

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

КУРСОВАЯ РАБОТА

Дисциплина: **Проектирование ОС и компонентов**

Тема: **Системная изоляция процессов в Windows средствами LPAC**

Выполнил студент группы 13541/3

(подпись) Д.В. Круминьш

Руководитель

(подпись) Е.В. Душутина

Санкт-Петербург
2018 г.

**ЗАДАНИЕ
НА ВЫПОЛНЕНИЕ КУРСОВОГО ПРОЕКТА**

студенту группы _13541/3_ _____ Круминьш Денис Валерьевич _____
(номер группы) (фамилия, имя, отчество)

1. Тема проекта (работы): Системная изоляция процессов в Windows средствами LPAC.

2. Срок сдачи законченного проекта (работы) 28.05.2018

3. Исходные данные к проекту (работе):
Платформа x64 на реальной системе Windows 10. Создание песочницы для приложения с использованием технологии LPAC.

4. Содержание пояснительной записки (перечень подлежащих разработке вопросов): введение, обзор LPAC, определение базовой функциональности, реализация изоляции, проверка изоляции, сравнение работы обычного и изолированного приложения, заключение, список использованных источников, приложения.

Дата получения задания: « 7 » апреля 2018 г.

Руководитель _____ Е.В. Душутина _____
(подпись) (инициалы, фамилия)

Задание принял к исполнению _____ Д.В. Круминьш _____
(подпись студента) (инициалы, фамилия)

7 апреля 2018 г.
(дата)

Содержание

Введение	4
1 Обзор LPAC	5
1.1 Принципы работы контейнера	5
1.2 Оркестраторы контейнера	8
1.3 Аналоги	9
1.3.1 FreeBSD Jail	9
1.3.2 Docker	10
2 Определение базовой функциональности	11
3 Реализация изоляции	11
3.1 Создание контейнера	11
3.2 Установка доступов контейнера	13
3.3 Переменные окружения	14
3.4 Доступ к ФС	15
3.5 Запуск приложения	17
4 Проверка изоляции	18
4.1 Проверка на нахождение в контейнере	18
4.2 Переменные окружения	20
4.3 Доступ к файлам	22
4.4 Доступ к сети	22
4.5 Список процессов	23
5 Сравнение работы обычного и изолированного приложения	24
Заключение	27
Список использованных источников	28
Приложения	30
Приложение 1. main.cpp	30
Приложение 2. ContainerCreate.h	30
Приложение 3. ContainerCreate.cpp	30
Приложение 4. ContainerTest.h	37
Приложение 5. ContainerTest.cpp	37
Приложение 6. Структура проекта	42

Введение

Современные серверы обладают избыточной производительностью, и приложения порой не используют даже их части. В результате системы какое-то время «простаивают», вместо выполнения полезной работы. Выходом стала виртуализация, позволяющая запускать несколько ОС на одном сервере, гарантированно разделяя их между собой и выделяя каждой нужное количество ресурсов. Но прогресс не стоит на месте. Следующий этап — **микросервисы**, когда каждая часть приложения развертывается отдельно, как самодостаточный компонент, который легко масштабируется под нужную нагрузку и обновляется.

Изоляция предотвращает вмешательство в работу микросервиса со стороны других приложений. С появлением проекта Docker, упростившего процесс упаковки и доставки приложений вместе с окружением, архитектура микросервисов получила дополнительный толчок в развитии.

Проект Docker нативен для Linux систем, но для систем семейства Windows, приходилось ставить виртуальную машину, что является достаточно громоздким решением.

И совсем недавно, в ОС семейства Windows появились свои, нативные контейнеры - **Application Container** и **LPAC** - Less Privileged Application Container.

1 Обзор LPAC

Контейнеры — это способ разместить приложение в собственной изолированной «коробке». У приложения в контейнере нет сведений о других приложениях или процессах, размещенных за пределами этой «коробки». Все необходимое приложению для успешной работы также находится в этом контейнере. Куда бы контейнер не переместить, приложение всегда будет работать, так как оно получит все необходимое для запуска.

Контейнеры появились сравнительно недавно, где LPAC является частным случаем Application Container:

- **AC** - Application Container
 - Появился в Windows Server 2016;
 - Разработчики были вдохновлены успехами Docker(Linux).
- **LPAC** - Less Privileged Application Container
 - Появился в Windows 10 Creators Update(2017 год).

Причины использования контейнеров:

- Легкая переносимость приложений;
 - Так как контейнер содержит все необходимое для запуска приложения, он отличается высокой переносимостью и может запускаться на любом компьютере под управлением Windows Server 2016 и ОС старше.
- Востребованность при использовании микросервисов.

1.1 Принципы работы контейнера

Контейнеры — это изолированная и переносимая среда выполнения с контролируемыми ресурсами, которая работает на хост-компьютере или виртуальной машине. Приложения или процессы, которые запускаются в контейнере, поставляются вместе со всеми необходимыми компонентами и файлами конфигурации. Они не имеют представления о других процессах, выполняющихся вне контейнера.

Узел контейнера предоставляет набор ресурсов, и контейнер будет использовать только их. Контейнер считает, что других ресурсов, помимо предоставленных, не существует, поэтому он не может взаимодействовать с ресурсами, выделенными соседнему контейнеру.

При создании контейнеров Windows и последующей работе с ними пригодятся перечисленные ниже основные понятия.

Узел контейнера. Физический или виртуальный компьютер, настроенный для работы с контейнерами Windows. На узле контейнера работает один или несколько контейнеров Windows.

Образ контейнера. По мере того как в файловую систему или реестр контейнера вносятся изменения (например, при установке программного обеспечения), они регистрируются в "песочнице". Во многих случаях может потребоваться зарегистрировать это состояние, чтобы применить внесенные изменения при создании новых контейнеров. В этом и заключается суть образа: после остановки работы контейнера можно либо отключить "песочницу" либо преобразовать ее в новый образ контейнера. Предположим, что в контейнер происходит установка MySQL. Создание нового образа на базе этого контейнера будет происходить аналогично его развертыванию. Этот образ будет содержать только внесенные изменения (MySQL) и при этом работать в виде слоя поверх образа ОС контейнера.

"Песочница". После запуска контейнера все операции записи (изменения файловой системы и реестра либо установка программного обеспечения) регистрируются на уровне "песочницы".

Образ ОС контейнера. Контейнеры развертываются из образов. Образ ОС контейнера — это первый из возможного множества слоев образа, составляющих контейнер. Этот образ представляет собой среду операционной системы. Образ ОС контейнера невозможно изменить.

Репозиторий контейнера. При каждом создании образа контейнера этот образ и его зависимости сохраняются в локальном репозитории. Эти образы можно использовать повторно много раз на узле контейнера. Образы контейнеров также можно хранить в открытом или закрытом реестре (например, Docker Hub), чтобы использовать на многих других узлах контейнеров.

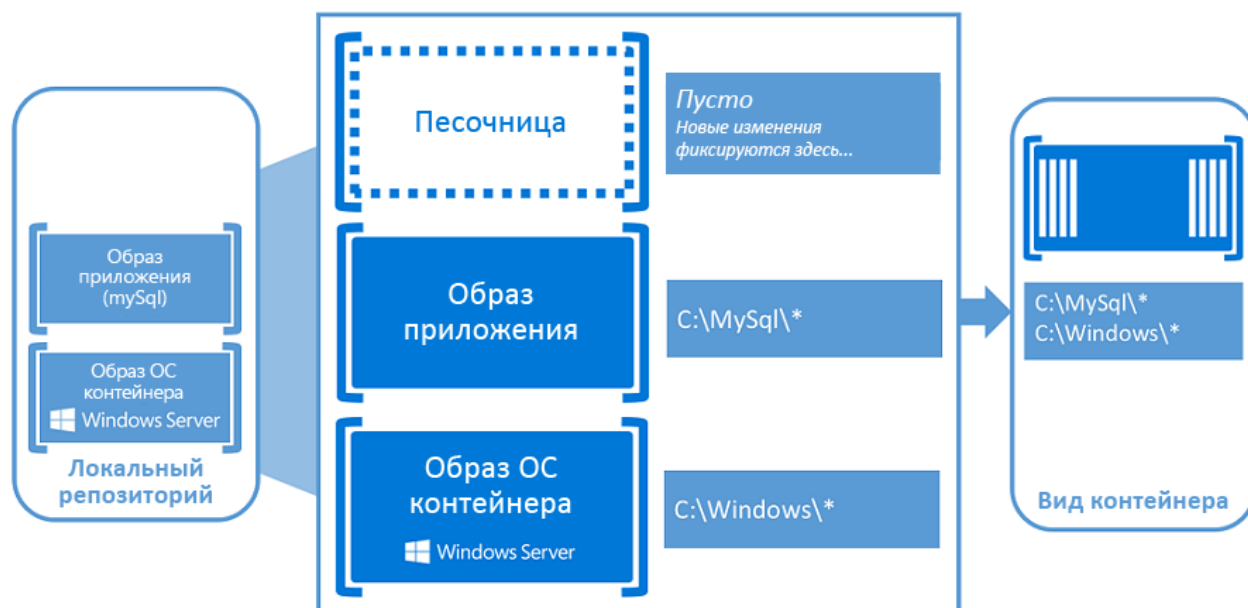


Рис. 1: Взаимодействие с контейнерами

Отличие контейнеров от виртуальных машин заключается в том, что контейнеры не загружают собственные копии ОС, библиотеки, системные файлы и прочее. операционная система как бы делится с контейнером. Единственное, что дополнительно требуется, — это ресурсы, необходимые для запуска приложения в контейнере. В результате контейнер стартует в считанные секунды и меньше нагружает систему, чем в случае применения виртуальных машин.

