

|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **«МИРЭА – Российский технологический университет» РТУ МИРЭА** |

Институт КБСП направление 10.05.05

Кафедра КБ-4 «Интеллектуальные системы информационной безопасности»

Дисциплина «Кроссплатформенная среда исполнения программного обеспечения»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе (проект) на тему:**

Система блокировки файлов AnViS

Студент Чуксеев Никита Максимович

подпись, дата

Группа ББСО-04-20

Работа защищена на оценку

Руководитель работы Ф.И.О.

подпись, дата

Члены комиссии Ф.И.О.

подпись, дата

Москва 2022г.



|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **«МИРЭА – Российский технологический университет» РТУ**  **МИРЭА** |

Институт КБСП направление 10.05.05

Кафедра КБ-4 «Интеллектуальные системы информационной безопасности»

Дисциплина «Кроссплатформенная среда исполнения программного обеспечения»

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студент: 2 курса, группа: ББСО-04-20

1. Тема: “Система блокировки файлов AnViS”
2. Срок представления проекта (работы) к защите 2022г.
3. Исходные данные для разработки
4. Содержание пояснительной записки:

Задание

Содержание

Описание предметной области

Список использованных источников

Руководитель работы Ф.И.О.

подпись, дата

Задание принял к исполнению Ф.И.О.

подпись, дата

Москва 2022г.

**Содержание**

[Введение 4](#__RefHeading___Toc608_2039763097)

[Цель разработки проекта 5](#__RefHeading___Toc612_2039763097)

[Процесс написания программы 5](#__RefHeading___Toc5056_2039763097)

[Инициализация 5](#__RefHeading___Toc5058_2039763097)

[Интерфейс 5](#__RefHeading___Toc5060_2039763097)

[Основные функции 7](#__RefHeading___Toc5062_2039763097)

[Поток блокировки файлов 10](#__RefHeading___Toc5064_2039763097)

[Результат 11](#__RefHeading___Toc5056_20397630971)

[Список использованных источников 14](#__RefHeading___Toc616_2039763097)

# Введение

Большинство данных, которые хранятся на ЭВМ в наше время, является достаточно важными. Это могут быть аудио-, фото- и видеоматериалы, программы, проекты, документы и т. д.

Бывают случаи, когда по той или иной причине данные изменяются и удаляются без нашего ведома. Какие же это причины? Их несколько: сбои ОС и ПО, вирусы, внешнее взаимодействие злоумышленников с ЭВМ и данными на ней, случайные действия посторонних пользователей и т. д.

Но как не допустить модификацию данных и их удаление или стороннее взаимодействие с ними? Безусловно, можно использовать антивирусы, которые отслеживают большинство событий ЭВМ. Но, как правило, не все они достаточно эффективны и не во всех случаях способны помочь сохранить данные и работоспособность ЭВМ. При том, большинство антивирусов используют очень много компьютерных ресурсов из-за контроля почти всего, что происходит с компьютером в определённый отрезок времени.

В данном случае, для защиты данных, в случае подозрения их утечки или дальнейшей модификации, было бы достаточно выполнять простейшую блокировку файлов на чтение и запись, что и делает «Система блокировки файлов AnViS».

# Цель разработки проекта

Главной целью данного проекта является создание системы блокировки файлов AnViS (AnViS от анг. аббр. - **An**ti**Vi**rus **S**ystem – система-антивирус). В ходе реализации был выбран язык программирования Java и компилятор Eclipse IDE for Java Developers.

# Процесс написания программы

## Инициализация

**public** **static** **boolean** *isRun* = **true**;

**public** **static** Thread [] *threadlist* = **new** Thread[0];

**public** **static** **long** *interval* = 5000L; //время перед завершением работы программы

**public** **static** **long** *lockTime* = 30000L; //время временной блокировки указанных файлов

**public** **static** **long** *maxmemoryfree* = 1024\*1024; //значение свободной оперативной памяти JVM в байтах, при достижении которого,

//создание новых потоков блокировки файлов приостанавливается

## Интерфейс

**try** {

System.***out***.println("Система блокировки файлов AnViS успешно запущена!\nДля получения инструкции введите команду \"info\"");

