Автоматическое обнаружение гонок при параллельных сборках на основе Make

Метод и реализация

Артем Климов, Владислав Иванишин, Александр Монаков

X июня 2024 г.

ИСП РАН

План презентации

- **Введение**. Состояния гонок в Makefile. Примеры и формулировка проблемы.
- **Обзор существующих решений**.
- **Теоретический метод** автоматического обнаружения состояний гонок.
- **Реализация** метода в виде сантизатора для Make.
- **Оценка** сантизатора на реальных проектах, **сравнение** с существующими решениями. **Заключение**.

Введение

Введение: Состояния гонок

Состояние гонки — ситуация, когда несколько потоков работают с одним и тем же ресурсом одновременно, и результат выполнения программы зависит от порядка выполнения потоков.

- Одновременное чтение и запись по одному и тому же адресу;
- Создание каталога и одновременная работа с его содержимым;
- Удаление ресурса без ожидания окончания его использования.

Введение: Состояния гонок

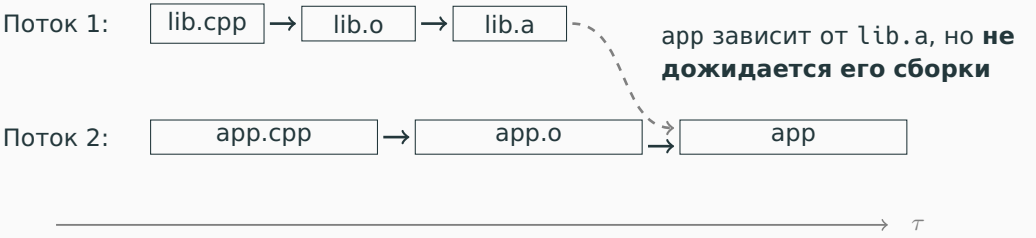
Пример: Два потока пытаются увеличить переменную `counter`, если она меньше 1.

```
1  std::atomic<int> counter = 0;
2
3  void thread1() {
4      if (counter < 1)
5          counter++;
6  }
7
8  void thread2() {
9      if (counter < 1)
10         counter++;
11 }
```

С неудачным планированием переменная может быть увеличена дважды, и окончательное значение будет равно 2.

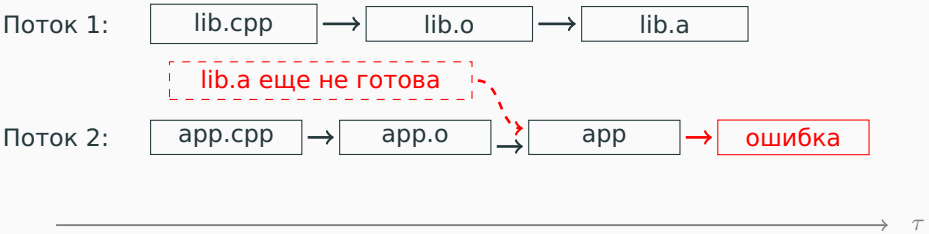
Введение: Состояния гонки в схемах сборки

- Цели сборки могут собираться параллельно;
- При отсутствии необходимой синхронизации могут возникать гонки.



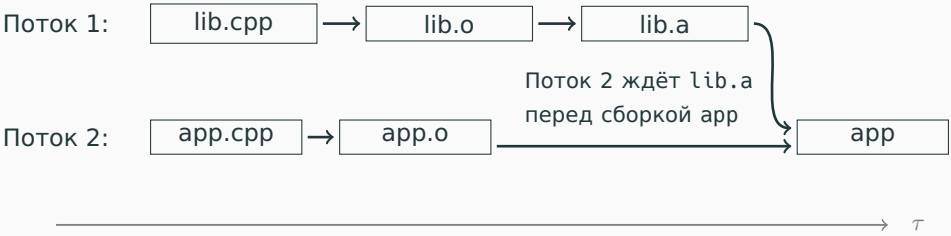
Введение: Состояния гонки в схемах сборки

- Если сборка библиотеки затянется, произойдет ошибка:



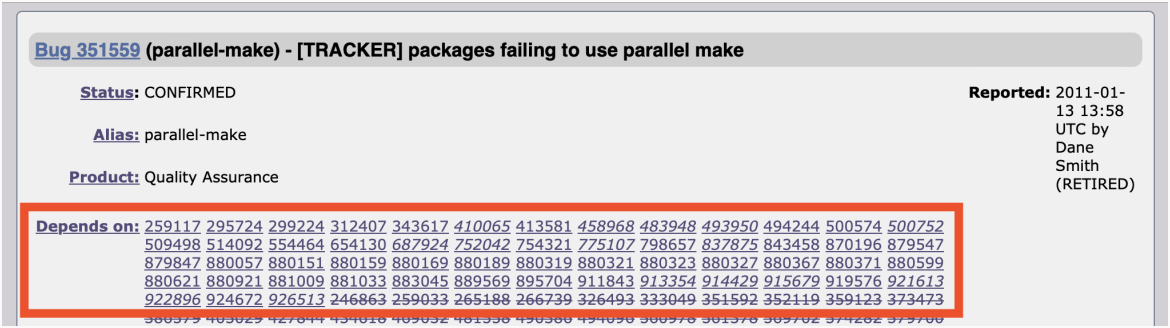
Введение: Состояния гонки в схемах сборки

- **Решение:** Определить пропущенную зависимость и добавить ее в схему сборки.



Введение: Состояния гонок в реальных проектах

Ошибки сборки, связанные с состояниями гонок, часто становятся предметом обсуждения на форумах:



Bug 351559 (parallel-make) - [TRACKER] packages failing to use parallel make

Status: CONFIRMED

Alias: parallel-make

Product: Quality Assurance

Reported: 2011-01-13 13:58 UTC by Dane Smith (RETIRED)

Depends on: 259117 295724 299224 312407 343617 *410065* 413581 *458968* *483948* *493950* 494244 500574 *500752* 509498 514092 554464 654130 *687924* *752042* 754321 *775107* 798657 *837875* 843458 870196 879547 879847 880057 880151 880159 880169 880189 880319 880321 880323 880327 880367 880371 880599 880621 880921 881009 881033 883045 889569 895704 911843 *913354* *914429* *915679* 919576 *921613* *922896* 924672 *926513* ~~246863~~ ~~259033~~ ~~265188~~ ~~266739~~ ~~326493~~ ~~333049~~ ~~351592~~ ~~352119~~ ~~359123~~ ~~373473~~

Введение: Формулировка проблемы

Состояния гонок в схемах сборки являются актуальной проблемой:

- Сценарий гонки может редко воспроизводиться;
- Состояния гонок могут приводить не только к ошибкам сборки, но и к уязвимостям и проблемам в успешно собраннном проекте.

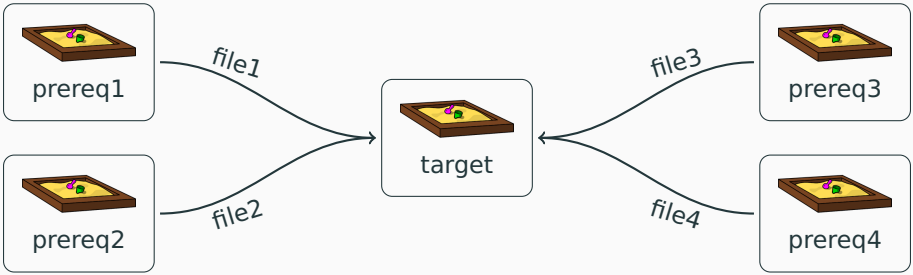
Цель исследования: Облегчить для разработчиков поиск и отладку состояний гонок в их схемах сборки путём разработки санитайзера. Требования к нему:

- Обнаруживать все гонки, связанные с отсутствующими зависимостями;
- Выдавать детерминированный результат;
- Легко встраиваться в проекты;
- Не требовать `-j1` или многократных пересборок.

Обзор существующих решений

Обзор существующих решений: Bazel

- Система сборки Bazel — использует песочницы для устранения случайности.
- Каждая цель собирается в своей песоцнице.
- Каждая песочница имеет только файлы, собранные целями-зависимостями.



Обзор существующих решений: make --shuffle

- `make --shuffle` — перемешивает список пререквизитов каждой цели.
- Это увеличивает вероятность проявления гонки, но помогает не всегда.

```
target: prereq1, prereq2, prereq3, ...
```

```
target: prereq5, prereq2, prereq4, ...
```

Теоретический метод

Теоретический метод: Типы гонок

Самые частые состояния гонок, встречающиеся в реальных проектах, можно разделить на три категории:

- **Гонка на содержимом файла.**
- **Гонка на пути к файлу.**
- **Гонка между созданием каталога и файлом в нем.**

Теоретический метод: Гонка на содержимом файла

- **Гонка на содержимом файла.**
- **Гонка на пути к файлу.**
- **Гонка между созданием каталога и файлом в нем.**

Теоретический метод: Гонка на содержимом файла

В следующем Makefile присутствует гонка на содержимом файла `file.out`

```
1  all: write_a append_b
2
3  write_a:
4          echo 'a' > file.out
5
6  append_b:
7          echo 'b' >> file.out
```

'a' записывается в `file.out` из цели `write_a`.