Подобно **Docker** имеются тенденции к развитию репозитория с контейнерами, то есть к распространению уже предварительно настроенных контейнеров, с определенными функциями.

Контейнеры Windows используют одно ядро с ОС, которое динамично разделяют между собой. Процесс распределения (CPU, ОЗУ, сеть) берет на себя ОС. При необходимости можно ограничить максимально доступные ресурсы, выделяемые контейнеру. Файлы ОС и запущенные службы проецируются в пространство имен каждого контейнера. Такой тип контейнера эффективно использует ресурсы, уменьшая накладные расходы, а значит, позволяет более плотно размещать приложения.

Микросервисы

Наиболее прибыльной и востребованной тенденцией для использования контейнеров являются микросервисы. Микросервис - это подход к разработке приложений, где каждая часть приложения развертывается как полностью автономный компонент, называемый микросервисом, который можно индивидуально масштабировать и обновлять.

Например, подсистема приложения, которая получает запросы из общедоступного Интернета, может быть отделена от подсистемы, которая помещает запрос в очередь для поддержки бэкэнд-подсистемы и перебрасывает их в базу данных.

Микросервис - это не новый подход, и он явно не привязан к контейнерам, но преимущества контейнеров увеличиваются при применении к сложному приложению на основе микросервиса. Это означает, что микросервис может быстро масштабироваться, чтобы удовлетворить повышенную нагрузку, пространство имен и изоляция ресурсов контейнеров не позволяет одному экземпляру микросервиса вмешиваться в другие.

1.2 Оркестраторы контейнера

Благодаря малому размеру и ориентированности на приложение контейнеры удобно применять в средах с гибкой настройкой доставки и архитектурах, основанных на микрослужбах. Однако, при использовании контейнеров и микрослужб можно внедрить в свою среду сотни и тысячи компонентов. Можно вручную управлять несколькими десятками виртуальных машин и физических серверов, но не существует способа управления средой контейнеров промышленного масштаба без средств автоматизации. Автоматизация большого количества контейнеров и управление ими, а также их взаимодействиями называется оркестрацией.

Стандартное определение оркестрации включает следующие задачи:

- **Планирование:** поиск подходящего компьютера для запуска контейнера с учетом образа контейнера и запроса на ресурс. Сходство или удаление сходства: указать, что контейнеры в наборе должны запускаться рядом друг с другом (для повышения производительности) или достаточно далеко друг от друга (для обеспечения доступности).
- **Наблюдение за работоспособностью:** отслеживание сбоев контейнера и автоматическое изменение расписания для него.
- **Отработка отказа:** отслеживание запущенных задач на каждой машине и переназначение контейнеров с машин, на которых возник сбой, на работоспособные узлы.
- **Масштабирование:** добавление или удаление экземпляров контейнера для соответствия запросу (ручное или автоматическое).
- **Сеть:** предоставление сети наложения для координации контейнеров при обмене данными между несколькими хост-машинами.

- **Обнаружение служб:** обеспечение автоматической локализации контейнеров даже при перемещении между хост-машинами и изменении IP-адреса.
- **Координация обновления приложений:** управление обновлениями контейнера во избежание простоев и откат до предыдущей версии в случае сбоя.

1.3 Аналоги

1.3.1 FreeBSD Jail

Jails основывается на концепции chroot (2), которая используется для изменения корневого каталога набора процессов. Это создает безопасную среду, отдельную от остальной части системы. Процессы, созданные в среде chrooted, не могут получить доступ к файлам или ресурсам за ее пределами. По этой причине компрометация службы, запущенной в chrooted-среде, не должна позволять злоумышленнику изменять внутренние системы. Однако, chroot имеет несколько ограничений. Он подходит для простых задач, которые не требуют большой гибкости или сложных, расширенных функций. Со временем было обнаружено, что многие способы избавиться от chrooted среды, что делает его менее идеальным решением для обеспечения безопасности.

Jails улучшают концепцию традиционной среды chroot несколькими способами. В традиционной среде chroot процессы ограничены только в той части файловой системы, к которой они могут получить доступ. Остальные системные ресурсы, системные пользователи, запущенные процессы и сетевая подсистема совместно используются chrooted процессами и процессами хост-системы. Jails расширяет эту модель путем виртуализации доступа к файловой системе, набору пользователей и сетевой подсистеме. Доступны более мелкозернистые элементы управления для настройки доступа к Jails. Jails можно рассматривать как тип виртуализации на уровне операционной системы.

Находясь "за решеткой root не имеет прав:

- Загружать модули ядра и каким-либо образом модифицировать ядро (например, через /dev/kmem).
- Изменять переменные ядра (за исключением kern.securelevel и kern.hostname).
- Создавать файлы устройств.
- Монтировать и демонтировать файловые системы.
- Изменять сетевые конфигурации.

- Создавать raw сокеты (поведение настраивается).
- Получать доступ к сетевым ресурсам, не ассоциированным с IP-адресом jail'a.
- Работать с System V IPC (поведение настраивается).
- Присоединяться к процессу и использовать ptrace(2).

1.3.2 Docker

Docker - одна из самых популярных платформ для работы с контейнерами для Linux.

В своем ядре docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере. Безопасная изоляция позволяет вам запускать на одном хосте много контейнеров одновременно. Легковесная природа контейнера, который запускается без дополнительной нагрузки гипервизора, позволяет вам добиваться больше от вашего железа.

Основа изоляции лежит на пространстве имен и контрольных группах.

Пространство имен(namespaces)

Docker использует технологию namespaces для организации изолированных рабочих пространств, которые мы называем контейнерами. Когда мы запускаем контейнер, docker создает набор пространств имен для данного контейнера.

Это создает изолированный уровень, каждый аспект контейнера запущен в своем пространстве имен, и не имеет доступ к внешней системе.

Список некоторых пространств имен, которые использует docker:

- pid: для изоляции процесса;
- net: для управления сетевыми интерфейсами;
- ipc: для управления IPC ресурсами. (IPC: InterProcess Communication);
- mnt: для управления точками монтирования;
- utc: для изолирования ядра и контроля генерации версий(UTC: Unix timesharing system).

Control groups (контрольные группы)

Docker также использует технологию cgroups или контрольные группы. Ключ к работе приложения в изоляции, предоставление приложению только тех ресурсов, которые вы

хотите предоставить. Это гарантирует, что контейнеры будут хорошими соседями. Контрольные группы позволяют разделять доступные ресурсы железа и если необходимо, устанавливать пределы и ограничения. Например, ограничить возможное количество памяти контейнеру.

Существует еще множество аналогов, но всех их объединяет следующее:

- Удобная инкапсуляция приложений.
- Быстрая загрузка;
- Отсутствие дополнительной нагрузки на гипервизор;
- Простое масштабирование.

Application Container и LPAC повторяют опыт Linux, внося на ос семейства Windows нативную контейнеризацию приложений.

2 Определение базовой функциональности

В данной работе просходит работы с LPAC контейнерами, и для запущенного в контейнере приложения предполагается некоторая функциональность, в частности:

- отдельное пространство имен, которое не должно затрагивать пространство имен системы;
- доступ к файловой системе должен быть ограничен предварительно заданными настройками;
- доступ к сетевым возможностям, а также разграничение локальной и общей сети;
- недоступность процессов вне контейнера.

3 Реализация изоляции

3.1 Создание контейнера

Для реализации изоляции, сперва необходимо создать контейнер и выполнить определенные настройки.

Процесс запуска выглядит следующим образом:

1. Создать контейнер, с помощью функции **CreateAppContainerProfile**;
2. [Опционально] Установка доступов, с помощью функции **CreateWellKnownSid**;
3. [Опционально] Установка конкретных доступов, например для файлов используя функцию **SetEntriesInAclA**;
4. [Опционально] Изменение переменных окружения, с помощью функции **ExpandEnvironmentStringsA**;
5. Запуск программы, с помощью **CreateProcessA**.

Контейнер создается с помощью функции **CreateAppContainerProfile**[1]

```
HRESULT WINAPI CreateAppContainerProfile (
    _In_   PCWSTR      pszAppContainerName ,
    _In_   PCWSTR      pszDisplayName ,
    _In_   PCWSTR      pszDescription ,
    _In_   PSID_AND_ATTRIBUTES pCapabilities ,
    _In_   DWORD        dwCapabilityCount ,
    _Out_  PSID         *ppSidAppContainerSid
);
```

Листинг 1: Прототип CreateAppContainerProfile

- pszAppContainerName - имя контейнера;
- pszDisplayName - отображаемое имя контейнера;
- pszDescription - описание контейнера;
- pCapabilities - разрешения для контейнера, для LPAC значение NULL;
- dwCapabilityCount - количество разрешений;
- ppSidAppContainerSid - выходной параметр - **SID**.

