Граф действий для восстановления дерева процессов

Горбунов Егор Алексеевич

8 мая 2017 г.

1 Основные понятия

1.1 Введение

Задача данного документа — это формализация задачи восстановления дерева процессов в Linux. Перед тем как вводить основные понятия, неформально опишем задачу восстановления:

Задача восстановления дерева процессов в Linux — это задача поиска и исполнения последовательности действий, которые, будучи исполненными, приведут исходных «пустой» процесс в состояние целевого дерева процессов.

Для этой задачи, целевое дерево процессов — это набор из нескольких процессов, объединённых в дерево отношением родитель-ребёнок. Каждый процесс из этого дерева, в рамках задачи восстановления, есть ни что иное, как статичный набор ресурсов — снимок состояния. Целевой процесс мы рассматриваем не как развивающийся во времени организм, а просто как состояние этого процесса в определённый момент времени.

Решение задачи восстановления предполагает, что найденная последовательность действий может быть выполнена достаточно быстро, чтобы удовлетворять требованиям живой миграции.

1.2 Ресурс и процесс

Для разработки наиболее общего подхода к алгоритму восстановления (генерации команд для восстановления), в плане покрываемых ресурсов, понятие ресурса также должно быть широким.

Определение 1. *Ресурс* — r — некоторая структура в ядре ОС, которая так или иначе используются *процессом* и операционной системой. Такая структура — это абстрактное понятие само по себе, она может представлять из себя реальный экземпляр сишной структуры, но можеть быть чем-то менее осязаемым.

Ресурсы существуют не просто так, а как описывается в определении, они используются процессами. Обычно, процесс не может взаимодействовать с ресурсом, как со структурой/объектом в ядре, напрямую. Вместо этого у процесса есть некоторый интерфейс к ресурсу.

Определение 2. *Интерфейс к ресурсу — handle* — это объект, через который процесс получает доступ к ресурсу. Будем обозначать его как h.

Таким образом владение ресурсом для процесса можно описать парой: (r, h) из ресурса и интерфейса к этому ресурсу. Например, r — это экземпляр `struct file`, а h — файловый дескриптор. (`В целесообразности введения термина handle ещё нужно удостовериться. Для некоторых ресурсов тяжело подобрать нужный handle: pid, один из таких =) Будет ли у него хэндл равен ресурсу? Нужно ли вообще третировать pid как ресурс?`).

Вообще говоря, процесс — это объект, который развивается во времени. Но под термином «процесс» в данном документе мы будем иметь ввиду некоторое его состояние в какой-то момент времени. Процессы будем обозначать заглавными буквами: P_1, P, A, B.

Факт того, что процесс P владеет ресурсом r, к которому получает доступ через ***handle*** h, обозначаем так: $(r, h) \in P$

Будем предполагать далее, что существует множество процессов $\mathcal{P}$, в которое входят всевозможные P, где процесс $P = \{(r_1, h_1), (r_2, h_2), \ldots, (r_n, h_n)\}$ представляет из себя набор ресурсов, которыми он «владеет». Все кванторы $\exists, \forall$, аргументами которых является процесс, будут предполагать, что процесс этот берётся из $\mathcal{P}$, т.е. $\exists P \iff \exists P \in \mathcal{P}$. $\mathcal{P}$ — это, по сути, множество из всевозможных множеств ресурсов (процессов).

Замечание: вообще говоря процесс — это более сложная конструкция, чем просто набор ресурсов. Было бы корректнее в данном документе использовать вместо процесса слова ***снимок процесса***, но это длиннее.

1.3 Разделяемые и неразделяемые ресурсы

Ресурсы можно разбить на два множества: те, что могут быть разделены между несколькими процессами и те, что индивидуальны для каждого процесса. Попытка формализовать эти понятия такова:

- r — разделяемый ресурс, если $\exists P_1, P_2 : P_1 \neq P_2 \wedge (r, h_1) \in P_1 \wedge (r, h_2) \in P_2$. То есть возможны таки два процесса, что оба процесса ссылаются на один и тот же ресурс r в ядре
- r — неразделяемый, соответственно, если двух таких процессов существовать не может

К примеру:

- идентификатор процесса (`pid`) — это неразделяемый ресурс.
- открытый файл — разделяемый ресурс

1.4 Создание ресурса

Вообще говоря процесс на то так и называется, что живёт во времени. При жизни процесс развивается (обычно): он открывает файлы, подключается к сети, создаёт потоки и другие процессы, производит

вычисления тем самым меняя значения регистров и ячеек памяти. Но для задачи восстановления дерева процессов такой подход к процессу излишний. Нам достаточно рассматривать процесс как замороженный набор ресурсов в определённой конфигурации. ***Задача восстановления*** заключается в том, чтобы сгенерировать, желательно, наиболее короткую последовательность простых действий, исполнение которых приведёт «пустой» процесс в целевое состояние.

