoSigaction;

int main\_handler(int type, int par1, int par2)

{

if (type==EVT\_SHOW)

{

show\_logo();

}

else if (type==EVT\_KEYPRESS)

{

switch (par1)

{

case KEY\_PREV:

mainview->forcePrev();

break;

case KEY\_NEXT:

mainview->forceNext();

break;

case KEY\_MENU:

mainview->open();

break;

}

}

}

void\* background\_process(void\* lpParam)

{

while(true)

{

usleep(1000000);

mainview->updateTray();

}

}

void close\_handler(int sign)

{

mainview->showLogo();

}

int main(int argc, char \*\*argv)

{

mainview = new MainView();

pthread\_create(&controller, NULL, background\_process, NULL);

logoSigaction.sa\_handler = close\_handler;

sigaction(SIGTERM, &logoSigaction, NULL);

InkViewMain(main\_handler);

return 0;

}

#include "inkview.h"

#include <iostream>

#include <sys/types.h>

#include <sys/wait.h>

#include "fscrawler.h"

extern const ibitmap background, buttons, charge, folder, back;

class MainView

{

private:

ifont \*font\_small;

ifont \*font\_big;

FSCrawler fscrawler;

int currentPosition;

int currentBase;

void drawUi();

void updateBase();

void updatePosition();

public:

MainView();

void updateTray();

void update();

void forceNext();

void forcePrev();

void open();

void showLogo();

};

#include "mainview.h"

MainView::MainView()

{

char font\_name[128];

sprintf(font\_name, "#%s,24,0", DEFAULTFONTB);

font\_small = GetThemeFont("clock.font", font\_name);

sprintf(font\_name, "#%s,36,0", DEFAULTFONTB);

font\_big = GetThemeFont("clock.font", font\_name);

currentPosition = 0;

OpenScreen();

SetAutoPowerOff(0);

ClearScreen();

updateTray();

drawUi();

}

void MainView::drawUi()

{

DrawBitmap(0, 750, &buttons);

DrawLine(0, 750, 600, 750, BLACK);

update();

}

void MainView::update()

{

static int fullUpdate;

if(fullUpdate > 10)

{

FullUpdate();

fullUpdate = 0;

}

else ++fullUpdate;

FillArea(0 , 51 , 600, 699, WHITE);

updatePosition();

updateBase();

SoftUpdate();

}

void MainView::updateBase()

{

SetFont(font\_big, BLACK);

for(int i = 0; i < 10; ++i)

{

if(fscrawler.isBack(currentBase + i)) DrawBitmap(0, 60 + 70 \* i, &back);

else if(fscrawler.isDirectory(currentBase + i)) DrawBitmap(0, 60 + 70 \* i, &folder);

DrawTextRect(50, 50 + 70 \* i, 550, 70, fscrawler.getName(currentBase + i), ALIGN\_CENTER | VALIGN\_MIDDLE);

DrawLine(50, 120 + 70 \* i, 550, 120 + 70 \* i, DGRAY);

}

}

void MainView::updatePosition()

{

int index = currentPosition + 1;

FillArea(0 , 50 + 70 \* index , 600, 70, LGRAY);

}

void MainView::updateTray()

{

SetFont(font\_small, BLACK);

SetTextStrength(1);

time\_t t = time(NULL);

struct tm \*t\_m;

t\_m=localtime(&t);

char time[11];

sprintf(time, "%2d:%02d\n",t\_m->tm\_hour, t\_m->tm\_min);

char capacity[5];

char status[12];

ifstream fbat("/sys/class/power\_supply/battery/capacity");

if (!fbat.is\_open())

cout << "battery capacity error" << endl;

else

{

fbat.getline(capacity, 5);

if (capacity[2] == NULL) capacity[1] = '%';

else if (capacity[3] == NULL) capacity[2] = '%';

else if (capacity[4] == NULL) capacity[3] = '%';

fbat.close();

}

ifstream fstat("/sys/class/power\_supply/battery/capacity");

if (!fstat.is\_open())

cout << "battery status error" << endl;

else

{

fstat.getline(status, 12)

fstat.close();