В случае если контейнер, с заданным именем существует, необходимо вызвать функцию **DeriveAppContainerSidFromAppContainerName**[2] для получения его SID.

```
HRESULT WINAPI DeriveAppContainerSidFromAppContainerName (
    _In_   PCWSTR pszAppContainerName ,
    _Out_  PSID   *ppsidAppContainerSid
);
```

Листинг 2: Прототип DeriveAppContainerSidFromAppContainerName

Security Identifier (SID) — идентификатор безопасности, структура данных в Windows, которая может идентифицировать системные объекты, например элементы управле-

ния доступом (Access Control Entries, ACE), токены доступа (Access Token), дескрипторы безопасности (Security Descriptor). SID всегда начинается с буквы S, далее идут числа, которые обозначают номер редакции ОС, источники выдачи, удостоверяющие центры и другую информацию.

Имеющиеся в системе SID можно посмотреть в следующей ветке реестра:

HKEY_CURRENT_USER/Software/Classes/Local Settings/Software/Microsoft/Windows/
CurrentVersion/AppContainerStorage/Mappings

3.2 Установка доступов контейнера

Под доступом подразумевается - атрибут **WELL_KNOWN_SID_TYPE**[3], список приведен на соответствующей странице **MSDN**[3].

Для установки доступа используется функция **CreateWellKnownSid**, которая возвращает SID доступа.

```
BOOL WINAPI CreateWellKnownSid(  
    _In_      WELL_KNOWN_SID_TYPE WellKnownSidType ,  
    _In_opt_  PSID                  DomainSid ,  
    _Out_opt_ PSID                  pSid ,  
    _Inout_   DWORD                 *cbSid  
);
```

Листинг 3: Прототип CreateWellKnownSid

Далее создается структура **SECURITY_CAPABILITIES**[4].

```
typedef struct _SECURITY_CAPABILITIES {  
    SID                      AppContainerSid ;  
    PSID_AND_ATTRIBUTES     Capabilities ;  
    DWORD                   CapabilityCount ;  
    DWORD                   Reserved ;  
} SECURITY_CAPABILITIES , *PSECURITY_CAPABILITIES ;
```

Листинг 4: Структура SECURITY_CAPABILITIES

- AppContainerSid - SID контейнера;
- Capabilities - список доступов, их SID;
- CapabilityCount - количество доступов;
- Reserved - зарезервированная на будущее переменная.

Данная структура используется в функции **UpdateProcThreadAttribute**[5].

```
BOOL WINAPI UpdateProcThreadAttribute(  
    _Inout_   LPPROC_THREAD_ATTRIBUTE_LIST lpAttributeList ,  
    _In_      DWORD dwFlags ,  
    _In_      DWORD_PTR Attribute ,  
    _In_      PVOID lpValue ,  
    _In_      SIZE_T cbSize ,  
    _Out_opt_ PVOID lpPreviousValue ,  
    _In_opt_  PSIZE_T lpReturnSize  
);
```

Листинг 5: Прототип UpdateProcThreadAttribute

- **Attribute** - должен быть задан как PROC_THREAD_ATTRIBUTE_SECURITY_CAPABILITIES - это означает, что следующий процесс будет создан в контейнере;
- **lpValue** - полученная ранее структура SECURITY_CAPABILITIES.

3.3 Переменные окружения

Изменить или добавить переменны окружения можно с помощью функции

SetEnvironmentVariable[6].

```
BOOL WINAPI SetEnvironmentVariable(  
    _In_      LPCTSTR lpName ,  
    _In_opt_  LPCTSTR lpValue  
);
```

Листинг 6: Прототип SetEnvironmentVariable

- lpName - имя переменной окружения;
- lpValue - значение переменной окружения.

Для извлечения значения используется функция **ExpandEnvironmentStrings**[7]

```
DWORD WINAPI ExpandEnvironmentStrings(  
    _In_      LPCTSTR lpSrc ,  
    _Out_opt_ LPTSTR lpDst ,  
    _In_      DWORD nSize  
);
```

Листинг 7: Прототип ExpandEnvironmentStrings

- lpSrc - имя переменной оркужения;

- lpDst - переменная, в которую будет выгружено значение переменной окружения;
- nSize - допустим размер значения для выгрузки.

В данной работе, производится добавление новой переменной окружения - **testName**, изменение значения существующей переменной - **USERDOMAIN**, а также получения значения переменных **temp** и **localappdata**.

```
SetEnvironmentVariable(L"testName", L"testValue");
SetEnvironmentVariable(L"USERDOMAIN", L"HELLOWORLD");

ExpandEnvironmentStringsA("%temp%", path, MAX_PATH - 1);
printf("New path of %temp%: %s\n", path);

ExpandEnvironmentStringsA("%localappdata%", path, MAX_PATH - 1);
printf("New path of %localappdata%: %s\n", path);
```

Листинг 8: Фрагмент кода по работе с переменными окружения

3.4 Доступ к ФС

Доступ к ФС(файлам) нельзя установить через **WELL_KNOWN_SID_TYPE** по той причине, что они не рассчитаны на какие-то конкретные случаи.

Для начала необходимо инициализировать структуру **EXPLICIT_ACCESS**[8]

```
typedef struct _EXPLICIT_ACCESS {
    DWORD          grfAccessPermissions;
    ACCESS_MODE     grfAccessMode;
    DWORD          grfInheritance;
    TRUSTEE         Trustee;
} EXPLICIT_ACCESS, *PEXPLICIT_ACCESS;
```

Листинг 9: Структура EXPLICIT_ACCESS

- grfAccessPermissions - маска с разрешениями;
- grfAccessMode - тип доступа(предоставление, изъятие);
- grfInheritance - флаги по наследованию доступов другими контейнерами;
- Trustee[9] - идентификация пользователя, группы или программы, которой будет предоставлен доступ.

```
typedef struct _TRUSTEE {
    PTRUSTEE          pMultipleTrustee;
    MULTIPLE_TRUSTEE_OPERATION MultipleTrusteeOperation;
```

```

TRUSTEE_FORM           TrusteeForm ;
TRUSTEE_TYPE           TrusteeType ;
LPCH                   ptstrName ;
} TRUSTEE, *PTRUSTEE;

```

Листинг 10: Структура TRUSTEE

- pMultipleTrustee - предоставление доступа прочим структурам типа TRUSTEE, сейчас может принимать только значение **NULL**;
- MultipleTrusteeOperation - взаимосвязь между структурами TRUSTEE;
- TrusteeForm - определения типа объекта **ptstrName** для которого предоставляется доступ;
- TrusteeType - для кого будет предоставлен доступ(пользователь, группа, компьютер...);
- ptstrName - указатель на объект для которого осуществляется доступ.

Далее необходимо инициализировать переменную типа **PACL** и выполнить функцию **GetNamedSecurityInfoA**[10].

```

DWORD WINAPI GetNamedSecurityInfo(
    _In_      LPTSTR          pObjectName ,
    _In_      SE_OBJECT_TYPE  ObjectType ,
    _In_      SECURITY_INFORMATION SecurityInfo ,
    _Out_opt_ PSID            *ppsidOwner ,
    _Out_opt_ PSID            *ppsidGroup ,
    _Out_opt_ PACL            *ppDacl ,
    _Out_opt_ PACL            *ppSacl ,
    _Out_opt_ PSECURITY_DESCRIPTOR *ppSecurityDescriptor
);

```

Листинг 11: Прототип GetNamedSecurityInfo

В данном случае важны следующие атрибуты:

- pObjectName - название объекта, если файл, то полный путь;
- ObjectType - тип объекта;
- SecurityInfo - флаг, который определяет какую информацию необходимо получить;
- ppDacl - доступы к объекту типа Access Control List.

Выходное значение ppSacl записывается в ранее созданную переменную типа PACL. Далее применяется функция **SetEntriesInAclA**[11], для дополнения уже существующих доступов.


```

DWORD WINAPI SetEntriesInAcl(
    _In_      ULONG          cCountOfExplicitEntries ,
    _In_opt_  PEXPLICIT_ACCESS pListOfExplicitEntries ,
    _In_opt_  PACL           OldAcl ,
    _Out_     PACL           *NewAcl
);

```

Листинг 12: Структура SetEntriesInAcl

- cCountOfExplicitEntries - количество элементов структуры типа EXPLICIT_ACCESS;
- pListOfExplicitEntries - ранее полученная структура, с доступами типа EXPLICIT_ACCESS;
- OldAcl - ранее полученный acl;
- NewAcl - обновленный acl, в который были добавлены заданные через EXPLICIT_ACCESS доступы.

