



АЛГОРИТМ ГЕНЕРАЦИИ КОМАНД ВОССТАНОВЛЕНИЯ ДЕРЕВА ПРОЦЕССОВ ОС LINUX НА ОСНОВЕ МОДЕЛИ ЖИЗНЕННОГО ЦИКЛА РЕСУРСОВ LINUX

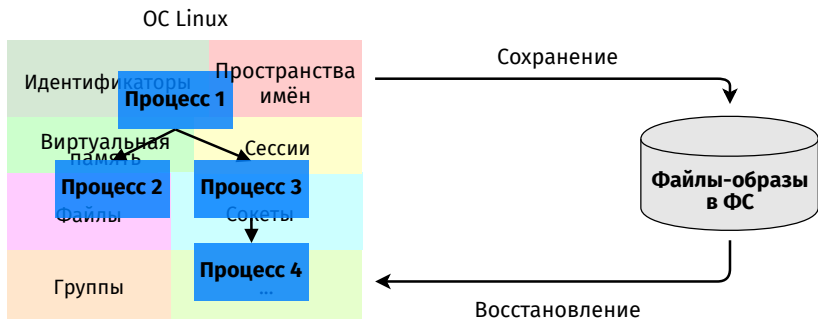
Горбунов Егор Алексеевич

научный руководитель: магистр прикладной математики и физики Е. А. Баталов

СПбАУ РАН

13 июня 2017 г.

Задача сохранения и восстановления дерева процессов



Использование

Живая миграция, ускорение запуска программ, отложенная отладка, обновление ядра без остановки программ

- **CRIU**

- + полностью в userspace
- + активно поддерживается на текущий момент времени

- BLCR¹ (2003)

- требует загрузки модуля в ядро

- DMTCP² (2004)

- к целевому процессу с момента запуска должна быть подключена библиотека
- перехватывает часть вызовов к `glibc`

- OpenVZ(2005)

- работает внутри собственного ядра Linux

¹Berkeley Lab Checkpoint/Restart

²Distributed MultiThreaded CheckPointing

Проблемы `crui` в подходе к восстановлению

Последовательность действий восстановления чётко зафиксирована в коде (и она очень большая), что приводит к проблемам:

- Код для восстановления каждого типа и зависимости ресурсов нужно аккуратно и согласованно добавить в эту последовательность \Rightarrow некоторые конфигурации ресурсов не поддерживаются из-за сложности
- Фиксированный порядок восстановления \Rightarrow потенциальное сужение множества допустимых деревьев для восстановления
- Отсутствие чёткого понимания того, какие конфигурации ресурсов дерева процессов `crui` гарантированно поддерживает

Подход с генератором и интерпретатором



- Для каждого конкретного дерева процессов получаем индивидуальную программу из команд
- Какие должны быть команды?
- Какие должны быть промежуточные представления?

Цель

Отойти от фиксированного порядка восстановления и найти обобщённый подход к восстановлению ресурсов

Задачи

- Разработать генератор команд для задачи восстановления дерева процессов в рамках подхода генератор-интерпретатор
- Разработать промежуточные представления

Требования

- Генерируемые команды должны быть исполнимы из пространства пользователя
- Возможность эффективной реализации предлагаемых алгоритмов

Модель дерева процессов

Ресурс — r — сущность в ядре ОС (file struct, group, session, ...)

Handle — h — объект, через который процесс получает доступ к ресурсу (file descriptor, gid, sid, ...)

Процесс

$$P = \{(r_1, h_1), (r_2, h_2), \dots, (r_n, h_n)\}$$

$pid(P)$ — идентификатор процесса

$parent(P)$ — процесс-родитель, $\neq P$

Дерево процессов

$$T = \{P_1, P_2, \dots, P_k\}$$

$$root = P_1$$

$$\forall P \in T \wedge P \neq root \ (parent(P) \in T)$$

Свойства ресурсов

isSharable(r) — ресурс, который можно разделить "при жизни"(file, group, namespace, ...)

isInherited(r) — ресурс, наследуемый ребёнком "при рождении"(file, private memory area, ...)

resourceDependencies(r) — множество ресурсов, от которых зависит создание r

canExistTogether(r_1, h_1, r_2, h_2) — могут ли 2 ресурса существовать вместе?

possibleCreators(r) — множество процессов, которые бы могли создать ресурс r

Действия, которые процессы совершают при жизни:

$$\mathcal{A} = \begin{cases} \text{ForkAction}(P_1, P_2) \\ \text{CreateAction}(P, r, h) \\ \text{ShareAction}(P_{\text{from}}, P_{\text{to}}, r, h_{\text{from}}, h_{\text{to}}) \\ \text{RemoveAction}(P, r, h) \end{cases}$$

