На правах рукописи 

Ефанов Николай Николаевич

МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ ВОССТАНОВЛЕНИЯ ДЕРЕВЬЕВ ПРОЦЕССОВ НА ГРАФАХ РЕКОНСТРУКЦИИ

Специальность 05.13.18 —
«Математическое моделирование, численные методы и комплексы программ»

Автореферат

диссертации на соискание учёной степени

кандидата физико-математических наук

Долгопрудный — 2020

Работа выполнена в Федеральном государственном автономном образовательном учреждении высшего образования «Московский Физико-Технический Институт (Национальный Исследовательский Университет)»

Научный руководитель: доктор физико-математических наук, профессор **Тормасов Александр Геннадьевич**

Научный консультант: **Емельянов Павел Владимирович,** кандидат физико-математических наук

Ведущая организация: Федеральное государственное бюджетное учреждение науки «Институт системного программирования им. В.П. Иванникова Российской академии наук»

Защита состоится 1 июня 2020 г. в 11 часов на заседании диссертационного совета ФПМИ.05.13.18.008 по адресу: 141701, Московская область, г. Долгопрудный, Институтский переулок, д.9.

С диссертацией можно ознакомиться в библиотеке и на сайте Московского физико-технического института (национального исследовательского университета) https://mipt.ru/education/post-graduate/soiskateli-fiziko-matematicheskie-nauki.php.

Работа представлена «4» марта 2020 г. в Аттестационную комиссию федерального государственного автономного образовательного учреждения высшего образования «Московский физико-технический институт (национальный исследовательский университет)» для рассмотрения советом по защите диссертаций на соискание ученой степени кандидата наук, доктора наук в соответствии с п. 3.1. ст. 4 Федерального закона «О науке и государственной научно-технической политике»

Общая характеристика работы

Актуальность темы. В диссертационной работе рассматривается построение математической модели реконструкции дерева процессов при восстановлении состояния набора приложений из снимка состояний или из упрощённого атрибутного описания. Данная задача связана с активно развивающимися в последнее время технологиями сохранения-восстановления состояний окружения исполнения в ОС, виртуализации, и, как следствие, является подзадачей ряда актуальных задач, решаемых программными комплексами: живой миграции и балансировки нагрузки, контейнерной виртуализации, восстановления после сбоев, отложенной отладки, ускорения загрузки приложений , остановки/возобновления работы приложений, поддержки инфраструктуры в системах высокопроизводительных вычислений.

Существующие на момент проведения исследования индустриальные и исследовательские системы сохранения-восстановления так или иначе решают задачу восстановления дерева процессов: любая такая система, работающая в Unix-подобной ОС, поддерживающая многозадачность и разделение ресурсов, должна обеспечивать реконструкцию иерархии процессов, их изоляцию – разделение на группы, сессии, пространства имён, осуществлять передачу наследуемых и разделяемых иными способами ресурсов в правильном порядке, что обуславливается переводом дерева процессов в надлежащее состояние. Способы такого восстановления различны. Существует класс решений, осуществляющий запись активности программ в ходе работы, с последующим воспроизведением данной активности при восстановлении. Такой подход имеет ряд недостатков, связанных как со сложностью обеспечения прозрачности переноса состояния относительно целевого приложения, так и со сложностью сохранения-воспроизведения активности: накладными расходами на запись, платформозависимостью. Другой класс решений производит восстановление на основе эвристического анализа данных снимков состояния. Как правило, такие решения имеют в своей основе фиксированные последовательности шагов по восстановлению, покрывающие конкретные случаи деревьев процессов. Такой подход также обладает недостатками в смысле надёжности и универсальности: существуют особые случаи, в которых эвристические решения генерируют некорректные последовательности действий или выходят из строя. Также следует отметить, что ряд решений требует моди-

фикации ядра ОС или использования специальных библиотек, компонуемых с целевым приложением, что является архитектурным недостатком.

Наконец, существует ряд исследований, решающих задачу восстановления строго для одного типа ресурса, к примеру, для групп процессов одного контейнера (Миркин) или множества сессий (Лаадан, Осман), либо обрабатывающие множество различных ресурсов, но имеющие определённые недостатки в моделях анализа и представления ресурсов (Горбунов, Баталов), из которых вытекает снижение надёжности. В виду недостатков вышеперечисленных решений, актуальна задача построения математической модели восстановления дерева процессов, обобщающей эвристические подходы и другие существующие модели, гарантирующая восстановление состояний процессов, владеющих системными ресурсами различных типов. С целью обеспечения широты использования, модель должна учитывать возможность восстановления на немодифицированном ядре, то есть построение на её базе системы восстановления из пространства пользователя, по аналогии с таким решением, как CRIU.

В работе предлагается построение модели реконструкции на базе промежуточного представления в виде специального ацикличного графа реконструкции, содержащего минимальным остовным покрытием подмножество рёбер с метками–системными вызовами.

Целью данной работы является построение вышеуказанной математической модели реконструкции дерева процессов, учитывающей некоторый ограниченный класс атрибутов ресурсов различного типа: групп процессов, сессий, идентификаторов процесса, с возможностью обобщения на более широкий класс атрибутов со схожими свойствами. Для деревьев, полученных из реальных системных конфигураций, модель должна гарантировать корректность выдачи последовательностей команд пространства пользователя, приводящих к реконструкции.

Для достижения поставленной цели необходимо было решить следующие **задачи**:

1. Исследовать свойства деревьев процессов, позволяющие производить обобщение эвристических принципов восстановления, оценить возможность непосредственной генерации деревьев.

2. Разработать полиномиальный алгоритм, строящий последовательность команд реконструкции дерева процессов, с учётом идентификаторов процесса, сессии, группы процессов, обеспечивающий восстановление дерева из корневой вершины при воспроизведении данных команд.
3. Провести формальное исследование зависимостей между различными атрибутами процессов, определить частичный порядок по разрешению зависимостей на множестве состояний процессов, исследовать свойства множеств состояний относительно вводимого порядка.
4. Обобщить метод реконструкции на более широкий класс атрибутов процессов, обладающих схожими свойствами с исследованными ранее атрибутами.
5. Связать построенную модель с корректностью восстановления деревьев, соответствующим реальным конфигурациям ОС.

Основные положения, выносимые на защиту:

1. 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.
2. Предложена обобщённая 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 обобщённой математической модели.
3. Разработан комплекс программ для проведения экспериментов по построению графов реконструкции и сравнению результатов с профилирующими решениями.
4. Проведена серия численных экспериментов с использованием разработанного программного комплекса. Результаты моделирования подтвердили теоретические оценки сложности задачи, а измерен-

ное время работы по реконструкции и получению списков команд в экспериментах сопоставимо с временными издержками на профилирование.

Научная новизна:

1. В данном исследовании впервые были определены характеристики деревьев процессов, позволяющие производить обобщение эвристических принципов совместного восстановления сессий, групп, пространств имён процессов, а также примитивов со схожей с ними семантикой, опираясь на особенности отображений ресурсов ОС в пространство пользователя как атрибутов процесса и следующие из них структурные свойства деревьев. Получены комбинаторные оценки числа деревьев при учёте различных атрибутов и системных вызовов.
2. Впервые предложены новые промежуточные представления в задаче восстановления дерева процессов, такие как граф реконструкции и граф реконструкции с зависимостями, проведён структурный анализ данных представлений, сформулирован и формально обоснован механизм восстановления команд реконструкции как ребёрного покрытия ациклического графа.
3. Впервые формально определены свойства частичного порядка, задаваемого зависимостями на множествах состояний процессов. Впервые представлен формальный критерий корректности дерева процессов на разрешаемых зависимостях.
4. Впервые получен метод реконструкции дерева процессов, гарантированно обеспечивающий восстановление для атрибутов некоторого типа и структуры за полиномиальное время.

Практическая значимость работы заключается в возможности использования предложенных алгоритмов в системах сохранения-восстановления с целью повышения их надёжности, так как заданных ограничениях на типы атрибутов реконструкция гарантируется. Критерий корректности деревьев процессов может служить основой для разработки методов валидации деревьев процессов и генерации синтетических тестов систем сохранения и восстановления. Разработанный в ходе работы программный

комплекс может служить инструментом исследования деревьев процессов и графов реконструкции.

Достоверность полученных результатов обеспечивается использованием формальных методов исследования, выполнением формальных доказательств и программными экспериментами. Результаты находятся в соответствии с результатами, полученными другими авторами.

