Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

—

Институт компьютерных наук и кибербезопасности

**Высшая школа кибербезопасности**

**К У Р С О В А Я Р А Б О Т А**

Изоляция ресурсов с помощью пространств имен: создание мини-контейнера

по дисциплине «Операционные системы»

Выполнил

студент гр. 5131001/30002 Мишенёв Н. С.

Руководитель

программист Огнёв Р. А.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г.

Санкт-Петербург

2025г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#__RefHeading___1)

[ОСНОВНАЯ ЧАСТЬ 4](#__RefHeading___33)

[Теоретическая ЧАСТЬ 4](#__RefHeading___32)

[1. Пространство имён или namespaces. 4](#__RefHeading___31)

[2. Типы пространств имен в Linux. 5](#__RefHeading___30)

[3. Создание пространств имён. 11](#__RefHeading___29)

[3.1. Unshare 11](#__RefHeading___34)

[4. Примеры использования пространств имён. 13](#__RefHeading___28)

[ПРАКТИЧЕСКАЯ ЧАСТЬ 15](#__RefHeading___27)

[1. Разработка программы контейнеризатора на С. 15](#__RefHeading___26)

[ЗАКЛЮЧЕНИЕ 15](#__RefHeading___25)

[ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ. 16](#__RefHeading___10)

# ВВЕДЕНИЕ

**Цель**

Изучение механизма пространств имен (namespaces) в Linux, лежащего в основе контейнеризации.

**Задачи**

* Изучить типы пространств имен в Linux (PID, Network, Mount, UTS, IPC, User).
* Рассмотреть создание namespaces через *clone* (с флагами CLONE\_NEW\*) и *unshare*.
* Привести примеры использования пространства имён.
* Написать программу на С, которая запускает процесс в отдельных *namespaces.*
* Сделать выводы о проделанной работе

# ОСНОВНАЯ ЧАСТЬ

## Теоретическая ЧАСТЬ

## Пространство имён или namespaces.

Пространство имён (от англ. namespaces) — особенность ядра Linux, позволяющая изолировать и виртуализировать глобальные системные ресурсы множества процессов.

Другими словами, пространства имен определяют набор ресурсов, которые может использовать процесс, поэтому намеренно скрыв от процесса какие-либо ресурсы, поместив их в другой namespace, мы можем запретить процессу ими пользоваться.

На высоком уровне они позволяют тонко разделять глобальные ресурсы операционной системы, такие как точки монтирования, сетевой стек и утилиты межпроцессного взаимодействия. В Linux они обычно представлены как файлы в директории **/proc/<pid>/ns.** (Рисунок 1)

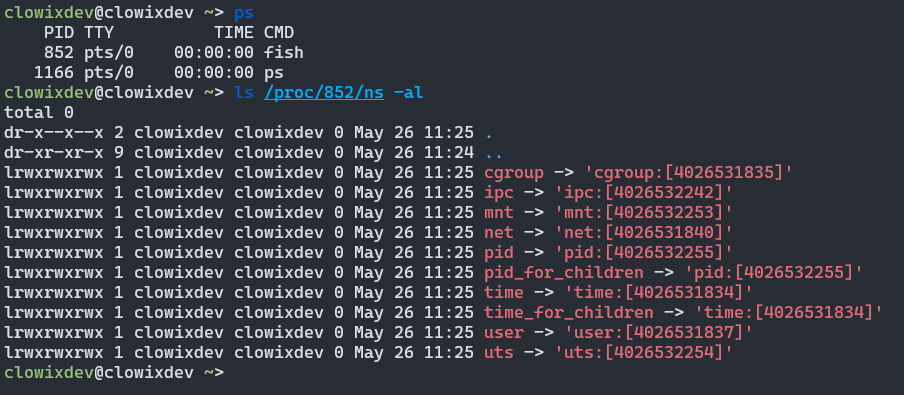


Рисунок 1 – пространство имён для запущенного терминала

Сильная сторона пространств имен в том, что они ограничивают доступ к системным ресурсам без информирования об этом выполняющегося процесса.

## Типы пространств имен в Linux.

В Linux существует несколько типов пространств имён. При запуске системы, инициализируется по 1 экземпляру каждого типа пространства имён, кроме пространства имён файловой системы. После инициализации, можно объединять или создавать дополнительные пространства имён.

Все пространства имён поддерживают вложенность, то есть между ними можно установить связь «родитель — потомок». Таким образом некоторые пространства наследуют все свойства от своего родительского пространства имён. Однако это верно не для всех пространств.

