Листинг проекта

Листинг 1.1: MainForm.cpp

#include "MainForm.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThreadAttribute]

int main(cli::array<String^>^ args)

{

Application::SetCompatibleTextRenderingDefault(false);

Application::EnableVisualStyles();

ScreenProject::MainForm mainForm;

Application::Run(%mainForm);

}

Листинг кода 1.2: MainForm.h

#include "SessionInfoForm.h"

#include "ScreenShotManager.h"

#include "ProcessManager.h"

#include "ThreadManager.h"

#include <string>

#include <list>

#include <msclr\marshal\_cppstd.h>

namespace ScreenProject

{

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace msclr::interop;

using namespace std;

/// <summary>

/// Сводка для MainForm

/// </summary>

public ref class MainForm : public System::Windows::Forms::Form

{

public:

MainForm(void)

{

InitializeComponent();

SetDefault();

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MainForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::ListBox^ TrackedProcessListBox;

protected:

protected:

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::ComboBox^ AllProcessesCB;

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::Button^ AddProcessToTrackB;

private: System::Windows::Forms::Button^ DeleteProcessFromTrack;

private: System::Windows::Forms::Button^ StartTrackingB;

private: System::Windows::Forms::Button^ CancelTrackingB;

private: System::Windows::Forms::Label^ TrackingStatusLabel;

private: System::Windows::Forms::ComboBox^ ScreenShotIntervalCB;

private: System::Windows::Forms::Label^ label3;

protected:

protected:

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

this->TrackedProcessListBox = (gcnew System::Windows::Forms::ListBox());

this->label1 = (gcnew System::Windows::Forms::Label());

this->AllProcessesCB = (gcnew System::Windows::Forms::ComboBox());

this->label2 = (gcnew System::Windows::Forms::Label());

this->AddProcessToTrackB = (gcnew System::Windows::Forms::Button());

this->DeleteProcessFromTrack = (gcnew System::Windows::Forms::Button());

this->StartTrackingB = (gcnew System::Windows::Forms::Button());

this->CancelTrackingB = (gcnew System::Windows::Forms::Button());

this->TrackingStatusLabel = (gcnew System::Windows::Forms::Label());

this->ScreenShotIntervalCB = (gcnew System::Windows::Forms::ComboBox());

this->label3 = (gcnew System::Windows::Forms::Label());

this->SuspendLayout();

//

// TrackedProcessListBox

//

this->TrackedProcessListBox->FormattingEnabled = true;

this->TrackedProcessListBox->ItemHeight = 20;

this->TrackedProcessListBox->Location = System::Drawing::Point(12, 37);

this->TrackedProcessListBox->Name = L"TrackedProcessListBox";

this->TrackedProcessListBox->Size = System::Drawing::Size(367, 364);

this->TrackedProcessListBox->TabIndex = 0;

this->TrackedProcessListBox->SelectedIndexChanged += gcnew System::EventHandler(this, &MainForm::TrackedProcessListBox\_SelectedIndexChanged);

//

// label1

//

this->label1->AutoSize = true;

this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 10.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label1->Location = System::Drawing::Point(8, 14);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(184, 17);

this->label1->TabIndex = 1;

this->label1->Text = L"Отслеживаемые процессы";

this->label1->TextAlign = System::Drawing::ContentAlignment::MiddleLeft;

//

// AllProcessesCB

//

this->AllProcessesCB->FormattingEnabled = true;

this->AllProcessesCB->Location = System::Drawing::Point(401, 37);

this->AllProcessesCB->Name = L"AllProcessesCB";

this->AllProcessesCB->RightToLeft = System::Windows::Forms::RightToLeft::Yes;

this->AllProcessesCB->Size = System::Drawing::Size(367, 28);

this->AllProcessesCB->TabIndex = 2;

this->AllProcessesCB->SelectedIndexChanged += gcnew System::EventHandler(this, &MainForm::AllProcessesCB\_SelectedIndexChanged);

//

// label2

//

this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 10.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label2->Location = System::Drawing::Point(434, 14);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(334, 20);

this->label2->TabIndex = 3;

this->label2->Text = L"Доступные процессы";

this->label2->TextAlign = System::Drawing::ContentAlignment::MiddleRight;