Апробация работы. Основные результаты работы докладывались на ряде международных и всероссийских конференций: 59–62 Научно–практических конференциях МФТИ, IX Московской международной конференции по исследованию операций, XV–XVII Международных конференциях "Алгебра, теория чисел и дискретная геометрия", международных конференциях "Software Engineering Conference in Russia (CEE–SECR)" 2017–2019. Доклад на конференции CEE–SECR 2017 был награждён премией Бертрана Мейера за лучшую научно-исследовательскую работу в области программной инженерии.

Личный вклад. все основные результаты диссертационной работы получены автором лично.

Публикации. Основные результаты по теме диссертации изложены в 16 публикациях, 4 из которых изданы в журналах, рекомендованных ВАК [1—4], 5 — в изданиях, входящих в Scopus/RSCI Web Of Science [3—6], 9 [7—15] — в тезисах докладов конференций, 1 [16] — в иных изданиях. Работы [5—7; 11; 12; 16] были выполнены в соавторстве. В работе [5] Н.Ефанову принадлежит идея атрибутно-управляемой трансформации дерева, алгоритм, анализ вычислительной сложности и постановка эксперимента, в работах [6; 16] – построение метода двухстадийного анализа, оценки вычислительной сложности и постановка эксперимента, в [11] – идеи и обоснование исследуемых метрик эффективности, техническая постановка эксперимента, в [7; 12] – идеи и обоснование использования конкретных алгоритмов.

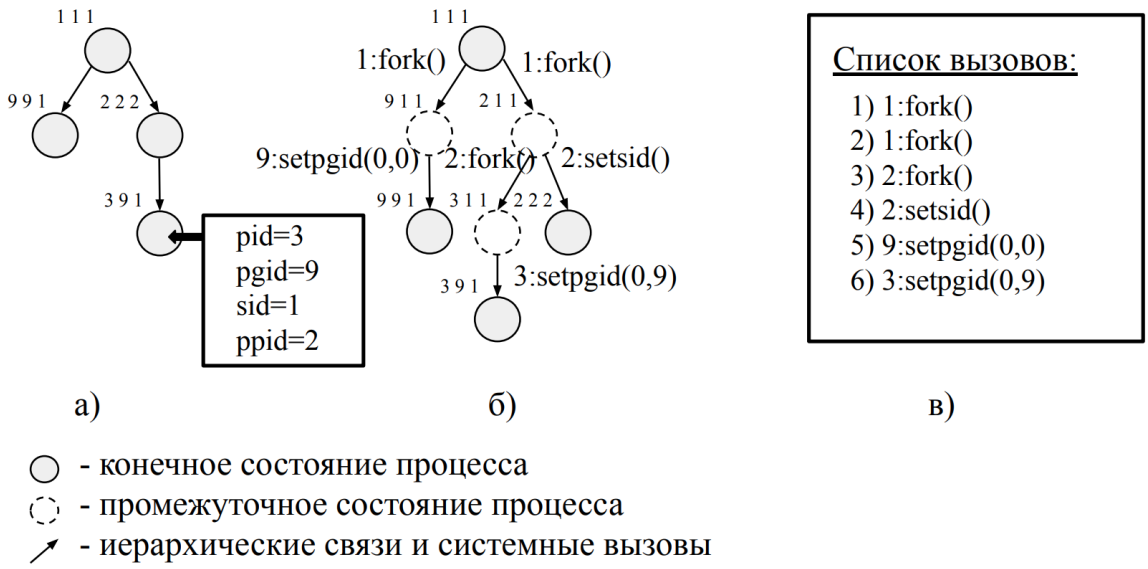


Рис. 1 — Дерево процессов T с атрибутами $attr = \{pid,pgid,sid,ppid\}$ (а), визуализация эволюции T из корневой вершины (б) и список вызовов, произведённых при построении в контексте соответствующих процессов (в).

Содержание работы

Во **введении** обосновывается актуальность исследований, выполненных в рамках диссертационной работы, приводится обзор научной литературы по изучаемой проблеме, формулируется цель, задачи, описывается научная новизна и практическая значимость представляемой работы.

Первая глава посвящена обзору области исследования и используемых в решении поставленных задач методов дискретной математики, информатики и теории операционных систем.

Вторая глава посвящена исследованию деревьев процессов Unix-подобных операционных систем, выявлению практически важных для решения поставленной задачи свойств наследования и сегментации состояний процессов по атрибутам, обоснованию необходимости анализа межатрибутных отношений в деревьях, определению эффективного математического аппарата для такого анализа, построению и обоснованию корректности алгоритма реконструкции дерева для подмножества атрибутов, включающего идентификатор процесса, родителя, сессии и группы процессов. Полученный алгоритм обобщает и развивает ряд существующих решений, гарантированно повышая их надёжность. **Раздел 2.1** посвящён оценкам количества различных деревьев в зависимости от атрибутов процессов методами перечислительной комбинаторики, а также получению некоторых свойств, связанных

изменением деревьев в ходе выполнения системных вызовов (Рис. 1). Пусть в ходе работы ОС создан снимок состояния S. Пусть P - множество замороженных процессов в S. Дерево процессов T рассматривается как ориентированное дерево, каждая вершина которого содержит представление состояния процесса из множества процессов снимка состояния, и атрибутирована словарём $attr$, хранящим идентификаторы предоставленных процессу ресурсов; пусть K - список ключей как имён атрибутов процесса: системных идентификаторов, файловых дескрипторов, сигналов, отображений памяти, так, что: $val(v.attr[key]) : K \to val\_with\_type(key)$ - отображение ключей на типизированные значения атрибутов. Рёбра E представляют отношения родитель-потомок между процессами:

$$\begin{gathered} T = T(V,E) : V = \{v : v \in P\}, \\ \forall v \in V \exists v.attr : v.key = val(v.attr[key]) | 'None', \forall key \in K, \\ E = \{(u,v) : v.ppid = u.pid \ \ \& \ \ u,v \subset V\} \end{gathered} \tag{1}$$

Оценивается количество деревьев, порождаемых системными вызовами fork и exit, суммированием количества различных деревьев из заданных вершин, полученного по формуле Кейли:

$$F(n) = \sum_{i=2}^{n-1} i^{i-2}, \tag{2}$$

где $n = 2^{16}$ – максимально возможное число процессов в ОС Linux. Легко доказать обобщённое утверждение: пусть $F(n)$ – число деревьев из n процессов с поддержкой некоторых системных вызовов. Рассмотрим новый вызов $k\_call(k)$, изменяющий неотрицательный целочисленный атрибут вызывающего процесса k, наследуемый при fork: $k\_call(k') = k'$, $k\_call(0) = pid$, где pid – идентификатор вызвавшего процесса. Тогда число деревьев процессов:

$$F(n, k\_call) = F(n) \sum_{m=0}^{n-1} \binom{n}{m} (m+1)^{n-m-1} \tag{3}$$

Экспоненциальный рост числа деревьев в при добавлении новых процессов и системных вызовов, даже с учётом образования симметрической группы идентификаторами некорневых процессов, говорит о неэффективности пе-

реборной генерации цепочек системных вызовов для решения задачи, что мотивирует разрабатывать более эффективные вычислительные методы, основанные на подробном анализе атрибутов в деревьях процессов.

В **разделе 2.2** исследуются синтаксические свойства деревьев процессов, строится соотвествующая атрибутная КС-грамматика, порождающая язык строк деревьев процессов в префиксной нотации. Разбором в данной грамматике можно построить атрибутированное дерево, и получить из него абстрактный семантический граф, изоморфный дереву процессов. Отдельный вопрос исследования атрибутов процессов как синтаксических составляющих выделен в Приложение А.

В **разделе 2.3** исследован ряд свойств сессий и групп процессов, упрощающих атрибутный анализ. Важным свойством является инвариантность наследуемой сессии от лидера:

Лемма 1. *Пусть* $\exists B$ *– цепь в дереве* T, $\exists p$ *– процесс, такие, что* $B \subset T$ & $p \in B$ & $p.sid \neq p.pid$. *Тогда верны следующие утверждения:*

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

Лемма описывает схему передачи атрибутов, наследуемых процессом при рождении, которые могут в процессе эволюции процессов либо сохраняться, либо однократно замещаться, либо уничтожаться. К таким атрибутам относятся сессии, IPC-дескрипторы семейства SYS V, созданные с приватным ключом доступа, и, в некоторых ограничениях, пространства имён. Возможность воссоздания завершённых процессов, потомки которых были переупорядочены к корню, также следует из данной леммы, причём без строгого определения позиции завершения в дереве.

