

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/341902610>

# МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ ВОССТАНОВЛЕНИЯ ДЕРЕВЬЕВ ПРОЦЕССОВ НА ГРАФАХ РЕКОНСТРУКЦИИ

Presentation · June 2020

DOI: 10.13140/RG.2.2.16866.56008

---

CITATIONS

0

READS

28

1 author:



Nikolay Efanov

Moscow Institute of Physics and Technology

12 PUBLICATIONS 14 CITATIONS

[SEE PROFILE](#)

# МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ ВОССТАНОВЛЕНИЯ ДЕРЕВЬЕВ ПРОЦЕССОВ НА ГРАФАХ РЕКОНСТРУКЦИИ

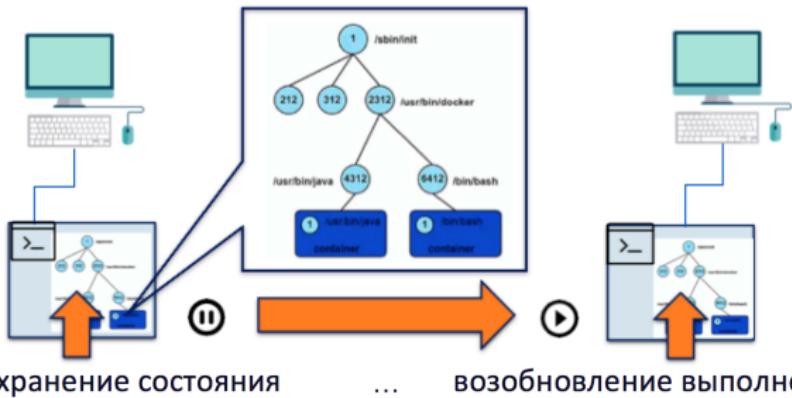
Докладчик: Н.Н. Ефанов

Научный руководитель: проф., д.ф.-м.н. А.Г. Томасов

Научный консультант: к.ф.-м.н. П.В. Емельянов

Долгопрудный, 2020

Восстановление дерева процессов для Unix-подобной ОС – реконструкция последовательностей команд, строящих дерево с заданными атрибутами процессов.



Является ключевым этапом в восстановлении состояния процессов и их разделяемых ресурсов в системах восстановления по контрольным точкам (checkpoint/restore). Приложения подобных систем:

- Контейнерная виртуализация, живая миграция
- Резервное копирование
- Восстановление после сбоев, отложенная отладка
- Ускорение загрузки программ, поддержка паузы средствами ОС

## ① В пространстве пользователя

- Профилировка системных вызовов (DMTCP)
  - + сохранение истинной последовательности команд
  - - возможное нарушение прозрачности
  - - снижение производительности
  - - компоновка со специальными библиотеками
- Эвристическое восстановление (CRIU)
  - + простота реализации
  - - полнота покрытия
  - - сложные фиксированные последовательности команд

## ② В ядре ОС (OpenVZ)

- - требует модификации ядра

## ③ Гибридные (BLCR)

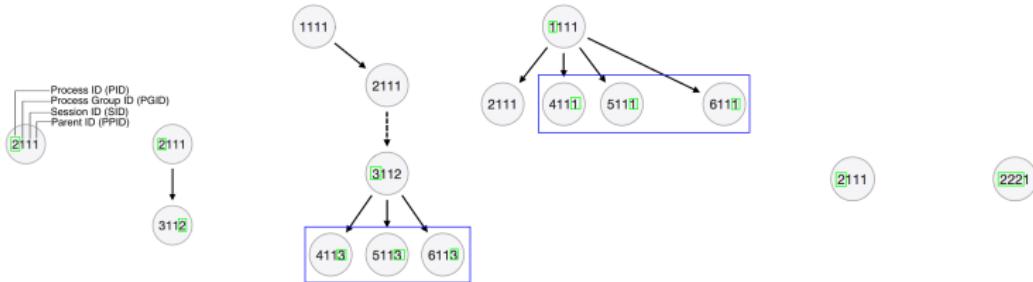
- - вышеперечисленные недостатки

## ④ Иные подходы

- Не гарантируют реконструкцию отдельных системных конфигураций и, в частности, атрибутов группы.
- Зачастую, не предлагают вычислительно эффективного способа решения.
- Перебор и хранение неэффективны в силу комбинаторного взрыва: только для *fork* число деревьев с *n* процессами:  $\sum_{i=2}^{n-1} i^{i-2}$ , где  $n \leq 2^{16} - 1$ .