//

// AddProcessToTrackB

//

this->AddProcessToTrackB->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(0)),

static\_cast<System::Int32>(static\_cast<System::Byte>(192)), static\_cast<System::Int32>(static\_cast<System::Byte>(0)));

this->AddProcessToTrackB->Enabled = false;

this->AddProcessToTrackB->ForeColor = System::Drawing::SystemColors::ButtonFace;

this->AddProcessToTrackB->Location = System::Drawing::Point(401, 76);

this->AddProcessToTrackB->Name = L"AddProcessToTrackB";

this->AddProcessToTrackB->Size = System::Drawing::Size(179, 42);

this->AddProcessToTrackB->TabIndex = 4;

this->AddProcessToTrackB->Text = L"Добавить";

this->AddProcessToTrackB->UseVisualStyleBackColor = false;

this->AddProcessToTrackB->Click += gcnew System::EventHandler(this, &MainForm::AddProcessToTrackB\_Click);

//

// DeleteProcessFromTrack

//

this->DeleteProcessFromTrack->BackColor = System::Drawing::Color::Crimson;

this->DeleteProcessFromTrack->Enabled = false;

this->DeleteProcessFromTrack->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 12, System::Drawing::FontStyle::Regular,

System::Drawing::GraphicsUnit::Point, static\_cast<System::Byte>(204)));

this->DeleteProcessFromTrack->ForeColor = System::Drawing::SystemColors::ButtonFace;

this->DeleteProcessFromTrack->Location = System::Drawing::Point(589, 76);

this->DeleteProcessFromTrack->Name = L"DeleteProcessFromTrack";

this->DeleteProcessFromTrack->Size = System::Drawing::Size(179, 42);

this->DeleteProcessFromTrack->TabIndex = 5;

this->DeleteProcessFromTrack->Text = L"Удалить";

this->DeleteProcessFromTrack->UseVisualStyleBackColor = false;

this->DeleteProcessFromTrack->Click += gcnew System::EventHandler(this, &MainForm::DeleteProcessFromTrack\_Click);

//

// StartTrackingB

//

this->StartTrackingB->BackColor = System::Drawing::Color::DarkTurquoise;

this->StartTrackingB->Enabled = false;

this->StartTrackingB->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 13.8F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->StartTrackingB->ForeColor = System::Drawing::SystemColors::HighlightText;

this->StartTrackingB->Location = System::Drawing::Point(401, 295);

this->StartTrackingB->Name = L"StartTrackingB";

this->StartTrackingB->Size = System::Drawing::Size(367, 50);

this->StartTrackingB->TabIndex = 6;

this->StartTrackingB->Text = L"Начать отслеживание";

this->StartTrackingB->UseVisualStyleBackColor = false;

this->StartTrackingB->Click += gcnew System::EventHandler(this, &MainForm::StartTrackingB\_Click);

//

// CancelTrackingB

//

this->CancelTrackingB->BackColor = System::Drawing::Color::Crimson;

this->CancelTrackingB->Enabled = false;

this->CancelTrackingB->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 13.8F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->CancelTrackingB->ForeColor = System::Drawing::SystemColors::ButtonFace;

this->CancelTrackingB->Location = System::Drawing::Point(401, 351);

this->CancelTrackingB->Name = L"CancelTrackingB";

this->CancelTrackingB->Size = System::Drawing::Size(367, 50);

this->CancelTrackingB->TabIndex = 7;

this->CancelTrackingB->Text = L"Прекратить отслеживание";

this->CancelTrackingB->UseVisualStyleBackColor = false;

this->CancelTrackingB->Click += gcnew System::EventHandler(this, &MainForm::CancelTrackingB\_Click);

//

// TrackingStatusLabel

//

this->TrackingStatusLabel->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 10.2F, System::Drawing::FontStyle::Regular,