Из леммы 1 тривиально следует механизм обратного переупорядочивания процессов, оказавшихся перенаследованными init-процессом после завершения родителя:

Утверждение 1. *пусть существует некорневой процесс* p, *тогда* $B_{full} = \langle init,p \rangle$ & $B_{full} \setminus init = \langle next,p \rangle$, $init \neq p$, *тогда* V *дополнимо временным процессом* ex, u:

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

Группа процессов при этом может быть реконструирована на следующем шаге: из того, что группа существует только в рамках одной сессии, то есть $\forall u, v \in V, u.pgid = v.pgid \Rightarrow u.sid = v.sid$, тривиально следует свойство:

Утверждение 2. $(\forall u,v \in V^+, \forall k,m \in K : (u.k = v.k) \Rightarrow (u.m = v.m)) \Leftrightarrow (\forall((V_1 = \{v_1 \in V^+ : v_1.k = u.k\}, V_2 = \{v_2 \in V^+ : v_2.k = v.k\}) \ \& \ (u.m \neq v.m)) \Rightarrow (V_1 \cap V_2 = \emptyset))$

Утверждение 2 предоставляет возможность последовательно реконструировать состояния процессов с вложенными атрибутами: к примеру, сначала реконструируются сессии, а потом происходит реконструкция каждой группы, не выходя за пределы её сессии.

Раздел 2.4 посвящён построению математической модели восстановления дерева процессов в терминах атрибутированных графовых структур.

Неформальная постановка задачи разработки способа восстановления: разработать метод построения средствами ОС Linux заранее заданного дерева процессов со своими атрибутами, начиная из корневой вершины.

Вводится ряд предложений, связанных с техническими свойствами процессов, леммой 1 и утверждением 2:

Предложение 1. *Команды по восстановлению дерева процессов могут быть представлены в терминах системных вызовов, изменяющих атрибуты процессов, при этом время выполнения каждого системного вызова должно быть много меньшим, чем время пребывания процесса в любом из состояний, за исключением, быть может, вызова waitpid, ожидающего завершения одного из процессов-потомков, и не изменяющего атрибутов процессов.*

Предложение 2. *Имеет место ряд технических свойств, следующих из внутреннего устройства ОС Linux:*

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

Предложение 3. *Построим на базе информации, полученной из дерева процессов, ориентированную корневую графовую структуру данных, содержащую в качестве вершин состояния, проходимые процессами в ходе реконструкции из корневого состояния, и с заданными на данной структуре бинарными отношениями, определяющими частичный порядок на множестве вершин, как ориентированными рёбрами. Информация об атрибутах промежуточных вершин и отношениях должна быть восстановлена на основании атрибутов конечных состояний и структурных свойств дерева процессов.*

Формальная постановка задачи реконструкции: Для входных данных – дерева процессов $T = (V,E)$, словарей ключей атрибутов K, системных вызовов L и предикатов проверок атрибутов A, построить граф реконструкции $G(T) = (V^+,E^+)$, такой, что:

$$
\begin{gathered}
(V^+ = V \cup V^{int}, V \cap V^{int} = \emptyset, E^+ = E^{syscalls} \cup E^{follow}, |E \setminus (E \cap E^+)| \geq 0) \\
\& \ \ (\forall v \in V^+ \, \exists v.attr : v.key = val(v.attr[key]) \mid 'None', \forall key \in K) \ \ \& \\
(\forall v \in V^+ \setminus \{v_{init}\} (\exists (e = (c,v) \in E^{syscalls} : c \in V^+) \mid \exists ((u,v) \in E^{follow}, \\
e = (c,v) \in E^{syscalls}, u,c \in V^+ : u.pid = v.pid \neq c.pid)) : (\exists (a \in A : \\
a(c.attr, v.attr) = True) \ \ \& \ \ \exists (e.label : E^{syscalls} \to L \times c.attr \times v.attr)) \ \ \& \\
(in_{syscalls}(v) = 1, in_{syscalls}(v_{init}) = 0))
\end{gathered}
\tag{4}
$$

Восстановление групп и сессий является частным случаем данной задачи, при котором размеры словарей A, L, K существенно меньшие, чем ограничение на

мощность V: $max(|L|) + max(|K|) + max(|A|) << max(|V|)$, что позволяет исследовать непосредственно особенности построения графа реконструкции: технические нюансы, вроде сложности работы с множеством предикатов, влияния числа атрибутов и типов системных вызовов на реконструкцию при этом должны асимптотически нивелироваться. Далее показано, что для большого числа атрибутов и системных вызовов со сходной к сессиям и группам семантикой задача сводима к многократному решению вышеупомянутого частного случая.

Моделирование реконструкции дерева процессов на графах заключается в процедуре генерации графа реконструкции $G(T) = (V^+,E^+)$ по входному дереву T, топологической сортировке и извлечению из полученного графа команд по реконструкции в порядке, заданном топологической сортировкой. Каждое ребро из $E^{syscalls}$ помечено командой, и для реконструкции в рамках модели требуется, чтобы $(V^+,E^{syscalls})$ являлось остовным деревом $G(T)$. Необходимым условием существования такого остова, покрывающего весь граф, является односвязность графа в слабом смысле. Это накладывает ограничения на формируемые процедуры трансформации, строящие граф реконструкции из дерева процессов: каждая трансформация должна сохранять слабую связность и локальную достижимость вершин, частичную упорядоченность и при этом фактически запрещать обратные рёбра в перезаписываемых подграфах: если до трансформации, изменяющей подграф (V^*,E^*) на (V^{**},E^{**}) в графе существовал простой ориентированный путь в $\langle u,v\rangle, u \in V^+ \setminus V^* \cup V^{**}, v \in V^* \cap V^{**}$, такой, что $\exists\{w_m : w_m \in V^+\} \subset \langle u,v\rangle : |\langle u,w_1\rangle| < |\langle u,w_2\rangle| < ... < |\langle u,w_m\rangle|$, то после её проведения должен существовать ориентированный простой путь $\langle u,v\rangle'$, такой, что $\exists\{w_{m_k} : w_{m_k} \in V^+\} \subset \langle u,v\rangle : |\langle u,w_{m_1}\rangle| < |\langle u,w_{m_2}\rangle| < ... < |\langle u,w_{m_k}\rangle|$, и, возможно, содержащий другие промежуточные вершины и рёбра, чем $\langle u,v\rangle$, то есть любой путь, проходящий через любую непустую подпоследовательность w_{m_k} последовательности вершин w_m, через которую существовал путь до трансформации, сохраняет порядок следования вершин.

Существование графа реконструкции с вышеуказанными свойствами обеспечивается так называемой корректностью дерева процессов.

Определение 1. *Дерево процессов T называется корректным, если для него выполнены условия леммы 1, утверждения 2, и из него строится граф*

реконструкции $G(T)$, содержащий команды, выполнение которых в порядке, определяемом топологической сортировкой $G(T)$, строит дерево T из корневой вершины.

Обоснование существования графа $G(T)$ для любого корректного T производится формальным моделированием роста дерева процессов набором конечных автоматов. Каждый автомат из набора моделирует поведение одного процесса при переходе из состояния в состояние: автомат распознаёт язык строк, символы которых соответствуют системным событиям, в ходе которых меняется состояние моделируемого процесса, либо символу пропуска, по которому автомат совершает петлевой переход. Все события выстроены в последовательность, что обоснованно можно сделать в виду того, что моделируется именно реконструкция, и важен её конечный результат. Каждый момент дискретизированного времени все автоматы, не завершившие работу, синхронно читают символ из специально подготовленной для каждого автомата строки и либо переходят в новое состояние, либо совершают петлевой переход, либо переходят в терминальное состояние. Если дерево процессов корректно, то в конце реконструкции все автоматы окажутся в своих терминальных состояниях, что равносильно тому, что все действия по реконструкции произошли. Далее по состояниям и непетлевым переходам можно построить граф, дополненный событиями (источник события считается известным), а также дерево на терминальных состояниях автоматов. Далее доказывается, что построенное дерево и редукция графа изоморфны соотвественно дереву процессов и его графу реконструкции.

Таким образом, по построению выполняется достижимость любой вершины графа из корневой, откуда следует слабая односвязность, отсутствие в графе петель и путей, повторно содержащих одну и ту же вершину, откуда следует ацикличность. Возможность получения такого графа из дерева процессов последовательными трансформациями следует непосредственно из формальной процедуры построения.