# Семантика операций эволюции дерева процессов

Операции, производимые системными вызовами над отдельными вершинами дерева процессов:



2:fork()

3:exit()

2:setsid()

2111

2211

3111

3111

2111

2311

4311

3311

3311

2:setpgid(0,0)

2:setpgid(0,3)

4:setpgid(2,3)

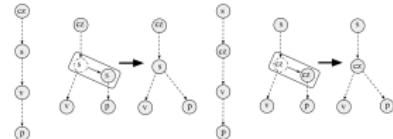
# Свойства дерева процессов, используемые при реконструкции

Пусть  $\exists B$  – цепь в дереве процессов  $T$ ,  $\exists p$  – процесс:  $B \subseteq T$ .

## Реконструкция сессии:

Пусть  $p.sid \neq p.pid$ . Тогда:

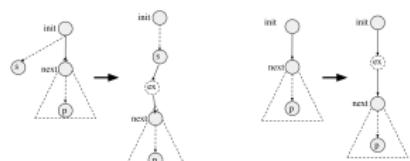
- ① Лидер сессии  $s : s.sid = s.pid = p.sid$  находится или существовал в  $T$  выше  $p$ .
- ② Пусть  $B' \subset B = \langle s, p \rangle$  – цепь от  $s$  к  $p$ , тогда  $\forall u \in B' \setminus \{s, p\} \Rightarrow u.sid = u.pid | u.sid = s.sid$ .



## Реконструкция завершенных процессов:

Пусть  $B_{full} = \langle init, p \rangle$  &  $B_{full} \setminus \{init\} = \langle next, p \rangle$ ,  
 $init \neq p$ , тогда  $V$  дополнимо процессом  $ex$ :

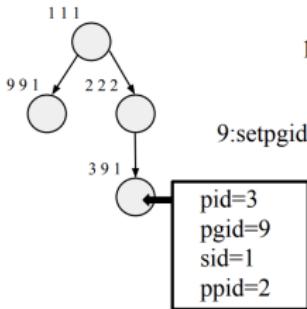
- ① если  $s \notin B_{full}$  &  $s \notin T$ , то  $s.ppid = ex.pid$  &  $ex.ppid = init.pid$  &  $ex.sid = ex.pid = s$ .
- ② если  $s \notin B_{full}$  &  $s \in T$ , то  $next.ppid = ex.pid$  &  $ex.ppid = s.pid$ .



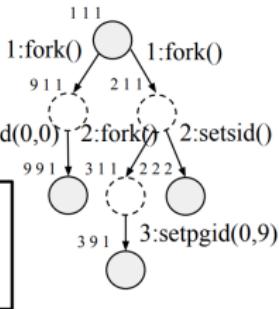
## Реконструкция группы:

$\forall u, v \in V, u.pgid = v.pgid \Rightarrow u.sid = v.sid$ , следовательно, реконструкцию может выполнить процесс той же сессии.

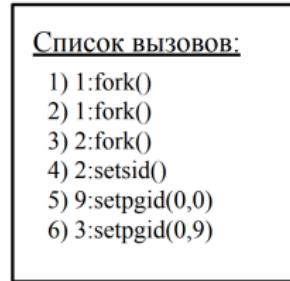
# Графовая модель реконструкции дерева процессов



a)



б)



в)

- - конечное состояние процесса
- - промежуточное состояние процесса
- ↗ - иерархические связи и системные вызовы

**Модель:** команды реконструируются как метки подмножества ребер ориентированного графа, описывающего возможный сценарий эволюции дерева процессов от корневой вершины до конечного состояния.

## Основания построения:

- Жизненный цикл процесса – цепь состояний  $B = \langle b_1, b_t \rangle$ , где  $b_1$  – состояние при рождении,  $b_t$  – в снимке состояний.
- Переходы между состояниями осуществляются в ходе выполнения системных вызовов как из контекста текущего процесса, так и из контекста других процессов.
- Большинство атрибутов, за исключением идентификатора процесса и идентификатора родителя, наследуются от процесса-родителя.
- Времена переходов между состояниями – много меньше, чем времена пребывания процессов в любых состояниях.

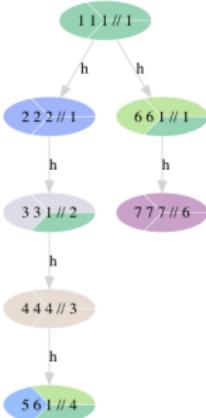
**Граф реконструкции**  $G(T) = (V^+, E^+)$  дерева  $T = (V, E)$  – ориентированный граф, содержащий:

- вершины  $V$  как конечные состояния процессов
- вершины  $V^+ \setminus V$  как промежуточные состояния процессов
- рёбра, соответствующие:
  - выполнению системных вызовов  $E^{syscalls} \subseteq E^+$
  - переходам между состояниями  $E^{follow} = E^+ \setminus E^{syscalls}$