System::Drawing::GraphicsUnit::Point, static\_cast<System::Byte>(204)));

this->TrackingStatusLabel->ForeColor = System::Drawing::SystemColors::AppWorkspace;

this->TrackingStatusLabel->Location = System::Drawing::Point(401, 269);

this->TrackingStatusLabel->Name = L"TrackingStatusLabel";

this->TrackingStatusLabel->Size = System::Drawing::Size(367, 23);

this->TrackingStatusLabel->TabIndex = 8;

this->TrackingStatusLabel->Text = L"Отслеживание начато...";

this->TrackingStatusLabel->TextAlign = System::Drawing::ContentAlignment::MiddleCenter;

this->TrackingStatusLabel->Visible = false;

//

// ScreenShotIntervalCB

//

this->ScreenShotIntervalCB->FormattingEnabled = true;

this->ScreenShotIntervalCB->Items->AddRange(gcnew cli::array< System::Object^ >(7) {

L"5 сек.", L"30 сек.", L"1 мин.", L"5 мин.",

L"10 мин.", L"30 мин.", L"1 час."

});

this->ScreenShotIntervalCB->Location = System::Drawing::Point(589, 191);

this->ScreenShotIntervalCB->Name = L"ScreenShotIntervalCB";

this->ScreenShotIntervalCB->Size = System::Drawing::Size(143, 28);

this->ScreenShotIntervalCB->TabIndex = 9;

this->ScreenShotIntervalCB->SelectedIndexChanged += gcnew System::EventHandler(this, &MainForm::ScreenShotIntervalCB\_SelectedIndexChanged);

//

// label3

//

this->label3->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 10.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label3->Location = System::Drawing::Point(437, 184);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(146, 40);

this->label3->TabIndex = 10;

this->label3->Text = L"Интервал создания скриншотов:";

this->label3->TextAlign = System::Drawing::ContentAlignment::MiddleRight;

//

// MainForm

//

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::None;

this->BackColor = System::Drawing::Color::AliceBlue;

this->ClientSize = System::Drawing::Size(780, 421);

this->Controls->Add(this->label3);

this->Controls->Add(this->ScreenShotIntervalCB);

this->Controls->Add(this->TrackingStatusLabel);

this->Controls->Add(this->CancelTrackingB);

this->Controls->Add(this->StartTrackingB);

this->Controls->Add(this->DeleteProcessFromTrack);

this->Controls->Add(this->AddProcessToTrackB);

this->Controls->Add(this->label2);

this->Controls->Add(this->AllProcessesCB);

this->Controls->Add(this->label1);

this->Controls->Add(this->TrackedProcessListBox);

this->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 12, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedSingle;

this->MaximizeBox = false;

this->MinimizeBox = false;

this->Name = L"MainForm";

this->ShowIcon = false;

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"ScreenProject";

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private:

void SetDefault()

{

TrackedProcessListBox->Items->Clear();

TrackedProcessListBox->SelectedIndex = -1;

AllProcessesCB->Items->Clear();

auto lis = ProcessManager::GetAllProcesses();

for (string process : lis)

{

AllProcessesCB->Items->Add(gcnew System::String(process.c\_str()));

}

}

void Check()

{

if (TrackedProcessListBox->Items->Count > 0 && ScreenShotIntervalCB->SelectedIndex != -1)

{

StartTrackingB->Enabled = true;

}

else

{

StartTrackingB->Enabled = false;

}

}

System::Void AllProcessesCB\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e)

{

if (AllProcessesCB->SelectedIndex != -1)

{

AddProcessToTrackB->Enabled = true;

}

else

{

AddProcessToTrackB->Enabled = false;

}

}

System::Void AddProcessToTrackB\_Click(System::Object^ sender, System::EventArgs^ e)

{

TrackedProcessListBox->Items->Add(AllProcessesCB->SelectedItem);

AllProcessesCB->Items->RemoveAt(AllProcessesCB->SelectedIndex);

//AllProcessesCB->SelectedIndex = -1;

AddProcessToTrackB->Enabled = false;

Check();

}

System::Void TrackedProcessListBox\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e)

{

if (TrackedProcessListBox->SelectedIndex != -1)

{

DeleteProcessFromTrack->Enabled = true;