Подаздел 2.6 посвящён построению алгоритма реконструкции сессий и групп – Алгоритма 1. Построение графа реконструкции происходит в 3 стадии Алгоритма 1:

1. Стадия пред-обработки обеспечивает восстановление состояний, в которых были созданы жестко наследуемые атрибуты сессии, а так-

же корректное обратное усыновление поддеревьев, переупорядоченных к корню после завершения нелистовых процессов, обеспеченное свойствами из леммы 1. Выходными данными является корректное дерево процессов T', из которого можно получить T, завершив все добавленные процессы.

2. Стадия обработки заключается в восстановлении промежуточных состояний для каждого процесса: сначала для каждого состояния p - вершины T' - атрибуты сверяются с родительскими. Если группа и сессия совпали, процесс получен через $fork$. Иначе, происходит восстановление сессии и группы в соответствии с существованием лидера сессии в T', а затем группы в сессии, производящиеся соответствующими трансформациями на основании наборов правил, составленных из предикатов в форме КНФ Хорна[1]. Трансформации соотвествуют требуемым свойствам, что обеспечено проверкой матриц достижимости подграфов до и после трансформаций, достаточно просты, не требуют сложных проверок графовых изоморфизмов.
3. Стадия пост-обработки обеспечивает перестраивание полученного на стадии обработки графа, добавляя промежуточные носители идентификаторов групп для гарантии существования группы в момент попытки перевода в неё некоторого процесса. В работе рассмотрен экстенсивный метод добавления носителя, приводящий к некоторой избыточности, а также обсуждены более эффективные методы внесения поправок.

Сформулированы и доказаны теоремы о завершаемости, корректности и вычислительной сложности предложенного алгоритма.

Теорема 1. *Алгоритм завершает работу для любых входных данных*[2].

Теорема 2. *Любой граф $G(T)$, построенный алгоритмом по корректному входному дереву процессов T, слабо-односвязен, ацикличен и содержит $E^{syscalls}$ остовным деревом.*

[1]Таким образом, проверка выполнимости любого набора – полиномиальна

[2]Всё же будем подразумевать данные о T в виде ориентированного графа, иначе считаем вход не прошедшим синтаксическую проверку

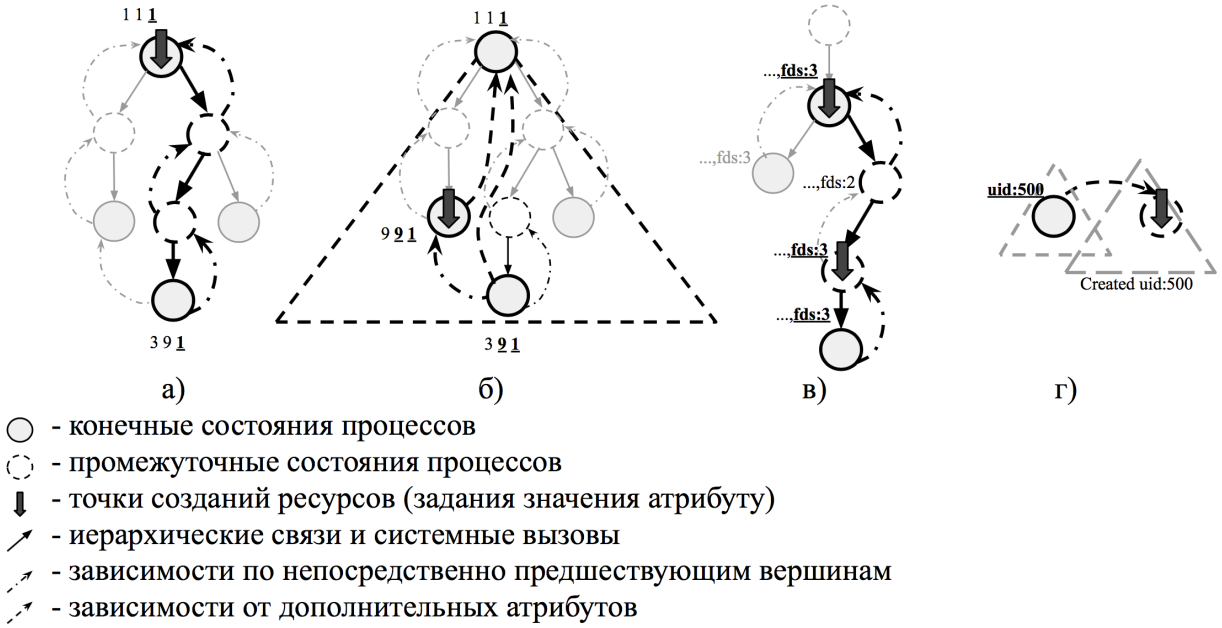


Рис. 2 — Выделенные типы атрибутов ресурсов: жестко наследуемые (а), устанавливаемые в подобласти (б), мягко-наследуемые (в), свободно выставляемые (г).

Лемма 2. *Алгоритм имеет временную сложность порядка* $O(|V^+|^2)$ *и пространственную сложность порядка* $O(|V^+|)$.

Теорема 3. *Для любого* $G(T) = (V^+,E^+)$, *полученного алгоритмом из корректного дерева процессов* $T = (V,E)$ *с атрибутами* $K = \{pid,sid,pgid\}$, *справедлива оценка* $\frac{|V^+|}{|V|} \leq O(1)$.

Третья глава посвящена строгому анализу межатрибутных зависимостей в частично упорядоченных множествах состояний процессов. Целью строгого анализа зависимостей является обобщение результатов, представленных во **второй главе**, на более широкий класс атрибутов. В работе выделяются 4 типа атрибутов:

1. Жестко наследуемые (Hardly Inherited, HI) – в G^{dep} существует цепь от создателя атрибута к наследующему потомку строго по E^+. Пример: сессии, пространства имён.
2. Устанавливаемые в подобласти (Subset Inherited, SI) -– носитель выставляемого значения, предшествующее состояние процесса и состояние, из контекста которого выполняется системный вызов, лежат в замыкании, определяемом некоторым оператором CL. Пример: группы процессов.

3. Мягко наследуемые (Mildly Inherited, MI) – ослабление SI в смысле возможного многократного изменения. Пример: файловые дескрипторы.
4. Свободно устанавливаемые в рамках соответствующего пространства имён. Пример: использование существующего идентификатора пользователя.

Для постоения обобщения разработанной модели реконструкции на данные типы атрибутов, следует рассмотреть структуру множества состояний относительно зависимостей между атрибутами, исследуемую в **разделе 3**.

Раздел 3.1 представляет теоретические результаты анализа V^+ как частично упорядоченного множества. Зависимость между состояниями u,$v \in V^+$ вводится в как двуместное отношение (u,v) с семантикой "для реализации u требуется сначала реализовать v". Если в некоторый момент в системе существует процесс в состоянии v, то выполнено необходимое (но не достаточное!) условие для реализации состояния u некоторого процесса, а зависимость называется разрешённой. Зависимость происходит по атрибуту $attr_i$, если v либо создаёт атрибут $attr_i$, либо в его контексте происходит системный вызов, выставляющий данный атрибут в u. Для анализа зависимостей вводится новое промежуточное представление - граф реконструкции, дополненный зависимостями $G^{dep}(T)$ - мультиорграф:

$$G^{dep}(T) = (V^+, E^{dep} \cup E^+) \tag{5}$$

где E^{dep} - множество зависимостей между состояниями процессов как вершинами.

Уточнение. E^{dep} содержит по меньшей мере множество рёбер транзитивного сокращения[3] всех зависимостей на V^+. При необходимости, все зависимости можно получить тразнизивным замыканием E^{dep}.

Зависимости определяют частичный порядок на V^+. Для получения важных свойств V^+ как частично-упорядоченного множества вводится ряд определений, следующих из технических свойств исследуемой структуры данных: отношение доминирования между атрибутами, образование атрибутами иерархий и др.

[3]Существование и единственность транзитивного сокращения следует из конечности графа и ацикличности, доказываемой в теореме 4.

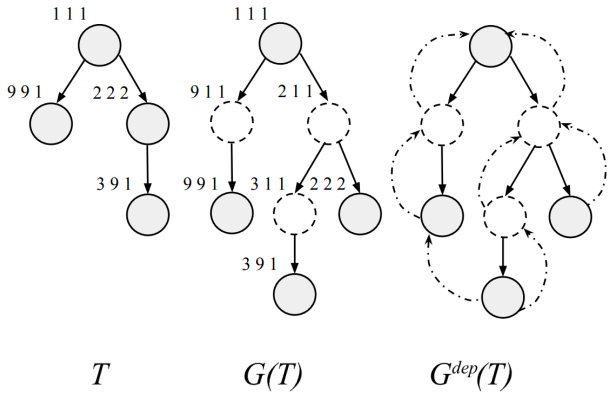


Рис. 3 — Входное дерево процессов T, граф реконструкции $G(T)$ и граф реконструкции с зависимостями $G^{dep}(T)$.