Функциональные возможности пространства имён одинаковы для всех типов: каждый процесс связан с пространством имён и может видеть или использовать только ресурсы, связанные с этим пространством имён, и, где это применимо, — с его потомками.

Рассмотрим типы пространств имён в Linux-системе:

* 1. **PID**

Исторически ядро Linux поддерживает одно дерево процессов. Древовидная структура данных содержит ссылку на каждый активный процесс в виде иерархии от родителя к потомку. Она также нумерует все выполняющиеся в ОС процессы (Рисунок 2). Эта структура поддерживается в файловой системе **procfs**, которая является свойством исключительно работающей ОС.

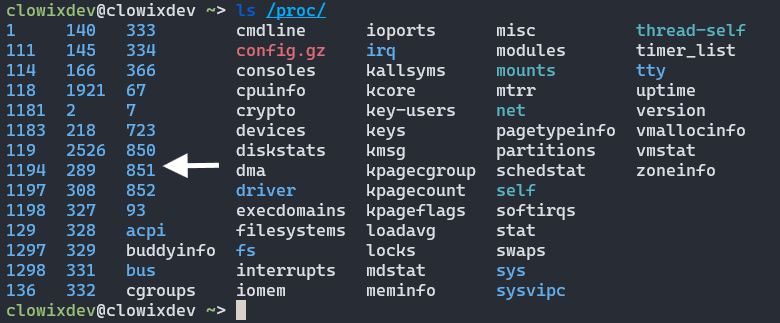


Рисунок 2 – файловая система **procfs**

Эта структура позволяет процессам с достаточными привилегиями прикрепляться к другим процессам, инспектировать эти процессы, обмениваться с ними информацией и/или завершать их.

Она также содержит информацию о корневом каталоге процесса, его текущем рабочем каталоге, дескрипторах открытых файлов, адресах виртуальной памяти, доступных точках монтирования и т.д. Пример структуры **procfs** и дерева процессов представлен на рисунке 3 и 4 соответственно:

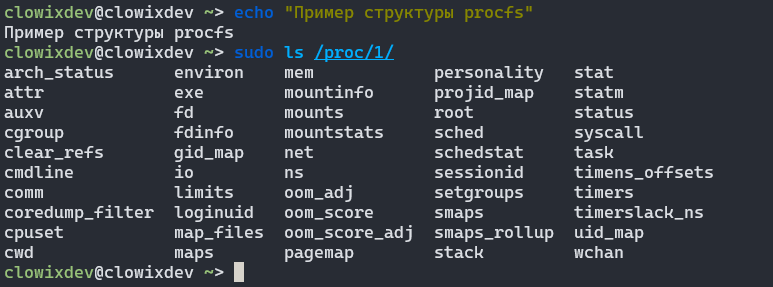


Рисунок 3 – пример структуры **procfs** процесса с **PID 1**

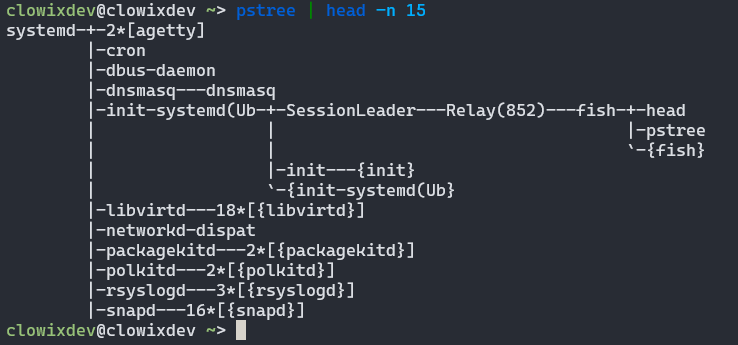


Рисунок 4 – пример структуры дерева процессов.