Scanner sc = **new** Scanner(System.***in***);

String command = ""; //переменная, хранящая команду

**while** (!command.equals("stop")) { //пока введённая команда не является командой stop - обрабатывать ввод команд и выполнять их

**try** { //начало области отслеживания обработки ошибок

command = sc.nextLine(); //ввод команды с клавиатуры

**if** (command.equals("stop")) {

System.***out***.println("Завершение работы программы!");

*clear*();

*isRun* = **false**;

Thread.*sleep*(*interval*);

}

**else** **if** (command.equals("list")) {

*optimize*();

*list*();

}

**else** **if** (command.equals("lock")) {

String path = "";

System.***out***.print("Пожалуйста, введите путь к файловому объекту: ");

path = sc.nextLine();

*add*(path, **false**);

}

**else** **if** (command.equals("lockfull")) {

String path = "";

System.***out***.print("Пожалуйста, введите путь к файловому объекту: ");

path = sc.nextLine();

*add*(path, **true**);

}

**else** **if** (command.equals("unlock")) {

String path = "";

System.***out***.print("Пожалуйста, введите путь к файловому объекту: ");

path = sc.nextLine();

*delete*(path);

}

**else** **if** (command.equals("get time")) {

System.***out***.println("Время интервала в миллисекундах: " + *interval*);

}

**else** **if** (command.equals("set time")) {

System.***out***.print("Пожалуйста, введите новое время интервала в миллисекундах (целое число): ");

**try** {

*interval* = Long.*parseLong*(sc.nextLine());

System.***out***.println("Время интервала успешно установлено!");

}

**catch** (Exception e) {

System.***out***.println("Было введено не число!");

}

}

**else** **if** (command.equals("get locktime")) {

System.***out***.println("Время продолжительности блокировки файловых объектов в миллисекундах: " + *lockTime*);

}

**else** **if** (command.equals("set locktime")) {

System.***out***.print("Пожалуйста, введите новое время продолжительности блокировки файловых объектов в миллисекундах (целое число): ");

**try** {

*lockTime* = Long.*parseLong*(sc.nextLine());

System.***out***.println("Время задержки успешно установлено!");

}

**catch** (Exception e) {

System.***out***.println("Было введено не число!");

}

}

**else** **if** (command.equals("clear")) {

*clear*();

}

**else** **if** (command.equals("info")) {

System.***out***.println("\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\* Информация \*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*\nАдрес технической поддержки: Chukseev.nikita@yandex.ru\nПроект \"Система блокировки файлов AnViS\"\nКоманды:\n info - вывести информацию о данном проекте\n stop - завершить работу программы\n lock - запустить поток блокировки файлового объекта с временной привязкой\n lockfull - запустить поток блокировки файлового объекта без временной привязки\n unlock - остановить поток блокировки файлового объекта\n list - отображение списка всех файловых объектов находящихся под блокировкой\n set time - установить время интервала\n get time - получить время интервала\n set locktime - установить время продолжительности блокировки файлового объекта\n get locktime - получить время продолжительности блокировки файлового объекта\n clear - остановить все потоки блокировки файловых объектов\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*");

}

**else** **if** (command.equals(" ")||command.length()==0) {}

**else** {

System.***out***.println("Команда не распознана!");

}

}

**catch** (Exception ee) { //обработка ошибок

ee.printStackTrace(); //вывод ошибок

}

}

sc.close();

}

**catch** (Exception e) { //обработка ошибок

e.printStackTrace(); //вывод ошибок

}

## Основные функции

**public** **static** **void** add(String path, **boolean** islockfull) **throws** Exception { //добавить поток блокировки файла по пути path

//islockfull - флаг, который отвечает за полную блокировку файла, а не временную

//если по указанному пути находится папка, то сканируем абсолютно все вложенные файлы и выполняем для них эту же самую функцию

**if** (Runtime.*getRuntime*().freeMemory()>*maxmemoryfree*) {

*optimize*();

**if** (**new** File(path).exists()) {

path = **new** File(path).getAbsolutePath();

File f = **new** File(path);