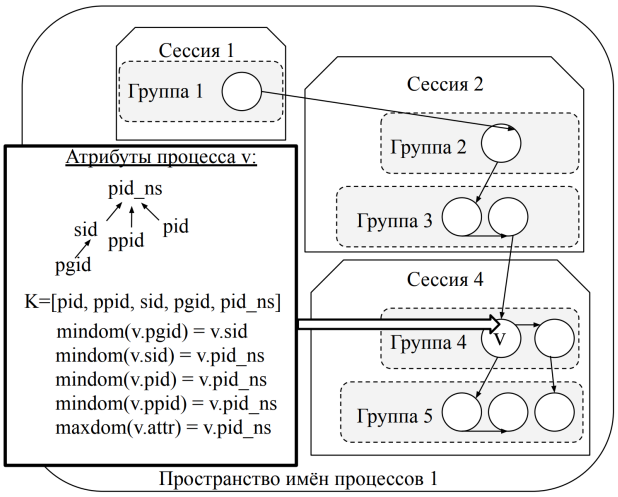


Рис. 4 — Иерархии атрибутов процессов для идентификаторов процессов, сессий, групп процессов в едином пространстве имён.

Определение 2. *Будем говорить, что атрибут* $attr_2$ *доминирует над* $attr_1$*, если* $\forall v,u \in V^+ : u.attr_1 = v.attr_1 \Rightarrow u.attr_2 = v.attr_2$.

Определение 3. *Доминирующий над* $attr_1$ *атрибут* $attr_2$ *назовём минимально доминирующим, и определим отображения* $mindom : K \to K$ *и* $Mindom : \mathbb{N} \cup \{0\} \to \mathbb{N} \cup \{0\}$*, такие, что* $mindom(attr_1) = attr_2, Mindom(u.attr_1) = u.attr_2$*, 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* $\forall v' \in V^+ \setminus \{v\} \Rightarrow v'.attr_2 \neq const$.

Отношение минимального доминирования задаёт множество минимальной мощности, включающее множество состояний с различными значениями атрибута $attr_1$, но с идентичными $attr_2$.

Вводится целочисленная оценка количества атрибутов, подлежащих реконструкции перед реконструкцией текущего – глубина изоляции:

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

К примеру, перед реконструкцией группы процессов следует создать сессию, в которой эта группа находится.

Очевидно, для любого атрибута глубина изоляции конечна, в противном случае для реализации состояния с таким атрибутом пришлось бы выполнить бесконечное число системных вызовов.

Определение 4. *Максимальным по мощности доминирующим атрибутом называется атрибут нулевой глубины изоляции:* $maxdom(attr) = attr_x : \forall v \in V^+ \Rightarrow depth(v.attr_x) = 0$.

Для деревьев процессов $maxdom(attr)$ соответствует корневому пространству имён процессов: построение дерева начинается с корневого процесса v_{init}, который находится в корневом пространстве имён процессов, и первые вызовы, создающие вложенные пространства, происходят в этом пространстве.

Пользуясь введённым понятием иерархии атрибутов, легко доказать, что введение V^+ как верхней полурешётки соответствует семантике восстановления дерева процессов.

Теорема 4. V^+ *с отношением зависимости образует полную верхнюю полурешетку.*

На полученной полурешётке вводятся операторы замыкания CL и предзамыкания PCL, экстенсивность и монотонность которых следует из свойств введённого частичного порядка на V^+. Операторы введены с целью получения состояний из V^+, соответствующих созданию или получению доступа процессов к ресурсам системы, а оператор замыкания $CL(v.attr_i)$ служит для перевода состояния v с некоторым атрибутом $attr_i$ в состояние, в котором был создан минимально доминирующий атрибут $mindom(attr_i)$. Существование и единственность неподвижной точки $CL(v.attr_i)$ доказывается из технически аргументированного требования к эксклюзивному присутствию создателя: появления в некоторый момент лидера сессии, корня пространства имён и др.

В разделе также исследуется применение механизма построения замыканий для консистентного восстановления подмножеств процессов всего дерева T.

Можно формально переопределить необходимое и достаточное условие корректности дерева процессов, представленное во **второй главе**.

Теорема 5. *Если T - корректное дерево процессов, то множество его вершин V дополнимо до верхне полурешёточно упорядоченного по зависимостям V^+ : $\big(\forall x,y \in V^+ \; \exists k : \forall n > k, n \leq |K| : attr_k,attr_n \in K, depth(attr_n) < |K| - k\big)$ & $\big(x.attr_n = y.attr_n\big) \Rightarrow \big(x \sqcup y = PCL(x.attr_k)|PCL(y.attr_k)|CL(x.attr_k)\big)$ & $\big(CL(x.attr_k) = CL(y.attr_k)\big)$, где CL,PCL - введённые выше операторы замыкания и предзамыкания на V^+, и единственной неподвижной точкой CL является создатель минимально доминирующего атрибута для $attr_k$.*

Для получения достаточного условия, требуется ввести классификацию PCL. Какой семантикой обладают образы, получаемые ими?

Предложение 4. *Для реконструкции состояния u : $u.attr_1 = w.attr_1$ с атрибутом $attr_1$, созданным в состоянии w_{new} и "хранимым" в состоянии w, из предшествующего состояния v : $u.pid = v.pid|(u.pid \neq v.pid$ & $syscall =' fork')$ системным вызовом, выполненным в контексте процесса с состоянием c, достаточно выполнения: $c.attr_2 = v.attr_2 = w_{new}.attr_2 = u.attr_2$, и оператор $CL(w.attr_1) = CL(c.attr_1) = CL(u.attr_1)$ возвращает состояние с минимальным доминирующим атрибутом $attr_2$ атрибута $attr_1$.*

Зависимости, разрешаемые в ходе построения G^{dep}, разделяются на 4 типа, и определяются упомянутыми выше операторами:

1. Зависимость от вершины-предшественника, $PCL(u.attr_i)_{pred}$.
2. Зависимость от вершины-создателя $attr_i$, $PCL(u.attr_i)_{creator}$
3. Зависимость от вершины-исполнителя системного вызова, добавившего состояние с атрибутом $attr_i$, $PCL(u.attr_i)_{syscaller}$
4. Зависимость от минимально доминирующего атрибута для $attr_i$, $CL(u.attr_i)$: все образы, определяемые PCL, лежат в замыкании, определяемом CL.

Легко показать, что 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 с частичным порядком, наследованным из V^+ (Рис. 5а).

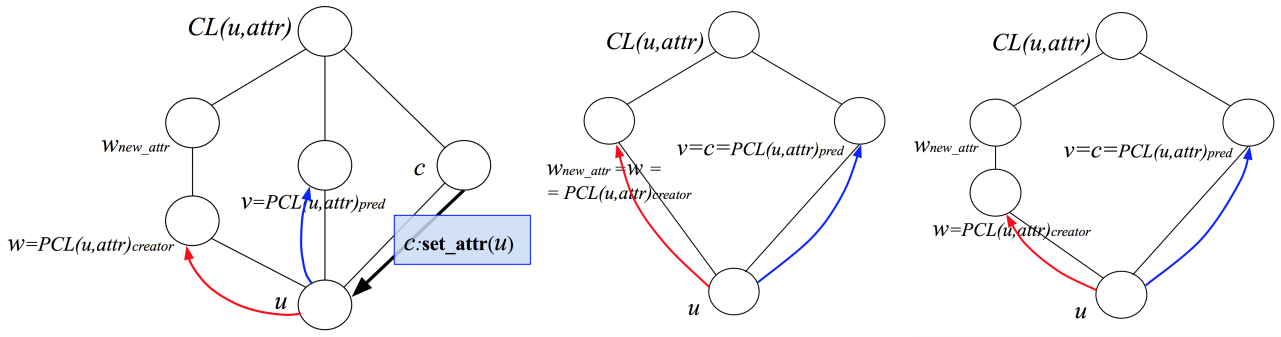
Опишем данную решётку системой неравенств:

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

Определение 5. *Назовём* $\{v,c,w,w_{new},CL(u.attr_i)\}$ *генерирующим множеством для* $u.attr_i$*:* $Gen(u.attr_i)$.