* 1. **Network (NET)**

Сетевое пространство имен ограничивает видимость процесса внутри сети. Оно позволяет процессу располагать собственной частью сетевого стека хоста (набором сетевых интерфейсов и правилами маршрутизации) (Рисунок 5)

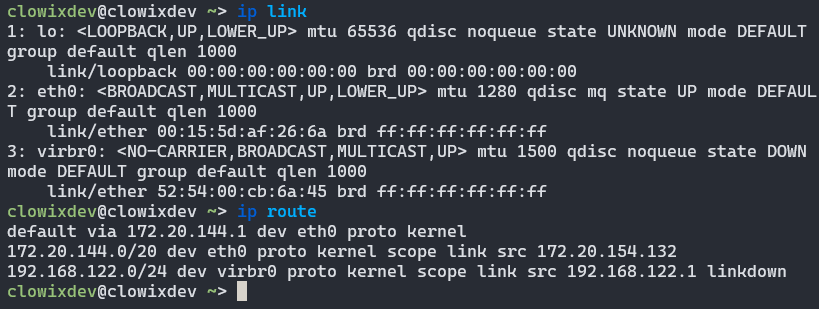


Рисунок 5 – пример сетевых интерфейсов и правил маршрутизации

* 1. **Mount (MNT)**

Пространства имен mount (MNT) позволяют создавать деревья файловых систем под отдельные процессы, тем самым создавая представления корневой файловой системы. Linux поддерживает структуру данных для всех различных файловых систем, смонтированных в системе.

Эта структура является индивидуальной для каждого процесса, а также пространства имен. В нее входит информация о том, какие разделы дисков смонтированы, где они смонтированы и тип монтирования (RO/RW). (Рисунок 6)

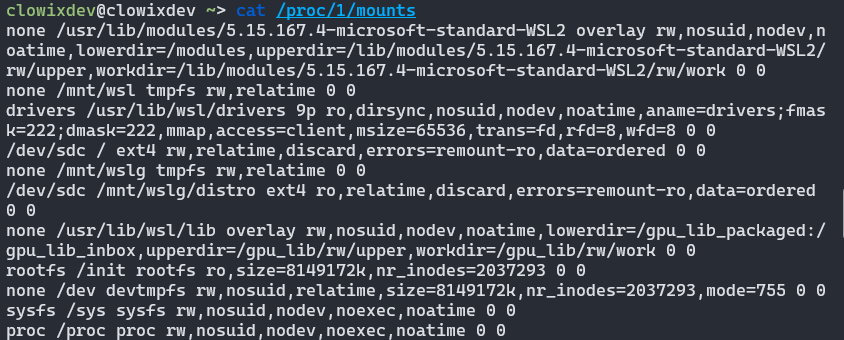


Рисунок 6 – **MNT** структурадля процесса с **PID 1**

Пространства имен в Linux дают возможность копировать эту структуру данных и передавать копию разным процессам. Таким образом, эти процессы могут изменять данную структуру (монтировать и размонтировать), не влияя на точки монтирования друг друга.

Предоставляя разные копии структуры файловой системы, ядро изолирует список точек монтирования, видимых процессу в пространстве имен. (Рисунок 7)

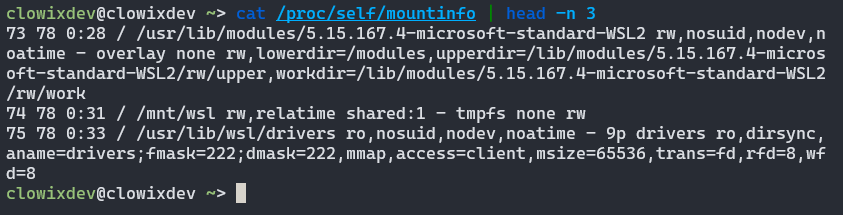


Рисунок 7 – список точек монтирования в namespace текущего процесса

* 1. **UTS**

Пространство имен UTS изолирует имя хоста системы для определенного процесса.

Большая часть взаимодействия с хостом выполняется через IP-адрес и номер порта. Однако для человеческого восприятия все сильно упрощается, когда у процесса имя. К примеру, выполнять поиск по файлам журналов гораздо проще, когда определено имя хоста. Также это связано с тем, что в динамической среде IP могут изменяться. (Рисунок 8, Рисунок 9)

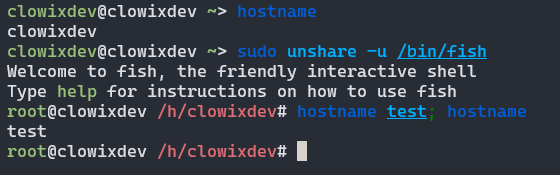


Рисунок 8 – изменение **hostname** в другом **UTS** **namespace**

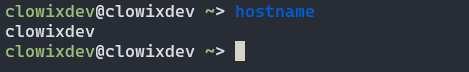


Рисунок 9 – проверка **hostname** в другом терминале

* 1. **IPC**

Пространство имен IPC предоставляет изоляцию для механизмов взаимодействия процессов, таких как семафоры, очереди сообщений, разделяемая память и т.д.

