ЧЕБЫШЕВСКИЙ СБОРНИК
Том 20. Выпуск 4.

УДК 519.[172-178] DOI 10.22405/2226-8383-2019-20-4-124-136

О полурешётке состояний процессов Linux

Н. Н. Ефанов

Ефанов Николай Николаевич — аспирант, ассистент/преподаватель кафедры информатики и вычислительной математики, Московский физико-технический институт (Национальный исследовательский университет) (г. Москва).
e-mail: nefanov90@gmail.com

Аннотация

В работе рассматривается структура данных - множество состояний процессов Linux, которая используется в задаче восстановления дерева процессов в Unix-подобных операционных системах. Целью исследования является анализ зависимостей в такой структуре, введение естественного порядка по зависимостям, предложение и обоснование возможности его введения как верхней полной полурешётки. Следующие из технических свойств прикладной задачи иерархии атрибутов позводяют ввести дополнительные ограничения на минимальные верхние границы в полурешётке. Ограничения формально описываются в виде подходящих операторов предзамыкания и замыкания. Из ограничений следует необходимое условие корректности дерева процессов. На основании свойств точек, возвращаемых введёнными операторами и схемы выполнения системного вызова, приводится достаточное условие корректности: для каждого атрибута, возникающего в контексте процесса, должно существовать решёточно упорядоченное относительно наследуемого порядка множество, содержащее промежуточные состояния процессов, через которые и разрешаются зависимости. Введённые условия формируют критерий корректности дерева процессов, что может быть полезно в таких задачах как генерация тестов для систем сохранения и восстановления состояний Unix-подобных ОС, поиск аномалий, повышение портабельности и надёжности программных комплексов. Приводятся также схемы зависимостей между атрибутами, которые вводят частные ограничения на реконструирующее множество. Рассматриваются открытые вопросы и предлагаются дальнейшие шаги.

Ключевые слова: математическое моделирование, прикладная алгебра, полугруппы, деревья, Unix-подобные операционные системы, системные вызовы, дерево процессов, восстановление по контрольным точкам, замыкания.

Библиография: 20 названий.

Для цитирования:
Н. Н. Ефанов. О полурешётке состояний процессов Linux // Чебышевский сборник, 2019, т. 20, вып. 4, с. 124–136.

CHEBYSHEVSKII SBORNIK
Vol. 20. No. 4.

UDC 519.[172-178] DOI 10.22405/2226-8383-2019-20-4-124-136

On semilattice of Linux processes' states

N. N. Efanov

Efanov Nikolay Nikolaevich — PhD Student Teaching assistant, Department of Informatics and Computational Mathematics, MIPT. Moscow Institute of Physics and Technology (National Research University) (Moscow).
e-mail: nefanov90@gmail.com

Abstract

The paper discusses a set of states of Linux processes as data structure, which is used in the task of process-tree reconstruction in Unix-like operating systems. The purpose of the study is to analyze dependencies in such structure, to introduce the natural order of dependencies, to propose the class of such reconstruction structure as upper complete semilattice. Following from the technical properties of the applied problem attributes' hierarchy allow to introduce additional restrictions on the minimum upper bounds in such semilattice.

Constraints are formally described as suitable pre-closure and closure operators. The constraints implies the necessary condition for the correctness of the process tree. Based on the properties of points returned by the proposed operators and system call execution scheme, a sufficient condition for correctness is given. The introduced conditions form the criterion for process-tree correctness, which can be useful in such tasks as generating tests for checkpoint-restore in Unix-like operating systems, anomalies detection, increasing portability and reliability of software. Dependency schemes between attributes that impose particular constraints on the reconstructing set are also shown. Opened questions are also highlighted and further steps are suggested.

Keywords: mathematical modeling, applied algebra, semi-groups, trees, Unix-like operating systems, system calls, process tree, checkpoint restore, closures.

Bibliography: 20 titles.

For citation:
N. N. Efanov, 2019, "On semilattice of Linux processes' states" , *Chebyshevskii sbornik*, vol. 20, no. 4, pp. 124–136.

1. Введение

Возможность сохранения и восстановления состояний процессов операционной системы (ОС) является важнейшей составляющей ряда системных, виртуализационных и прикладных программных технологий [1, 2, 3, 4, 5, 6], представляет основу многих технологий отложенной отладки программ, симуляции исполнения, обновления компонентов системы без перезагрузки, ускорения запуска программ, защиты от сбоев [2, 3]. В Unix-подобных ОС процессы объединяются в ориентированное дерево – дерево процессов [1, 4, 5, 6, 7]. Воспроизведение дерева процессов, соответствующего некоторому дереву, полученному при сохранении состояния, является важнейшей частью процедуры восстановления, а техника восстановления последовательностями системных вызовов следует из особенностей управления ресурсами системы из пользовательских программ. Такая техника применяется в случае восстановления состояний из пространства пользователя [2, 4, 5].