Теорема 6. *Пусть* V^+ *- множество вершин графа* G^{dep}*, проверяемого на соответствие графу реконструкции с зависимостями, и два множества* $V = \{v \in V^+ : v \neq x \sqcup y, \forall x,y \in V^+ \setminus \{v\} \ \& \ y.pid = v.pid\}, E = \{(x,y) : x,y \in V \ \& \ y.ppid = x.pid\}$ *– непустые. Пусть* $\forall u \in V^+ \setminus \{v_{init}\} \ \forall attr_i \in K \ \exists (Gen(u.attr_i) \subseteq V^+) \mid (Gen(u.attr_i) = \emptyset \iff u.attr_i = none)$*. Тогда* (V,E) *– корректное дерево процессов, а* G^{dep} *– граф реконструкции* (V,E) *с зависимостями.*

Раздел 3.2 рассматривает связь стадии пост-обработки и полурешёточной упорядоченности V^+, представляя исправление аномалий в графах реконструкции как преобразование графа в граф полурешётки (Рис. 5).



а) б) в)

Рис. 5 — Общая схема реконструкции состояния u с атрибутом "*attr*" по генерирующему множеству (а), фактическая схема из алгоритма до стадии пост-обработки (б) и вариант исправления в пост-обработке (в).

На основе требования к существованию нужного генерирующего множества *Gen* для каждого атрибута каждого процесса, предложено обобщение

алгоритма атрибутной реконструкции из **второй главы** на любую древесную иерархию атрибутов:

Предложение 5. *Будем считать, что порядок реконструкции атрибутов важен тогда и только тогда, атрибуты подчинены отношению доминирования. То есть в ходе работы алгоритма будет построен такой граф реконструкции, что, при выполнении команд из него, нужное значение атрибута $attr_i$ для любого процесса будет выставлено не ранее, чем будет выставлено значение $attr_j$ для этого же процесса. Если же атрибуты несравнимы по иерархии, что, очевидно, равносильно тому, что $\exists x,y \in \mathbb{N} : \forall m \geq x, n \geq y$, такие, что, если n и m-кратными применениями оператора $PCL(...)_{creator}$ получены создатели данных атрибутов как неподвижные точки: $p_1 = PCL^m(u.attr_i)_{creator} = PCL^{m+1}(u.attr_i)_{creator}, p_2 = PCL^n(u.attr_i)_{creator} = PCL^{n+1}(u.attr_i)_{creator}$, и $p_1 \sqcup p_2 = v : v.pid \neq u.pid$, то порядок реконструкции таких атрибутов не важен для любого процесса.*

Предложение 6. *Будем считать, что синтаксис и семантика системных вызовов, устанавливающих атрибуты SI и HI, а также правила разделения этих атрибутов между процессами аналогичны таковым для атрибутов группы процессов и сессии, с точностью до переименования вызова и атрибута.*

Вообще говоря, конструирование самой иерархии атрибутов должно восходить к семантике работы с конкретными ресурсами. Иерархия должна определяться:

1. Атрибутами процессов.
2. Синтаксисом и семантикой системных вызовов. Будем считать, что синтаксис и семантика однозначно определяются типом атрибута.

Соответственно, считаем, что входных данных задачи достаточно для извлечения однозначной иерархии атрибутов. Вводим описание иерархии формально. Пусть по K можно построить однозначно словарь типов атрибутов K_t вида $\{k,t\}$, где $k \in K, t \in \{HI,SI\}$, и пусть для изменения, создания, чтения либо удаления каждого атрибута из K существует системный вызов $l \in L$, синтаксис и семантика которого однозначно определяется типом атрибута.

Тогда H-корневое дерево, задающее иерархию атрибутов.

$$\begin{gathered}\text{H}=(\text{K}_t,DR) : ((\exists!(u,v) \in DR) \Leftrightarrow (u = mindom(v))) \ \ \& \\ (\forall w \in K_t \ \exists! root = maxdom(w)\end{gathered} \tag{6}$$

Очевидно, перечисление атрибутов, согласованно с вложениями состояний процессов в соответсвующие замыкания, можно организовать, выполняя перечисление вершин H: можно полагать на каждом шаге некоторую вершину v - описанием минимально доминирующего атрибута атрибутов, являющихся непросредственными потомками данной вершины в H - $v.children$. То есть на каждом шаге перечисления v для любой вершины $node \in V^+, \forall attr_i \in v.children$ будет справедливо, что $CL(node.attr_i) = creator$, где $creator$ - создатель минимально доминирующего атрибута. Следовательно, каждому шагу перечисления соответствует ситуация, аналогичная алгоритму для групп и сессий: имеем атрибут и минимально доминирующий атрибут, нужно произвести их совместную реконструкцию. Но произведено обобщение на конечное число HI и SI атрибутов. Так как типы атрибутов известны, и предполагается синтаксис и семантика системных вызовов, определяемые типом атрибута, можно обобщить результаты, полученые для групп процессов, на атрибуты типа SI, а результаты для сессий - на HI. При этом при построении графа можно применять те же правила трансформации, что и ранее, в виду схожести свойств системных вызовов. Также доказано, что анализ и трансформации для MI-атрибутов можно свести к анализу и трансформациям, сходным с HI-атрибутами, а также что для F-атрибутов анализ и трансформации сходны с простейшим случаем для SI-атрибутов.

Предлагается сведение обобщённой реконструкции к пошаговому использованию соответствующих процедур алгоритма. Для HI-атрибутов будут использоваться пред-обработка и восстановление сессии, для SI–постобработка и восстановление группы. Введём процедуру переименования атрибутов и системных вызовов $renameAttrs(G,K,L,rA = oldattr : attr,rS = oldcall : syscall)$, заменяющую ключ атрибута и системный вызов для каждой пары ключ:значение любых вхождений соответственно G,K и L. Считаем, что переименование - относительно быстрая операция, проводимая перечислением вершин G и записей K и L за $O(|V^+| + |K| + |L|)$. Прямые и обратные переименования атрибутов сессии или группы на нужное имя атрибута про-

изводятся до и после вызова процедур из Алгоритма 1, то есть в промежутке между ними подзадача сводится к его соотвествующей подзадаче, что можно сделать согласно Предложению 6.

В ходе работы каждой стадии Алгоритма 1 обработанные вершины помечаются, поэтому ситуация, когда для нескольких вершин вставляются идентичные команды, либо все вершины уже отмечены, может быть исключена процедурой выборочной "реактивацией" вершин, временная сложность которой $O(|V^+|)$: так как правилами трансформации обеспечивается, что на каждом этапе множество восстановленных состояний каждого процесса $\{u\}_k \in V^+ : u.pid = k$ образует цепь, будем снимать метку $handled$ для каждого элемента цепи, не имеющего исходящего ребра в $\{u\}_k \cap (E^{follow} \cup E^{syscalls}) : \forall u_{last} \in \{u\}_k : out(u_{last})_{\{u\}_k} = 0 \Rightarrow u_{last}.handled = 0$.

Аналогично Алгоритму 1 **главы 2**, разобъем реконструкцию на 3 стадии:

1. Пред-обработка: формируется иерархия H производится обход H, и, для каждого потомка текущего атрибута в H, если его тип HI, выполняется сведение к пред-обработке из Алгоритма 1 **главы 2**, решение, обратная конвертация к исходным именам атрибутов.
2. Обработка: $genProcess$: производится обход H, и, для каждого потомка текущего атрибута в H, в соответствии с типом атрибута выполняется сведение к соответствующей подзадаче, решение подзадачи, обратная конвертация и реактивация вершин для следующего шага обработки. Переименование при возврате из подзадачи $fork$ в h нужно для технического обеспечения обобщения подзадачи: в противном случае в графе будут выставлены рёбра с $fork$, ведущие в вершины, для которых анализ ещё не закончен. Переименование в h, с учётом предполагаемой неразличимости кратных сонаправленных рёбер с одинаковой меткой, ликвидирует этот нюанс. Обратное переименование в конце процедуры переводит все h в $fork$, то есть возвращает нужные метки нужным соответстующим рёбрам.
3. Пост-обработка: производится обход H, и, для каждого потомка текущего атрибута в H, если его тип SI, выполняется пост-обработка

из Алгоритма 1, решение, обратная конвертация к исходным именам атрибутов.

Прямые и обратные переименования атрибутов сессии на нужное имя атрибута производятся до и после вызова процедур из Алгоритма 1, то есть в промежутке между ними подзадача сведена к его соотвествующей подзадаче.

Далее проведён анализ роста множеств состояний V^+ относительно V для различных типов атрибутов, позволяющие получить временные оценки сложности от количества вершин $|V|$ во входном дереве процессов при обобщённой реконструкции. На основании анализа сформулированы и доказаны 3 леммы.