**if** (f.isFile()) {

**boolean** isFree = **true**;

**for** (**int** c = 0; c < *threadlist*.length; c++) {

**if** (*threadlist*[c].getName().equals(path)) {

isFree = **false**;

}

}

**if** (isFree) {

lock l = **new** lock(path, islockfull);

l.start();

l.setName(path);

//добавляем в массив потоков новый поток

Thread [] ta = **new** Thread[*threadlist*.length+1];

**for** (**int** c = 0; c < *threadlist*.length; c++) {

ta[c] = *threadlist*[c];

}

ta[*threadlist*.length] = l;

*threadlist* = ta.clone();

}

**else** {

System.***out***.println("Поток блокировки файла \"" + path + "\" уже запущен!");

}

}

**else** {

**int** length = f.listFiles().length;

**for** (**int** c = 0; c < length; c++) {

*add*(f.listFiles()[c].getAbsolutePath(), islockfull);

}

}

}

**else** {

System.***out***.println("Такого файла не существует!");

}

}

**else** {

System.***out***.println("Достигнута верхняя граница использования оперативной памяти для данной программы!");

}

}

**public** **static** **void** delete(String path) **throws** Exception { //удаление потока блокировки файла по указанному пути path

//если по указанному пути находится папка, то сканируем абсолютно все вложенные файлы и выполняем для них эту же самую функцию

path = **new** File(path).getAbsolutePath();

File f = **new** File(path);

**if** (f.exists()) {

**if** (f.isFile()) {

**for** (**int** c = 0; c < *threadlist*.length; c++) {

**if** (*threadlist*[c]!=**null**&&*threadlist*[c].getName().equals(path)) {

*threadlist*[c].interrupt();

}

}

}

**else** {

**for** (**int** c = 0; c < **new** File(path).listFiles().length; c++) {

*delete*(**new** File(path).listFiles()[c].getAbsolutePath());

}

}

}

**else** {

System.***out***.println("Такого файла не существует!");

}

*optimize*();

}

**public** **static** **void** clear () **throws** Exception { //остановить все потоки блокировки файлов

System.***out***.println("Остановка всех потоков блокировки файлов...");

**for** (**int** c = 0; c < *threadlist*.length; c++) {

*delete*(*threadlist*[c].getName());

}

System.***out***.println("Все потоки блокировки файлов успешно остановлены!");

*optimize*();

}

**public** **static** **void** optimize() **throws** Exception { //данная функция удаляет из массива потоков threadlist те потоки,

//которые являются не активными или пустыми

**int** counter = 0;

**for** (**int** c = 0; c < *threadlist*.length; c++) {

**if** (*threadlist*[c]==**null**||!*threadlist*[c].isAlive()) {

counter++;

}

}

**if** (counter > 0) {

Thread [] temp = **new** Thread[*threadlist*.length-counter];

**int** s = 0;

**for** (**int** c = 0; c < *threadlist*.length; c++) {

**if** (*threadlist*[c]!=**null**&&*threadlist*[c].isAlive()) {

temp[s++] = *threadlist*[c];

}

}

*threadlist* = temp.clone();

}

}

**public** **static** **void** list() **throws** Exception { //перечисление всех активных потоков блокировки файлов

**if** (*threadlist*.length>0) {

System.***out***.print("Активные потоки блокировки файлов: ");

System.***out***.print("\n[1] " + *threadlist*[0].getName());

**for** (**int** c = 1; c < *threadlist*.length; c++) {

System.***out***.print(";\n[" + (c+1) + "] " + *threadlist*[c].getName());

}

System.***out***.println(".");

}

**else** {

System.***out***.println("На данный момент активных потоков блокировки файлов нет!");

}

}

## Поток блокировки файлов

**public** **static** **class** lock **extends** Thread { //поток блокировки файлов

**public** String path = ""; //путь к файлу

**public** **long** time = 0L; //время начала блокировки

**public** **boolean** isFullLock = **false**; //флаг полной блокировки

RandomAccessFile raf; //объект, благодаря которому мы и будем блокировать файл

**public** lock(String name, **boolean** islockfull) { //инициализация в конструкторе класса

path = name;

time = System.*currentTimeMillis*(); //присваеваем переменной начального времени текущее значение времени

isFullLock = islockfull;

