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.
- **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

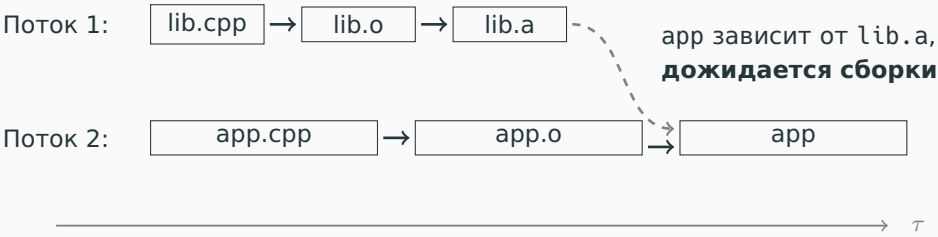
Пример: Два потока пытаются увеличить переменную `counter`, если она меньше 1.

```
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.

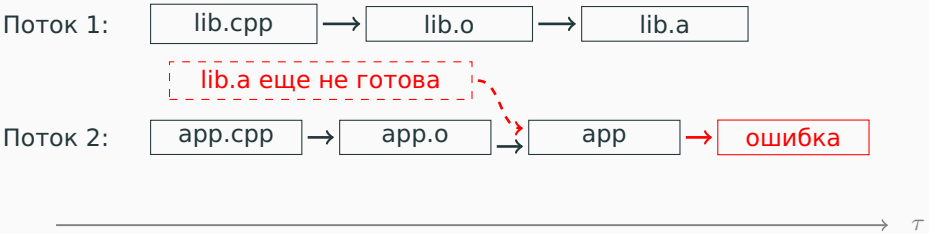
Введение: Состояния гонки в схемах сборки

- Системы сборки могут компилировать несколько целей одновременно.
- Если две цели взаимодействуют с одним и тем же файлом во время параллельной сборки, может возникнуть состояние гонки.



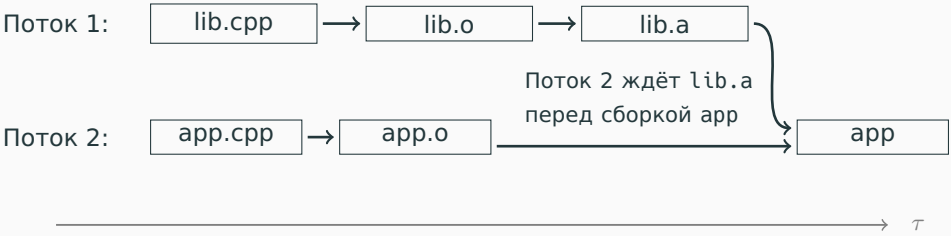
Введение: Состояния гонки в схемах сборки

- Если сборка библиотеки затянется, произойдет ошибка:



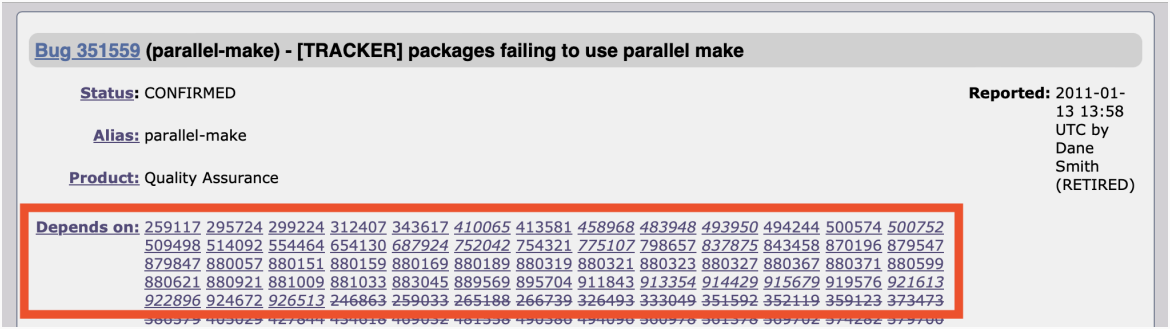
Введение: Состояния гонки в схемах сборки

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



Введение: Состояния гонок в реальных проектах

бки сборки, связанные с состояниями гонок, часто становятся предметом обсуждения на фору



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 могут быть **трудны для отладки:**

- Сценарий гонки может редко воспроизводиться.
- `make --shuffle` — случайная переупорядочивание независимых целей. Это помогает, но не гарантирует обнаружение гонки.

Введение: Формулировка проблемы

Состояния гонок в схемах сборки могут быть **трудны для отладки:**

- Сценарий гонки может редко воспроизводиться.
- `make --shuffle` — случайная переупорядочивание независимых целей. Это помогает, но не гарантирует обнаружение гонки.

Цель исследования: Облегчить для разработчиков поиск и отладку состояний гонок в их схемах сборки.

В качестве целевой системы сборки была выбрана Make из-за ее популярности.

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 путь к файлу.**
- **Гонка между созданием каталога и файлом в нем.**

Теоретический метод: Гонка на содержимом файла

- **Гонка на содержимом файла.**
- **Гонка на путь 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'`.

Исправление: Сделать `append_b` зависимым от `write_a`

Теоретический метод: Гонка на содержимом файла

Как можно обнаружить такие состояния гонок?

Теоретический метод: Гонка на содержимом файла

Запустите сборку под **strace**.

Strace - это 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 +++
...
```