Лемма 3. *Для любого $G(T) = (V^+,E^+)$, полученного алгоритмом из корректного дерева процессов $T = (V,E)$ с атрибутами $K = \{pid,sid,pgid\}$, справедлива оценка $\frac{|V^+|}{|V|} \leq O(1)$.*

Лемма 4. *Для любого $G(T) = (V^+,E^+)$, полученного алгоритмом из корректного дерева процессов $T = (V,E)$ с атрибутами K, K не содержит мягко-наследуемых атрибутов, справедлива оценка $\frac{|V^+|}{|V|} \leq O(|K|)$.*

Лемма 5. *Для любого $G(T) = (V^+,E^+)$, полученного алгоритмом из корректного дерева процессов $T = (V,E)$ с атрибутами K, справедлива оценка $\frac{|V^+|}{|V|} \leq O(|K|)$.*

Лемма 5 заключает эффект фактического рассмотрения мягко-наследуемых атрибутов как жестко-наследуемых, но уникально именованных. Например, переоткрытие файла на дескрипторе с тем же номером формально будет считаться в такой модели созданием нового атрибута. Количество различных атрибутов при этом может быть довольно большим. К примеру, в случае восстановления файловых дескрипторов и отображений регионов виртуальной памяти непосредственно в дереве процессов, можно столкнуться с ограничениями, настраиваемыми через системный интерфейс `sysctl`, например, `sysctl fs.file-max` и `sysctl vm.max_map_count` на Ubuntu 16.04 с mainline-ядром версии 4.10 по умолчанию равны 1024 и 65530 соответственно. Сумма этих значений больше потенциального числа процессов, равного $2^{16} - 1$. Вероятно, предпочтительнее реализовать обработку подобных ресурсов уже после восстановления дерева процессов, как это и реализовано в современных системах восстановления по контрольным точкам. Альтернативой

может служить рассмотрение таких ресурсов как атрибутов свободного типа, для которых потеря наследования на одном из состояний исправляется повторной установкой значения системным вызовом в соответствующем замыкании. С другой стороны, описание древесной иерархией любых атрибутов имеет ряд недостатков, связанных, в первую очередь, с несравнимостью атрибутов на одном уровне иерархии. К примеру, пусть стоит задача восстановить в рамках одного процесса набор потоков и мьютексов, на которых они синхронизируются. Восстановить потоки в рамках построенной модели не составит труда: в процессе есть главный поток, идентификатор tid которого равен номеру группы потоков $tgid$. Остальные потоки жестко наследуют этот номер, что и определяет реконструкцию потоков как состояний с HI – атрибутом группы потоков в замыкании одного pid. Но блокировки в такой модели являются несравнимыми: любой поток может захватить блокировку, если она открыта – порядок не определён, что может вызвать взаимоблокировки. В таком случае модель требует внесения уточнений, к примеру, явным добавлением зависимостей типа "конфликт доступа к ресурсу" .

Следствие 1. *Временная сложность алгоритма обобщённой реконструкции от размера входа составляет в худшем случае* $O(|V|^2|K|^3)$ *по времени и* $O(|V||K|)$ *по памяти.*

Действительно, по лемме 5 $O(|V^+|) = O(|K||V|)$, следовательно, подзадачи, решаемые за квадрат от размера входа, в виду вложенных обходов всех вершин, требуют порядка $O(|K|^2|V|^2)$, и решение подзадач происходит для $|K|$ различных атрибутов. Данный результат можно существенно улучшить, обрабатывая в подзадачах только реактивированные вершины, тогда можно получить сложность порядка $O(|K||V|^2)$ для каждой из $|K|$ подзадач. Отнесём данную модификацию к возможным улучшениям алгоритма. Оценка по памяти в любом случае составит $O(|K||V|)$, так как конструируемый граф хранится полностью.

В главе также производится рассматривается подзадача восстановления подмножества процессов, не изолированных в отдельное пространство имён. Такое подмножество соответсвует некоторому подграфу дерева процессов, и реконструкция сводится к построению замыкания по всем минимально доминирующим атрибутам, которые отличаются на данном подмножестве процессов, плюс требуется проверить,

возможна ли реконструкция выполнениями системных вызовов только из данных процессов, либо добавленных, процессов. Временная сложность в худшем случае составит $O(|H^-|max(|H'|)max(|subtree(T',rt)|^2)$, где $|H^-|, max(|H'|)$,$max(|subtree(T',rt)|^2)$, соотвественно число деревьев в лесе, корнями которого являются создатели вышеуказанных минимально доминирующих атрибутов, максимальное число атрибутов в иерархии и размер максимального по мощности замыкания, построенного из поддерева процессов, растущего из корня, являющегося неподвижной точкой соответствующего построенному замыканию оператора. $max(|subtree(T',rt)|^2)$ может быть порядка $O(V^+)$, то есть сложность может быть порядка $O(|H^-|max(|H'|)|V^+|^2)$. При этом нет гарантий успешной реконструкции без проверки на возможность выполнения вызовов, поэтому рекомендованным решением является всё же изоляция процессов в отдельный контейнер, либо хотя бы в сессию, группу.

В **четвертой главе** приведено описание программного комплекса, разработанного для выполнения экспериментов по исследованию графов реконструкции. Комплекс позволяет производить снятие дампов деревьев процессов, загрузку дампов из файлов, анализ исследуемыми в работе способами, генерацию команд, воспроизведение синтетических нагрузок в изолированном окружении – отдельном пространстве имён процессов, трассировку системных вызовов, сохранение результатов и визуализацию данных. Схема комплекса приведена на (Рис. 6). В **разделе 4.2** приводятся результаты экспериментального исследования, заключавшегося в сравнении времени построения графа реконструкции и извлечения из него списка команд с временем, затрачиваемым на трассировку построения идентичного реконструируемому дерева процессов посредством профилировщиков perf и strace. Использовались деревья из двух типов синтетических тестов, обеспечивающих фиксированное число промежуточных состояний при построении деревьев. Соответственно, Тест 1 – "simple load" – имитировал случай с реконструкцией групп процессов, что соответсвует восстановлению пользовательского сеанса, а Тест 2 – "context load" – реконструкцию групп и сессий, что соответствует сложной конфигурации дерева процессов: работе операционной системы в целом, либо контейнера с произвольными гостевыми приложениями. В обо-