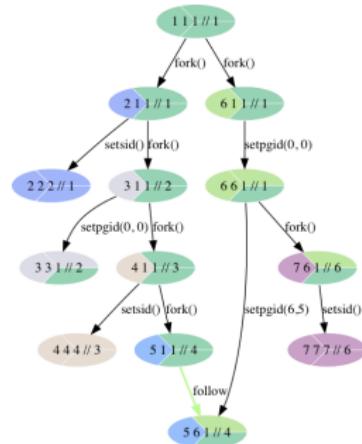
$$G(T) = (V^+, E^+) :$$

$$(V^+ = V \cup V^{int}, V \cap V^{int} = \emptyset, E^+ = E^{syscalls} \cup E^{follow}, |E \setminus (E \cap E^+)| \geq 0) \\ \& (\forall v \in V^+ \exists v.attr : v.key = val(v.attr[key]) \mid 'None', \forall key \in K) \& \\ (\forall v \in V^+ \setminus \{v_{init}\} (\exists (e = (c, v)) \in E^{syscalls} : c \in V^+) \mid \exists ((u, v) \in E^{follow}, \\ e = (c, v) \in E^{syscalls}, u, c \in V^+ : u.pid = v.pid \neq c.pid) \& \\ (in_{syscalls}(v) = 1, in_{syscalls}(v_{init}) = 0)) \quad (1)$$

# Свойства $G(T)$ и корректность дерева $T$



a)  $T$



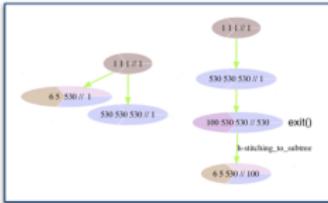
b)  $G(T)$

Доказано, что:

- ①  $G(T)$  слабо-односвязен.
- ②  $G(T)$  ацикличен.
- ③  $(V^+, E^{syscalls})$  образует оставное дерево  $G(T)$ .

**Корректным** деревом процессов именовано любое дерево процессов  $T$ , из которого за конечное число шагов строится граф его реконструкции  $G(T)$ .

Доказано формальным моделированием эволюции дерева на наборе конечных автоматов, что любое реальное дерево процессов – корректно. Построен полиномиальный алгоритм получения  $G(T)$  из корректного  $T$ .



**Input :  $T, A, K, L$**

**Output:  $G$**

**I: пред-обработка дерева  $T'$ , CR=preprocess( $T, A, K$ )**

**II: обработка**

**Реконструкция сессий**

$G = \text{process1}(T', A, K, L, CR)$

$G = \text{process2}(G, \dots)$

**Реконструкция группы**

$G = \text{process3}(G, \dots)$

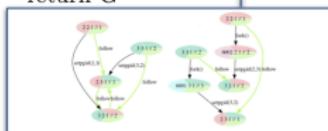
$G = \text{process4}(G, \dots)$

$G = \text{process5}(G, \dots)$

**III: пост-обработка**

$G = \text{GCorr}(G, K)$

return  $G$



```

function process4 (G,A,K,L,CR);
Input :  $G, A, K, L, CR$ 
Output:  $G$ 
//Стадия 4: реконструкция промежуточных лидеров
for item in dfs(G,ignoreHandled=True) do
    pleader = None
    parent = parent(item)
    if not as(item,item) then
        for node in df s(CR[item.sid]) do
            if as(node,item) then
                //лидер найден
                pleader=node
                break
            end
        else
            if item.pgid == node.pid then
                найден процесс-кандидат, но это не лидер
                PG.append(node)
            end
        end
    end
    if pleader == None & |PG| > 0 then
        if ∃i ∈ PG : (i,x).label = pred, x ∈ V+ then
            TR3()
            continue
        end
    else
        TR4()
        continue
    end
end
if pleader == None & |PG| == 0
    TR5()
    continue
end

```

**Проверка формулы над предикатами  $A$ :**  
 $\sim as(item) \& a_+(node, item)$



$G(T)$  строится из дерева  $T$  последовательностью трансформаций:

$$TRx(G_{i-1}(T))|_{f(a)=True} \Rightarrow G_i(T)$$

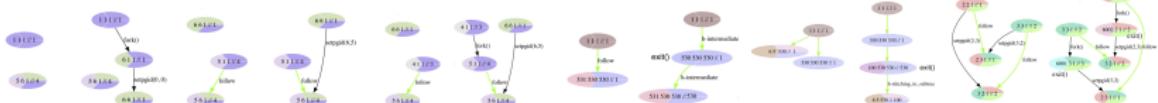
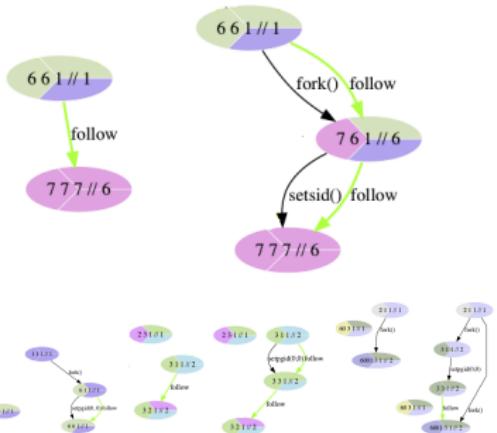
$TR1:$