Журнал strace показывает, что `file.out` был открыт из двух разных процессов для записи и дозаписи (как флаги `O_TRUNC` и `O_APPEND` указывают).

Теоретический метод: Гонка на содержимом файла

Путем обработки журнала и сопоставления процессов с целями можно получить более информативный журнал событий.

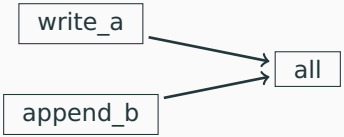
```
...
target write_a has performed write access on 'file.out'
...
target append_b has performed read-write access on 'file.out'
```

Теоретический метод: Гонка на содержимом файла

Путем обработки журнала и сопоставления процессов с целями можно получить более информативный журнал событий.

```
...
target write_a has performed write access on 'file.out'
...
target append_b has performed read-write access on 'file.out'
```

Граф зависимостей схемы сборки выглядит следующим образом:

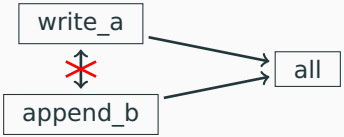


Теоретический метод: Гонка на содержимом файла

Путем обработки журнала и сопоставления процессов с целями можно получить более информативный журнал событий.

```
...
target write_a has performed write access on 'file.out'
...
target append_b has performed read-write access on 'file.out'
```

Граф зависимостей схемы сборки выглядит следующим образом:



Так как между `write_a` и `append_b` нет ориентированного пути, цели являются независимыми. Можно сообщить о гонке.

Теоретический метод: Гонка на содержимом файла

Обзор алгоритма:

1. Получить журнал событий:
   - Запустите сборку под strace.
   - Сопоставьте процессы с целями.
   - Переформатируйте журнал.
2. Получите граф зависимостей схемы сборки.
3. Для каждой пары конфликтующих доступов к одному и тому же пути файла :
   - **Найти зависимость** между соответствующими целями.
   - **Если отсутствует**, сообщить о **состоянии гонки**.

Теоретический метод: Гонка на содержимом файла

Обзор алгоритма:

1. Получить журнал событий:
   - Запустите сборку под strace.
   - Сопоставьте процессы с целями. — Как?
   - Переформатируйте журнал.
2. Получите граф зависимостей схемы сборки. — Как?
3. Для каждой пары конфликтующих доступов к одному и тому же `пути файла` :
   - **Найти зависимость** между соответствующими целями.
   - **Если отсутствует**, сообщить о **состоянии гонки**.

Теоретический метод: Гонка на содержимом файла

Обзор алгоритма:

1. Получить журнал событий:
   - Запустите сборку под strace.
   - Сопоставьте процессы с целями. — Как?
   - Переформатируйте журнал.
2. Получите граф зависимостей схемы сборки. — Как?
3. Для каждой пары конфликтующих доступов к одному и тому же пути файла :
   — Что насчет жестких ссылок?
   - **Найти зависимость** между соответствующими целями.
   - **Если отсутствует**, сообщить о **состоянии гонки**.