Имея обновленный acl, необходимо вызвать функцию **SetNamedSecurityInfoA**[12] для обновления прав к объекту.

```

DWORD WINAPI SetNamedSecurityInfo(
    _In_      LPTSTR          pObjectName ,
    _In_      SE_OBJECT_TYPE  ObjectType ,
    _In_      SECURITY_INFORMATION SecurityInfo ,
    _In_opt_  PSID            psidOwner ,
    _In_opt_  PSID            psidGroup ,
    _In_opt_  PACL            pDacl ,
    _In_opt_  PACL            pSacl
);

```

Листинг 13: Структура SetNamedSecurityInfo

В данном случае важны следующие атрибуты:

- pObjectName - название объекта, если файл, то полный путь;
- ObjectType - тип объекта;
- SecurityInfo - флаг, который определяет какую информацию необходимо обновить;
- ppDacl - обновленные доступы к объекту типа Access Control List.

3.5 Запуск приложения

Для запуска приложения вызывается стандартная функция по запуску - **CreateProcessA**[13].

```

BOOL WINAPI CreateProcess(
    _In_opt_ LPCTSTR lpApplicationName ,
    _Inout_opt_ LPTSTR lpCommandLine ,
    _In_opt_ LPSECURITY_ATTRIBUTES lpProcessAttributes ,
    _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes ,
    _In_ BOOL bInheritHandles ,
    _In_ DWORD dwCreationFlags ,
    _In_opt_ LPVOID lpEnvironment ,
    _In_opt_ LPCTSTR lpCurrentDirectory ,
    _In_ LPSTARTUPINFO lpStartupInfo ,
    _Out_ LPPROCESS_INFORMATION lpProcessInformation
);

```

Листинг 14: Прототип CreateProcessA

В данном случае важны следующие атрибуты:

- lpApplicationName - путь к запускаемой в контейнере программе;
- dwCreationFlags - флаг приоритета запускаемого процесса;
- lpStartupInfo - структура для предоставления процессу информации о системе;
- lpProcessInformation - возвращаемая структура с данными о процессе.

В данном случае, для проверки изоляции используется самозапуск приложения.

4 Проверка изоляции

4.1 Проверка на нахождение в контейнере

Для проверки на нахождение в контейнере была написана соответствующая функция - **IsInAppContainer**.

```

BOOL IsInAppContainer()
{
    HANDLE process_token;
    BOOL is_container = 0;
    DWORD return_length;

    OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &process_token);

    if (!GetTokenInformation(process_token, TokenIsAppContainer, &is_container,
        ↪ sizeof(is_container), &return_length))

```

```

        return false;

    return is_container;
}

```

Листинг 15: Фрагмент кода, по проверки работы в контейнере

Спервая используется функция **OpenProcessToken**[14].

```

BOOL WINAPI OpenProcessToken(
    _In_   HANDLE   ProcessHandle ,
    _In_   DWORD    DesiredAccess ,
    _Out_  PHANDLE  TokenHandle
);

```

Листинг 16: Прототип OpenProcessToken

Где первым аргументом идет дескриптор текущего процесса, далее требуемый тип для вывода, в данном случае токен, и последним аргументом - TokenHandle выходная переменная для получения дескриптора токена.

Далее с помощью функции **GetTokenInformation**[15] происходит проверка токена на нахождение в контейнере.

```

BOOL WINAPI GetTokenInformation(
    _In_      HANDLE                TokenHandle ,
    _In_      TOKEN_INFORMATION_CLASS TokenInformationClass ,
    _Out_opt_ LPVOID                TokenInformation ,
    _In_      DWORD                 TokenInformationLength ,
    _Out_     PDWORD                ReturnLength
);

```

Листинг 17: Прототип GetTokenInformation

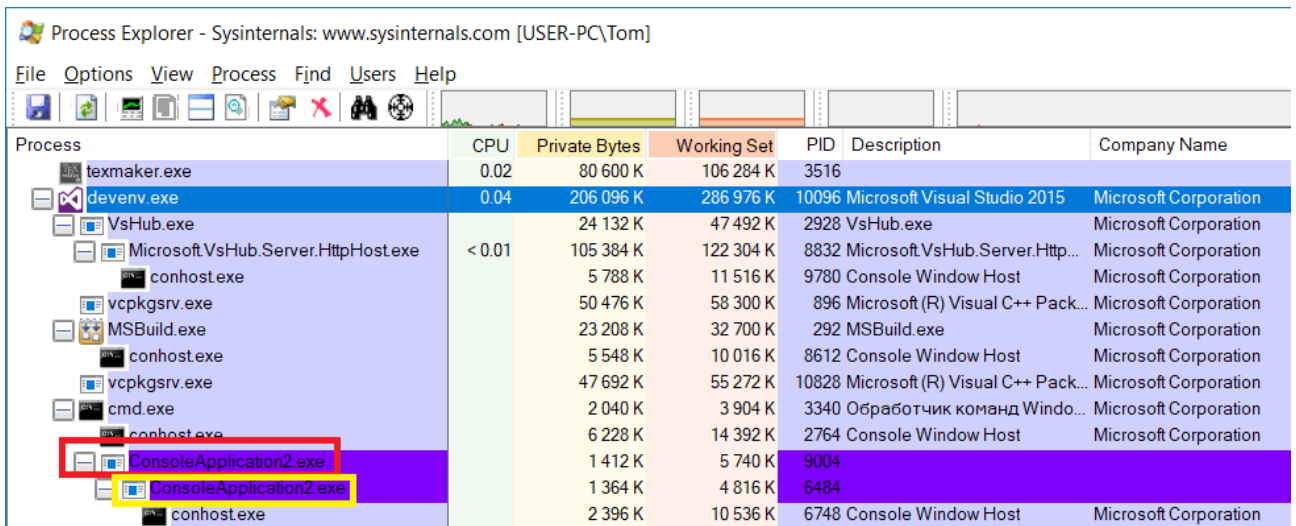
- TokenHandle - дескриптор токена;
- TokenInformationClass - определение типа возвращаемой информации;
- TokenInformation - указатель на переменную в которую вернуть результат;
- TokenInformationLength - размер;
- ReturnLength - указатель на переменную для хранения размера ответа.

Таким образом, если возвращаемое значение переменной **TokenInformation** является TRUE, это означает нахождение в контейнере.

4.2 Переменные окружения

В данной работе, производится добавление новой переменной окружения - **testName**, изменение значения существующей переменной - **USERDOMAIN**, а также получения значения переменных **temp** и **localappdata**.

Для тестирования используется программа **Process Explorer 16.21**. Работа производилась в Visual Studio, на рисунке процесс с именем **devenv.exe**.



Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
texmaker.exe	0.02	80 600 K	106 284 K	3516		
devenv.exe	0.04	206 096 K	286 976 K	10096	Microsoft Visual Studio 2015	Microsoft Corporation
VsHub.exe		24 132 K	47 492 K	2928	VsHub.exe	Microsoft Corporation
Microsoft.VsHub.Server.HttpHost.exe	< 0.01	105 384 K	122 304 K	8832	Microsoft.VsHub.Server.Http...	Microsoft Corporation
conhost.exe		5 788 K	11 516 K	9780	Console Window Host	Microsoft Corporation
vcpkgssrv.exe		50 476 K	58 300 K	896	Microsoft (R) Visual C++ Pack...	Microsoft Corporation
MSBuild.exe		23 208 K	32 700 K	292	MSBuild.exe	Microsoft Corporation
conhost.exe		5 548 K	10 016 K	8612	Console Window Host	Microsoft Corporation
vcpkgssrv.exe		47 692 K	55 272 K	10828	Microsoft (R) Visual C++ Pack...	Microsoft Corporation
cmd.exe		2 040 K	3 904 K	3340	Обработчик команд Windo...	Microsoft Corporation
conhost.exe		6 228 K	14 392 K	2764	Console Window Host	Microsoft Corporation
ConsoleApplication2.exe		1 412 K	5 740 K	9004		
ConsoleApplication2.exe		1 364 K	4 816 K	6484		
conhost.exe		2 396 K	10 536 K	6748	Console Window Host	Microsoft Corporation

Рис. 2: Структура процессов Visual Studio

На рисунке выделено два процесса **ConsoleApplication2.exe**, родительский с PID 9004, и дочерний, запущенный в контейнере с PID 6484.

Откроем каждый из процессов более подробно, и сравним их переменный окружения.