'b' добавляется к тому же файлу из цели `append_b`.

Теоретический метод: Гонка на содержимом файла

В следующем Makefile присутствует гонка на содержимом файла `file.out`

```
1  all: write_a append_b
2
3  write_a:
4          echo 'a' > file.out
5
6  append_b:
7          echo 'b' >> file.out
```

'a' записывается в `file.out` из цели `write_a`.

'b' добавляется к тому же файлу из цели `append_b`.

`write_a` и **`append_b`** независимы, их порядок выполнения не определен.

`file.out` может содержать `'ab'` или `'a'`.

Теоретический метод: Гонка на содержимом файла

В следующем Makefile присутствует гонка на содержимом файла `file.out`

```
1  all: write_a append_b
2
3  write_a:
4          echo 'a' > file.out
5
6  append_b: write_a
7          echo 'b' >> file.out
```

'a' записывается в `file.out` из цели `write_a`.

'b' добавляется к тому же файлу из цели `append_b`.

`write_a` и **`append_b`** независимы, их порядок выполнения не определен.

`file.out` может содержать `'ab'` или `'a'`.

Решение: Добавить зависимость между `append_b` и `write_a`

Теоретический метод: Гонка на содержимом файла

Как можно обнаружить такие состояния гонок?

Теоретический метод: Гонка на содержимом файла

Strace — утилита Linux, которая перехватывает и выводит системные вызовы и сигналы.

```
$ strace -f -e trace=%file make
...
# echo 'a' > file.out:
[pid  1060] openat(AT_FDCWD, "file.out", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
[pid  1060] +++ exited with 0 +++
...
# echo 'b' >> file.out:
[pid  1061] openat(AT_FDCWD, "file.out", O_WRONLY|O_CREAT|O_APPEND, 0666) = 3
[pid  1061] +++ exited with 0 +++
...
```

Согласно журналу strace, при сборке `file.out` открывается на запись двумя процессами.

Теоретический метод: Гонка на содержимом файла

Зная соответствие между процессами и породившими их целями, можно утверждать, что:

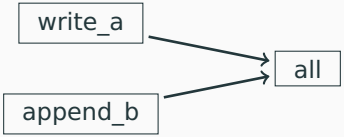
1. Цель **`write_a`** производит запись в `file.out`
2. Цель **`append_b`** тоже производит запись в `file.out`

Теоретический метод: Гонка на содержимом файла

Зная соответствие между процессами и породившими их целями, можно утверждать, что:

1. Цель **`write_a`** производит запись в `file.out`
2. Цель **`append_b`** тоже производит запись в `file.out`

Граф зависимостей схемы сборки выглядит следующим образом:

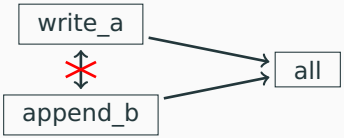


Теоретический метод: Гонка на содержимом файла

Зная соответствие между процессами и породившими их целями, можно утверждать, что:

1. Цель **`write_a`** производит запись в `file.out`
2. Цель **`append_b`** тоже производит запись в `file.out`

Граф зависимостей схемы сборки выглядит следующим образом:



Так как между `write_a` и `append_b` нет ориентированного пути, цели являются независимыми и образуют гонку.

Теоретический метод: Гонка на содержимом файла

Обзор алгоритма:

1. Запустить сборку под strace;
2. Сопоставить процессы с целями;
3. Получить граф зависимостей;
4. Найти недостающие зависимости между доступами к одному и тому же `пути` .

Теоретический метод: Гонка на содержимом файла

Обзор алгоритма:

1. Запустить сборку под strace;
2. Сопоставить процессы с целями; — Как?
3. Получить граф зависимостей; — Как?
4. Найти недостающие зависимости между доступами к одному и тому же `пути` .

Теоретический метод: Гонка на содержимом файла

Обзор алгоритма:

1. Запустить сборку под strace;
2. Сопоставить процессы с целями; — Как?
3. Получить граф зависимостей; — Как?
4. Найти недостающие зависимости между доступами к одному и тому же пути. — Всегда ли путь = файл?