}

else

{

DeleteProcessFromTrack->Enabled = false;

}

}

System::Void DeleteProcessFromTrack\_Click(System::Object^ sender, System::EventArgs^ e)

{

AllProcessesCB->Items->Add(TrackedProcessListBox->SelectedItem);

TrackedProcessListBox->Items->RemoveAt(TrackedProcessListBox->SelectedIndex);

DeleteProcessFromTrack->Enabled = false;

Check();

}

void SwitchControlsEnable()

{

AllProcessesCB->Enabled = !AllProcessesCB->Enabled;

TrackedProcessListBox->Enabled = !TrackedProcessListBox->Enabled;

CancelTrackingB->Enabled = !CancelTrackingB->Enabled;

StartTrackingB->Enabled = !StartTrackingB->Enabled;

TrackingStatusLabel->Visible = !TrackingStatusLabel->Visible;

ScreenShotIntervalCB->Enabled = !ScreenShotIntervalCB->Enabled;

}

System::Void StartTrackingB\_Click(System::Object^ sender, System::EventArgs^ e)

{

SwitchControlsEnable();

int interval = 0;

if (ScreenShotIntervalCB->Text == "5 сек.")

{

interval = 5000;

}

else if (ScreenShotIntervalCB->Text == "30 сек.")

{

interval = 30000;

}

else if (ScreenShotIntervalCB->Text == "1 мин.")

{

interval = 60000;

}

else if (ScreenShotIntervalCB->Text == "5 мин.")

{

interval = 300000;

}

else if (ScreenShotIntervalCB->Text == "10 мин.")

{

interval = 600000;

}

else if (ScreenShotIntervalCB->Text == "30 мин.")

{

interval = 1800000;

}

else if (ScreenShotIntervalCB->Text == "1 час.")

{

interval = 3600000;

}

list<string> trackingProcesses = TrackedProcesesToList();

ThreadManager::ScreenShotThreadStart(trackingProcesses, interval);

}

System::Void ScreenShotIntervalCB\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e)

{

Check();

}

list<string> TrackedProcesesToList()

{

list<string> trackingProcesses;

for (int i = 0; i < TrackedProcessListBox->Items->Count; i++)

{

string item = marshal\_as<std::string>(TrackedProcessListBox->Items[i]->ToString());

trackingProcesses.emplace\_back(item);

}

return trackingProcesses;

}

System::Void CancelTrackingB\_Click(System::Object^ sender, System::EventArgs^ e)

{

ThreadManager::ScreenShotThreadClose();

list<string> nameOfProceses;

for (string process : TrackedProcesesToList())

{

string processName = "";

for (int i = 0; i < process.size() - 4; i++)

{

processName += process[i];

}

nameOfProceses.emplace\_back(processName);

}

SessionInfoForm^ sessionInfoForm = gcnew SessionInfoForm(nameOfProceses);

sessionInfoForm->ShowDialog();

SetDefault();

SwitchControlsEnable();

Check();

}

};

}

Листинг 1.3: SessionInfoForm.h

#include "ScreenShotManager.h"

#include "ProcessManager.h"

#include "ThreadManager.h"

#include <string>

#include <list>

#include <msclr\marshal\_cppstd.h>

#include "ShellAPI.h"

#include "FolderManager.h"

namespace ScreenProject

{

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace msclr::interop;

using namespace std;

/// <summary>

/// Сводка для SessionInfoForm

/// </summary>

public ref class SessionInfoForm : public System::Windows::Forms::Form

{

public:

SessionInfoForm(list<string> nameOfProcesses)

{

InitializeComponent();

SetListBoxInfo(nameOfProcesses);

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~SessionInfoForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::ListView^ InfoListView;

protected:

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::ColumnHeader^ columnHeader1;

private: System::Windows::Forms::ColumnHeader^ columnHeader2;

private: System::Windows::Forms::ColumnHeader^ columnHeader3;

private: System::Windows::Forms::ColumnHeader^ columnHeader4;

protected:

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

this->InfoListView = (gcnew System::Windows::Forms::ListView());

this->columnHeader1 = (gcnew System::Windows::Forms::ColumnHeader());

this->columnHeader2 = (gcnew System::Windows::Forms::ColumnHeader());

this->columnHeader3 = (gcnew System::Windows::Forms::ColumnHeader());

this->columnHeader4 = (gcnew System::Windows::Forms::ColumnHeader());

this->label1 = (gcnew System::Windows::Forms::Label());

this->SuspendLayout();

//

// InfoListView

//

this->InfoListView->Columns->AddRange(gcnew cli::array< System::Windows::Forms::ColumnHeader^ >(4) {

this->columnHeader1,

this->columnHeader2, this->columnHeader3, this->columnHeader4

});

this->InfoListView->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 12, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->InfoListView->FullRowSelect = true;

this->InfoListView->GridLines = true;

this->InfoListView->HideSelection = false;

this->InfoListView->Location = System::Drawing::Point(12, 68);

this->InfoListView->Name = L"InfoListView";

this->InfoListView->Size = System::Drawing::Size(778, 406);

this->InfoListView->TabIndex = 0;

this->InfoListView->UseCompatibleStateImageBehavior = false;

this->InfoListView->View = System::Windows::Forms::View::Details;

this->InfoListView->SelectedIndexChanged += gcnew System::EventHandler(this, &SessionInfoForm::InfoListView\_SelectedIndexChanged);

//

// columnHeader1

//

this->columnHeader1->Text = L"";

this->columnHeader1->Width = 1;

//

// columnHeader2

//

this->columnHeader2->Text = L"Имя процесса";

this->columnHeader2->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

this->columnHeader2->Width = 311;

//

// columnHeader3

//

this->columnHeader3->Text = L"Время использования";

this->columnHeader3->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

this->columnHeader3->Width = 223;

//

// columnHeader4

//

this->columnHeader4->Text = L"Подробнее";

this->columnHeader4->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

this->columnHeader4->Width = 230;

//

// label1

//

this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 15.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label1->Location = System::Drawing::Point(12, 9);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(778, 56);

this->label1->TabIndex = 1;

this->label1->Text = L"Сводка за день";

this->label1->TextAlign = System::Drawing::ContentAlignment::MiddleCenter;

//

// SessionInfoForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->BackColor = System::Drawing::Color::AliceBlue;

this->ClientSize = System::Drawing::Size(802, 486);

this->Controls->Add(this->label1);

this->Controls->Add(this->InfoListView);

this->MaximizeBox = false;

this->MinimizeBox = false;

this->Name = L"SessionInfoForm";

this->ShowIcon = false;

this->ResumeLayout(false);

}

#pragma endregion

void SetListBoxInfo(list<string> nameOfProcesses)

{

for (string process : nameOfProcesses)

{

ListViewItem^ item = gcnew ListViewItem("");

item->SubItems->Add(gcnew System::String(process.c\_str()));

item->SubItems->Add(gcnew System::String(ProcessManager::GetUsedProcessTime(process).c\_str()));

item->SubItems->Add("перейти к каталогу->");

InfoListView->Items->Add(item);

}

}

System::Void InfoListView\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e)

{

if (InfoListView->SelectedIndices->Count == 0)

{

return;

}

string processName = marshal\_as<std::string>(InfoListView->SelectedItems[0]->SubItems[1]->Text->ToString());

time\_t timeNow = time(0);

tm\* currentTime = localtime(&timeNow);

const time\_t tmDate = time(nullptr);

char bufDate[64];

strftime(bufDate, std::size(bufDate), "%d.%m.%Y", localtime(&tmDate));

string path = FolderManager::GetMainFolderPath() + processName + "\\" + bufDate;

LPCSTR folderPath = path.c\_str();

ShellExecuteA(GetForegroundWindow(), "open", folderPath, NULL, NULL, SW\_RESTORE); //Открывает локальный диск C

InfoListView->SelectedIndices->Clear();

}

};

}

Листинг 1.4: ThreadManager.h

#include <list>

#include <string>

static public ref class ThreadManager

{

private:

static void ScreenShotThread(std::list<std::string> treckedProceses, int createScreenShotInterval);

public:

static void ScreenShotThreadStart(std::list<std::string> treckedProceses, int createScreenShotInterval);

static void ScreenShotThreadClose();