System.***out***.println("Поток блокировки файла \"" + path + "\" успешно запущен!");

}

**public** **void** run () {

**try** {

**if** (**new** File(path).exists()) { //выполняем только если файловый объект существует

File f = **new** File(path);

**if** (f.isFile()) { //выполняем только если файловый объект является не папкой, а файлом

raf = **new** RandomAccessFile(path, "rw"); //блокируем файл

**while** ((*isRun* && (System.*currentTimeMillis*()-time) < *lockTime* && !isFullLock) || isFullLock) { //ожидаем, пока выполняется условие ожидания

//если блокировка временная, то ждём, пока истечёт время, а если полная, то ждём ручной разблокировки

*sleep*(1L);

}

raf.close(); //закрываем объект, блокирующий указанный файл

}

}

**else** {

System.***out***.println("Файлового объекта по пути \"" + path + "\" не существует!");

}

System.***out***.println("Время блокировки файла \"" + path + "\" истекло! Его поток блокировки успешно остановлен!");

}

**catch** (Exception e) { //в случае исключения тоже закрываем объект, блокирующий указанный файл

**try** {

raf.close();

}

**catch** (Exception ex) {}

System.***out***.println("Поток блокировки файла \"" + path + "\" остановлен!");

}

}

}

# Результат

Перед нами три иконки — программа проекта, скрипт запуска и текстовый файл (Рисунок 1). Запускаем программу AnViS.jar скриптом start.cmd (содержимое на рисунке 2) и пробуем заблокировать файл file.txt (содержимое на рисунке 3). Вводим команду ”info” для ознакомления с инструкцией программы, после чего пишем команду блокировки ”lock” и вводим относительный или абсолютный путь (Рисунок 4). Относительный путь в любом случае будет преобразован программой в абсолютный путь автоматически. В результате, наш файл file.txt успешно блокируется (Рисунок 5). По истечении времени блокировки файла его поток блокировки автоматически останавливается (Рисунок 6). После того, как мы закончили работать в программе, вводим команду ”stop” и видим, что программа вывела сообщение о завершении работы и остановке активных потоков.



Рисунок 1. Иконки программы AnViS.jar, скрипта запуска start.cmd и текстового файла file.txt

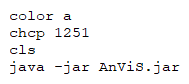


Рисунок 2. Содержимое скрипта запуска start.cmd

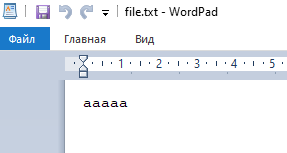


Рисунок 3. Содержимое файла file.txt

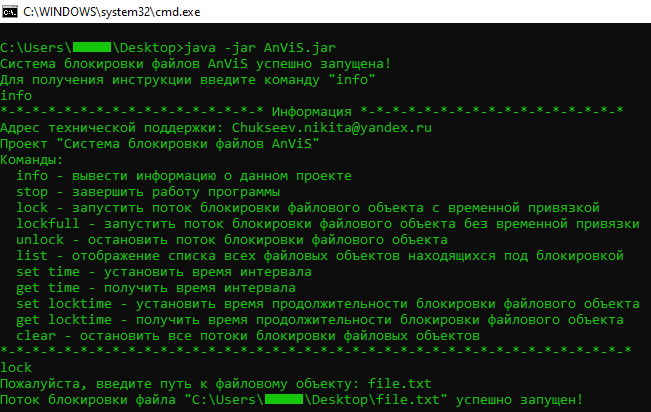


Рисунок 4. Запуск программы AnViS.jar и начало работы с ней. Блокировка файла file.txt