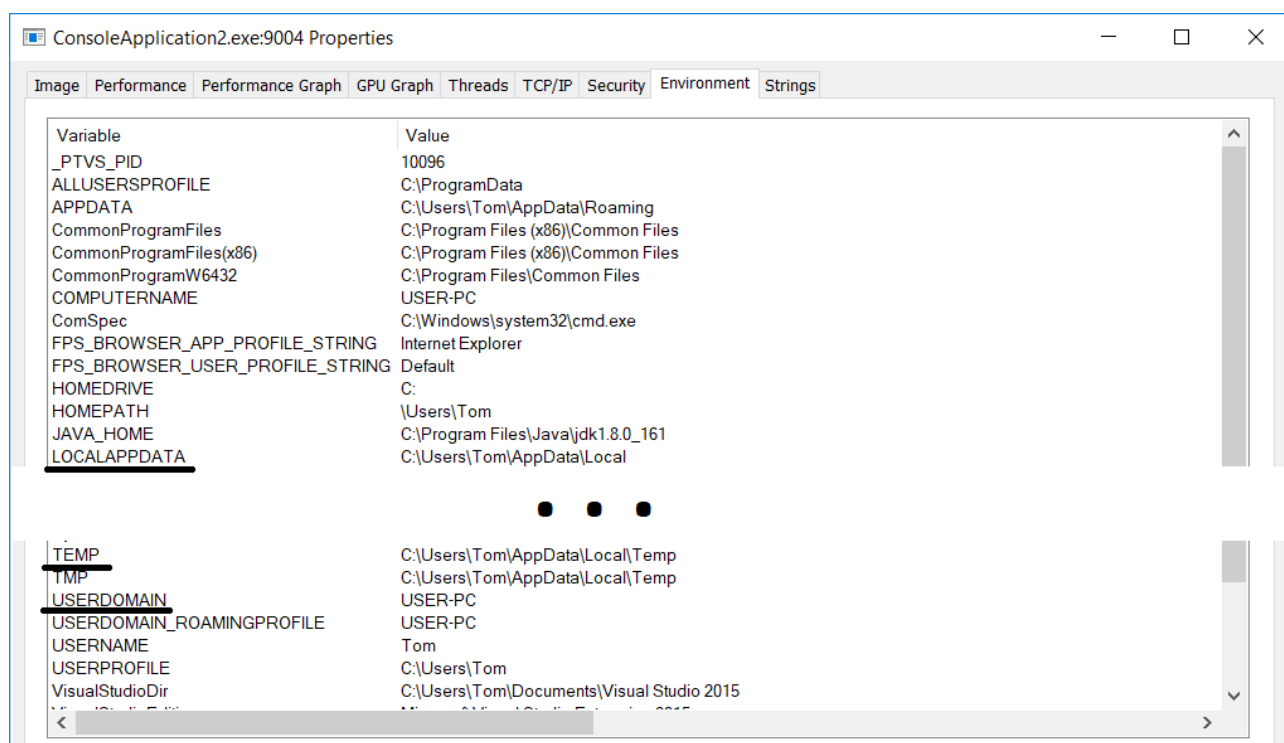


Рис. 3: Переменные окружения родительского процесса

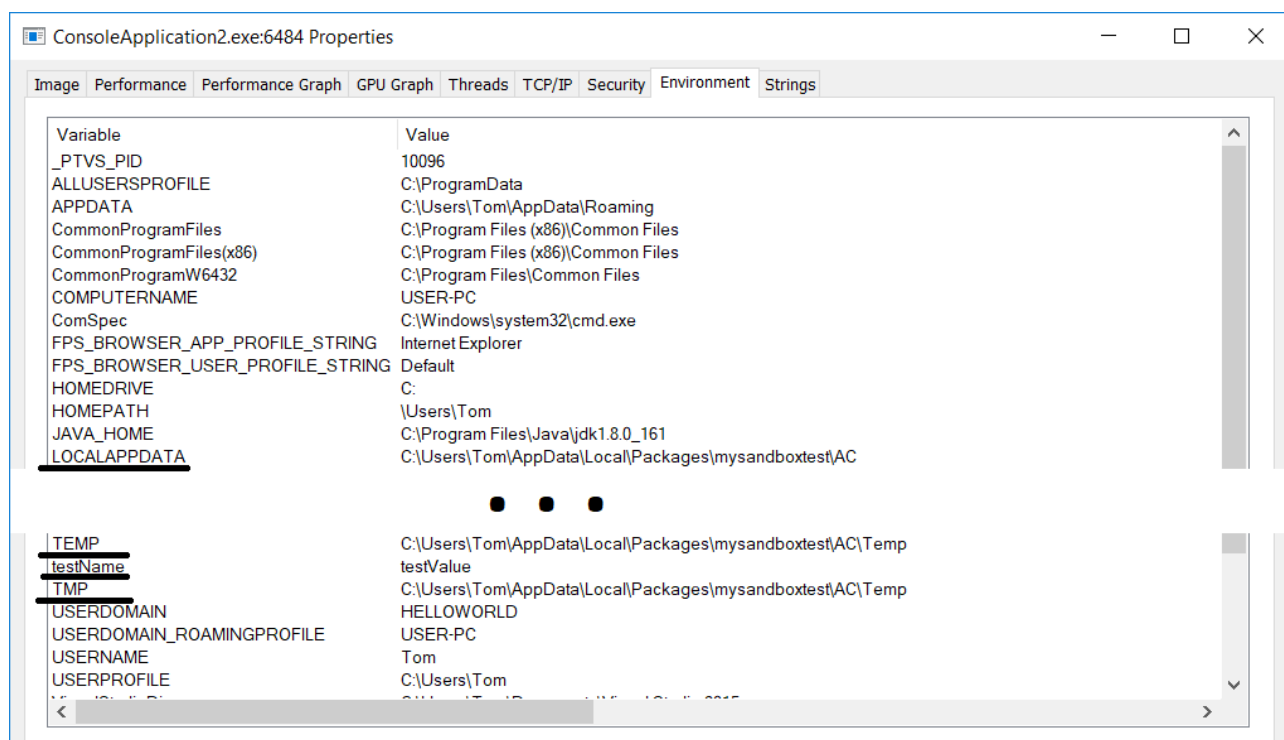


Рис. 4: Переменные окружения процесса в контейнере

Как видно из рисунков:

1. Переменная с именем **testName** присутствует лишь в процессе запущенном в контейнере;
2. Значение переменной **USERDOMAIN** в контейнере является измененным;
3. Переменные **temp** и **localappdata** имеют иные пути, они были автоматически изменены без ручного изменения. Соответственно процесс в контейнере имеет доступ к этим папкам.
 - Новый путь - C:\Users\Tom\AppData\Local\Packages\mysandboxtest\AC\Temp

4.3 Доступ к файлам

Будет совершена проверка на доступ к:

1. Файлу находящимся по пути
 - C:\Users\Tom\AppData\Local\Packages\mysandboxtest\AC\Tempбез предоставления каких-либо дополнительных разрешений;
2. Файлу на рабочем столе, к которому был дан доступ;
3. Файлу на рабочем столе, к которому не был дан доступ.

```
[+] Running filesystem test ...
Opening of file C:\Users\Tom\AppData\Local\Packages\mysandboxtest\AC\Temp\
↳ allowed_test.txt was successful
Opening of file C:\Users\Tom\desktop\allowed_test.txt was successful
Opening of file C:\Users\Tom\desktop\blocked_test.txt returned access denied
```

Листинг 18: Результат проверки

Как видно из лога, доступ к файлу **blocked_test.txt** был запрещен, а доступ к другим файлам был успешным, так как были выданы соответствующие права доступа.

4.4 Доступ к сети

Тестирование будет производиться путем открытия 80 порта, к следующим адресам:

- **108.177.14.138** (google.com);
- **192.168.1.1** (локальный адрес роутера).

Тесты будут выполнены с использованием следующих: WELL_KNOWN_SID_TYPE:

- WinCapabilityPrivateNetworkClientServerSid - доступ к локальной сети;
- WinCapabilityInternetClientServerSid - доступ к сети "Интернет".

Без задачи каких-либо WELL_KNOWN_SID_TYPE:

```
Connection to 108.177.14.138 was blocked
Connection to 192.168.1.1 was blocked
```

Листинг 19: Полный запрет сети

WinCapabilityPrivateNetworkClientServerSid:

```
Connection to 108.177.14.138 was blocked
Connection to 192.168.1.1 was successful
```

Листинг 20: Доступ только к локальной сети

WinCapabilityInternetClientServerSid:

```
Connection to 108.177.14.138 was successful
Connection to 192.168.1.1 was blocked
```

Листинг 21: Доступ только к сети "Интернет"

Вместе:

```
Connection to 108.177.14.138 was successful
Connection to 192.168.1.1 was successful
```

Листинг 22: Полный доступ к сети

Как видно из результатов, тестирование сети прошло успешно, доступ были предоставлены согласно выбранным WELL_KNOWN_SID_TYPE.

4.5 Список процессов

Была написана функция **ProcessListTest** прикрепленная в приложении 5, суть которой заключается в выводе всех процессов в системе. Данная функция была запущена как из родительском процессе, так и в дочернем, изолированном процессе.

```
Found process: [System Process]
Found process: ConsoleApplication2.exe
Found process: conhost.exe
```

Листинг 23: Список процессов из изолированного процесса

```
Found process: [System Process]
Found process: System
Found process: smss.exe
Found process: csrss.exe
Found process: wininit.exe
Found process: services.exe
Found process: lsass.exe
Found process: svchost.exe
Found process: WUDFHost.exe
Found process: fontdrvhost.exe
Found process: svchost.exe
Found process: WUDFHost.exe
Found process: svchost.exe
...
```

Листинг 24: Список процессов из обычного процесса

Как видно из логов, изолированный процесс не видит прочих процессов в системе.