имя объекта	вершина №	parent	item	parent	interm	item
-------------	-----------	--------	------	--------	--------	------

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \xrightarrow{F-W} \begin{pmatrix} 0 & 1 \\ \infty & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \xrightarrow{F-W} \begin{pmatrix} 0 & 1 & 2 \\ \infty & 0 & 1 \\ \infty & \infty & 0 \end{pmatrix}$$

Операции в трансформациях:

- сохраняют слабую связность
- сохраняют достижимость
- не удаляют вершины
- не добавляют обратные ребра



- Алгоритм завершает работу за конечное число шагов.
- Граф  $G(T) = (V^+, E^+)$ , построенный алгоритмом по корректному входному дереву процессов  $T$ , слабо-односвязен, ацикличен и содержит  $(V^+, E^{syscalls})$  оставным деревом.
- Временная сложность алгоритма –  $O(|V^+|^2)$ , пространственная –  $O(|V^+|)$ .
- $\forall G(T) = (V^+, E^+) : T = (V, E)$  – корректное дерево процессов,  $K = \{pid, sid, pgid\} \Rightarrow \frac{|V^+|}{|V|} \leq O(1)$ .

Следовательно, сложность от размера входа  $O(|V|^2)$  и  $O(|V|)$  соответственно.

**Зависимость** между состояниями  $u, v \in V^+$ : двуместное отношение  $(u, v)$  с семантикой "для реализации  $u$  требуется реализовать  $v$ ".

**Отношение доминирования.** Атрибут  $attr_2$  доминирует над  $attr_1$ , если  $\forall v, u \in V^+ : u.attr_1 = v.attr_1 \Rightarrow u.attr_2 = v.attr_2$ . **Доминирующий** над  $attr_1$  атрибут  $attr_2$  назовём **минимально доминирующим**, и определим отображение  $mindom : K \rightarrow K$ , такое, что  $mindom(attr_1) = attr_2$  тогда и только тогда, когда  $\forall u \in V^+ \exists \{v\} \in V^+ : v.attr_2 = u.attr_2, \forall V' = \{v\} \cup v',$  и  $\forall v' \in V^+ \setminus \{v\} \Rightarrow v'.attr_2 \neq const$ .

**Граф реконструкции с зависимостями:**

$$G^{dep}(T) = (V^+, E^{dep} \cup E^+) \quad (2)$$

где  $E^{dep}$  - множество зависимостей.

**Глубина изоляции атрибута:**

$$depth(attr_1) = \sum_{i=0}^{|K|} k(i) : k(i) = \begin{cases} 1 & attr_i \text{ доминирует над } attr_1 \\ 0 & \text{иначе} \end{cases}$$

**Максимальный по мощности доминирующий атрибут:**

$$maxdom(attr) = attr_x : \forall v \in V^+ \Rightarrow depth(v.attr_x) = 0.$$

**Лемма:**  $V^+$  с отношением зависимости образует полную верхнюю полурешётку.

**Доказательство:** Рассмотрим операцию взятия минимальной верхней границы  $\sqcup$  на  $V^+ : x \sqcup x = x; x \sqcup y = y \iff (x, y) \in E^{dep}$ .  $\sqcup$  по определению идемпотентна. Доказано, что:

$\forall x, y \in V^+ \exists! c \in V^+ : (x, c) \in E^{dep}, (y, c) \in E^{dep}$ . Параллельно доказывается коммутативность и ассоциативность  $\sqcup$ . Рассмотрим ситуации:

- ①  $(x, y) \in E^{dep}$  - тогда, очевидно,  $c = y$ .
- ② Доказывается  $\{(x, y), (y, x)\} \subset E^{dep} \iff x = y$  (от противного)  
 $\Rightarrow y = x \sqcup y = y \sqcup x = x$
- ③ Пусть  $(x, y, attr_1), (y, x, attr_1) \notin E^{dep}$ . Единственный  $c$  – создатель ближайшего общего доминирующего атрибута:  $\max(depth(attr_k))$  – находится индукцией по доминирующими атрибутам. Ассоциативность доказывается рассмотрением 2 композиций таких вершин.