С целью построения гибкой процедуры реконструкции, разработаны различные математические модели, такие как анализ графа ресурсов системы [6], последовательная трансформация семантическими действиями входного дерева процессов как абстрактного синтаксического дерева в атрибутной грамматике [5], а также неявный атрибутный анализ результата LR - разбора строчной записи дерева, в которой иерархия кодируется правильными скобочными последовательностями [4]. В работах вышеперечисленных работах изучены свойства, получены полиномиальные алгоритмы восстановления некоторых типов деревьев процессов. Тем не менее, строгое исследование зависимостей между состояниями в деревьях, ровно как и разработка формальных методов проверки деревьев на корректность, являются открытыми вопросами. В работе представляются некоторые результаты исследования зависимостей между состояниями процессов, на основании изучения свойств графа восстановления - промежуточного представления, введённого в [5] (Рис. 1).

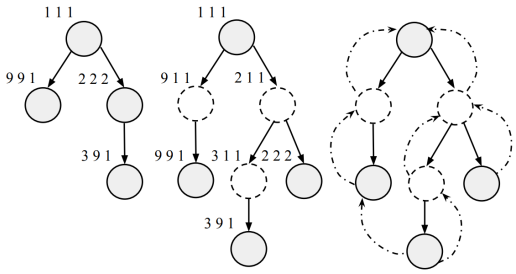

Рис. 1: Исходное и промежуточные представления (слева направо): дерево процессов T, некоторый его граф восстановления $G(T)$ и дополненный зависимостями (мульти)граф $G(T) \cup DEP(T)$. Числа при вершинах – атрибуты-идентификаторы процесса P, группы G и сессии S соответственно. Корневой процесс всегда имеет идентификаторы “1,1,1”.

2. Промежуточные представления и их анализ

Рассматриваются два орграфа - дерево процессов

$$T = (V, E) \tag{1}$$

и граф восстановления - промежуточное представление, на базе анализа которого производится восстановление дерева в работе [4]:

$$G(T) = (V^+, E^+) \tag{2}$$

где $V^+ = V \cup V^{int}$ - конечные вершины, дополненные промежуточными V^{int}, соответствуют состояниям процессов в ходе выполнения системных вызовов, описываемых E^+ как переходами между состояниями и иерархическими зависимостями [4, 5]. Вершины маркированы словарями атрибутов: $\forall v \in V^+ \exists v.attr : v.key = val(v.attr[key]) | 'None', \forall key \in K$, где K - список ключей как имён атрибутов процесса: системных идентификаторов, файловых дескрипторов, сигналов, отображений памяти и др, $val(v.attr[key]) : K \to val\_with\_type(key)$ - отображение ключей на типизированные значения атрибутов. Определяется также множество вершин с равным значением атрибута $u.attr_1$:$D(attr_1, u.attr_1) = \{v \in V^+ | v.attr_1 = u.attr_1\}$

Definition 1. *Определим зависимость между вершинами $u, v \in V^+$ как бинарное отношение (u, v) с семантикой "для реализации и требуется сначала реализовать v". Следовательно, зависимости определяют частичный порядок на V^+. Будем говорить, что зависимость происходит по атрибуту attr, если v либо создаёт атрибут attr, либо в его контексте происходит системный вызов, выставляющий данный атрибут в v.*

Используя данное определение, рассморим ещё одно промежуточное представление:

Definition 2. *Графом зависимостей $DEP(T)$ называется мультиорграф:*

$$DEP(T) = (V^+, E^{dep}) \tag{3}$$

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

Definition 3. *Будем говорить, что атрибут $attr_2$ доминирует над $attr_1$, если*

$$\forall u \in V^+ : \forall v \in D(attr_1, u.attr_1) \to val(u.attr_2) = const.$$

Definition 4. *Доминирующий над $attr_1$ атрибут $attr_2$ назовём минимально доминирующим,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 $\forall v' \in V^+ \setminus \{v\} \to v'.attr_2 \neq const$.*

Такое определение задаёт минимальное по мощности множество, в которое включается и множество состояний с различными значениями атрибута $attr_1$, но с идентичными $attr_2$, и его замыкание.

Definition 5. *Глубиной изоляции атрибута $attr_1$ называется число доминирующих над $attr_1$ атрибутов: $depth(attr_1) = \sum_{i=2}^{|K|} attr_i : \forall u \in V^+ : \forall v \in D(attr_1, val(u.attr_1)) \to val(u.attr_i) = const$.*

$\forall attr_{key} \in K \to depth(attr_{key}) < \infty$, в противном случае, для реализации состояния с таким атрибутом пришлось бы выполнить бесконечное число системных вызовов, так как каждый новый атрибут создаётся через отдельный вызов.

