Московский физико-технический институт
(государственный университет)

Применение механизмов контейнерной виртуализации в бинарном компиллляторе приложений x86-Эльбрус

Выпускная калификационная работа
(бакалаврская работа)

Студент: Толмачев Д.В.

Научный руководитель: к.ф-м.н. Рожин А.Ф.

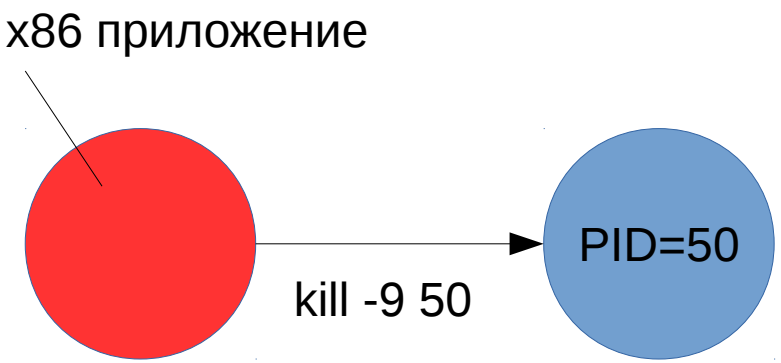
Москва, 2017

Проблема трансляции системных вызовов

В архитектуре Эльбрус бинарный компилятор рассматривается как обычное *Linux*-приложение и работает под управлением ОС Эльбрус.

Одной из задач бинарного компилятора является трансляция системных вызовов гостевого приложения в системные вызовы ОС Эльбрус.

В ряде задач бинарный компилятор обладает правами суперпользователя, поэтому он может влиять на работу процессов пользователей нативной системы.



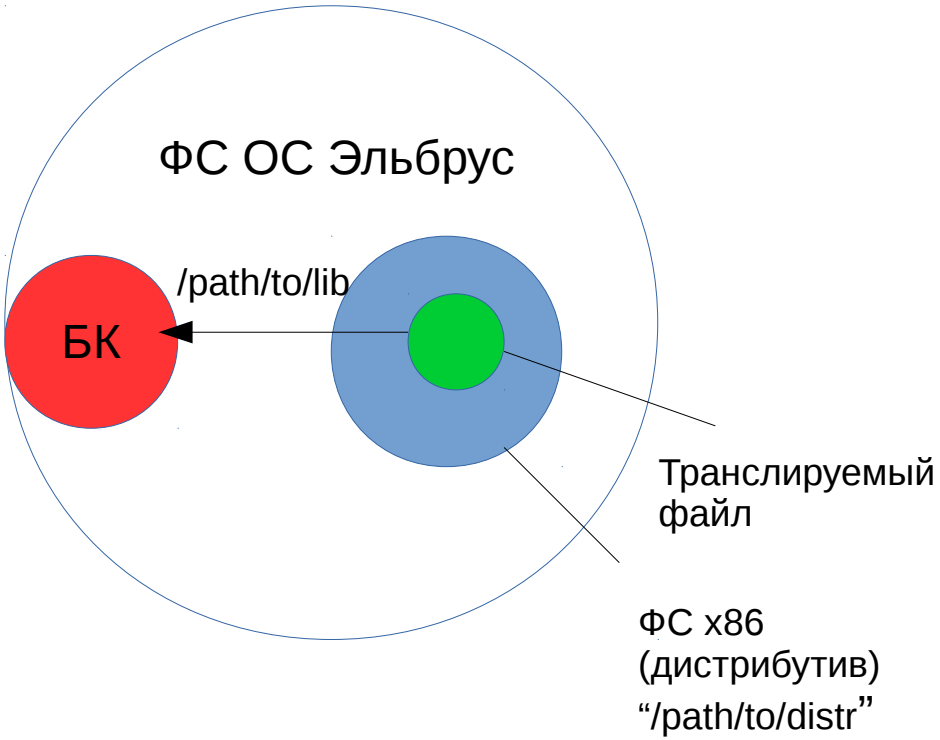
Данная команда выполнится успешно. Таким образом x86 приложение запущенное с помощью бинарного компилятора завершит процесс нативной системы ОС Эльбрус, что не должно случаться

При ошибочной работе x86 приложения работоспособность нативной системы может быть нарушена.

Проблема использования различных гостевых дистрибутивов

Для запуска x86 приложений через бинарный компиллятор необходим набор x86 библиотек, стандартных конфигурационных файлов, сокетов, которые расположены внутри x86 файловой системы.