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`.

```
1  all: file.out write_a append_b
2  file.out:
3      touch file.out
4      ln -f file.out hardlink_a
5      ln -f file.out hardlink_b
6
7  write_a: file.out
8      echo 'a' > hardlink_a
9
10 append_b: file.out
11     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 событий для приведенного выше примера:

```
...
target file.out has performed write access on 'file.out'
...
target write_a has performed write access on 'hardlink_a'
...
target append_b has performed read-write access on 'hardlink_b'
```

Все три пути в журнале событий различаются. Состояние гонки не обнаружена.

Теоретический метод: Гонка на содержимом файла

Журнал событий для приведенного выше примера:

```
...
target file.out has performed write access on 'file.out'
...
target write_a has performed write access on 'hardlink_a'
...
target append_b has performed read-write access on 'hardlink_b'
```

Все три пути в журнале событий различаются. Состояние гонки не обнаружена.

Исправление: Используйте **номера inode** вместо путей к файлам.

Теоретический метод: Гонка на содержимом файла

inode - это уникальный идентификатор файла в файловой системе. Он гарантированно будет иметь одно и то же значение для жестких ссылок на один и тот же файл.

Замените пути к файлам на номера inode:

```
...
target file.out has performed write access on inum 100000
...
target write_a has performed write access on inum 100000
...
target append_b has performed read-write access on inum 100000
```

Теоретический метод: Гонка на содержимом файла

Обновленный алгоритм:

1. Получить лог событий
   - Запустить сборку под strace с логированием номера inode.
   - Сопоставьте процессы с целями.
   - Переформатируйте журнал.
2. Получите граф зависимостей схемы сборки.
3. Для каждой пары конфликтующих доступов к одному и тому же ~~пути файла~~ номеру inode:
   - Найдите зависимость между соответствующими целями.
   - Если отсутствует, сообщить о состоянии гонки.

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

Примечание: номера inode могут быть повторно использованы.

Теоретический метод: Гонка на путь к файлу

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

Теоретический метод: Гонка на путь 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 для хранения некоторых промежуточных данных. Он удаляется после сборки цели.

То же имя используется для хранения промежуточных данных в другой независимой цели.

Это состояние гонка, потому что команда `rm` может быть выполнена, когда другая цель все еще использует файл.

Теоретический метод: Гонка на путь к файлу

Журнал событий для приведенного выше примера:

```
...
target write_a has performed write access on inum 100000
target write_a has performed unlink access on inum 100000
...
target append_b has performed write access on inum 100001
target append_b has performed unlink access on inum 100001
```

Файл `tmp_file` изменил свой номер inode после его удаления и повторного создания. Алгоритм, основанный на номерах inode, не сможет найти гонку здесь.

Теоретический метод: Гонка на путь к файлу

Журнал событий для приведенного выше примера:

```
...
target write_a has performed write access on inum 100000
target write_a has performed unlink access on inum 100000
...
target append_b has performed write access on inum 100001
target append_b has performed unlink access on inum 100001
```

Файл `tmp_file` изменил свой номер inode после его удаления и повторного создания. Алгоритм, основанный на номерах inode, не сможет найти гонку здесь.

Исправление: Используйте пути к файлам в дополнение к номерам inode только для обнаружения гонок, включающих удаление файла.

Теоретический метод: Гонка на каталог

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

Теоретический метод: Гонка на каталог

Другой распространенный шаблон состояния гонок:

```
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 команда `echo` будет выполнена до команды `mkdir`, произойдет ошибка.

Теоретический метод: Гонка на каталог

Другой распространенный шаблон состояния гонок:

```
all: build build/file.out

build:
    mkdir -p build

build/a.out:
    echo "a" > build/a.out
```

Если команда `echo` будет выполнена до команды `mkdir`, произойдет ошибка.