Для разрешения зависимостей вводятся операторы

$$PCL(*.attr_i) : V^+ \rightarrow V^+ : \forall v \in V^+ \exists f \in V^+ : v \sqcup PCL(v.attr_i) =$$

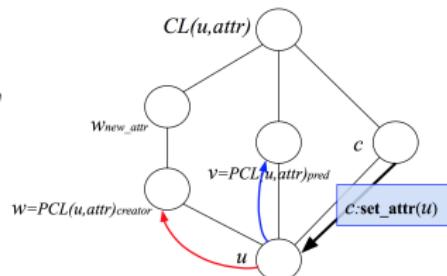
$PCL(v.attr_i) = f \ \& \ f.mindom(attr_i) = v.mindom(attr_i)$ . Доказано, что:

- $PCL$  – монотонный и экстенсивный.
- Если в вершине  $f$   $mindom(attr_i)$  был создан, то  $PCL$  – идемпотентный.  
В таком случае  $PCL$  обозначается как  $CL$  – оператор замыкания.

Зависимости, разрешаемые в  $G^{dep}$ , нестрого разделяются на 4 типа:

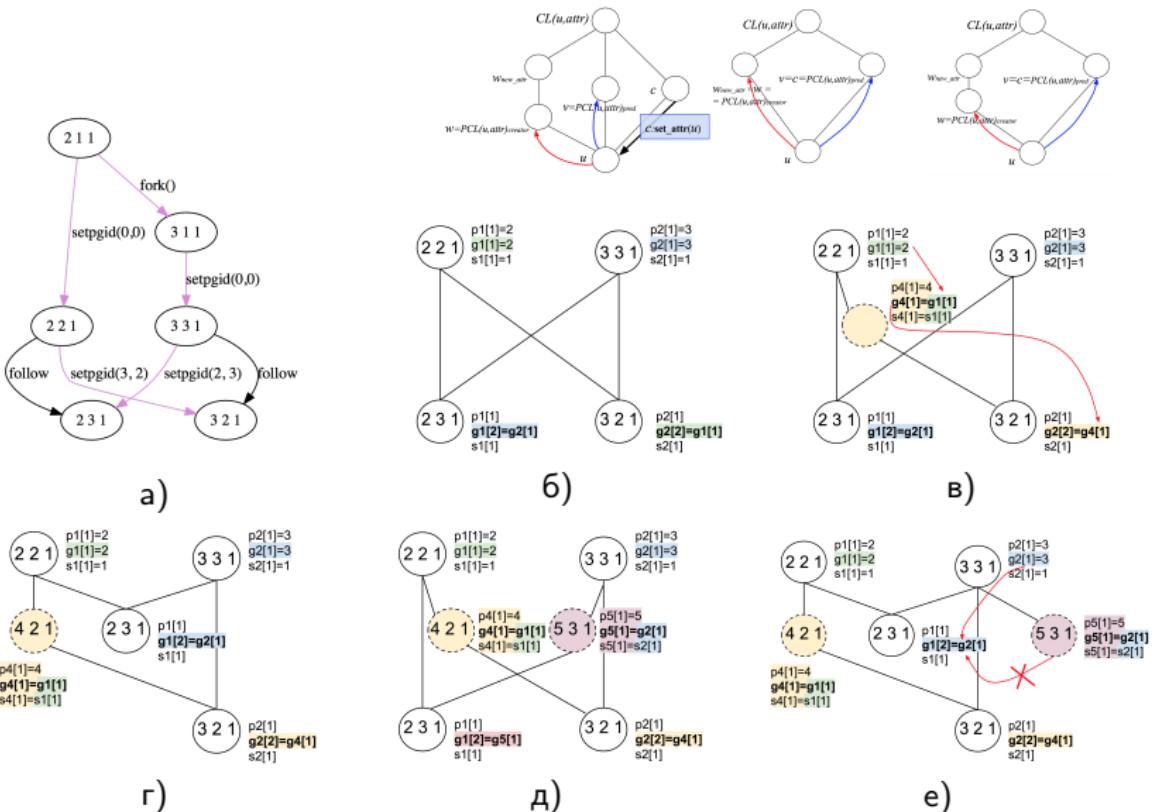
- 1 Зависимость от вершины-предшественника,  $PCL(u.attr_i)_{pred}$ .
- 2 Зависимость от вершины-создателя  $attr_i$ ,  $PCL(u.attr_i)_{creator}$ , ею же можно описать зависимость от промежуточного носителя.
- 3 Зависимость от вершины-исполнителя системного вызова, добавившего состояние с атрибутом  $attr_i$ ,  $PCL(u.attr_i)_{syscaller}$ .
- 4 Зависимость от  $CL(u.attr_i)$ .