Однако сам компилятор расположен в нативной системе вне x86 дистрибутива, поэтому должен знать пути до данного окружения. Для корректной работы трансляции системных вызовов необходимо дополнять пути до файлов путем до гостевого дистрибутива.



Кроме того различные гостевые дистрибутивы имеют разные наборы специальных файлов, для которых пути не должны меняться. Следовательно, при изменении гостевого дистрибутива необходимо изменять исходный код бинарного компилятора

Цель

Создание изолированного управляемого через терминал окружения для запуска x86 приложения внутри ОС Эльбрус с помощью бинарного компиллятора приложений с применением механизмов контейнерной виртуализации

Задачи

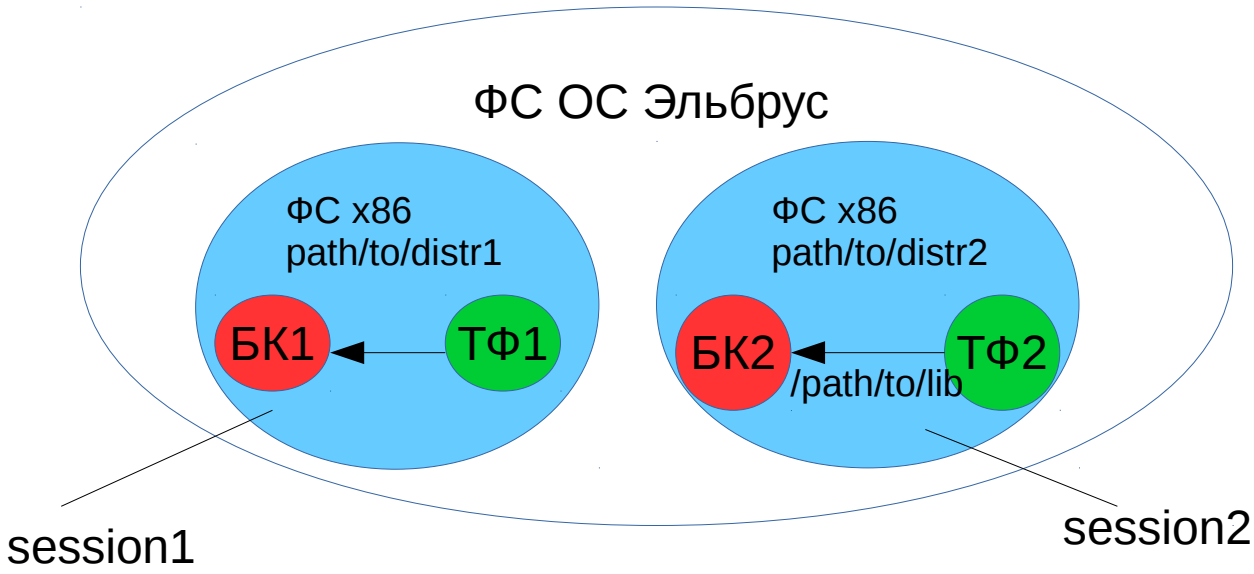
- Определить оптимальную конфигурацию ядра для работы с механизмами контейнерной виртуализации (namespace, cgroups)
- Определить необходимые сервисы, запускаемые внутри контейнера менеджером систем и служб systemd
- Создание утилиты для запуска x86 дистрибутива(контейнера), управляемого через терминал, внутри ОС Эльбрус
- Проверить работоспособность утилиты на гостевом дистрибутиве Ubuntu 16.10

Предлагаемое решение

Внутри ОС Эльбрус создается управляемое через терминал изолированное x86 окружение с запущенными через init сервисами, внутри которого запускается x86 приложение.

Данное решение удовлетворяет требованиям:

- БК не может влиять на работу других пользователей в нативной системе
- Не требуется никаких преобразований путей при обработке системных вызовов
- Не требуется производить изменения в исходном коде бинарного компилятора при изменении гостевого дистрибутива



При трансляции системного вызова в данном случае не требуется преобразование пути /path/to/lib, так как БК запускается внутри x86 файловой системы.

Конфигурация ядра

Стандартное ядро ОС Эльбрус, поставляемое пользователю не поддерживает необходимые для контейнерной виртуализации механизмы namespace и cgroups.
В ходе исследования была определена конфигурация ядра для решения поставленной задачи.