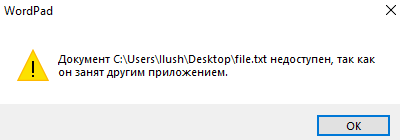


Рисунок 5. Результат блокировки файла file.txt

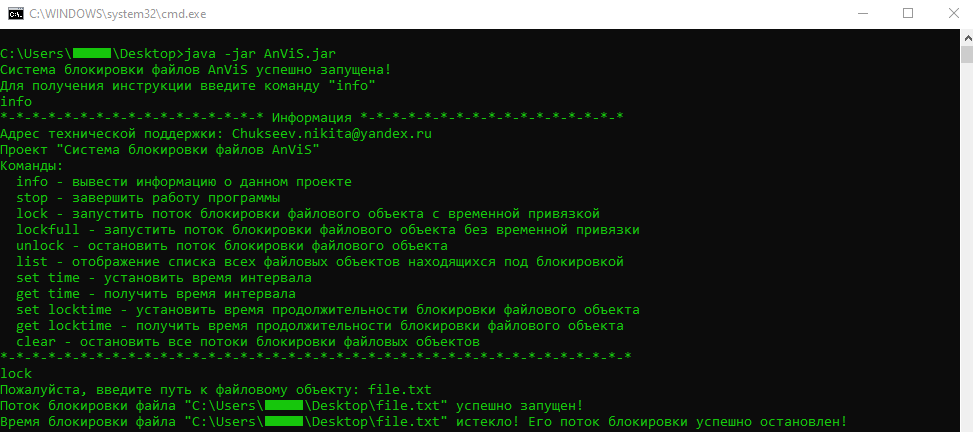


Рисунок 6. Остановка потока блокировки файла file.txt по причине истечения времени блокировки

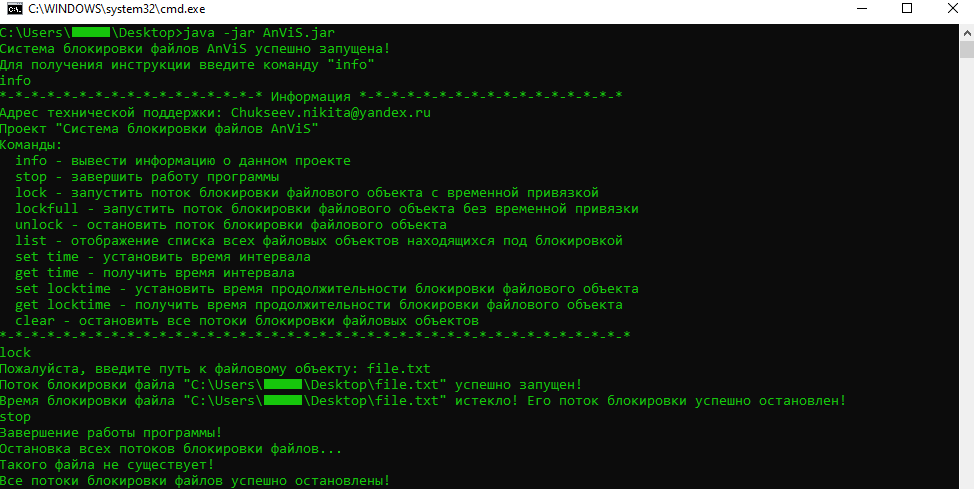


Рисунок 7. Ввод команды ”stop” и завершение работы программы

Программа успешно справляется со своей задачей и обрабатывает почти все возможные исключения. Код будет работать на версиях Java 8 и выше. Так же код может использоваться на разных платформах Java: ME, SE, EE.

Главная цель была успешно осуществлена. Создана кроссплатформенная и быстрая программа с русскоязычным дружественным интерфейсом. Программа успешно блокирует файлы почти всех типов. Также можно блокировать ярлыки программ. Тогда они не смогут запуститься до того момента, пока блокировка не будет снята.

Программу можно использовать и в качестве антивируса, и в качестве теста-вируса, и в качестве заморозки файла, т. е. его защиты от модификации и удаления и т. д.

Алгоритм может использоваться в любых антивирусах как дополнительная система контроля данных и обеспечения их безопасности.

Программа построена таким образом, что любые операции с потоками и массивами не приведут к переполнению как оперативной памяти, выделенной JVM, так и всей оперативной памяти в целом.

# Список использованных источников

1. https://habr.com/
2. https://javarush.ru/
3. https://metanit.com/
4. [https://ru.wikipedia.org](https://ru.wikipedia.org/)/
5. <https://ru.stackoverflow.com/>