5 Сравнение работы обычного и изолированного приложения

Запустим все тесты сразу для обычного и изолированного процесса.

```
[+] Running filesystem test...
Path of %temp%: C:\Users\Tom\AppData\Local\Temp
Path of %localappdata%: C:\Users\Tom\AppData\Local
Opening of file C:\Users\Tom\AppData\Local\Temp\allowed_test.txt was successful
Opening of file C:\Users\Tom\desktop\allowed_test.txt was successful
Opening of file C:\Users\Tom\desktop\blocked_test.txt was successful
[+] Filesystem testing done

[+] Running network test...
Connection to 108.177.14.138 was successful
Connection to 192.168.1.1 was successful
[+] Network testing done

[+] Running process list testing...
Found process: [System Process]
Found process: System
Found process: smss.exe
Found process: csrss.exe
```



```
Found process: wininit.exe
Found process: csrss.exe
Found process: services.exe
...
Found process: svchost.exe
Found process: cmd.exe
Found process: conhost.exe
Found process: ConsoleApplication2.exe
[+] Process list testing done
```

Листинг 25: Лог обычного процесса

```
[+] Running filesystem test...
Path of %temp%: C:\Users\Tom\AppData\Local\Packages\mysandboxtest\AC\Temp
Path of %localappdata%: C:\Users\Tom\AppData\Local\Packages\mysandboxtest\AC
Opening of file C:\Users\Tom\AppData\Local\Packages\mysandboxtest\AC\Temp\
    ↳ allowed_test.txt was successful
Opening of file C:\Users\Tom\desktop\allowed_test.txt was successful
Opening of file C:\Users\Tom\desktop\blocked_test.txt returned access denied
[+] Filesystem testing done

[+] Running network test...
Connection to 108.177.14.138 was blocked
Connection to 192.168.1.1 was blocked
[+] Network testing done

[+] Running process list testing...
Found process: [System Process]
Found process: ConsoleApplication2.exe
Found process: conhost.exe
[+] Process list testing done
```

Листинг 26: Лог изолированного процесса

Как видно из логов, для обычного процесса доступ к сети был полностью разрешен, а в изолированном полностью заблокирован. Тоже касается файлов, обычный процесс смог получить доступ ко всем файлам, а изолированный лишь к доступным.

Рассмотрим работу прочих приложений.

Запуск блокнота по пути - C:\Windows\notepad.exe происходит без каких-либо проблем, что говорит о возможности подгрузки GUI. Из изолированного блокнота был сохранен файл по пути

C:\Users\Tom\AppData\Local\Packages\mysandboxtest\AC\Temp

А из не изолированного по пути **E:**. Далее было произведено сравнение свойств файлов.

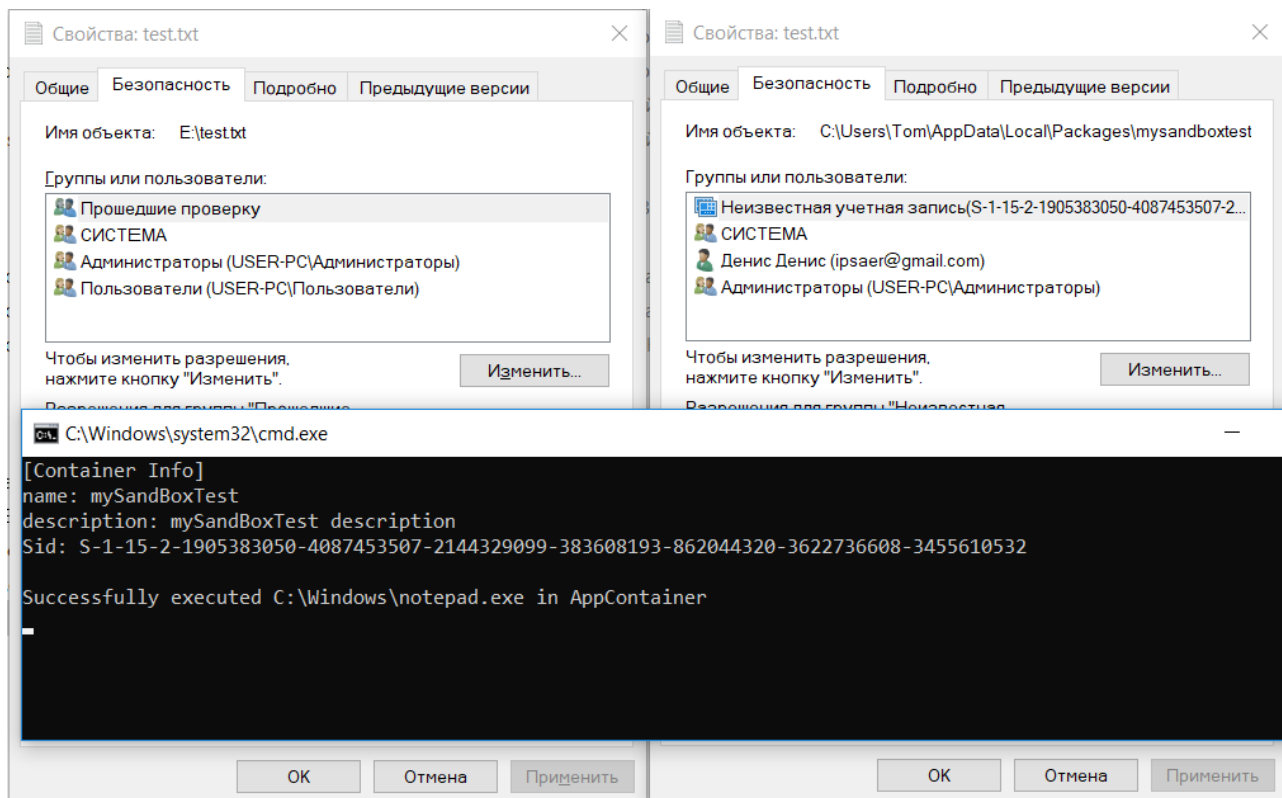


Рис. 5: Сравнение свойств файлов

Как видно из рисунка, на вкладке безопасность, у файла справа первой строкой записано - **Неизвестная учетная запись(S-15-2-19...)**. И если взглянуть на вывод консоли, становится понятно что неизвестная учетная запись является SID контейнера. То есть в данном случае вместо привычного пользователя или группы выступает идентификатор контейнера.

Запуск более масштабных приложений: Google Chrome, Internet Explorer и т.д. не приводил к успешному запуску. Процесс данных программ был создан, но никаких ошибок или окон GUI не последовало.

Это скорее всего связано с тем, что у программ имеются зависимости от прочих сервисов, настроек реестра, прочих файлов(то что происходит во время установки программы)

Заключение

В данной работе была рассмотрена изоляция процессов в Windows средствами LPAC. Технология Application Container и LPAC являются сравнительно новыми(2017 год). Во время чтения используемых функций на msdn было видно что некоторые параметры еще не используется в полной мере или вовсе пока являются заглушками.

Если сравнивать с Linux решениями по контейнеризации, то пока использование Application Container несколько затруднено из-за:

- Отсутствует какой-либо репозиторий с готовыми контейнерами;
- Средство пока еще слабо документировано.

Но данные проблемы решаемы, и уже сейчас идет создание открытого репозитория с контейнерами.

В работе была продемонстрирована работы с ФС, сетевыми функциями, переменными окружения, процессами системы из изолированного процесса. Все возможности изолированного процесса как и ожидалось были жестко ограничены, без расширения соответствующих прав доступа.

В работе были произведены попытки запуска более масштабных приложений, наподобии браузеров, но это не увенчалось успехом. Это связано с тем, что первоначальная цель контейнеризации была именно изоляция процессов, что сейчас широко применяется в микросервисах.

Помимо микросервисов, технология контейнеризации широко применяется в антивирусных программах. В которых необходимо оградить внутренние процесс от внешних вмешательств. С приходом технологии Application Container на ОС семейства Windows появляется возможность изолировать процессы используя нативные возможности системы.

Список использованных источников

- [1] CreateAppContainerProfile function [Электронный ресурс]. — URL: [https://msdn.microsoft.com/ru-ru/library/windows/desktop/hh448539\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/hh448539(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [2] CreateAppContainerProfile function [Электронный ресурс]. — URL: [https://msdn.microsoft.com/en-us/library/windows/desktop/hh448541\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/hh448541(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [3] WELL KNOWN SID TYPE enumeration [Электронный ресурс]. — URL: [https://msdn.microsoft.com/en-us/library/windows/desktop/aa379650\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa379650(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [4] SECURITY_CAPABILITIES structure [Электронный ресурс]. — URL: [https://msdn.microsoft.com/en-us/library/windows/desktop/hh448531\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/hh448531(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [5] UpdateProcThreadAttribute function [Электронный ресурс]. — URL: [https://msdn.microsoft.com/en-us/library/windows/desktop/ms686880\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms686880(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [6] SetEnvironmentVariable function [Электронный ресурс]. — URL: [https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms686206\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms686206(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [7] ExpandEnvironmentStrings function [Электронный ресурс]. — URL: [https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms724265\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms724265(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [8] EXPLICIT_ACCESS structure [Электронный ресурс]. — URL: [https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa446627\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa446627(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [9] TRUSTEE structure [Электронный ресурс]. — URL: [https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa379636\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa379636(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [10] GetNamedSecurityInfo function [Электронный ресурс]. — URL: [https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa446645\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa446645(v=vs.85).aspx) (дата обращения: 2017-06-02).