Definition 6. *Максимальным по мощности доминирующим атрибутом называется $attr_x : \forall v \in V^+ \to depth(v.attr_x) = 0$.*

Для деревьев процессов такой $attr_x$ соответствует корневому PID-пространству имён [7], ввиду того, что построение дерева процессов начинается с корневого процесса v_{init}, который находится в корневом PID-пространстве, и первые вызовы, создающие вложенные пространства, происходят именно в нём.
Свойства объектов, введённых в определениях 5 и 6, следуют из технических особенностей прикладной задачи, поэтому принимаются без доказательства.
Структурные свойства V^+, связанные с частичной упорядоченностью, ранее формально не исследовались. Покажем, что введение V^+ как верхней полурешётки [8] соответствует семантике восстановления дерева процессов.

Утверждение 1. *V^+ с отношением зависимости образует конечную верхнюю полурешетку.*

Доказательство. Выберем произвольные состояния $x, y, z, \in V^+ : x \neq y$. Рассмотрим бинарную операцию минимальной верхней границы $\sqcup$ на $V^+ : x \sqcup x = x; x \sqcup y = y \iff (x, y) \in E^{dep}$. $\sqcup$ по определению идемпотентна. Докажем, что $\forall x, y \in V^+ \exists z \in V^+ : (x, z) \in E^{dep}, (y, z) \in E^{dep}$, доказывая параллельно коммутативность и ассоциативность $\sqcup$. Рассмотрим ситуации:

1. $(x, y) \in E^{dep}$ - тогда, очевидно, $c = y$. Случай (y, x) симметричен.
2. Покажем, что $\{(x, y), (y, x)\} \subset E^{dep} \iff x = y$.
   - Допустим, что $\exists x, y \in V^+ : \{(x, y), (y, x)\} \subset E^{dep}$, и $x \neq y$. о определению зависимости, для реализации x нужно реализовать сначала y, но для этого нужно реализовать состояние, идентичное x. То есть на данном шаге нужно выполнить минимум 2 системных вызова, и требуется снова воспроизвести x.

- Пусть выполнено $k-1$ итераций воспроизведения x,y, тогда нужно снова воспроизвести x, и выполнено $2(k-1)$ системных вызовов.
- После шага k требуется снова воспроизвести x, следовательно, число системных вызовов не ограничено снизу, что противоречит самой схеме восстановления.

В результате заключаем, что циклические зависимости запрещены, и допустимы лишь формально в случае $x = y$, откуда следует коммутативность: $y = x \sqcup y = y \sqcup x = x$.

3. Пусть $(x, y, attr_1), (y, x, attr_1) \notin E^{dep}$.

- Рассмотрим минимальный доминирующий атрибут: $attr_2$. Если $y.attr_2 = x.attr_2$, и $\exists x'(attr_2) \in D(attr_2, x.attr_2)$ - создатель ресурса, описываемого атрибутом $attr_2$, доказательство завершено. Иначе, $y.attr_2 \neq x.attr_2$, и процедура повторяется.
- Пусть выполнено $k-1 = depth(attr_1)$ шагов процедуры выше, и $attr_k$ суть атрибут корневого PID-пространства имён.
- корневое PID-пространство имён, по Определению 6, и, после $k < \infty$-шагов, $z = v_{init}$.

Следовательно, $(V^+, \sqcup)$ - конечная верхняя полурешетка, с задаваемым зависимостями естественным частичным порядком и максимальным элементом v_{init}.**Что и требовалось доказать.** □

3. Критерий корректности дерева процессов и другие результаты

Полурешеточная упорядоченность V^+ - крайне важный результат. В частности, для проверки корректности дерева процессов. Понятие корректности дерева процессов введено в работах [1, 4, 5] прагматично: дерево процессов $T = (V, E)$ является корректным, если $\exists G = (V^+, E^+)$ – граф реконструкции, такой, что E^+ содержат метки-системные вызовы, и после топологической сортировки G последовательное выполнение таких меток восстанавливает V из начального состояния v_{init}. Такое определение соответствует выразительному смыслу реконструкции: те деревья, которые заведомо невозможно восстановить, полагаются некорректными. Кроме того, все деревья, полученные из настоящих рабочих снимков [9] состояний операционных систем, полагаются корректными, так как получены именно последовательностями системных вызовов. Предыдущие работы [4, 5] рассматривали реальные, следовательно, корректные деревья процессов. С другой стороны, валидация произвольных деревьев на корректность может оказаться полезной, к примеру, в задачах генерации синтетических тестов [10] для систем сохранения и восстановления состояний ОС. Структурные свойства V^+ могут представлять интерес для построения критериев такой валидации деревьев. Для обоснования данного факта, введём и докажем ряд вспомогательных утверждений.

Лемма 1. *Пусть $CL(v.attr_1) : V^+ \to V^+$ - оператор, $\forall v \in V^+$ определяющий $f \in V^+$ с минимальным доминирующим атрибутом для некоторого $attr_1$, такой, что $v \sqcup CL(v) = CL(v)$. Тогда CL монотонный и экстенсивный.*

Доказательство. Из свойств ресурсов ядра ОС, CL обладает:

1. Экстенсивностью: $v \leq CL(v.attr_1)$. Это свойство очевидно из факта, что атрибут не может появиться в дереве, пока соответствующий ресурс не создан.