};

Листинг 1.5: ThreadManager.cpp

#include "ThreadManager.h"

#include "ScreenShotManager.h"

#include "ProcessManager.h"

#include <thread>

#include <chrono>

#include <list>

#include <string>

static bool threadIsPerformed = false;

void ThreadManager::ScreenShotThread(std::list<std::string> treckedProceses, int createScreenShotInterval)

{

while (threadIsPerformed)

{

for (string process : treckedProceses)

{

if (ProcessManager::GetActiveWindowProccesName()->ToLower() == (gcnew System::String(process.c\_str()))->ToLower())

{

ScreenShotManager::DoScreenShot();

}

if (ProcessManager::GetActiveWindowProccesName()->ToLower() == "explorer.exe")

{

ScreenShotManager::DoScreenShot();

}

}

std::this\_thread::sleep\_for(std::chrono::milliseconds(createScreenShotInterval));

}

}

void ThreadManager::ScreenShotThreadStart(std::list<std::string> treckedProceses, int createScreenShotInterval)

{

threadIsPerformed = true;

std::thread th(ScreenShotThread, treckedProceses, createScreenShotInterval);

th.detach();

}

void ThreadManager::ScreenShotThreadClose()

{

threadIsPerformed = false;

}

Листинг 1.6: ScreenShotManager.h

#include <string>

static public ref class ScreenShotManager

{

private:

static std::string CreatePath();

public:

static int DoScreenShot();

};

Листинг 1.7: ScreenShotManager.cpp

#include "ScreenShotManager.h"

#include "ProcessManager.h"

#include "FolderManager.h"

#include <Windows.h>

#include <algorithm>

#include <ctime>

#include <string>

using namespace std;

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_CRT\_NONSTDC\_NO\_DEPRECATE

#define \_CRT\_SECURE\_NO\_DEPRECATE

#pragma warning(disable : 4996)

std::string ScreenShotManager::CreatePath()

{

time\_t now = time(0);

tm\* ltm = localtime(&now);

const time\_t tmDate = time(nullptr);

char bufDate[64];

strftime(bufDate, std::size(bufDate), "%d.%m.%Y", localtime(&tmDate));

char bufTime[64];

strftime(bufTime, sizeof(bufTime), "%H-%M-%S", ltm);

System::String^ processName = ProcessManager::GetActiveWindowProccesName();

string appName;

for each (char letter in (processName->Remove(processName->Length - 4))->ToCharArray())

{

appName += letter;

}

FolderManager::CheckFolder(appName);

FolderManager::CheckFolder(appName + "\\" + bufDate);

string path = FolderManager::GetMainFolderPath() + appName + "\\" + bufDate + "\\" + bufTime + ".bmp";

return path;

}

int ScreenShotManager::DoScreenShot()

{

int x1, y1, x2, y2, w, h;

if (ProcessManager::GetActiveWindowProccesName()->ToLower() == "screen\_project.exe")

{

HWND hwndC = GetActiveWindow();

ShowWindow(hwndC, SW\_MINIMIZE);

}

HWND hwnd = GetForegroundWindow();

Sleep(200);

x1 = GetSystemMetrics(SM\_XVIRTUALSCREEN);

y1 = GetSystemMetrics(SM\_YVIRTUALSCREEN);

x2 = GetSystemMetrics(SM\_CXVIRTUALSCREEN);

y2 = GetSystemMetrics(SM\_CYVIRTUALSCREEN);

HDC hScreen = GetDC(NULL);

float g\_DPIScale = 1.0f;

float dpi = GetDpiForWindow(hwnd);

g\_DPIScale = dpi / 96.0f;

w = x2 - x1;

h = y2 - y1;

w \*= g\_DPIScale;

h \*= g\_DPIScale;

HDC hDC = CreateCompatibleDC(hScreen);

HBITMAP hBitmap = CreateCompatibleBitmap(hScreen, w, h);

HGDIOBJ old\_obj = SelectObject(hDC, hBitmap);