Теоретический метод: Гонка на содержимом файла

Жесткие ссылки - это несколько имен для одного и того же файла в файловой системе. Они создаются с помощью команды `ln`.

```
all: file.out write_a append_b
file.out:
    touch file.out
    ln -f file.out hardlink_a
    ln -f file.out hardlink_b

write_a: file.out
    echo 'a' > hardlink_a

append_b: file.out
    echo 'b' >> hardlink_b
```

hardlink_a и hardlink_b - жесткие ссылки на file.out

'a' записывается в hardlink_a из цели write_a

'b' добавляется к hardlink_b из цели append_b

Теоретический метод: Гонка на содержимом файла

Журнал strace для предыдущей сборки укажет на следующее:

- Цель **`file.out`** производит запись в `file.out`
- Цель **`write_a`** производит запись в `hardlink_a`
- Цель **`append_b`** производит запись в `hardlink_b`

Все три пути различаются. Состояние гонки найти не удастся.

Теоретический метод: Гонка на содержимом файла

Решение: Использовать **номера inode** вместо путей к файлам.

Номер inode — уникальный идентификатор файла в файловой системе. У жестких ссылок на один и тот же файл номер inode совпадает. Для предыдущего примера:

- Цель **`file.out`** производит запись на inode `100000`
- Цель **`write_a`** производит запись на inode `100000`
- Цель **`append_b`** производит запись на `100000`

Теоретический метод: Гонка на содержимом файла

Обновленный алгоритм:

1. Запустить сборку под трассировщиком с логированием inode;
2. Сопоставить процессы с целями;
3. Получить граф зависимостей;
4. Найти недостающие зависимости между доступами к одному и тому же ~~пути файла~~ номеру inode

Этот алгоритм может найти состояния гонок даже при использовании жестких ссылок.

Теоретический метод: Гонка на пути к файлу

- Гонка на содержимом файла.
- **Гонка на пути к файлу.**
- Гонка между созданием каталога и файлом в нем.

Теоретический метод: Гонка на пути к файлу

В следующем Makefile присутствует гонка на пути `tmp_file`

```
1  all: something something_else
2
3  something:
4      generate_something > tmp_file
5      do_something_with tmp_file
6      rm tmp_file
7
8  something_else:
9      generate_something_else > tmp_file
10     do_something_else_with tmp_file
11     rm tmp_file
```

`tmp_file` используется как временный файл. Он удаляется после сборки цели.

То же имя используется для временного файла в другой цели.

Цели образуют гонку: `rm` может удалить файл, пока другая цель ещё использует его.

Теоретический метод: Гонка на пути к файлу

Последовательность событий для примера выше:

- Цель **`something`** создаёт inode 10000
- Цель **`something`** удаляет inode 10000
- Цель **`someting_else`** создаёт inode 10001
- Цель **`something_else`** удаляет inode 10001

Номера inode не помогут обнаружить гонку на пути к файлу.

Теоретический метод: Гонка на пути к файлу

Последовательность событий для примера выше:

- Цель **`something`** создаёт inode 10000
- Цель **`something`** удаляет inode 10000
- Цель **`someting_else`** создаёт inode 10001
- Цель **`something_else`** удаляет inode 10001

Номера inode не помогут обнаружить гонку на пути к файлу.

Решение: Использовать и пути, и номера inode для поиска гонок.

Теоретический метод: Гонка на каталоге

- Гонка на содержимом файла.
- Гонка на пути к файлу.
- **Гонка между созданием каталога и файлом в нем.**

Теоретический метод: Гонка на каталоге

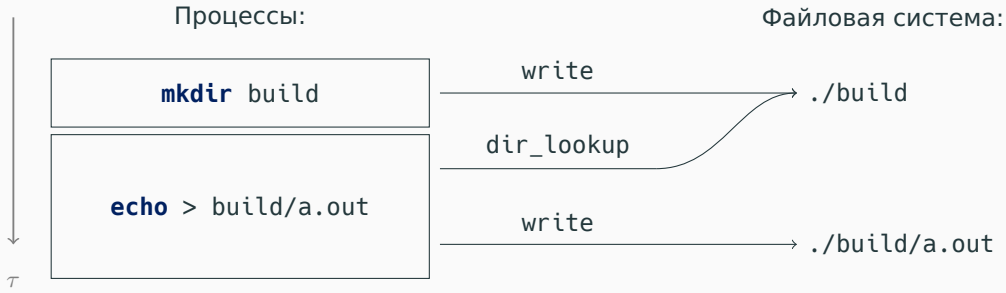
Makefile ниже содержит гонку между созданием каталога и файлом в нем:

```
1  all: build build/file.out
2
3  build:
4      mkdir -p build
5
6  build/a.out:
7      echo "a" > build/a.out
```

Если доступ к директории (**echo**) выполнится раньше её создания (**mkdir**), произойдет ошибка.