2. Монотонностью: $\forall v, u \in V^+ : v \leq u \to CL(v.attr_1) \leq CL(u.attr_1)$. Предположим, что минимальный доминирующий атрибут $attr_1$ это $attr_2$, и $v \leq u$. Пусть $u.attr_2 \neq v.attr_2$. Тогда ресурс, описываемый $u.attr_2$ должен быть создан ранее, чем ресурс того же типа для v, либо они не сравнимы. Иначе они равны.

□

Definition 7. *Будем обозначать оператор предзамыкания[11] CL, если идемпотентность не обязана выполняться, как PCL.*

Лемма 2. *Пусть CL возвращает состояние, в котором минимальный доминирующий атрибут был создан. Тогда CL - оператор замыкания [12], со строго одной неподвижной точкой для каждого $f = CL(v.attr_1)$,$\forall v \in V^+, \forall attr_1 \in attr$ в случае, если V^+ соответствует порождению корректного дерева процессов.*

Доказательство. Рассмотрим $CL(CL(v.attr_1).attr_1)$.
$CL(CL(v.attr_1).attr_1) = CL(v.attr_1) \Leftrightarrow CL(v.attr_1)$- создатель. Следовательно, CL идемпотентный.

Случай, если CL для вершины v с заданным атрибутом $v.attr_1$ будет иметь более одной неподвижной точки, будет означать создание более одного ресурса с одним и тем же идентификатором в одной области действия данного идентификатора, что означает конфликт идентификаторов. □

Приведём некоторые результаты, следующие из полурешеточной упорядоченности V^+ и приведённых выше лемм:

Теорема 1. *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 $attr_k$.*

Доказательство. (От противного).

Пусть либо V^+ не верхняя полурешётка, либо $\exists x, y \in V^+$

$$\forall k \exists n > k : \big(attr_k, attr_n \in K, depth(attr_n) < |K| - depth(attr_k)\big) \;\&\; \big(x.attr_n = y.attr_n\big),$$

для которых $x \sqcap y$ не равен ни $PCL(x.attr_k)$, ни $PCL(y.attr_k)$, ни $CL(x.attr_k)$, либо $\big(CL(x.attr_k) \neq CL(y.attr_k)\big)$.

Пусть $x \sqcap y$ не удовлетворяет состояниям, которые можно получить введением $PCL(x.attr_k)$ и $PCL(x.attr_k)$, то есть минимально доминирующие атрибуты не совпадают по значениям. Рассмотрим минимально доминирующий атрибут $attr_{k+1}$ для $attr_k$; так как глубина изоляции $depth(x.attr_{k+1}) < |K| - depth(attr_k)$ как доминирующего, следовательно, $x.attr_{k+1} = y.attr_{k+1}$. Противоречие. $x \sqcap y = CL(x.attr_k)$ соответствует случаю, когда x, y являются прямыми потомками создателя атрибута $x.attr_{k+1}$, и для него рассуждения аналогичны. Кроме того, получение CL соответствует конечному числу применений PCL, при котором в силу монотонности достигается создатель атрибута $x.attr_{k+1}$ и $y.attr_{k+1}$.

Пусть теперь $CL(x.attr_k) \neq CL(y.attr_k)$. Но $x.attr_{k+1} = y.attr_{k+1}$, значит, CL обязан вернуть единственную неподвижную точку - создателя $attr_{k+1}$. В противном случае имеет место конфликт по идентификаторам.

Пусть теперь V^+ - не полурешётка. Покажем, что полурешёточная упорядоченность V^+ восходит к процедуре его построения из V, если это вообще возможно. Возможны случаи:

1. Пусть $\exists x, y \in V^+ : x \sqcup y$ не определяется однозначно, то есть существует несколько верхних границ, из любых из которых можно удовлетворить зависимости по $attr_k$. В следствие выполнимости правой части коньюнкции из условия теоремы, все такие границы лежат в замыкании, задаваемом $CL(x.attr_k)$. Следовательно, V^+ следует урезать

выбором любой из данных границ в качестве наименьшей, что производится удалением лишних зависимостей. Удаление всех избыточных зависимостей из $\left(V^+, E^{dep}\right)$ даст транзитивный остов [13] $\left(V^+, E^{excl(dep)}\right)$ как граф искомой полурешётки [14].

2. Пусть для некоторых $x, y \in V^+$ таких, что в условиях правой части коньюнкции не существует наименьшей верхней границы, и её не получить из процедуры выше. Но тогда $\forall k \exists n > k : \left(attr_k, attr_n \in K, depth(attr_n) < |K| - depth(attr_k)\right)$ & $\left(x.attr_n = y.attr_n\right)$, для которых $x \sqcap y$ не равен ни $PCL(x.attr_k)$, ни $PCL(y.attr_k)$, ни $CL(x.attr_k)$, что, как доказано ранее, противоречит предположению о корректности дерева процессов.

□

Доказанная теорема представляет необходимое условие корректности дерева процессов.