Задача восстановления

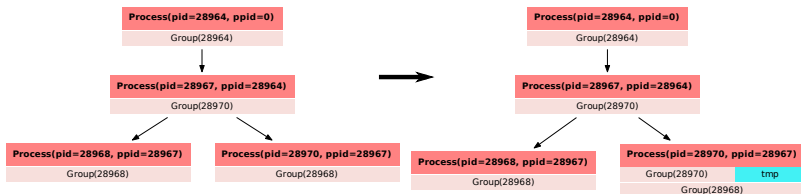
Имея исходное дерево процессов T , найти последовательность действий $[a_i]$, ($a_i \in \mathcal{A}$) такую, что:

$$\{P_0\} \xrightarrow{\mathcal{A}} \{P_0\} \cup T$$

Общий план алгоритма

1. Считать файлы-образы процесса, полученные после `criu dump`
2. Инициализировать дерево процессов в рамках модели
3. Дополнить дерево процессов ресурсами ("Замыкание")
4. Построить множество действий ($\subset \mathcal{A}$), которые нужно выполнить для восстановления
5. Упорядочить множество действий \Rightarrow список действий

"Замыкание" исходного дерева процессов



- Создание каждого ресурса *требует определённого состояния* дерева
- "Замыкание" исходного снимка ресурсов — это объединение таких состояний
 - Полученное дерево может быть некорректным с точки зрения ОС Linux \Rightarrow алгоритм должен об этом позаботиться
- Добавленные к исходному снимку процесса $P \in T$ ресурсы: $Tmp(P)$ — временные ресурсы

Построение множества действий

Process(pid=0, ppid=-1)
Fork
Process(pid=28964, ppid=0)

Process(pid=28964, ppid=0)
Creates
Session(28964)

Process(pid=28964, ppid=0)
Fork
Process(pid=28967, ppid=28964)

Process(pid=28964, ppid=0)
Creates
Group(28964)

Process(pid=28967, ppid=28964)
Fork
Process(pid=28970, ppid=28967)

Process(pid=28970, ppid=28967)
Creates
Group(28970)

Process(pid=28967, ppid=28964)
Fork
Process(pid=28968, ppid=28967)

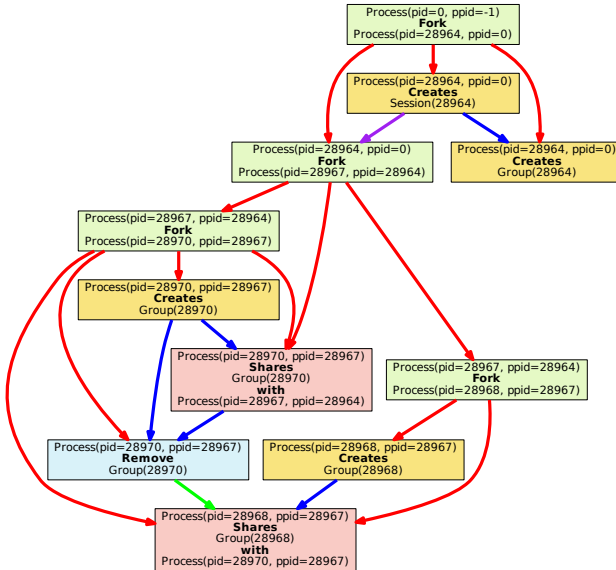
Process(pid=28970, ppid=28967)
Shares
Group(28970)
with
Process(pid=28967, ppid=28964)

Process(pid=28968, ppid=28967)
Creates
Group(28968)

Process(pid=28970, ppid=28967)
Remove
Group(28970)

Process(pid=28968, ppid=28967)
Shares
Group(28968)
with
Process(pid=28970, ppid=28967)

Построение рёбер предшествования

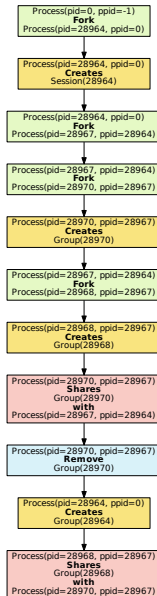


Упорядочение графа действий

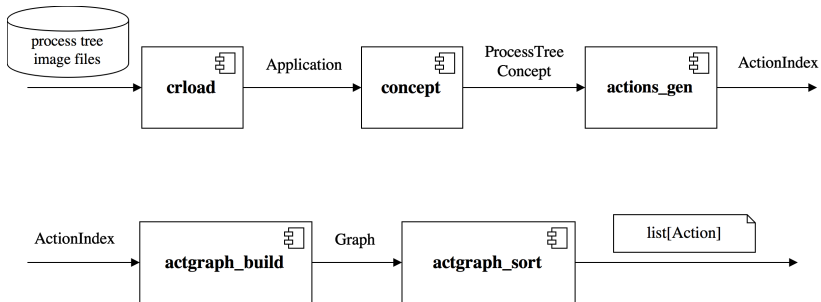
- Топологическая сортировка графа + добавление недостающих действий удаления в силу наследуемых ресурсов
- Сложность всего алгоритма построения и сортировки:

$$O\left(\sum_{P \in T} |P| \cdot (1 + |Tmp(P)|)\right)$$

- Если граф содержит цикл, то в рамках текущей модели восстановление невозможно



Модули генератора



Итоги и дальнейшие планы

Итоги

- Введена формальная обобщённая модель ресурсов ОС Linux для решения задачи восстановления
- Предложен и реализован алгоритм генерации промежуточного представления в виде графа действий и последовательности действий для восстановления дерева процессов
- <https://github.com/egorbunov/criugen>