Обычно, когда процесс ответвляется, он наследует все IPC, открытые его родителем. Процессы внутри IPC namespace не могут видеть или взаимодействовать с ресурсами IPC вышестоящего пространства имен. Пример структуры **ipcs** приведён на рисунке 10:

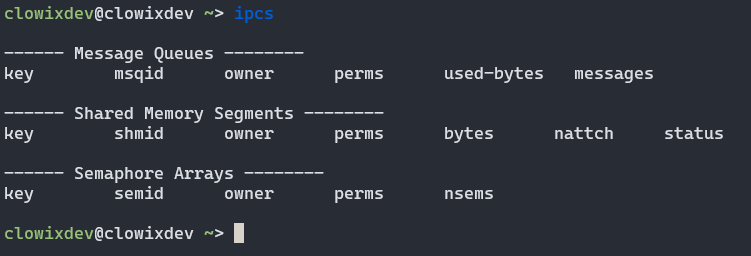


Рисунок 10 – пример IPC namespace для текущего процесса

* 1. **USER**

Все процессы в Linux имеют родительский процесс. Также существуют привилегированные и непривилегированные процессы, что определяется их пользовательским ID или же UID. В зависимости от этого UID процессы получают разные привилегии в ОС. Пользовательское пространство имен – это функционал ядра, позволяющий выполнять виртуализацию этого атрибута для каждого процесса.

Пользовательские пространства имен изолируют связанные с безопасностью идентификаторы и атрибуты. В частности, ID пользователей, групповые ID, корневой каталог, ключи и возможности.

Для примера, запустим процесс в новом **user namespace** и посмотрим на его родителя (Рисунок 11)

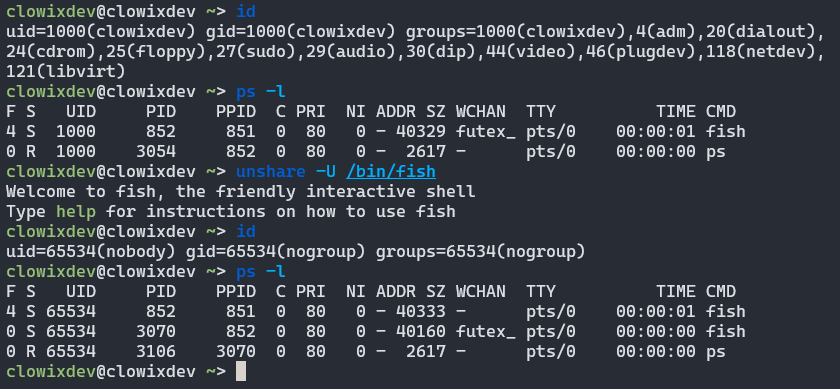


Рисунок 11 – пример создания нового пользовательского пространства имён

Как можно заметить, PID родительского процесса не изменился, но его обладатель теперь кто-то, о ком нет информации, потому что у запущенного процесса в новом **namespace**, нет доступа за его пределы.

## Создание пространств имён.

Создание пространств имён происходит с помощью системного вызова **clone()** и с помощью команды **unshare.** Рассмотрим саму суть способов создания пространств и их отличия:

## Unshare

Команда unshare запускает выбранный процесс в новом namespace, который пользователь задаёт флагом: Рисунок (11)

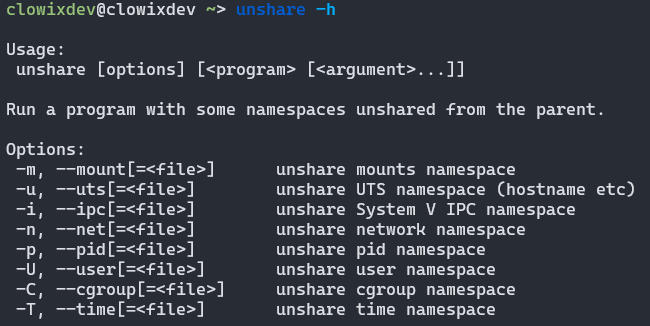


Рисунок 11 – синтаксис и доступные флаги команды unshare

Таким образом, не прибегая к самописным решениям и установке каких либо дополнительных утилит, можно запустить нужный пользователю процесс в отдельном от родителя namespace.