$$\begin{cases} w \leq w_{new} = PCL(u.attr_i)_{creator} \leq CL(u.attr_i), \\ v = PCL(u.attr_i)_{pred} \leq CL(u.attr_i), \\ c = PCL(u.attr_i)_{syscaller} \leq CL(u.attr_i), \\ u \leq v, u \leq w, u \leq c. \end{cases}$$



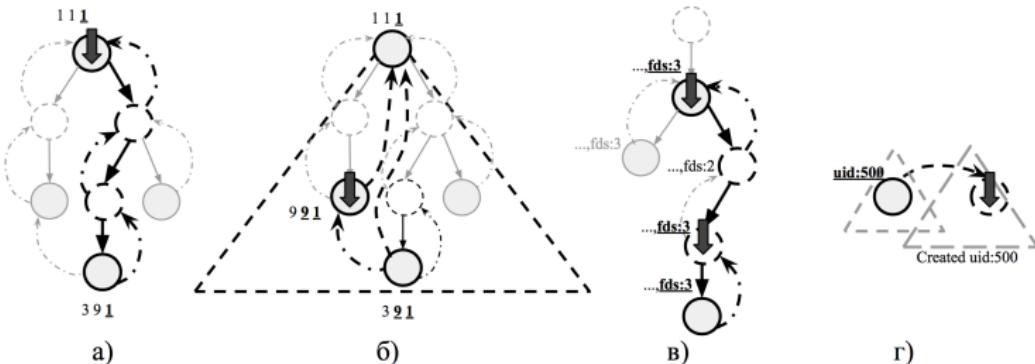
Элементы  $u, v = PCL(u.attr_i)_{pred}, w, w_{new} = PCL(u.attr_i)_{creator}, c = PCL(u.attr_i)_{syscaller}, CL(u.attr_i)$  формируют полную решетку.

# Пример: порядок состояний процессов на стадии пост-обработки



Изменение конфликтного подмножества  $V^+$  (a,6) на этапе пост-обработки – добавление носителя (в) и диаграмма Хассе исправленного подмножества (r). Подмножество с избыточно созданными носителями (д) можно переупорядочить (е) и вложить в полурешётку, не нарушая её свойств.

## Обобщённый метод реконструкции (1/2)



- - конечные состояния процессов
- ◐ - промежуточные состояния процессов
- ↓ - точки создания ресурсов (задания значения атрибуту)
- ↗ - иерархические связи и системные вызовы
- ↖ - зависимости по непосредственно предшествующим вершинам
- ↗ - зависимости от дополнительных атрибутов

Атрибуты: жестко наследуемые HI (а), устанавливаемые в подобласти SI (б), мягко-наследуемые MI (в), свободно выставляемые F (г).

Разработан обобщённый метод реконструкции дерева процессов с атрибутами, близкими по семантике операций к сессиям, группам, пространствам имён.

## Иерархия атрибутов:

Пусть  $\exists! K_t(K)$  вида  $\{k, t\}$ , где  $k \in K, t \in \{HI, SI\}^*$ ,  $DR$  – множество отношений доминирования на  $K_t$ , тогда  $H = (K_t, DR) : ((\exists!(u, v) \in DR) \Leftrightarrow (u = mindom(v))) \ \& \ (\forall w \in K_t \exists! root = maxdom(w))$

## Обобщённый алгоритм реконструкции:

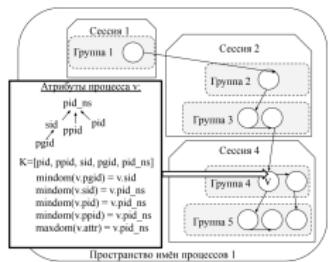
Построен на основе использования процедур Алгоритма 1.

Использует вспомогательные процедуры переименования меток графа и выделения обрабатываемых вершин, требующие  $O(|K||V^+|)$  и  $O(|V^+|)$  действий соответственно.

## Доказаны:

- Конечносъ.
- Корректность.
- Сложность:  $O(|K|^3||V^2|)$ -time,  $O(|K||V|)$ -space.
- Возможно улучшение до  $O(|K|^2||V^2|)$ -time.