Теоретический метод: Гонка на каталоге

Решение: Сопровождать каждый доступ к файлу событием **dir_lookup** для родительской директории:



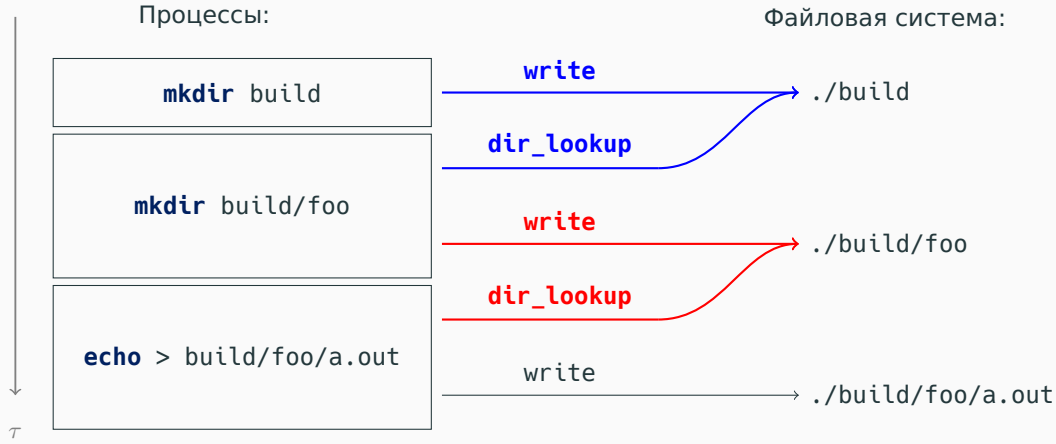
Теоретический метод: Гонка на каталоге

Решение: Сопровождать каждый доступ к файлу событием **dir_lookup** для родительской директории:

Конфликт между write и dir_lookup для build позволит найти гонку.

Теоретический метод: Гонка на каталоге

Такой подход работает с любой глубиной вложенности:



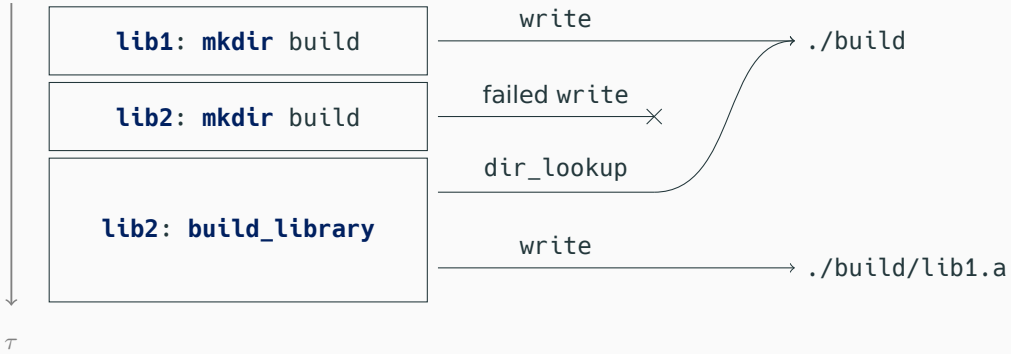
Теоретический метод: Гонка на каталоге

Проблема: Каталоги могут создаваться по необходимости из разных мест:

```
1  all: lib1 ... lib9
2
3  lib1:
4      mkdir -p build
5      build_library build/lib1.a
6  ...
7
8  lib9:
9      mkdir -p build
10     build_library build/lib9.a
```