* 1. **clone()**

Системный вызов **clone()** можно использовать с некоторыми флагами, при указании которых в аргументах функции, создадутся определённые пространства имён:

* **CLONE\_NEWNS** – новое пространство имен MNT
* **CLONE\_NEWUTS** – новое пространство имен UTS
* **CLONE\_NEWIPC** – новое пространство имен IPC
* **CLONE\_NEWPID** – новое пространство имен PID
* **CLONE\_NEWNET** – новое пространство имен NET
* **CLONE\_NEWUSER** – новое пространство имен USR

Разница между **clone** и **unshare** в том, что первый порождает новый процесс внутри нового набора пространств имен, а последний перемещает в новый набор пространств имен текущий процесс.

## Примеры использования пространств имён.

Пространство имён даёт процессам, запущенным в контейнерах, иллюзию, что они имеют свои собственные ресурсы. Основная цель изоляции процессов состоит в предотвращении вмешательства процессов одного контейнера в работу других контейнеров, а также работу хостовой машины.

На данный момент существует несколько технологий контейнеризации, основными же являются **LXC** и **Docker**. И в основе этих технологиях лежат пространства имён и они имеют одинаковый принцип работы.

Технология контейнеризации обеспечивает изоляцию процессов путём переноса процессов, запущенных в контейнере, в собственное пространство имён и ограничения прав и видимости процессов, запущенных в других контейнерах и на самой хостовой машине. Этот механизм работает при поддержке пространства имён PID, которое изолирует пространство идентификаторов процесса контейнера от пространства хоста. Поскольку пространство имён PID иерархичное, то процесс может видеть только процессы в своём собственном пространстве имён, либо в его «дочернем» пространстве имён. Вследствие этого, после создания и присвоения пространства имён контейнеру, хостовая машина может видеть и взаимодействовать с пространством идентификаторов процесса контейнера, но процесс внутри контейнера не может видеть и что-либо делать с пространством хоста или другого контейнера.

Для достижения защищённости контейнеров и хостовой машины от нелегитимных процессов и модификаций с файловой системой необходима изоляция файловых систем.

Технология контейнеризации использует пространство имён файловой системы для изоляции файловой системы, иерархично связанной с другими контейнерами. Это пространство предоставляет процессам каждого контейнера различное представление дерева файловой системы и ограничивает все события монтирования, происходящие внутри контейнера.

Процессы, запущенные в контейнерах, должны быть ограничены так, чтобы они могли взаимодействовать только через определённый набор ресурсов IPC и им было запрещено вмешиваться в работу других контейнеров и хост-машины.

Технология контейнеризации достигает изоляции IPC использованием пространства имён IPС. Процесс в пространстве имён IPC не может писать или читать IPC-ресурсы, принадлежащие другому пространству имён. Технология контейнеризации присваивает свое пространство имён IPC каждому контейнеру, таким образом, не давая процессам в контейнере вмешиваться в другие контейнеры.

Изоляция сети важна для предотвращения сетевых атак, таких как «человек посередине». Контейнеры должны быть настроены таким образом, чтобы они не могли подслушивать или управлять сетевым трафиком других контейнеров или хоста.

Для каждого контейнера технология контейнеризации создаёт независимый сетевой стек при помощи пространства имён сети. Таким образом, каждый контейнер имеет свои собственные IP-адреса, таблицы IP-маршрутизации, сетевые устройства и так далее. Это позволяет контейнерам взаимодействовать друг с другом через свои соответствующие сетевые интерфейсы, что аналогично тому, как они взаимодействуют с внешними хостами.

Только с появлением пространства имён пользователей стала возможна полная изоляция контейнеров от хостовой машины. Поскольку до появления этого пространства, пользователь имел ID 0 в контейнере, то есть был пользователем root в этом контейнере, и при этом же имел такой же ID внутри хостовой машины, что создавал большую угрозу для всей системы. И только благодаря пространству имён пользователей удалось разделить эти ID, то есть пользователь остается привилегированным внутри контейнера и при этом непривилегированным внутри хостовой машины.

Благодаря пространству имён UTS каждый контейнер может иметь свое собственное доменное имя NIS domainname и имя контейнера nodename . Это является важным фактором для многих приложений, веб-сервисов, журналирования и других возможностей.

## ПРАКТИЧЕСКАЯ ЧАСТЬ

## Разработка программы контейнеризатора на С.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы ...

# ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ.

|  |
| --- |
|  |