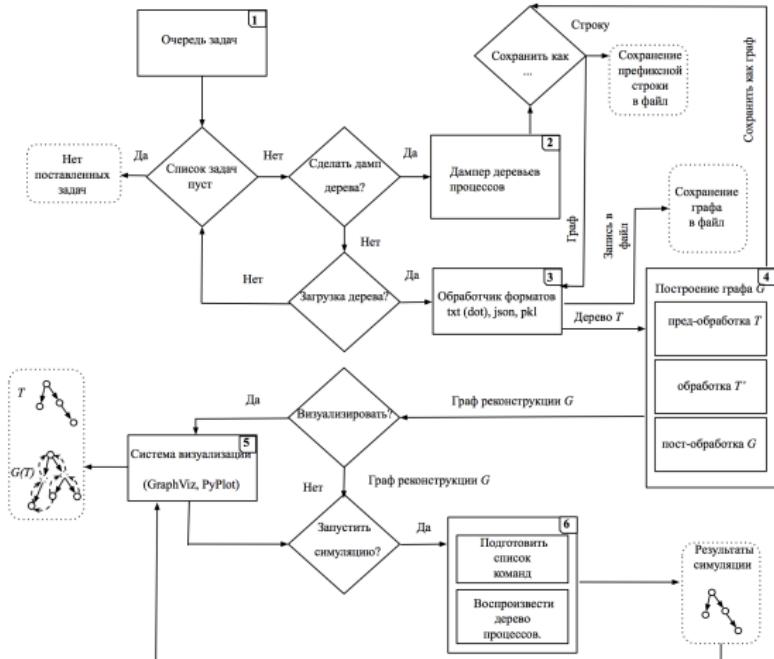
\*  $F, MI$  сводятся соответственно к  $SI, HI$



```

function genProcess (T,A,K,L,CR);
Input : T',A,K,L,CR
Output: G
G = T'
H = makeH(K,L)
for s ∈ df(H) do
    for attr ∈ children do
        if type(attr) == "H" then
            G.K,L = renameAttrs(G.K,L,'sid': attr, setsid: L[attr])
            G=processS(G,A,K,L,CR)
            G=processS(G,A,K,L,CR)
            G.K,L = renameAttrs(G.K,L,attr: 'sid',
                L[attr].setsid, 'fork': 'N')
            reactivate(G)
        end
    else
        G.K,L = renameAttrs(G.K,L,'pgid': attr, setpgid: L[attr])
        G=processS(G,A,K,L,CR)
        G=processS(G,A,K,L,CR)
        G=processS(G,A,K,L,CR)
        G.K,L = renameAttrs(G.K,L,attr: 'pgid',
            L[attr].setpgid, 'fork': 'N')
        reactivate(G)
    end
end
end
G.K,L = renameAttrs(G.K,L,None, 'N': 'fork')

```



**Схема программного комплекса. Комплекс позволяет сохранять состояние дерева процессов,**

**восстанавливать последовательность команд через построение графа реконструкции, генерировать код по реконструкции на базе восстановленных команд, воспроизводить дерево процессов в Linux (в отдельном PID, UID-пространстве имён), логировать системные вызовы.**

Комплекс доступен по адресу: <https://github.com/nefanov/reconstructor>

Сохранение-восстановление деревьев, генерируемых синтетическими тестами:

- ① simple load – *fork, setpgid, exit*
- ② context load – любые вызовы\*

Сопоставлено время построения и извлечения команд из  $G(T)$  с временными издержками на профилирование истинных команд посредством:

- strace
- perf

Метрика издержек:  $C = C_s + C_r$

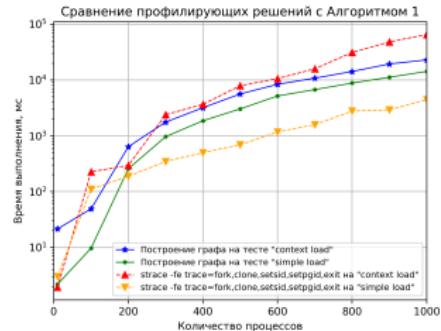
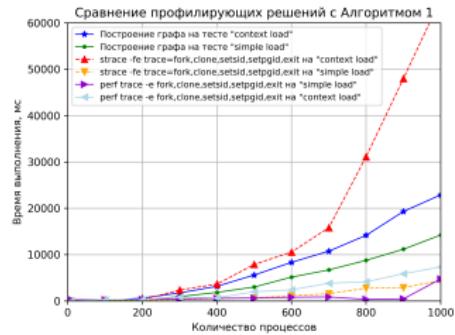
$C_s$  – время сохранения

$C_r$  – восстановления

Результаты экспериментов:

- Издержки на построение графа и извлечение команд сопоставимы с затратами на профилирование.
- Характер роста времени работы при увеличении числа процессов соответствует теоретической оценке.

\*Рассматриваемые вызовы – *fork, vfork, clone, exit, setsid, setpgid, wait, waitpid*



- Разработана математическая модель восстановления дерева процессов на базе построения графа реконструкции. Разработан полиномиальный алгоритм восстановления сессий, групп процессов и обратных усыновлений, гарантирующий реконструкцию в смысле разрешения требуемых зависимостей в момент исполнения системного вызова.
- Предложена обобщенная математическая модель разрешения атрибутных зависимостей между состояниями процессов, в рамках которой возможно восстановление деревьев с любой древесной иерархией атрибутов, восстановление подмножества процессов. Сформулирован формальный критерий корректности деревьев процессов в терминах обобщенной математической модели.
- Разработан комплекс программ для проведения экспериментов по построению графов реконструкции и сравнению результатов с профилирующими решениями.
- Проведена серия численных экспериментов с использованием разработанного программного комплекса. Результаты моделирования подтвердили теоретические оценки сложности задачи, а измеренное время работы по реконструкции и получению списков команд в экспериментах сопоставимо с временными издержками на профилирование.