}

FillArea(0 , 0 , 600, 50, WHITE);

DrawLine(0, 50, 600, 50, BLACK);

DrawTextRect(500, 0, 100, 50, capacity, ALIGN\_CENTER | VALIGN\_MIDDLE);

if (status[0] == 'C')

DrawBitmap(450, 0, &charge);

DrawTextRect(0, 0, 600, 50, time, ALIGN\_CENTER | VALIGN\_MIDDLE);

SoftUpdate();

}

void MainView::forceNext()

{

if (currentPosition < 7 && currentPosition < fscrawler.filesCount() - 3)

++currentPosition;

else if (currentBase < fscrawler.filesCount() - 9)

++currentBase;

else return;

update();

}

void MainView::forcePrev()

{

if (currentPosition > 0)

--currentPosition;

else if (currentBase > 0)

--currentBase;

else return;

update();

}

void MainView::open()

{

if (fscrawler.isDirectory(currentBase + currentPosition))

{

fscrawler.open(currentBase + currentPosition);

currentPosition = 0;

currentBase = 0;

update();

}

else

OpenBook(fscrawler.getPath(currentBase + currentPosition), "", 0);

}

void MainView::showLogo()

{

FillArea(0 , 0 , 600, 800, WHITE);

DrawBitmap(0, 0, &background);

FullUpdate();

}

#include <sys/types.h>

#include <dirent.h>

#include <iostream>

#include <string>

#include <string.h>

#include <vector>

using namespace std;

const int BACK = -1;

class FSCrawler

{

private:

DIR\* dir;

string currentPath;

dirent\* backBath;

dirent\* emptyPath;

vector<dirent\*> files;

public:

FSCrawler();

bool isDirectory(int index);

bool isBack(int index);

int filesCount();

void update();

void open(int index);

char\* getName(int index);

char\* getPath(int index);

char\* operator[] (int index);

};

#include "fscrawler.h"

FSCrawler::FSCrawler()

{

backBath = new dirent;

strcpy(backBath->d\_name, "Back");

backBath->d\_type = DT\_DIR;

backBath->d\_ino = BACK;

emptyPath = new dirent;

strcpy(emptyPath->d\_name, "");

files.push\_back(emptyPath);

files.push\_back(backBath);

currentPath = ".";

update();

}

void FSCrawler::open(int index)

{

if(index < 0 || index >= files.size() || !isDirectory(index))

return;

else if(files[index]->d\_ino == BACK)

{

int pos = currentPath.find\_last\_of('/');

currentPath = currentPath.substr(0, currentPath.length() - pos - 1);

}

else

{

currentPath += '/';

currentPath += string(files[index]->d\_name);

}

files.clear();

files.push\_back(backBath);

update();

}

void FSCrawler::update()

{

if(dir) closedir(dir);

dir = opendir(currentPath.c\_str());

if(!dir)

cout << "Error opening directory" << endl;

else

{

dirent\* item;

while((item = readdir(dir)) != NULL)

{

if(!strcmp(item->d\_name, ".") || !strcmp(item->d\_name, ".."))

continue;

cout << item->d\_name << " = ";

cout << (item->d\_type == DT\_DIR ? "dir" : "file") << endl;

files.push\_back(item);

}

}

files.push\_back(emptyPath);

}

char\* FSCrawler::getName(int index)

{

if(index < 0 || index >= files.size()) return "";

return (char\*)string(files[index]->d\_name).c\_str();

}

int FSCrawler::filesCount()

{

return files.size() - 1;

}

bool FSCrawler::isDirectory(int index)

{

if(index < 0 || index >= files.size()) return false;

return files[index]->d\_type == DT\_DIR;

}

bool FSCrawler::isBack(int index)

{

if(index < 0 || index >= files.size()) return false;

return files[index]->d\_ino == BACK;

}

char\* FSCrawler::getPath(int index)

{

static string path;

path = currentPath;

path += files[index] + '/';

return path.c\_str();

}

char\* FSCrawler::operator[] (int index)

{

return files[index]->d\_name;

}