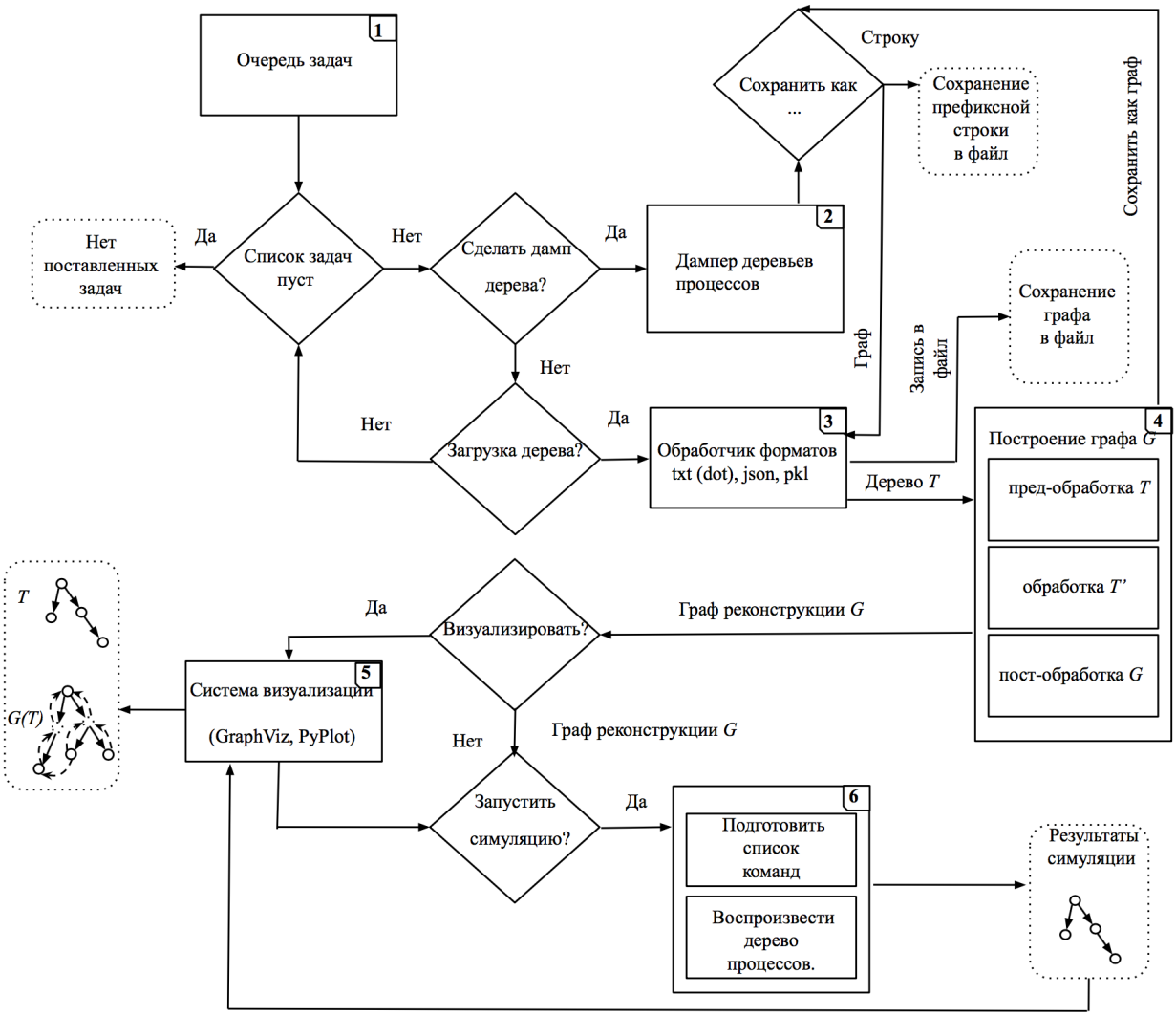


Рис. 6 — Схема программного комплекса.

их тестах также содержалась фиксированная доля процессов, завершаемых ранее своих потомков.

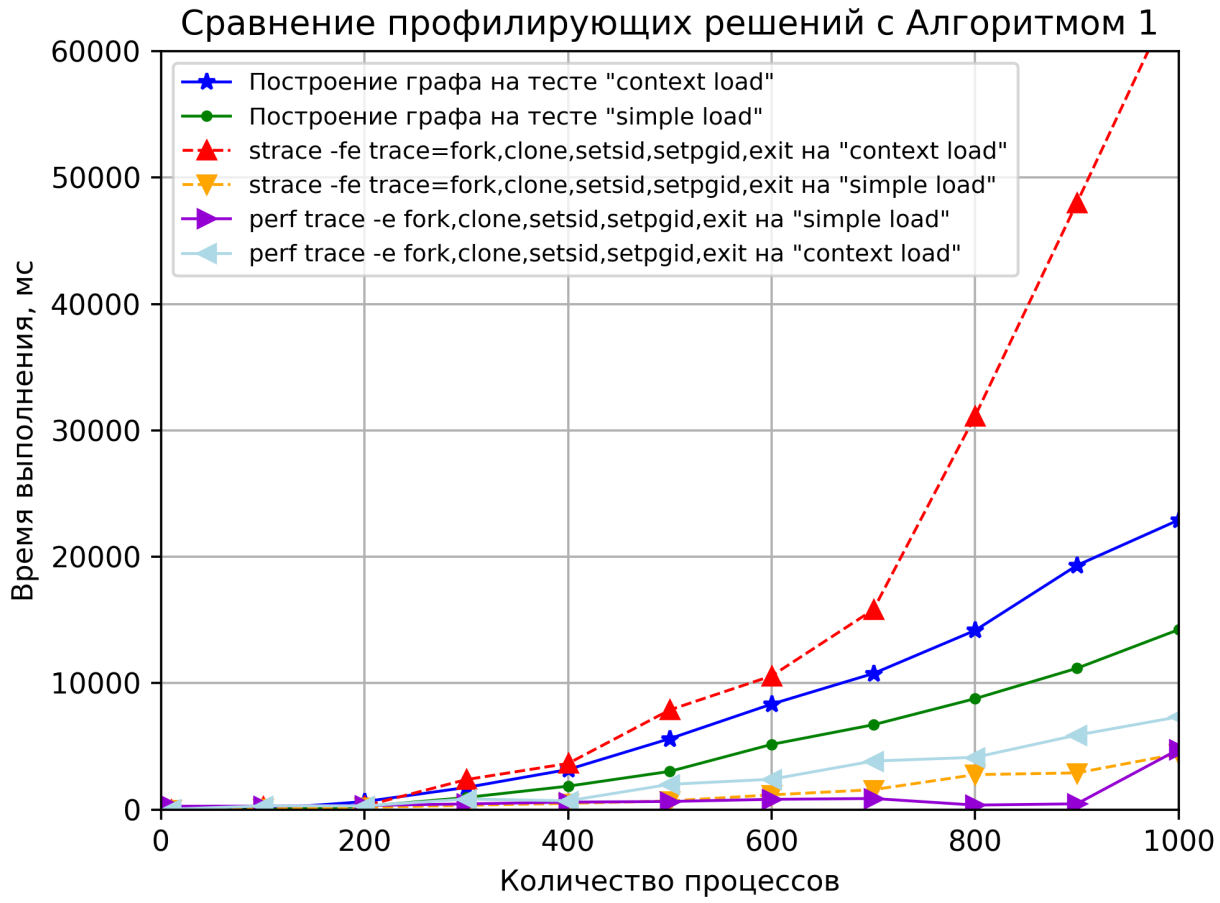


Рис. 7 — Результаты эксперимента.

Из результатов эксперимента можно заключить, что временные затраты на построение графа реконструкции сопоставимы с затратами на трассировку посредством strace. Для небольшого числа процессов, в пределах 100, алгоритм демонстрирует сопоставимые и даже немного лучшие результаты в сравнении с профилировщиками. Это говорит о практической эффективности использования разработанного алгоритма для восстановления контейнеров или изолированных в отдельных пространствах имён процессов. На большем числе процессов, от 100 до 200-300, в зависимости от нагруженности теста, результаты несколько хуже, но сопоставимы с профилировщиками по порядку величины. Для большого числа процессов различие между perf и остальными способами более существенно, но для strace и построенного алгоритма сохраняется вышеупомянутое отношение: для обоих тестов результаты

алгоритма лежат между результатами strace: трассировка strace занимает существенное время на большом числе процессов для Теста 2.

В **заключении** приведены основные результаты работы, которые заключаются в следующем:

1. На основе анализа деревьев процессов был выявлен ряд свойств, позволяющих производить восстановление сессий, групп процессов и производить обратное упорядочивание за $O(|V|^2)$ по времени и линейно по памяти. При этом для корректных деревьев гарантируется корректность реконструкции.
2. На основе анализа межатрибутных зависимостей был построен формализм, позволяющий обобщить результаты анализа на любую древесную иерархию атрибутов, корректное восстановление которой выполняется за $O(|K|^3|V|^2)$ по времени и $O(|K||V|)$ по памяти, где $|K|$ - число атрибутов. Внесено замечание, как можно улучшить данный результат до $O(|K|^2|V|^2)$ по времени. Тем не менее, применение такого способа восстановления к любым атрибутам ресурсов может быть затруднительным, в связи с несравнимостью атрибутов на одном уровне иерархии, что отмечено в заключении Главы 3. Построение полиномиального алгоритма, учитывающего, к примеру, восстановление файловых блокировок, средств синхронизации и пространств имён монтирования может быть рекомендован к дальнейшему исследованию как поправки к предложенной модели.
3. Полученный формальный критерий корректности дерева процессов может служить серьёзным подспорьем для валидации деревьев и генерации синтетических тестов для систем сохранения и восстановления. Тем не менее, вопрос поиска эффективного метода валидации по полученному критерию является открытым.
4. Проведённые эксперименты находятся в соответствии с теорией и демонстрируют сопоставимость времени атрибутного анализа деревьев процессов, даже без различных оптимизаций, с временными расходами на трассировку системных вызовов, что, с учётом технических недостатков профилирующих решений, делает атрибутный анализ более предпочтительным.

Публикации автора 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. — DOI: 10.22405/2226-8383-2018-19-2-151-162.
4. *Ефанов Н.* О полурешётке 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. — DOI: 10.1145/3290621.3290626. — URL: http://doi.acm.org/10.1145/3290621.3290626.
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. — DOI: 10.1145/3166094.3166106. — URL: http://doi.acm.org/10.1145/3166094.3166106.
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.* Атрибутная 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 состояний 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, Май.2019. — С. 165—168.

15. *Ефанов Н. Н.* О формальной корректности атрибутного 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 // Информационное обеспечение математических моделей. — М., 2017. — С. 83—90.