Какие промежуточные состояния дают операторы PCL? Ответ на этот вопрос формулирует достаточное условие корректности дерева процессов. В частности, из него следует способ проверки произвольного дерева на корректность.

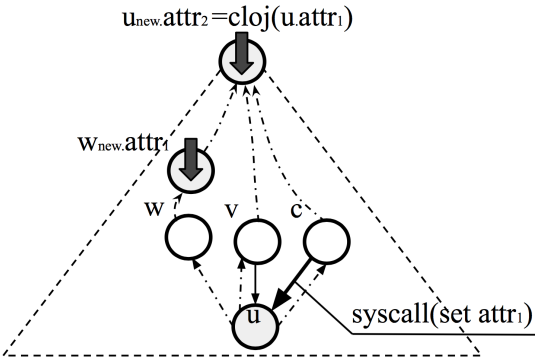

Рис. 2: Элементарный шаг восстановления состояния $u : u.attr_1 = w.attr_1$.

Рассмотрим некоторое состояние процесса, получаемое системным вызовом, производящимся успешно при выполнении следующих условий:

Предложение 1. *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* $v : u.pid = v.pid | (u.pid \neq v.pid \quad \& \quad syscall =' fork')$ *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* $attr_1$.

Проверка данного утверждения технически производится покрытием введённой конструкцией всех рассматриваемых случаев установки атрибута процессу, что является следствием из свойств иерархии процессов Linux [15] и особенностей использования интерфейса системных вызовов [4, 16]. Классификация атрибутных зависимостей и некоторые частные случаи таких восстанавливающих конструкций рассматриваются в Разделе 4.

Лемма 3. $u, v, w, w_{new}, c, CL(u.attr_1)$ *формируют полную решетку с частичным порядком, наследованным из* V^+.

Доказательство. $c.attr_2 = v.attr_2 = w_{new}.attr_2 = u.attr_2$, и $CL(u.attr_1)$ единица решетки. v наименьший элемент, и нет зависимостей от v, но он зависит от c, w_{new},u,$CL(u.attr_1)$, атрибуты $attr_2$ которых равны между собой. То есть v - meet-элемент для любого подмножества из

подмножеств $c, u, w_{new}, CL(u.attr_1)$ и $c, u, w, CL(u.attr_1)$, а для w, w_{new} - w является meet-элементом. следовательно, $u, v, w, w_{new}, c, CL(u.attr_1)$ - полная решетка с нулем v и единицей $CL(u.attr_1)$. □

Опишем данную решётку системой неравенств:

$$\begin{cases} w \leq w_{new} \leq CL(u.attr_1), \\ v \leq CL(u.attr_1), \\ c \leq CL(u.attr_1), \\ u \leq v, \\ u \leq w, \\ u \leq c. \end{cases}$$

Definition 8. *Назовём v, c, w, w_{new} генерирующим множеством $Gen(u.attr_1)$.*

Существование генерирующего множества $Gen(u.attr_1) \forall u \in V^+ \setminus v_{init}, \forall attr_1 \in K$, подходящих для реконструкции - достаточное условие восстановления дерева процессов.
Сформулируем критерий корректности.

Теорема 2. *Дерево процессов T корректно $\iff$ ($\forall u \in V^+ \setminus v_{init}$, V^+ такое, что выполнены условия теоремы о необходимости, и $\forall attr_i \in K \Rightarrow \exists Gen(u.attr_i) : Gen(u.attr_i) \subseteq V^+$).*

Доказательство. Необходимость следует из Теоремы 1. Обоснуем достаточность. Для этого проведём некоторую процедуру реконструкции по предвосстановленным состояниям из V^+, которая восстанавливает отношение зависимости на нём. Итак, $\forall u \in V^+$ восстановление разбивается на 3 (возможно, совпадающие в частных случаях) цепочки, состояния в которых можно получить через $PCL(u.attr)$ такой, что образ не меньше исходного состояния, причём все такие состояния лежат в замыкании, задаваемом $CL(u.attr)$:

1. Восстанавливается создатель или "хранитель"нужного $attr_k$. Создатель будет получен из хранителя в любом случае через конечное число применений PCL в силу монотонности. В виду восстановления в замыкании, сохраняется иерархия атрибутов: получая $\forall attr_k \in K$, что $CL(u.attr_k)$, является создателем минимально доминирующего атрибута, повторяя данную процедуру, за конечное число шагов достигается максимально доминирующий - корневое пространсово имён.

2. Восстанавливается порядок следования состояний процессов: через отношение PCL находится в замыкании состояние-предшественник. При этом большая часть атрибутов наследуется. В конечном счёте, многократное применение PCL приводит к CL за исключением, когда $attr_k$ - это PID-пространство имён. Тогда естественным образом такие применения должны приводить к состоянию, в котором процесс был порождён, то есть его PID не совпадает с PID предшественника, и данное состояния является верхне неразложимым элементом по зависимостям: практически все атрибуты наследуются от предшественника, кроме PID, а ненаследуемые выставляются локально, без внешних зависимостей, то есть $\exists! dep(PCL(PCL(...(u))), parent(u))$ - зависимость первого состояния процесса от его родителя. Таким образом, строится вся цепочка состояний целевого процесса.