Одно из основных необходимых действий - это создание ресурса. Любой ресурс так или иначе должен быть кем-то создан. (какие-то ресурсы, конечно, могут существовать "всегда"и попадать во владение к другим процессам при помощи механизмов разделения ресурсов, о которых в сл. секции).

Вводим следующее:

- Пустой процесс: $P = \{\}$ (такой процесс у нас в распоряжении в начале восстановления)
- Действие создания ресурса: ***CreateAction***(P, r, h) — процесс P создаёт ресурс r с интерфейсом (***handle***) h к нему

Сам процесс является ресурсом. После исполнения действия ***CreateAction***(P, r, h) верно: $(r, h) \in P$.

1.5 Разделение ресурсов

Процессы в Linux, как известно, выстраивают собою дерево. И каждый процесс — это одна вершина всего дерева процессов. У каждой вершины в дереве есть процесс-родитель. Различные ресурсы могут разделяться несколькими процессами. Как один ресурс может быть «передан» от одного процесса к другому, т.е. как получилось так, что какой-то ресурс разделяется? Тут всё как в жизни:

- Наследование: ребёнок унаследовал ресурс от родителя при рождении
- Сделка (share): один процесс поделился ресурсом с другим процессом при жизни; один процесс воспользовался ресурсом, который уже используется кем-то другим

Есть такие ресурсы, которыми поделиться при жизни невозможно (или невозможно сделать это эффективно) — это, например, private mappings: это, казалось бы, неразделяемый ресурс, который при наследовании, всё же, становится разделяемым из-за механизма Copy On Write.

Так же отдельного внимания заслуживают `pid namespace` (?).

Введём формальные обозначения для наследования и «сделки». Заметим, что наследование — это по сути не действие, которое производится процессом для достижения нужного результата, наследование ресурса представляет из себя «эффект» или свойство процедуры восстановления. В свою очередь сделка — это действие, которое может входить в последовательность действий необходимых для восстановления дерева процессов.

- ***InheritProperty***(Q, r) — обозначает, что процесс Q, при создании, наследует от своего отца P ресурс r, причём если $(r, h) \in P$, то после создания Q: $(r, h) \in Q$

- ***ShareAction***(P_1, P_2, r, h) — обозначает действие: процесс P_1 разделяет ресурс r процессу P_2 (т.е. у P_2 до выполнения действия не было этого ресурса) так, что процесс P_2 получает доступ к r посредством ***handle*** r. После выполнения действия верно: $(r, h) \in P_2$

1.6 Зависимость между ресурсами

Одни ресурсы могут зависеть от других. В моём текущем понимании, зависимости между ресурсами проявляются при создании этих ресурсов и этим можно ограничиться. Примеры зависимостей:

- Не приватная Virtual Memory Area зависит от того или иного файла, который необходим для создания этого маппинга (`mmap(...)`)
- ...

Зависимость между ресурсами влияет на порядок выполнения действий при восстановлении. Введём обозначение для краткого описания того, что один ресурс зависит от другого:

- ***DependsProperty***(r_1, r_2) — ресурс r_1 зависит от ресурса r_2

Замечание: может случаться так, что процесс из дерева, в целевой своей конфигурации, имеет ресурс r, т.е. $(r, h) \in P$. При этом ***DependsProperty***(r, q), но $(q, \_) \notin P$. Это значит, что в последовательности действий для восстановления должно фигурировать действие по удаления ресурса q из процесса, после того, как ресурс r был создан. Для того, чтобы обслуживать такую ситуацию, введём действие по удалению ресурса, а точнее пары (r, h). Под удалением ресурса понимается не глобальное удаление структуры в ядре, а откреплление ресурса от процесса по одному из интерфейсов:

- ***RemoveAction***(P, r, h) — действие, после которого верно: $(r, h) \notin P$

PS: также, помимо обычных действий (***CreateAction***(P, r, h), ***ShareAction***(P_1, P_2, r, h)) можно вводить их «временные» аналоги. Временное действие будет обозначать, что эффект этого действия должен быть устранён по окончании восстановления. Но мы будем использовать подход с ***RemoveAction***(P, r, h)

2 Последовательно действий для восстановления

Выше мы ввели понятие ресурса, процесса, команд и свойств.

3 Классификация ресурсов

3.1 Идентификаторы процесса

Ресурс	***handle***	**Наследование**	**Разделение (share)**
Сам процесс	pid	нет	нет
Группа процесса	pgid	да	да, `setpgid()`
Сессия процесса	ssid	да	нет
Идентификатор пользователя	uid	да	да, `setuid()`