Для обнаружения таких состояний гонок необходимо записывать два события для каждого доступа к файлу:

- Сам доступ к файлу
- Специальное событие **`dir_lookup`** для родительского каталога

Новый журнал событий:

```
build performed write on 'build'...
build/a.out performed dir_lookup on 'build'...
build/a.out performed unlink on 'build/a.out'...
```

Теперь алгоритм, основанный на номерах inode, обнаружит гонку на каталоге `build`

Теоретический метод: Гонка на каталог

Проблема: В некоторых проектах каталог сборки создается в каждой цели отдельно с помощью `mkdir -p`:

```
all: library_1 ... library_9

library_1:
    mkdir -p build
    build_library build/lib1.a
...

library_9:
    mkdir -p build
    build_library build/lib9.a
```

Теоретический метод: Гонка на каталог

Проблема: В некоторых проектах каталог сборки создается в каждой цели отдельно с помощью `mkdir -p`:

```
1  all: library_1 ... library_9
2
3  library_1:
4      mkdir -p build
5      build_library build/lib1.a
6  ...
7
8  library_9:
9      mkdir -p build
10     build_library build/lib9.a
```

Следствие: Записывается несколько событий `write` для одного и того же каталога. Только первое успешно. Остальные завершаются с ошибкой EEXIST.

```
library_1 performed write on 'build'...
library_1 performed dir_lookup on 'build'...
library_1 performed write on 'build/lib1.a'...

library_2 failed write on 'build'...
library_2 performed dir_lookup on 'build'...
library_2 performed write on 'build/lib2.a'...
...
```

Теоретический метод: Гонка на каталог

Проблема: В некоторых проектах каталог сборки создается в каждой цели отдельно с помощью `mkdir -p`:

```
1  all: library_1 ... library_9
2
3  library_1:
4      mkdir -p build
5      build_library build/lib1.a
6  ...
7
8  library_9:
9      mkdir -p build
10     build_library build/lib9.a
```

Следствие: Записывается несколько событий `write` для одного и того же каталога. Только первое успешно. Остальные завершаются с ошибкой EEXIST.

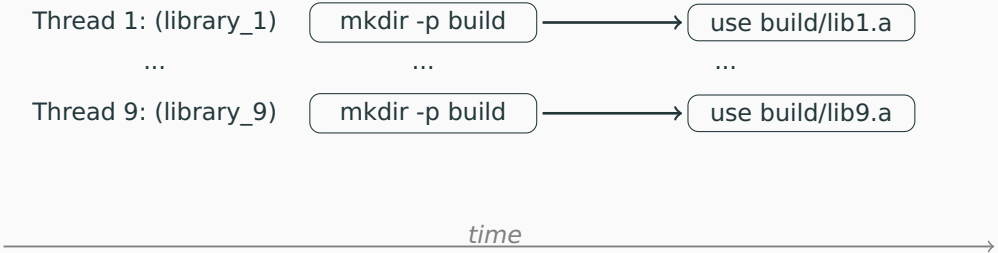
```
library_1 performed write on 'build'...
library_1 performed dir_lookup on 'build'...
library_1 performed write on 'build/lib1.a'...

library_2 failed write on 'build'...
library_2 performed dir_lookup on 'build'...
library_2 performed write on 'build/lib2.a'...
...
```

Алгоритм сообщит о ложной гонке для любой пары доступов к каталогу `build` из разных целей `library_n`. **Как избежать ложных положительных**

Теоретический метод: Гонка на каталог

Решение: Создайте отдельный алгоритм для обнаружения состояний гонок с использованием **`dir_lookup`**:

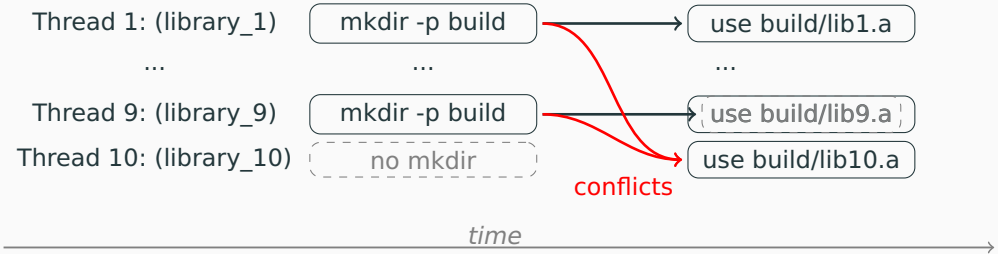
- Все попытки создать каталог сохраняются в наборе.
- Состояние гонки сообщается, если событие **`dir_lookup`** для этого каталога конфликтует со **всеми** попытками создания.