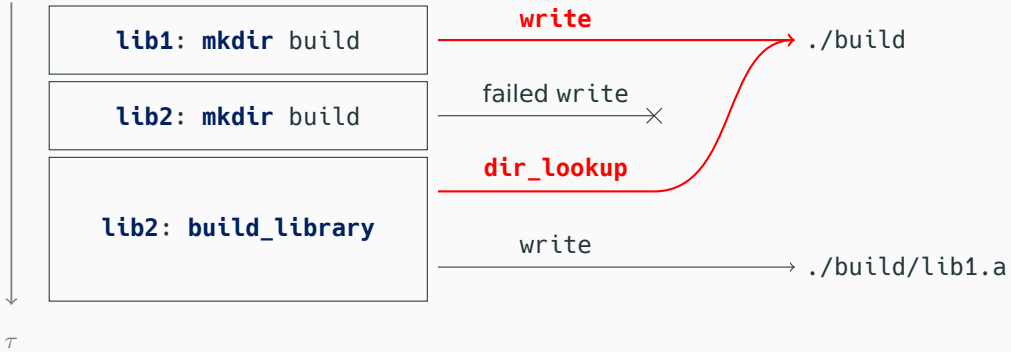
Теоретический метод: Гонка на каталоге

- При множественных **mkdir** алгоритм выдаст ложные срабатывания.



Теоретический метод: Гонка на каталоге

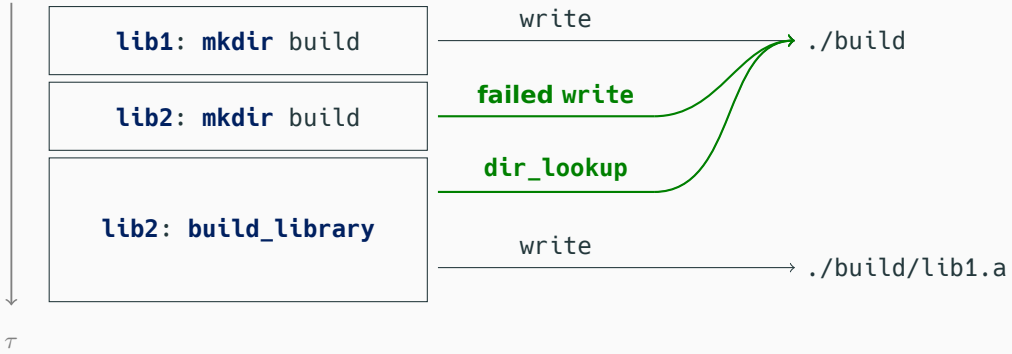
- При множественных **mkdir** алгоритм выдаст ложные срабатывания.



- Доступы из **lib1** и **lib2** будут считаться конфликтующими, хотя это не так.

Теоретический метод: Гонка на каталоге

- **Решение**: Для **dir_lookup** искать любой зависимый предшествующий **write**, даже если он неуспешный.



- Оба выделенных доступа относятся к **lib2**, поэтому гонки нет.

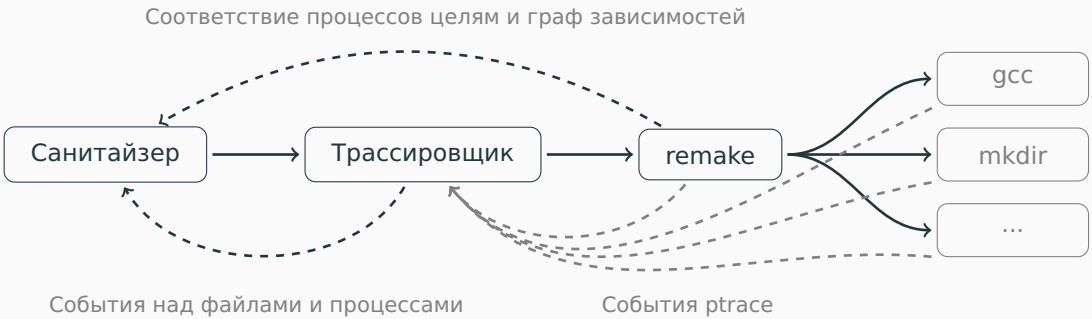
Реализация

Реализация: Архитектура

Сбор данных:

- **Граф зависимостей, соответствие процессов и целей** — патч для Make.
- **Журнал событий** — `ptrace`-трассировщик на Си вместо `strace`.

Архитектура:



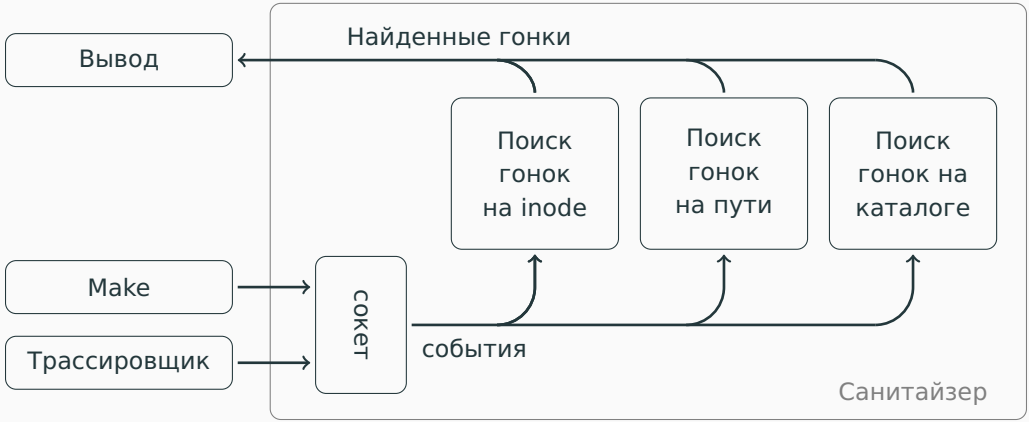
Реализация: Перехват системных вызовов

Системный вызов	Событие для санитайзера
`open(at)(at2)`	**`read`** или **`write`**
`mkdir(at)`	**`write`**
`creat`	**`write`**
`rmdir`	**`unlink`**
`unlink(at)`	**`unlink`**
`rename(at)(at2)`	**`unlink`** целевого пути, если он существует.

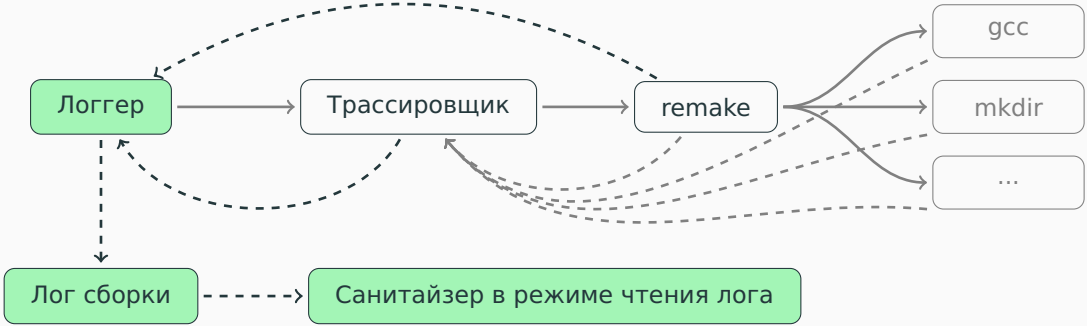
- Каждый доступ сообщается с **pid**, номером **inode** и **путём**.
- Для родительского каталога сообщается **`dir_lookup`**.
- Остальные системные вызовы фильтруются через `seccomp` BPF.

Реализация: Санитайзер

Санитайзер обрабатывает полученные события с помощью трех алгоритмов:



Реализация: Воспроизведение сборки



- Архитектура инструмента позволяет сохранять журнал сборки в файл и воспроизводить его после.
- Это позволяет быстро запускать санитайзер несколько раз с разными опциями или точками останова.

Тестирование

Тестирование: Список проектов

Сергей Трофимович с помощью `make --shuffle` обнаружил условия гонок в 29 проектах с открытым исходным кодом, включая:

1. Vim
2. GCC
3. strace
4. Ispell

`https://trofi.github.io/posts/249-an-update-on-make-shuffle.html`

Тестирование: Vim

- Сообщаемая ошибка сборки Vim:

```
cp: cannot create regular file '.../bin/vimtutor': No such file or directory
make[1]: *** [Makefile:2546: installtutorbin] Error 1
make[1]: Leaving directory '/build/source/src'
```

Тестирование: Vim

- Сообщаемая ошибка сборки Vim:

```
cp: cannot create regular file '.../bin/vimtutor': No such file or directory
make[1]: *** [Makefile:2546: installtutorbin] Error 1
make[1]: Leaving directory '/build/source/src'
```

- `bin/vimtutor` кажется создается слишком поздно для цели **`installtutorbin`**.
- Журнал событий:

```
target inst_dir/bin has performed write access on 'inst_dir/bin'...
target installtutorbin has performed dir_lookup access 'inst_dir/bin'...
```

Тестирование: Vim

- Сообщаемая ошибка сборки Vim:

```
cp: cannot create regular file '.../bin/vimtutor': No such file or directory
make[1]: *** [Makefile:2546: installtutorbin] Error 1
make[1]: Leaving directory '/build/source/src'
```

- `bin/vimtutor` кажется создается слишком поздно для цели **`installtutorbin`**.
- Журнал событий:

```
target inst_dir/bin has performed write access on 'inst_dir/bin'...
target installtutorbin has performed dir_lookup access 'inst_dir/bin'...
```