Для изоляции гостевых приложений от нативной системы:

- CONFIG_NAMESPACES
- CONFIG_UTS_NS
- CONFIG_IPC_NS
- CONFIG_PID_NS
- CONFIG_USER_NS
- CONFIG_NET_NS
- DEVPTS_MULTIPLE_INSTANCES

Для ограничения использования ресурсов гостевой системой

- CONFIG_CGROUPS
- CONFIG_CGROUP_NS
- CONFIG_CGROUP_DEVICE
- CONFIG_CGROUP_SCHED
- CONFIG_CGROUP_CPUACCT
- CONFIG_MEMCG
- CONFIG_CPUSETS

Необходимые сервисы systemd

Systemd – менеджер систем и служб, инициализирующий сервисы в Linux и управляющий ими.

В результате изучения systemd были определены необходимые сервисы, запускаемые внутри контейнера, основные из них:

- Dbus – сервис для межпроцессного взаимодействия внутри контейнера
- Cron – сервис для периодического выполнения заданий в определённое время
- Systemd-journal – сервис для записи отладочной информации о работе systemd

Механизм контейнерной виртуализации namespace

Основным инструментом для реализации контейнера является подсистема ядра: namespace.

Namespace (пространство имен) – это механизм ядра Linux обеспечивающий изоляцию процессов, точек монтирования, IPC ресурсов и т.д. друг от друга.

Механизм namespaces позволяет создавать отдельное ответвление дерева процессов с собственным PID 1, а также отдельное дерево файловой системы.

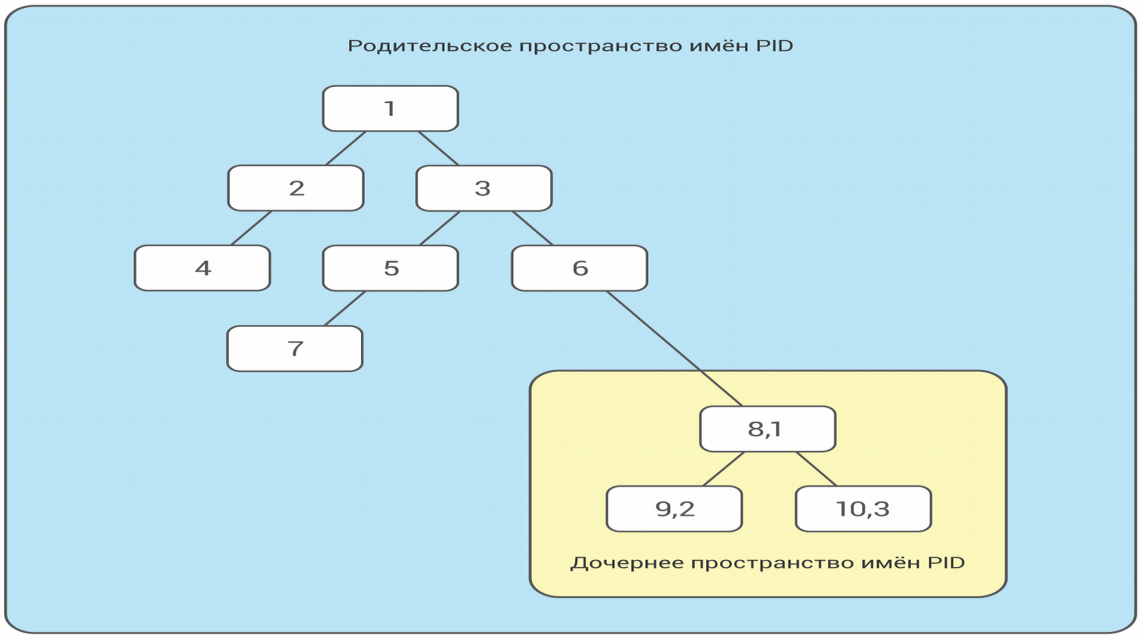
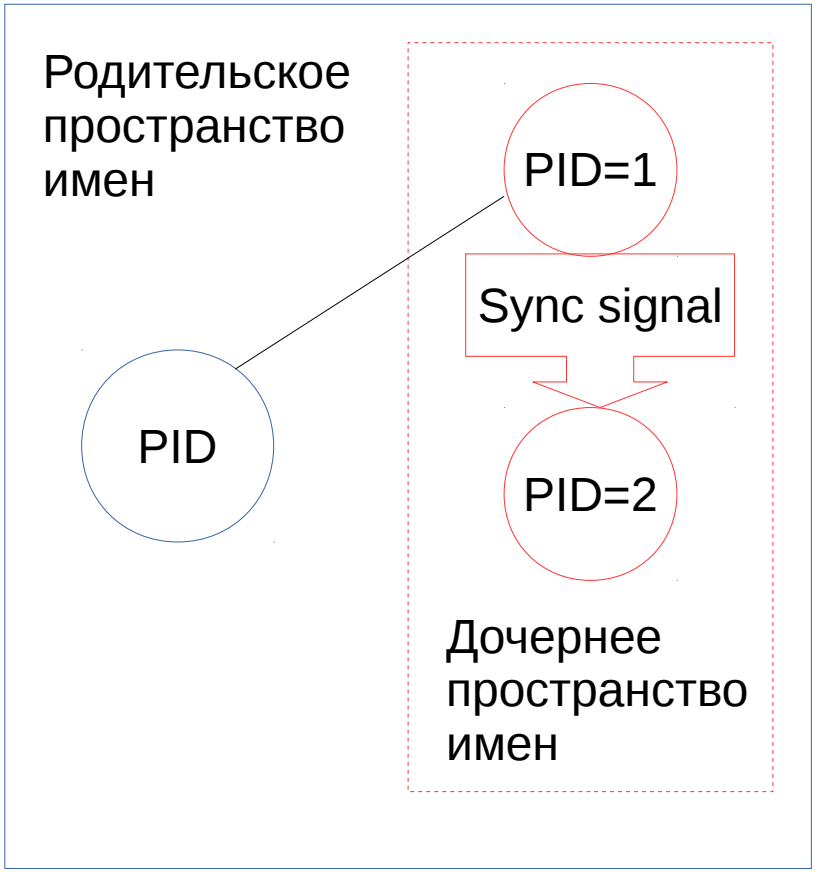