3. Для каждой последовательной пары u, v состояний из пункта выше находится процесс, который будучи в состоянии c, выполняет системный вызов, порождающий u из v.

Восстановления выше выполняются $\forall u \in V^+ \setminus v_{init}, \forall attr \in K : u.attr \neq v.attr$, остальные атрибуты наследуются. Для всех найденных состояний $w, w_{new}, c, v \in V^+$ и процедура поиска

также завершается успешно найденными промежуточными состояниями. Наконец, из 2-й и 3-й цепочек следует, что $\forall u \in V^+ \setminus v_{init}\ \exists v$ и системный вызов, восстанавливающий отношение (u, v) в правильном порядке следования атрибутов, и такая конструкция ациклична, как подмножество решеточно упорядоченного V^+ относительно наследуемого порядка. Следовательно, она сортируема топологически, и требование корректности выполнено. □

4. Типы атрибутных зависимостей на практике

В соответствии с введёнными в Разделах 1 и 2 определениями и свойствами процессов и системных вызовов ОС Linux [15], выделим условно 4 типа атрибутных зависимостей (Рис. 3):

1. Жестко наследуемые (Hardly Inherited, HI) - в G существует цепь от создателя атрибута к наследующему ребёнку по E^+. Создатель не может изменить значение атрибута после создания соответствующего ресурса. Пример: сессии, PID-пространства имён.

2. Устанавливаемые в подобласти (Subset Inherited, SI) – "хранитель"значения, выставляемого атрибуту, предшествующее состояние процесса и состояние, из контекста которого выполняется системный вызов, лежат в замыкании, определяемом оператором CL. Данный случай представляет наиболее общее описание генерирующего множества как антицепи w, c, v, возможны также частные случаи: выставление процессу атрибута носителем $c = w$, выставление атрибута самому себе: $c = v$ и др.

3. Мягко наследуемые (Mildly Inherited, MI): Создатель и наследник могут изменить значение атрибута сколь угодно раз, произведя соответствующий системный вызов. Усиление SI.

4. Свободно устанавливаемые в рамках текущего пространства имён. Согласно такой формулировке, это как наследуемые UID, GUID (Free Inherited, FI) , так и ненаследуемые (Free Non-inheritable, FN) — маски сигналов и др. На практике, большинство выставлений таких атрибутов происходит с частными условиями

$$u \leq v = c \leq CL(u.attr), w = w_{new} \leq CL(u.attr),$$

а $CL(u.attr)$ является создателем пространства имён. Имеет место доупорядочивание генерирующего множества: $c = v \leq w \leq w_{new} \leq CL(u.attr)$

Приведённая классификация атрибутных зависимостей и частные случаи во многом упрощают поиск подходящих генерирующих множеств за счёт частных случаев.

5. Заключение

В работе были исследованы свойства множества состояний процессов Linux, восстанавливающего дерево процессов, показано построение такого множества как полурешёточно упорядоченного по зависимостям. На основании свойств атрибутов процессов и полурешёточной упорядоченности был получен критерий корректности дерева процессов, то есть соответствия реальной конфигурации ОС и восстановимости. Тем не менее, представляет интерес построение формального алгоритма проверки по схеме из достаточного условия, исследования его временной и пространственной сложности. Получение и исследование частного случая критерия для деревьев процессов с набором атрибутов, урезанном до групп, сессий и процессов в едином пространстве имён, из работ [4, 5], будут исследованы и приведены в дальнейших работах автора. Одновременно с этим, представляет интерес получения частных случаев критерия для эвристических процедур из существующих систем сохранения и восстановления