# Публикации по теме диссертации:

1. Ефанов Н. Н. Исправление аномалий в графах реконструкции деревьев процессов Linux // Труды МФТИ. — 2019. — Т. 3. — С. 50–60.
2. Ефанов Н. Н. Классификация правил проверки атрибутов в деревьях процессов Linux // Естественные и технические науки. — М., 2018. — № 6. — С. 216–221.
3. Ефанов Н. On some combinatorial properties of LINUX process trees [О некоторых комбинаторных свойствах деревьев процессов LINUX] // Чебышевский сборник. — 2018. — Т. 19, № 2. — С. 151–162.
4. Ефанов Н. О полурешетке состояний процессов Linux // Чебышевский сборник. — 2019. — Т. 20, № 4. — С. 124–136.
5. Efanov N., Emelyanov P. Linux Process Tree Reconstruction Using The Attributed Grammar-Based Tree Transformation Model // Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia. — Moscow, Russian Federation : ACM, 2018. — 2:1–2:7. — (CEE- SECR '18). — ISBN 978-1-4503-6176-7.
6. Efanov N., Emelyanov P. Constructing the Formal Grammar of System Calls // Proceedings of the 13th Central Eastern European Software Engineering Conference in Russia. — St. Petersburg, Russia : ACM, 2017. — 12:1–12:5. — (CEE-SECR '17). — ISBN 978-1-4503-6396-9.
7. Efanov N., Shtypa E. Optimization of syscall sequences using minimal spanning trees search // Труды IX Московской международной конференции по исследованию операций (ORM2018-GERMEYER100). — М., 2018. — С. 12–18.
8. Ефанов Н. Н. Восстановление состояния Linux-процесса оптимальными последовательностями системных вызовов на основе марковских преобразователей // 59-я Научно-Практическая Конференция МФТИ. — М. : МФТИ, 2017. — URL: <http://conf59.mipt.ru/static/prog.html>.
9. Ефанов Н. Н. Формальная грамматика системных вызовов // 60-я Научно-Практическая Конференция МФТИ. ФПМИ. — М. : МФТИ, 2017. — С. 71–72.
10. Ефанов Н. Н. Атрибутивная грамматика деревьев процессов и восстановление порождающих цепочек системных вызовов Linux // 61-я Научно-Практическая Конференция МФТИ. ФПМИ. — М. : МФТИ, 2018. — С. 84–85.
11. Ефанов Н. Н., Щекотихина Д. Д. Измерение времени выполнения системных вызовов для выработки метрик эффективности восстановления состояния исполнения в OS Linux // 61-я Научно-Практическая Конференция МФТИ. ФПМИ. — М. : МФТИ, 2018. — С. 89–90.
12. Ефанов Н., Михайлов В. О генерации команд восстановления дерева процессов Linux // 62-я Научно-Практическая Конференция МФТИ. ФПМИ. — М. : МФТИ, 2019.
13. Ефанов Н. Н. Комбинаторные и групповые свойства деревьев процессов Linux // Сборник трудов XV международной конференции «Алгебра, теория чисел и дискретная геометрия: современные проблемы и приложения». — Тула, Май .2018. — С. 184–187.
14. Ефанов Н. Н. О некоторых полурешеточных свойствах состояний процессов Linux // Сборник трудов XVI международной конференции «Алгебра, теория чисел и дискретная геометрия: современные проблемы, приложения и проблемы истории». — Тула, Маи .2019. — С. 165–168.
15. Ефанов Н. Н. О формальной корректности атрибутивного алгоритма реконструкции деревьев процессов Linux // Сборник трудов XVII международной конференции «Алгебра, теория чисел и дискретная геометрия: современные проблемы, приложения и проблемы истории». — Тула, Сентябрь.2019. — С. 88–91.
16. Ефанов Н. Н., Емельянов П. В. Построение формальной грамматики системных вызовов //

Спасибо за внимание!

- В разделе 4.2.2 представлены тестовые нагрузки в виде двух синтетических профилей. На них проведено сравнение с логирующими решениями perf и strace, но не приведено сравнение с эвристическими методами, например, с используемыми в CRIU.
- В экспериментах для генерации деревьев использованы синтетические тесты. Не хватает понимания, каким образом использованные профили конфигураций соотносятся с конфигурациями, получаемыми как снимки реальных систем