- Цели **`inst_dir/bin`** и **`installtutorbin`** являются **неупорядоченными**
- Санитайзер сообщил об гонке:

```
race found on file 'inst_dir/bin':
 - write at target inst_dir/bin
 - dir_lookup at target installtutorbin
```

Тестирование: Другие гонки в Vim

- Санитайзер также сообщил о ранее неизвестных гонках:

```
race found at file 'src/po/LINGUAS':
 - write at target gvim.desktop
 - write at target vim.desktop
```

Тестирование: Другие гонки в Vim

- Санитайзер также сообщил о ранее неизвестных гонках:

```
race found at file 'src/po/LINGUAS':
 - write at target gvim.desktop
 - write at target vim.desktop
```

- Оказалось, что LINGUAS является временным файлом, используемым в двух независимых целях:

```
216 vim.desktop: ...
217         echo ... > LINGUAS
218         $(MSGFMT) ...
219         rm -f LINGUAS
220
221 gvim.desktop: ...
222         echo ... > LINGUAS
223         $(MSGFMT) ...
224         rm -f LINGUAS
```

Тестирование: Другие гонки в Vim

- Санитайзер также сообщил о ранее неизвестных гонках:

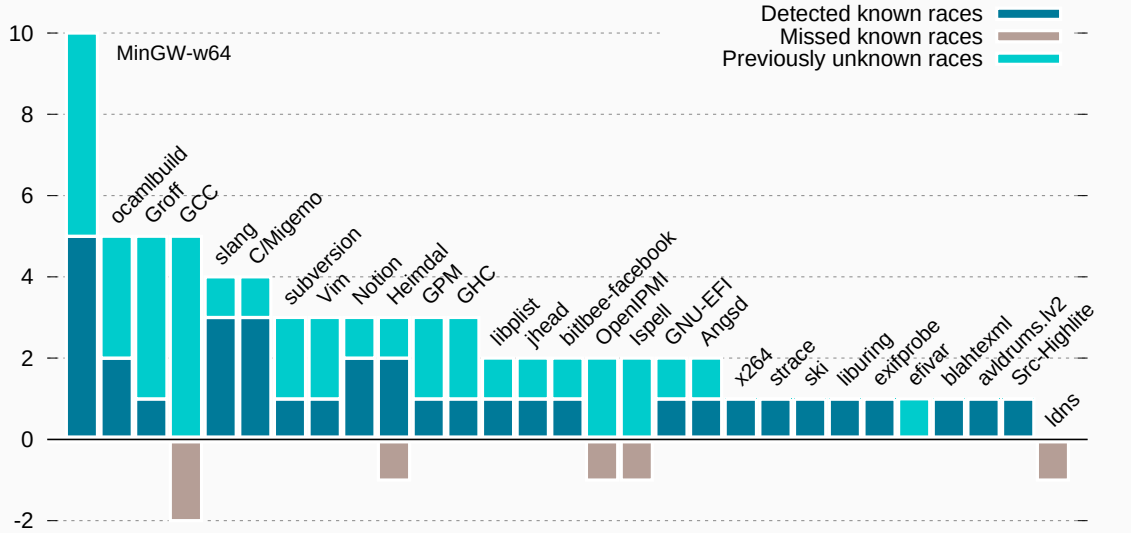
```
race found at file 'src/po/LINGUAS':
 - write at target gvim.desktop
 - write at target vim.desktop
```

- Оказалось, что LINGUAS является временным файлом, используемым в двух независимых целях:

```
216 vim.desktop: ...
217         echo ... > LINGUAS
218         $(MSGFMT) ...
219         rm -f LINGUAS
220
221 gvim.desktop: ...
222         echo ... > LINGUAS
223         $(MSGFMT) ...
224         rm -f LINGUAS
```

- Эта гонка не может быть обнаружена с помощью `make --shuffle`

Тестирование: Результаты



Тестирование: Выводы

Новый инструмент продемонстрировал свою надежность и помог достичь цели исследования. Он был назван **parmasan** — **Par**allel **ma**ke **san**itizer.

Дальнейшие улучшения:

- Улучшение учета символических ссылок в алгоритме поиска гонок;
- Интеграция инструмента в системы непрерывной интеграции;
- Добавление поддержки системы сборки **Ninja**;
- Поддержка внешних отладчиков.

Репозитории:

- `https://github.com/ispras/parmasan`
- `https://github.com/ispras/parmasan-remake`