- [11] SetEntriesInAcl function [Электронный ресурс]. — URL: [https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa379576\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa379576(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [12] SetNamedSecurityInfo function [Электронный ресурс]. — URL: [https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa379579\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa379579(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [13] CreateProcess function [Электронный ресурс]. — URL: [https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms682425\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms682425(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [14] OpenProcessToken function [Электронный ресурс]. — URL: [https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa379295\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa379295(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [15] GetTokenInformation function [Электронный ресурс]. — URL: [https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa446671\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa446671(v=vs.85).aspx) (дата обращения: 2017-06-02).
- [16] Контейнеры Windows [Электронный ресурс]. — URL: <https://docs.microsoft.com/ru-ru/virtualization/windowscontainers/about/> (дата обращения: 2017-06-02).
- [17] The Way FreeBSD Jail Work [Электронный ресурс]. — URL: <https://opensourceforu.com/2015/03/the-way-freebsd-jail-work/> (дата обращения: 2017-06-02).
- [18] Понимая Docker [Электронный ресурс]. — URL: <https://habr.com/post/253877/> (дата обращения: 2017-06-02).

Приложение 1. Main.cpp

Точка входа в программу

```
#include <windows.h>
#include <stdio.h>
#include "ContainerCreate.h"
#include "ContainerTest.h"

void main(int argc, char *argv[])
{
    //ProcessListTest();
    if (!IsInAppContainer())
    {
        RunContainerTests();
        RunExecutableInContainer(argv[0]);
        //RunExecutableInContainer("C:\\Windows\\notepad.exe");
    }
    else {
        RunContainerTests();
    }
    getchar();
}
```

Листинг 27: Main.cpp

Приложение 2. ContainerCreate.h

Заголовочный файл ContainerCreate.h

```
#pragma once
BOOL IsInAppContainer();
BOOL RunExecutableInContainer(CHAR *executable_path);
void ProcessListTest();
```

Листинг 28: ContainerCreate.h

Приложение 3. ContainerCreate.cpp

Файл по запуску контейнера, настройке и запуску приложения

```
#include <windows.h>
#include <strsafe.h>
```

```

#include <Sddl.h>
#include <Userenv.h>
#include <AccCtrl.h>
#include <Aclapi.h>

#pragma comment(lib, "Userenv.lib")

//Лист всех capabilities доступов() применимых к изолированному процессу
extern WELL_KNOWN_SID_TYPE app_capabilities[] =
{
    WinCapabilityInternetClientServerSid
};

WCHAR container_name[] = L"mySandBoxTest";
WCHAR container_desc[] = L"mySandBoxTest description";

BOOL IsInAppContainer();
BOOL SetSecurityCapabilities(PSID container_sid, SECURITY_CAPABILITIES *
    ↪ capabilities, PDWORD num_capabilities);
BOOL GrantNamedObjectAccess(PSID appcontainer_sid, CHAR *object_name,
    ↪ SE_OBJECT_TYPE object_type, DWORD access_mask);

/*Создание
контейнера, предварительная настройка и запуск приложения
*/
BOOL RunExecutableInContainer(CHAR *executable_path)
{
    PSID sid = NULL;
    HRESULT result;
    SECURITY_CAPABILITIES SecurityCapabilities = { 0 };
    DWORD num_capabilities = 0, attribute_size = 0;;
    STARTUPINFOEX startup_info = { 0 };
    PROCESS_INFORMATION process_info = { 0 };
    CHAR desktop_file[MAX_PATH];
    HANDLE file_handle = INVALID_HANDLE_VALUE;
    CHAR *string_sid = NULL;
    BOOL success = FALSE;

    do
    {
        result = CreateAppContainerProfile(container_name, container_name,
        ↪ container_desc, NULL, 0, &sid);
        if (!SUCCEEDED(result))
        {
            if (HRESULT_CODE(result) == ERROR_ALREADY_EXISTS)

```

```

        {
            result = DeriveAppContainerSidFromAppContainerName(
↪ container_name, &sid);
            if (!SUCCEEDED(result))
            {
                printf("Failed to get existing AppContainer name, error
↪ code: %d", HRESULT_CODE(result));
                break;
            }
        }
        else {
            printf("Failed to create AppContainer, last error: %d\n",
↪ HRESULT_CODE(result));
            break;
        }
    }

    printf("[Container Info]\nname: %ws\ndescription: %ws\n",
↪ container_name, container_desc);

    if (ConvertSidToStringSidA(sid, &string_sid))
        printf("Sid: %s\n\n", string_sid);

    if (!SetSecurityCapabilities(sid, &SecurityCapabilities, &
↪ num_capabilities))
    {
        printf("Failed to set security capabilities, last error: %d\n",
↪ GetLastError());
        break;
    }

    ExpandEnvironmentStringsA("%userprofile%\\desktop\\allowed_test.txt",
↪ desktop_file, MAX_PATH - 1);

    file_handle = CreateFileA(desktop_file, GENERIC_ALL, NULL, NULL,
↪ OPEN_ALWAYS, NULL, NULL);
    if (file_handle == INVALID_HANDLE_VALUE)
    {
        printf("Failed to create file %s, last error: %d\n", desktop_file);
        break;
    }

    if (!GrantNamedObjectAccess(sid, desktop_file, SE_FILE_OBJECT,
↪ FILE_ALL_ACCESS))

```



```

    {
        printf("Failed to grant explicit access to %s\n", desktop_file);
        break;
    }

    InitializeProcThreadAttributeList(NULL, 1, NULL, &attribute_size);
    startup_info.lpAttributeList = (LPPROC_THREAD_ATTRIBUTE_LIST)malloc(
↪ attribute_size);

    if (!InitializeProcThreadAttributeList(startup_info.lpAttributeList, 1,
↪ NULL, &attribute_size))
    {
        printf("InitializeProcThreadAttributeList() failed, last error: %d"
↪ , GetLastError());
        break;
    }

    if (!UpdateProcThreadAttribute(startup_info.lpAttributeList, 0,
↪ PROC_THREAD_ATTRIBUTE_SECURITY_CAPABILITIES,
        &SecurityCapabilities, sizeof(SecurityCapabilities), NULL, NULL))
    {
        printf("UpdateProcThreadAttribute() failed, last error: %d",
↪ GetLastError());
        break;
    }

    if (!CreateProcessA(executable_path, NULL, NULL, NULL, FALSE,
↪ EXTENDED_STARTUPINFO_PRESENT, NULL, NULL,
        (LPSTARTUPINFOA)&startup_info, &process_info))
    {
        printf("Failed to create process %s, last error: %d\n",
↪ executable_path, GetLastError());
        break;
    }

    printf("Successfully executed %s in AppContainer\n", executable_path);
    success = TRUE;

} while (FALSE);

if (startup_info.lpAttributeList)
    DeleteProcThreadAttributeList(startup_info.lpAttributeList);

if (SecurityCapabilities.Capabilities)
    free(SecurityCapabilities.Capabilities);

```

```

    if (sid)
        FreeSid(sid);

    if (string_sid)
        LocalFree(string_sid);

    if (file_handle != INVALID_HANDLE_VALUE)
        CloseHandle(file_handle);

    if (file_handle != INVALID_HANDLE_VALUE && !success)
        DeleteFileA(desktop_file);

    return success;
}

/*Проверка
на нахождение в контейнере
*/
BOOL IsInAppContainer()
{
    HANDLE process_token;
    BOOL is_container = 0;
    DWORD return_length;

    OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &process_token);

    if (!GetTokenInformation(process_token, TokenIsAppContainer, &is_container,
↪ sizeof(is_container), &return_length))
        return false;

    return is_container;
}

/*Установка
атрибутов безопасности, доступов
*/
BOOL SetSecurityCapabilities(PSID container_sid, SECURITY_CAPABILITIES *
↪ capabilities, PDWORD num_capabilities)
{
    DWORD sid_size = SECURITY_MAX_SID_SIZE;
    DWORD num_capabilities_ = sizeof(app_capabilities) / sizeof(DWORD);
    SID_AND_ATTRIBUTES *attributes;
    BOOL success = TRUE;

```

```

    attributes = (SID_AND_ATTRIBUTES *)malloc(sizeof(SID_AND_ATTRIBUTES) *
↪ num_capabilities_);

ZeroMemory(capabilities, sizeof(SECURITY_CAPABILITIES));
ZeroMemory(attributes, sizeof(SID_AND_ATTRIBUTES) * num_capabilities_);

    for (unsigned int i = 0; i < num_capabilities_; i++)
    {
        attributes[i].Sid = malloc(SECURITY_MAX_SID_SIZE);
        if (!CreateWellKnownSid(app_capabilities[i], NULL, attributes[i].Sid, &
↪ sid_size))
        {
            success = FALSE;
            break;
        }
        attributes[i].Attributes = SE_GROUP_ENABLED;
    }

    if (success == FALSE)
    {
        for (unsigned int i = 0; i < num_capabilities_; i++)
        {
            if (attributes[i].Sid)
                LocalFree(attributes[i].Sid);
        }

        free(attributes);
        attributes = NULL;
        num_capabilities_ = 0;
    }

    capabilities->Capabilities = attributes;
    capabilities->CapabilityCount = num_capabilities_;
    capabilities->AppContainerSid = container_sid;
    *num_capabilities = num_capabilities_;

    return success;
}

/*Предоставление
доступа к конкретному объекту
*/
BOOL GrantNamedObjectAccess(PSID appcontainer_sid, CHAR *object_name,
↪ SE_OBJECT_TYPE object_type, DWORD access_mask)
{