BOOL bRet = BitBlt(hDC, 0, 0, w, h, hScreen, x1, y1, SRCCOPY);

SelectObject(hDC, hBitmap);

TCHAR M[200];

string path = CreatePath();

std::wstring stemp = std::wstring(path.begin(), path.end());

LPCWSTR filePath = stemp.c\_str();

wsprintf(M, filePath, 1);

BitBlt(hDC, 0, 0, w, h, hScreen, 0, 0, SRCCOPY);

BITMAPINFO bmp\_info = { 0 };

bmp\_info.bmiHeader.biSize = sizeof(bmp\_info.bmiHeader);

bmp\_info.bmiHeader.biWidth = w;

bmp\_info.bmiHeader.biHeight = h;

bmp\_info.bmiHeader.biPlanes = 1;

bmp\_info.bmiHeader.biBitCount = 24;

bmp\_info.bmiHeader.biCompression = BI\_RGB;

int bmp\_padding = (w \* 3) % 4;

if (bmp\_padding != 0) bmp\_padding = 4 - bmp\_padding;

BYTE\* bmp\_pixels = new BYTE[(w \* 3 + bmp\_padding) \* h];;

GetDIBits(hDC, hBitmap, 0, h, bmp\_pixels, &bmp\_info, DIB\_RGB\_COLORS);

BITMAPFILEHEADER bmfHeader;

HANDLE bmp\_file\_handle = CreateFile(M, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

DWORD dwSizeofDIB = (w \* 3 + bmp\_padding) \* h + sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER);

bmfHeader.bfOffBits = (DWORD)sizeof(BITMAPFILEHEADER) + (DWORD)sizeof(BITMAPINFOHEADER);

bmfHeader.bfSize = dwSizeofDIB;

bmfHeader.bfType = 0x4D42;

DWORD dwBytesWritten = 0;

WriteFile(bmp\_file\_handle, (LPSTR)&bmfHeader, sizeof(BITMAPFILEHEADER), &dwBytesWritten, NULL);

WriteFile(bmp\_file\_handle, (LPSTR)&bmp\_info.bmiHeader, sizeof(BITMAPINFOHEADER), &dwBytesWritten, NULL);

WriteFile(bmp\_file\_handle, (LPSTR)bmp\_pixels, (w \* 3 + bmp\_padding) \* h, &dwBytesWritten, NULL);

CloseHandle(bmp\_file\_handle);

DeleteDC(hDC);

DeleteObject(hBitmap);

ReleaseDC(NULL, hScreen);

delete[] bmp\_pixels;

return 0;

}

Листинг 1.8: ProcessManager.h

#include <windows.h>

#include <stdio.h>

#include <tchar.h>

#include <psapi.h>

#include <string>

#include <iostream>

#include <list>

using namespace std;

using namespace System;

static public ref class ProcessManager

{

private:

static string GetProcessName(DWORD processID);

public:

static String^ GetActiveWindowProccesName();

static list<string> GetAllProcesses();

static string GetUsedProcessTime(string processName);

};

Листинг 1.9: ProcessManager.cpp

#include "ProcessManager.h"

#include "FolderManager.h"

#include <windows.h>

#include <stdio.h>

#include <tchar.h>

#include <psapi.h>

#include <string>

#include <iostream>

#include <list>

#include <ctime>

#include <filesystem>

using namespace std;

using namespace System;

using namespace filesystem;

string ProcessManager::GetProcessName(DWORD processID)

{

TCHAR szProcessName[MAX\_PATH] = L"<unknown>";

HANDLE hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION |

PROCESS\_VM\_READ,

FALSE, processID);

if (hProcess != NULL)

{

HMODULE hMod;

DWORD cbNeeded;

if (EnumProcessModules(hProcess, &hMod, sizeof(hMod),

&cbNeeded))

{

GetModuleBaseName(hProcess, hMod, szProcessName,

sizeof(szProcessName) / sizeof(TCHAR));

}

}

int i = 0;

string name;

while (szProcessName[i] != '\0')

{

name += szProcessName[i];

i++;