Планы

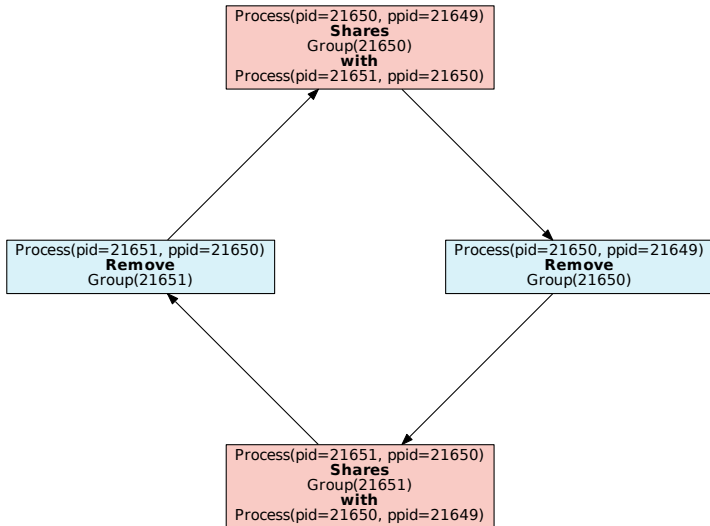
- Улучшение алгоритма для борьбы с разрешимыми циклами в графе действий
- Реализация интерпретатора команд
- Параллельное исполнение графа действий

- Для каждого процесса $P \in T$ создаём $ForkAction(parent(P), P)$
- Для каждого ресурса r в дереве выбираем его создателя P , handle h и добавляем действие $CreateAction(P, r, h)$
- Для каждого $isSharable(r)$ ресурса, добавляем $ShareAction(P, \dots)$ от создателя P к остальным процессам, держащим ресурс
- Добавляем $RemoveAction(r)$ для всех "временных" ресурсов

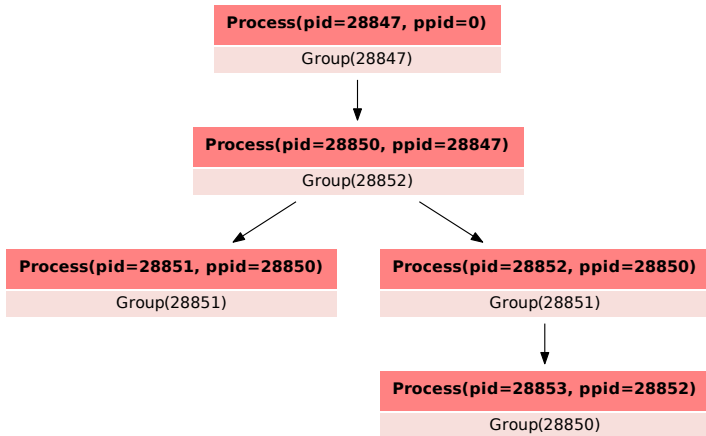
Построение ведётся в несколько этапов, каждый из которых строит часть необходимых рёбер:

- $ForkAction(_, P)$ предшествует любому действию, которое как-то нуждается в процессе P
- $CreateAction(_, r, h)$ предшествует любому действию, которое использует (r, h)
- Создание ресурса процессом P должно **учитывать наличие зависимостей** этого ресурса у процесса P
- Действия должны быть так упорядочены, что в любой момент времени состояние дерева процессов корректно
 - Вводим предикат $canExistTogether(r_1, h_1, r_2, h_2)$ и выстраиваем рёбра так, что неудовлетворяющие ему пары не существуют одновременно
- ...

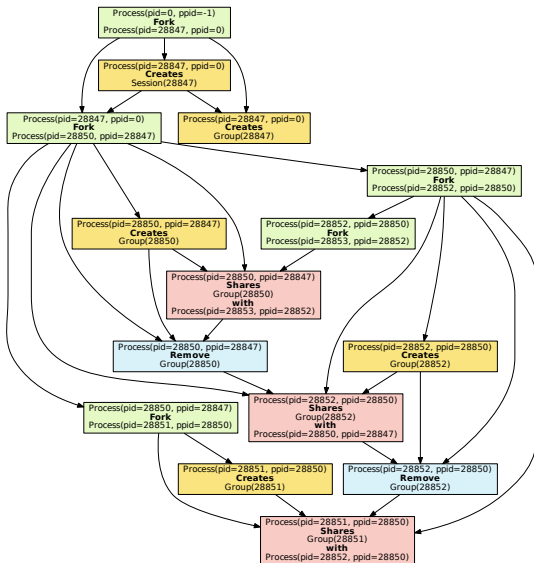
Циклический обмен конфликтующими ресурсами



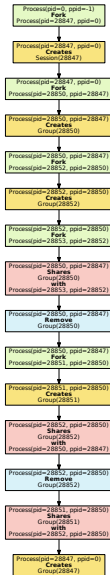
Пример с группами: дерево процессов



Пример с группами: граф действий



Пример с группами: упорядоченные действия



Иерархия ресурсов