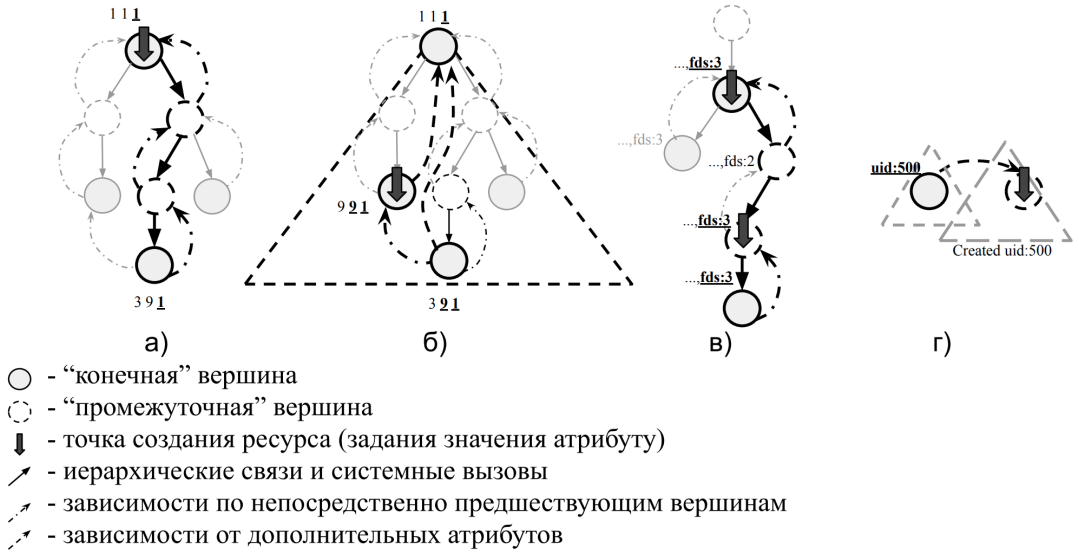

Рис. 3: Типы атрибутных зависимостей: а)строго наследуемые; б)выставляемые в подобласти; в)нестрого наследуемые; г)свободные.

состояния, таких как CRIU [2], BLCR [17], DMTCP [18], а также других приложений, к примеру, восстановления fork-join-последовательностей для многопоточного программирования [19] и для различных представлений задачи восстановления дерева процессов, например, в матричной форме [20]. На основании такого исследования будет проверена гипотеза о повышении эффективности вышеперечисленных методов за счёт уменьшения мощности множеств для поиска нужных состояний, так как состояния с нужными значениями атрибутов предположительно будут также замыкаться в соответствующих областях.

СПИСОК ЦИТИРОВАННОЙ ЛИТЕРАТУРЫ

1. Ефанов Н. Н. О некоторых комбинаторных свойствах деревьев процессов LINUX. Чебышевский сборник. 2018. 19(2). 151-162 сс.

2. Checkpoint-Restore In Userspace (CRIU) usage scenarios. 2019. `https://criu.org/Usage_scenarios` (Дата обращения: 24.05.2019)

3. Kraemer, S., Leupers, R., Petras, D.,Philipp, T., Hoffmann, A. Checkpointing SystemC-based virtual platforms // International Journal of Embedded and Real-Time Communication Systems, 2 (4), RWTH Aachen University, Germany. 2011. pp. 21-37.

4. Efanov N. N., Emelyanov P. V. Constructing the formal grammar of system calls // In Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR'17). 2017. Article 12. 5 pages.

5. Efanov N. N., Emelyanov P. V. Linux Process Tree Reconstruction Using The Attributed Grammar-Based Tree Transformation Model // In Proceedings of the 14th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR'18). 2018. ACM, NY, USA. Article 2. 7 pages.

6. Горбунов Е. «Алгоритм генерации команд восстановления дерева процессов ОС Linux на основе модели жизненного цикла ресурсов ОС». Магистерская диссертационная работа, Труды Кафедры Математических и Информационных технологий, СПбАУ РАН, 2017.

7. Linux namespaces. https://en.wikipedia.org/wiki/Linux_namespaces (Дата обращения: 24.05.2019)

8. Alfred V. Aho , Jeffrey D. Ullman , The theory of parsing, translation, and compiling, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972. URL: dl.acm.org/citation.cfm?id=578789 (Дата обращения: 24.05.2019).

9. Sukadev Bhattiprolu, Eric W. Biederman, Serge Hallyn, and Daniel Lezcano. Virtual servers and checkpoint/restart in mainstream Linux. SIGOPS Oper. Syst. Rev. 42, 5. July 2008, pp 104-113. doi:10.1145/1400097.1400109

10. Kalaee, A., Rafe, V. Model-based test suite generation for graph transformation system using model simulation and search-based techniques, Information and Software Technology, Elsevier, 108, 2019, pp. 1-29.

11. A.V. Arkhangelskii, L.S.Pontryagin, General Topology, Vol. 1, Springer-Verlag, Berlin, 1990.

12. Биркгоф, Г. Теория решеток. — М. : Наука. Главная редакция физико-математической литературы, 1984. — 568 с.

13. Aho, A. V., Garey, M. R., Ullman, J. D., The transitive reduction of a directed graph // SIAM Journal on Computing, 1 (2), 1972, pp 131–137.

14. Freese R. Automated lattice drawing //International Conference on Formal Concept Analysis. – Springer, Berlin, Heidelberg, 2004. – pp 112-127.

15. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.

16. Linux system calls. MANual pages(2). 2019. http://man7.org/linux/man-pages/man2/syscalls.2.html (Дата обращения: 24.05.2019)

17. Hargrove P. H., Duell J. C. Berkeley lab checkpoint/restart (BLCR) for linux clusters // Journal of Physics: Conference Series. – IOP Publishing, 2006. – Т. 46. – №. 1. – p. 494.

18. Arya K., Cooperman G. DMTCP: Bringing Checkpoint-Restart to Python // Proceedings of the 12th Python in Science Conference. – 2013. 2-7 cc.