}

CloseHandle(hProcess);

return name;

}

System::String^ ProcessManager::GetActiveWindowProccesName()

{

HWND hwnd;

hwnd = GetForegroundWindow();

Sleep(250);

DWORD pid;

GetWindowThreadProcessId(GetForegroundWindow(), &pid);

HANDLE hProc = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ,

FALSE, pid);

TCHAR nameP[MAX\_PATH] = L"<unknown>";

HMODULE hMod;

DWORD cbNeeded;

if (EnumProcessModules(hProc, &hMod, sizeof(hMod), &cbNeeded))

{

GetModuleBaseName(hProc, hMod, nameP, sizeof(nameP) / sizeof(TCHAR));

}

int i = 0;

System::String^ name;

while (nameP[i] != '\0')

{

name += nameP[i];

i++;

}

return (System::String^)name;

}

list<string> ProcessManager::GetAllProcesses()

{

list<string> tmpList;

DWORD aProcesses[1024], cbNeeded{}, cProcesses;

unsigned int i;

if (!EnumProcesses(aProcesses, sizeof(aProcesses), &cbNeeded))

{

return tmpList;

}

cProcesses = cbNeeded / sizeof(DWORD);

for (i = 0; i < cProcesses; i++)

{

if (aProcesses[i] != 0)

{

auto name = GetProcessName(aProcesses[i]);

tmpList.emplace\_back(name);

}

}

list<string> processesList;

for (string process : tmpList)

{

if (process != "<unknown>")

{

processesList.emplace\_back(process);

}

}

processesList.unique();

return processesList;

}

string ProcessManager::GetUsedProcessTime(string processName)

{

time\_t timeNow = time(0);

tm\* currentTime = localtime(&timeNow);

const time\_t tmDate = time(nullptr);

char bufDate[64];

strftime(bufDate, std::size(bufDate), "%d.%m.%Y", localtime(&tmDate));

string folderPath = FolderManager::GetMainFolderPath() + processName + "\\" + bufDate + "\\";

string processFile;

try

{

for (auto& p : directory\_iterator(folderPath))

{

path path = p;

processFile = path.generic\_string();

break;

}

}

catch (const std::exception&)

{

return "00:00:00";

}

string fileCreateTimeStr = "";

processFile = processFile.substr(processFile.find\_last\_of("/") + 1, string::npos);

for (int i = 0; i < processFile.size() - 4; i++)

{

fileCreateTimeStr += processFile[i];

}

tm\* fileCreateTime = new tm;

int hyphenCount = 0;

string tmp = "";

int sizee = fileCreateTimeStr.size();

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

{

if (fileCreateTimeStr[i] == '-' && i != sizee - 1)

{

switch (hyphenCount)

{

case 0:

fileCreateTime->tm\_hour = stoi(tmp);

break;

case 1:

fileCreateTime->tm\_min = stoi(tmp);

break;

}

hyphenCount++;

tmp = "";

continue;

}

if (i == sizee - 1)

{

fileCreateTime->tm\_sec = stoi(tmp + fileCreateTimeStr[i]);

}

tmp += fileCreateTimeStr[i];

}

timeNow -= fileCreateTime->tm\_hour \* 3600;

timeNow -= fileCreateTime->tm\_min \* 60;

timeNow -= fileCreateTime->tm\_sec;

struct tm\* usedTime = localtime(&timeNow);

strftime(bufDate, sizeof(bufDate), "%H:%M:%S", usedTime);

return bufDate;

}

Листинг 1.10: ProcessManager.h

#include <string>

static public ref class FolderManager

{

public:

static void CheckFolder(std::string folderName);

static std::string GetMainFolderPath();

};

Листинг 1.11: ProcessManager.cpp

#include "FolderManager.h"

#include <filesystem>

#include <string>

using namespace std;

void FolderManager::CheckFolder(string folderName)

{

string folderPath = GetMainFolderPath() + folderName;

filesystem::create\_directory(GetMainFolderPath());

filesystem::create\_directory(folderPath);

}

string FolderManager::GetMainFolderPath()

{

return "C:\\ScreenProjectScreens\\";

}