Теоретический метод: Гонка на каталог

Решение: Создайте отдельный алгоритм для обнаружения состояний гонок с использованием **`dir_lookup`**:

- Все попытки создать каталог сохраняются в наборе.
- Состояние гонки сообщается, если событие **`dir_lookup`** для этого каталога конфликтует со **всеми** попытками создания.



- Команда `mkdir build` не была выполнена для `library_10`, поэтому любое использование директории `build` будет сообщать о гонке.

Реализация

Реализация: Архитектура

- Граф зависимостей, сопоставление процессов с целями: измененный Make.
- Журнал событий: монитор на основе `ptrace` вместо парсинга журнала `strace`.
- Компонент `ptrace` перемещен в отдельный процесс для изоляции процесса сборки от санитайзера.

Реализация: Трейсинг системных вызовов

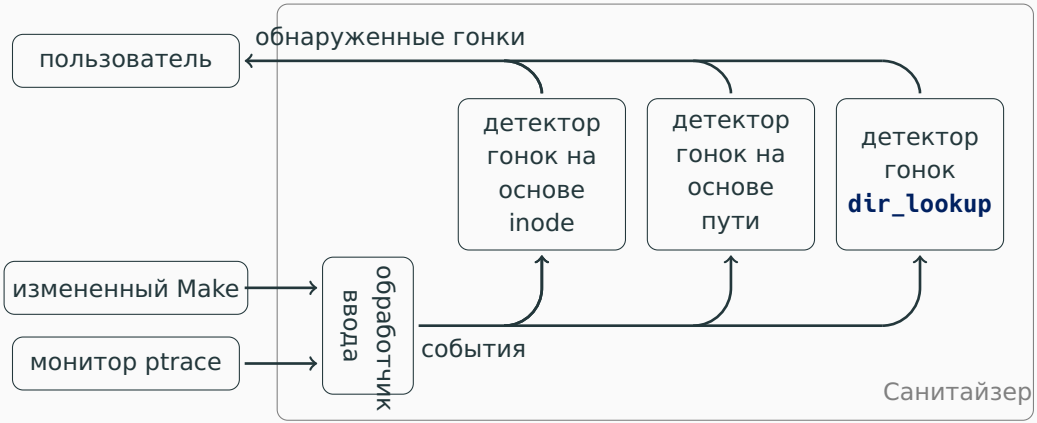
Перехватываемые системные вызовы:

системный вызов	событие для санитайзера
`open(at)(at2)`	доступ на **чтение**, **запись** или **чтение-запись**
`mkdir(at)`	доступ на **запись** к созданному каталогу
`creat`	доступ на **запись**
`rmdir`	доступ на **удаление**
`unlink(at)`	доступ на **удаление**
`rename(at)(at2)`	**удаление** целевого пути, если он существует.

- Каждый доступ сообщается с указанием **процесса**, номера **inode** и **пути к файлу**.

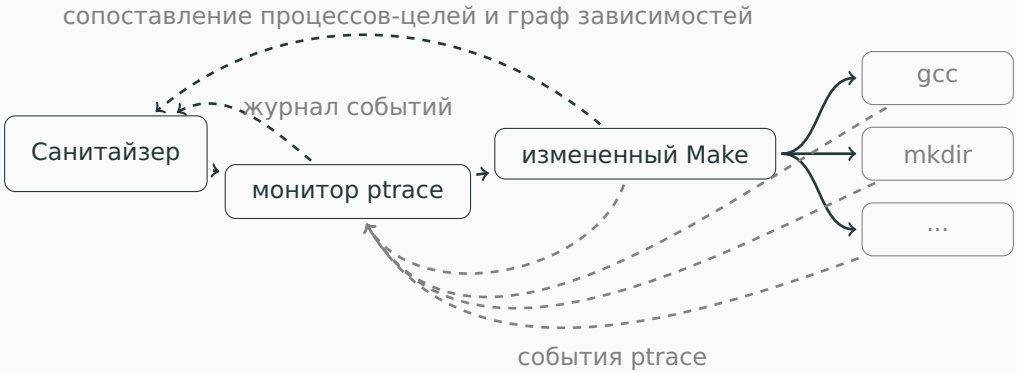
Реализация: Санитайзер

Санитайзер получает все сообщения и обрабатывает их с помощью трех алгоритмов:



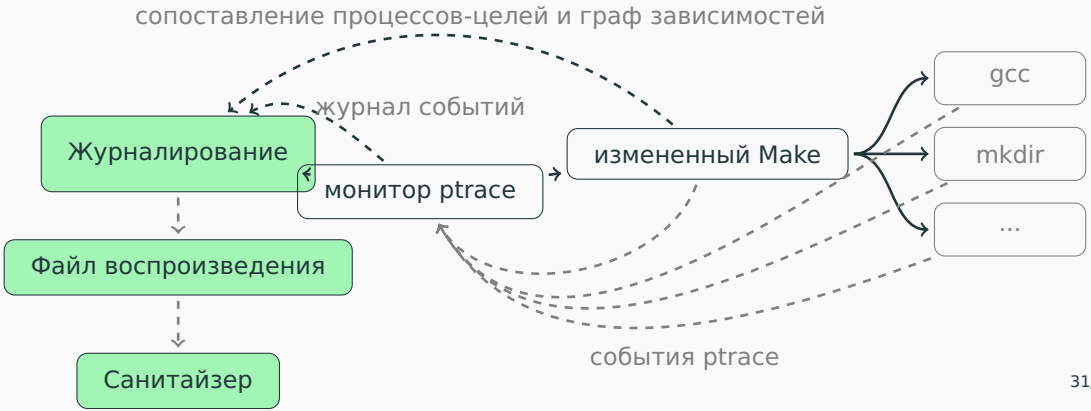
Реализация: Воспроизведение сборки

- Структура инструмента позволяет сохранять журнал сборки в файл и воспроизводить его после.



Реализация: Воспроизведение сборки

- Структура 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:

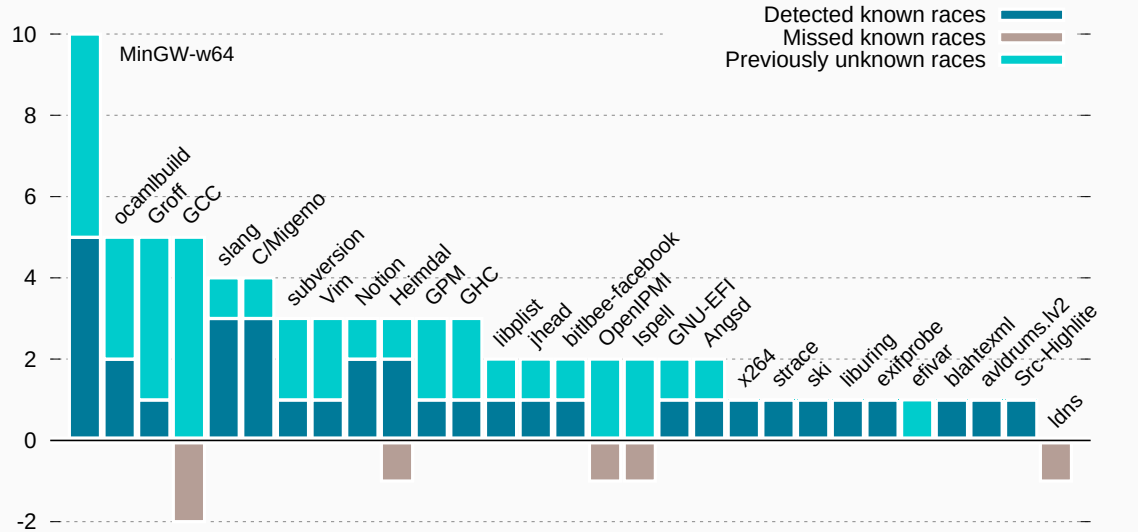
```
race found at file 'src/po/LINGUAS':
 - write at target gvim.desktop
 - write at target vim.desktop
```

- 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 (Tinderbox-cluster)
- Добавление поддержки системы сборки **Ninja**.

Репозитории:

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

Благодарю за внимание!