```

```

EXPLICIT_ACCESS_A explicit_access;
PACL original_acl = NULL, new_acl = NULL;
DWORD status;
BOOL success = FALSE;

do
{
    explicit_access.grfAccessMode = GRANT_ACCESS;
    explicit_access.grfAccessPermissions = access_mask;
    explicit_access.grfInheritance = OBJECT_INHERIT_ANCE |
↪ CONTAINER_INHERIT_ANCE;

    explicit_access.Trustee.MultipleTrusteeOperation = NO_MULTIPLE_TRUSTEE;
    explicit_access.Trustee.pMultipleTrustee = NULL;
    explicit_access.Trustee.ptstrName = (CHAR *)appcontainer_sid;
    explicit_access.Trustee.TrusteeForm = TRUSTEE_IS_SID;
    explicit_access.Trustee.TrusteeType = TRUSTEE_IS_WELL_KNOWN_GROUP;

    status = GetNamedSecurityInfoA(object_name, object_type,
↪ DACL_SECURITY_INFORMATION, NULL, NULL, &original_acl,
    NULL, NULL);
    if (status != ERROR_SUCCESS)
    {
        printf("GetNamedSecurityInfoA() failed for %s, error: %d\n",
↪ object_name, status);
        break;
    }

    status = SetEntriesInAclA(1, &explicit_access, original_acl, &new_acl);
    if (status != ERROR_SUCCESS)
    {
        printf("SetEntriesInAclA() failed, error: %d\n", object_name,
↪ status);
        break;
    }

    status = SetNamedSecurityInfoA(object_name, object_type,
↪ DACL_SECURITY_INFORMATION, NULL, NULL, new_acl, NULL);
    if (status != ERROR_SUCCESS)
    {
        printf("SetNamedSecurityInfoA() failed for %s, error: %d\n",
↪ object_name, status);
        break;
    }
}

```

```

        success = TRUE;

    } while (FALSE);

    if (original_acl)
        LocalFree(original_acl);

    if (new_acl)
        LocalFree(new_acl);

    return success;
}

```

Листинг 29: ContainerCreate.cpp

Приложение 4. ContainerTest.h

Заголовочный файл ContainerTest.h

```

void RunContainerTests();

```

Листинг 30: ContainerTest.h

Приложение 5. ContainerTest.cpp

Файл с тестами для изолированного процесса

```

#include <windows.h>
#include <lphlpapi.h>
#include <stdio.h>
#include <TIHelp32.h>

#pragma comment(lib, "ws2_32.lib")
#pragma comment(lib, "lphlpapi.lib")

void NetworkTest();
void FilesystemTest();
void ProcessListTest();
CHAR *GatewayIp();
void ConnectTest(CHAR *ip, SHORT port);
void CreateFileTest(CHAR *file_path);

```

```

void RunContainerTests ()
{
    printf("We are running in an app container\n\n");

    FilesystemTest ();
    NetworkTest ();
    ProcessListTest ();
}

void NetworkTest ()
{
    printf("[+] Running network test...\n");

    CHAR external_ip [] = "108.177.14.138"; //Google dns
    CHAR *network_ip = GatewayIp(); //Default Gateway

    // Соединение с внешним адреса
    ConnectTest(external_ip , 80);

    // Соединение с внутренним адресов
    ConnectTest(network_ip , 80);;

    printf("[+] Network testing done\n\n");
}

void FilesystemTest ()
{
    printf("[+] Running filesystem test...\n");

    CHAR path[MAX_PATH];
    CHAR test1_path[MAX_PATH];
    CHAR test2_path[MAX_PATH];
    CHAR test3_path[MAX_PATH];
    CHAR test4_path[MAX_PATH];

    //SetEnvironmentVariable(L"testName", L"testValue");
    //SetEnvironmentVariable(L"USERDOMAIN", L"HELLOWORLD");

    ExpandEnvironmentStringsA("%temp%", path, MAX_PATH - 1);
    printf("Path of %temp%: %s\n", path);

    ExpandEnvironmentStringsA("%localappdata%", path, MAX_PATH - 1);
    printf("Path of %localappdata%: %s\n", path);

    ExpandEnvironmentStringsA("%temp%\\allowed_test.txt", test1_path, MAX_PATH

```

```

↪ - 1);
    ExpandEnvironmentStringsA("%userprofile%\\desktop\\allowed_test.txt",
↪ test2_path, MAX_PATH - 1);
    ExpandEnvironmentStringsA("%userprofile%\\desktop\\blocked_test.txt",
↪ test3_path, MAX_PATH - 1);

    //Запись файла allowed_test.txt в директорию %temp% по( умолчанию есть доступ)
    CreateFileTest(test1_path);

    //Запись файла allowed_test.txt на рабочий стол было( явно выдано разрешение)
    CreateFileTest(test2_path);

    //Запись файла blocked_test.txt на рабочий стол должно( быть заблокировано)
    CreateFileTest(test3_path);

    printf("[+] Filesystem testing done\n\n");
}

/*Список
  всех процессов в системе
*/
void ProcessListTest()
{
    printf("[+] Running process list testing...\n");
    tagPROCESSENTRY32W process_entry;
    HANDLE snapshot;

    process_entry.dwSize = sizeof(process_entry);

    snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (snapshot)
    {
        if (Process32First(snapshot, &process_entry))
        {
            do
            {
                printf("Found process: %ws\n", process_entry.szExeFile);
            } while (Process32Next(snapshot, &process_entry));
        }
        CloseHandle(snapshot);
    }
    else {
        printf("Failed to get process list\n");
    }
}

```

```

    printf("[+] Process list testing done\n\n");
}

CHAR *GatewayIp()
{
    static CHAR ip[16];
    PVOID memory_buffer;
    PIP_ADAPTER_INFO adapter_info = NULL;
    DWORD buffer_size = 0;
    CHAR *return_buffer = NULL;

    GetAdaptersInfo(adapter_info, &buffer_size);

    memory_buffer = malloc(buffer_size);
    adapter_info = (PIP_ADAPTER_INFO)memory_buffer;

    if (GetAdaptersInfo(adapter_info, &buffer_size) == ERROR_SUCCESS)
    {
        while (adapter_info)
        {
            if (strcmp(adapter_info->GatewayList.IpAddress.String, "0.0.0.0"))
            {
                memcpy(ip, &adapter_info->GatewayList.IpAddress.String, 16);
                return_buffer = ip;
                break;
            }
            adapter_info = adapter_info->Next;
        }
    }

    free(memory_buffer);
    return return_buffer;
}

void ConnectTest(CHAR *ip, SHORT port)
{
    WSADATA wsadata;
    SOCKET sock;
    sockaddr_in addr;

    WSStartup(MAKEWORD(2, 2), &wsadata);

    sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock != INVALID_SOCKET)
    {

```



```

    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = inet_addr(ip);
    addr.sin_port = htons(port);

    if (connect(sock, (SOCKADDR *)&addr, sizeof(addr)) == SOCKET_ERROR)
    {
        if (WSAGetLastError() == WSAEACCES)
        {
            printf("Connection to %s was blocked\n", ip);
        }
        else {
            printf("Connection to %s was unsuccessful but not blocked\n",
↪ ip);
        }
    }
    else {
        printf("Connection to %s was successful\n", ip);
        closesocket(sock);
    }
}
else {
    printf("Failed to create socket, error code: %d\n", WSAGetLastError());
}

WSACleanup();
return;
}

void CreateFileTest(CHAR *file_path)
{
    HANDLE file_handle;

    file_handle = CreateFileA(file_path, GENERIC_ALL, 0, NULL, OPEN_ALWAYS,
↪ NULL, NULL);
    if (file_handle != INVALID_HANDLE_VALUE)
    {
        printf("Opening of file %s was successful\n", file_path);
        CloseHandle(file_handle);
        DeleteFileA(file_path);
    }
    else {
        if (GetLastError() == ERROR_ACCESS_DENIED)
        {
            printf("Opening of file %s returned access denied\n", file_path);
        }
    }
}

```

```

        else {
            printf("Opening of file %s failed but was not blocked\n", file_path
↪ );
        }
    }
}

```

Листинг 31: ContainerTest.cpp

Приложение 6. Структура проекта

Проект состоит из:

- 2 заголовочных файлов:
 - ContainerCreate.h;
 - ContainerTest.h.
- 3 файлов с исходным кодом:
 - Main.cpp;
 - ContainerCreate.cpp;
 - ContainerTest.cpp.