Иллюстрация работы PID namespace (изоляция процессов)

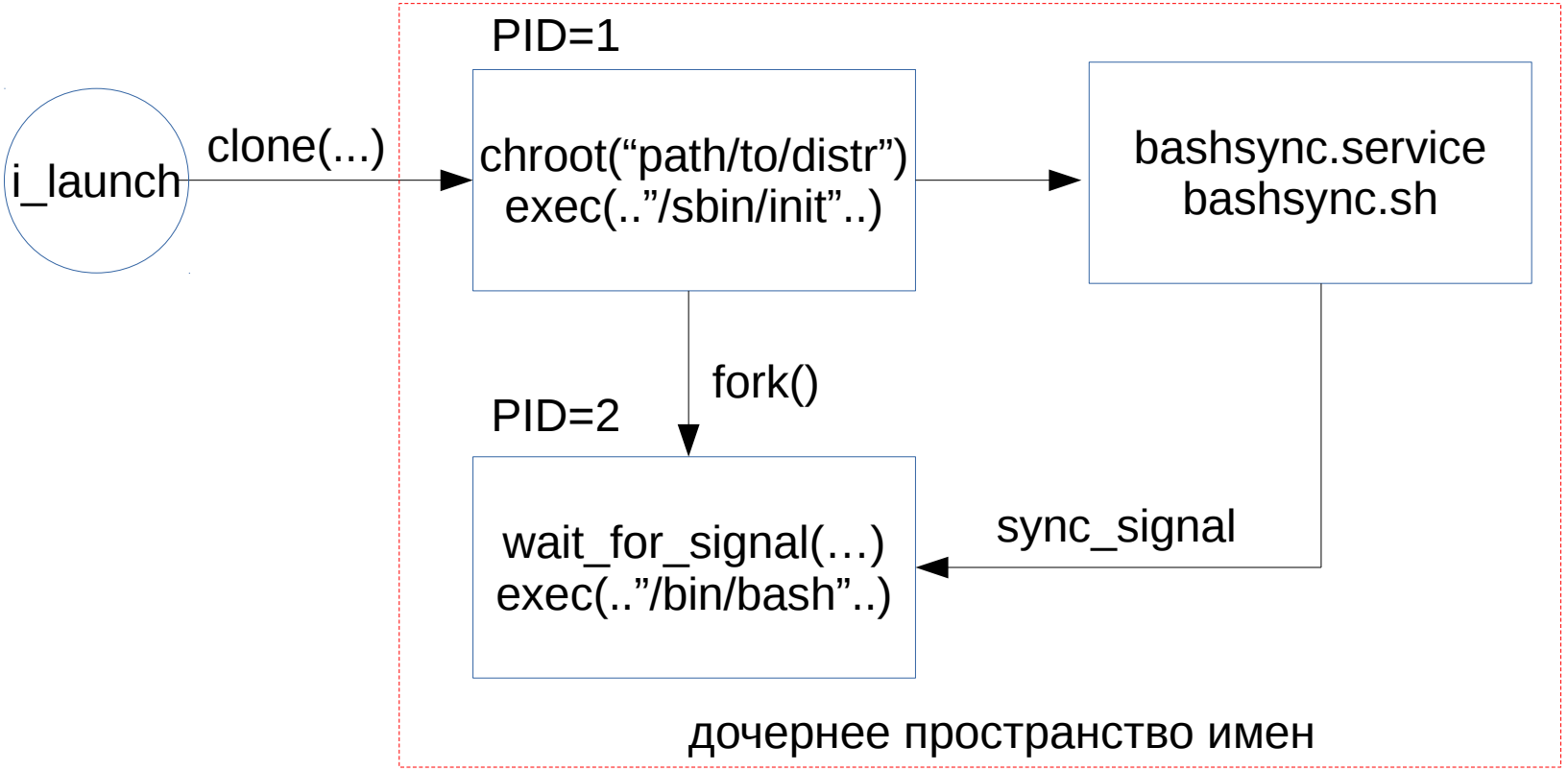
Утилита запуска контейнера



- PID – процесс, создающий новый namespace
- Процесс с PID=1 превращается в бинарный компиллятор, запускающий systemd внутри контейнера
- Процесс с PID=2 ожидает синхронизирующий сигнал от systemd и превращается в управляющий терминал

Пользователь не должен получить доступ к управляющему терминалу раньше чем будут загружены все сервисы systemd. Для синхронизации используется отдельный systemd сервис, запускающийся последним, который посылает синхронизирующий сигнал процессу с PID=2.

Алгоритм работы утилиты



i_launch - утилита запуска контейнера. Написана на языке C.

bashsync.service - сервис для синхронизации с управляющим терминалом. Написан на специальном языке для описания systemd сервисов.

bashsync.sh - вспомогательный shell-скрипт для синхронизации.

Проверка работоспособности утилиты на гостевом дистрибутиве Ubuntu 16.10

systemd,1

```
|               `-systemd,6872
|                 |-agetty,7324 --noclear tty1 linux
|                 |-bash,6873
|                 |   `-top,7541
|                 |-cron,7133 -f
|                 |-dbus-daemon,7148,messagebus --system --address=systemd: --nofork --nopidfile ...
|                 |-rsyslogd,7137,syslog -n
|                 |   |-{in:imklog},7259
|                 |   |-{in:imuxsock},7258
|                 |   `-{rs:main Q:Reg},7260
|                 |-sshd,7325
|                 |-systemd-journal,6920
|                 `-systemd-logind,7138
```

- В дереве процессов нативной системы видно, что init процесс запущен дважды: для ОС Эльбрус, для гостевого дистрибутива Ubuntu 16.10

```
top - 08:03:09 up  1:19,  0 users,  load average: 4.04, 3.89, 3.62
Tasks:  10 total,   1 running,   9 sleeping,   0 stopped,   0 zombie