19. Zengin M., Vafeiadis V. A programming language approach to fault tolerance for fork-join parallelism // Proceedings of the 7th International Symposium on Theoretical Aspects of Software Engineering (TASE 2013), Max Planck Institute for Software Systems (MPI-SWS), Saarsbruchen, Germany, 2013.

20. Marina Kudinova and Pavel Emelyanov. Building Mathematical Model for Restoring Processes Tree during Container Live Migration. // IVth International Conference on Engineering and Telecommunication (EnT), November 2017, Dolgoprudniy. doi: 10.1109/ICEnT.2017.41

REFERENCES

1. Efanov N.N. 2018. “On some combinatorial properties of LINUX process trees”. Chebyshevskii Sbornik, 19 (2), pp. 151-162.

2. Checkpoint-Restore In Userspace (CRIU) usage scenarios. 2019. https://criu.org/Usage_scenarios

3. Kraemer, S., Leupers, R., Petras, D.,Philipp, T., Hoffmann, A. .2011. Checkpointing SystemC-based virtual platforms. International Journal of Embedded and Real-Time Communication Systems, 2 (4), RWTH Aachen University, Germany. pp 21-37.

4. Efanov N. N., Emelyanov P. V. "Constructing the formal grammar of system calls", In Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR '17), ACM, New York, NY, USA, Article 12, 5 pages. doi: 10.1145/3166094.3166106

5. Efanov N. N., "Emelyanov P. V. Linux Process Tree Reconstruction Using The Attributed Grammar-Based Tree Transformation Model", In Proceedings of the 14th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR'18), ACM, New York, NY, USA, Article 2, 7 pages. doi: 10.1145/3290621.3290626

6. Gorbunov E. 2017. "Algoritm generatsii komand vosstanovleniya dereva processov OS Linux na osnove modeli zhiznennogo tsykla resursov". Master's thesis, MIT SPbAU.

7. Leonardo Passos, Jianmei Guo, Leopoldo Teixeira, Krzysztof Czarnecki, Andrzej Wąsowski, and Paulo Borba 2013, "Coevolution of variability models and related artifacts: a case study from the Linux kernel", In Proceedings of the 17th International Software Product Line Conference (SPLC '13), ACM, New York, NY, USA, 91-100. doi: 10.1145/2491627.2491628

8. Alfred V. Aho , Jeffrey D. Ullman 1972, The theory of parsing, translation, and compiling, Prentice-Hall, Inc., Upper Saddle River, NJ, USA. Available at: `dl.acm.org/citation.cfm?id=578789`

9. Sukadev Bhattiprolu, Eric W. Biederman, Serge Hallyn, and Daniel Lezcano, 2008, "Virtual servers and checkpoint/restart in mainstream Linux", SIGOPS Operation Systems Rev. 42, no. 5, July 2008, pp. 104-113. doi:10.1145/1400097.1400109

10. Kalaee, A., Rafe, V. 2019, "Model-based test suite generation for graph transformation system using model simulation and search-based techniques Information and Software Technology, 108, Elsevier, pp. 1-29.

11. A.V. Arkhangelskii, L.S.Pontryagin, 1990, General Topology, Vol 1, Springer-Verlag, Berlin.

12. Birkhoff, G. 1973, Lattice Theory, 3rd ed., Providence, RI : American Mathematical Society.

13. Aho, A. V., Garey, M. R., Ullman, J. D., 1972, "The transitive reduction of a directed graph", SIAM Journal on Computing, 1 (2), pp 131–137.

14. Freese, R., 2004, "Automated lattice drawing". In International Conference on Formal Concept Analysis, pp. 112-127. Springer, Berlin, Heidelberg.

15. Tanenbaum, A. S., Bos, H. 2014. Modern Operating Systems. 4th. edition.

16. Linux system calls. MANual pages(2). 2019. `http://man7.org/linux/man-pages/man2/syscalls.2.html`

17. Hargrove, P. H., & Duell, J. C. 2006. Berkeley lab checkpoint/restart (BLCR) for linux clusters. In Journal of Physics: Conference Series Vol. 46, No. 1, p. 494. IOP Publishing.

18. Arya, K., & Cooperman, G. 2013. DMTCP: Bringing Checkpoint-Restart to Python. In Proceedings of the 12th Python in Science Conference, pp. 2-7.

19. Zengin M, Vafeiadis V. 2013, “A programming language approach to fault tolerance for fork-join parallelism”, Proceedings of the 7th International Symposium on Theoretical Aspects of Software Engineering (TASE 2013), Max Planck Institute for Software Systems (MPI-SWS), Saarsbruchen, Germany, 2013. Available at:
`http://plv.mpi-sws.org/ftpar/tase2013-ftpar.pdf`

20. Marina Kudinova and Pavel Emelyanov. 2017, “Building Mathematical Model for Restoring Processes Tree during Container Live Migration”, IVth International Conference on Engineering and Telecommunication (EnT), November 2017, Dolgoprudniy. doi: 10.1109/ICEnT.2017.41

Получено 30.05.2019 г.
Принято в печать 20.12.2019 г.