%Cpu(s): 85.2 us,  3.9 sy,  0.0 ni, 11.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3508596 total,   702584 free,  1416168 used,  1389844 buff/cache
KiB Swap:  2928636 total,  2928636 free,        0 used.  1783668 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0  136636   6496   5092 S   0.0  0.2   0:00.37 systemd
    2 root      20   0   21240   3708   3164 S   0.0  0.1   0:00.01 bash
   25 root      20   0   44000   5156   4688 S   0.0  0.1   0:00.16 systemd-journal
  105 root      20   0   28160   2336   2100 S   0.0  0.1   0:00.00 cron
  106 syslog    20   0  260640   3224   2792 S   0.0  0.1   0:00.02 rsyslogd
  107 root      20   0   35888   1988   1620 S   0.0  0.1   0:00.01 systemd-logind
  112 message+  20   0   43140   3560   3144 S   0.0  0.1   0:00.06 dbus-daemon
  174 root      20   0   16064   1760   1624 S   0.0  0.1   0:00.00 agetty
  175 root      20   0   67824   3104   2368 S   0.0  0.1   0:00.00 sshd
  230 root      20   0   39792   3120   2648 R   0.0  0.1   0:00.82 top
```

- Запустив утилиту top внутри контейнера видим, что запущены необходимые сервисы systemd и управляющий терминал. Изнутри контейнера мы не можем наблюдать процессы нативной системы и взаимодействовать с ними

Результаты

С целью создания изолированного управляемого через терминал окружения для запуска x86 приложения внутри ОС Эльбрус с помощью бинарного компиллятора приложений были выполнены следующие задачи:

- Определена оптимальная конфигурация ядра
- Определены необходимые сервисы, запускаемые внутри контейнера менеджером систем и служб systemd
- Создана утилита для запуска x86 дистрибутива, управляемого через терминал, внутри ОС Эльбрус с помощью бинарного компиллятора приложений
- Проверена работоспособность утилиты на гостевом дистрибутиве Ubuntu 16.10