Автоматическое обнаружение гонок при параллельной сборке с использованием утилиты Make

Артем Климов, Владислав Иванишин, Александр Монаков

20 июня 2024 г.

ИСП РАН

План презентации

- **Введение**. Состояния гонок в Makefile. Примеры и формулировка проблемы.
- **Обзор существующих решений**.
- **Демонстрация** разработанного инструмента в виде сантайзера для Make.
- **Тестирование** решения 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

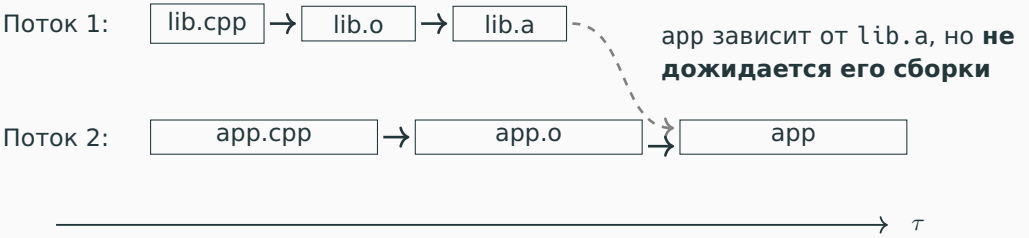
Пример: Два потока пытаются увеличить переменную `counter`, если она меньше 1.

```
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.

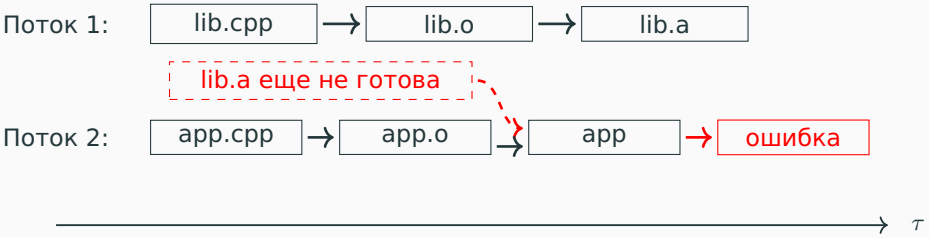
Введение: Состояния гонки в схемах сборки

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



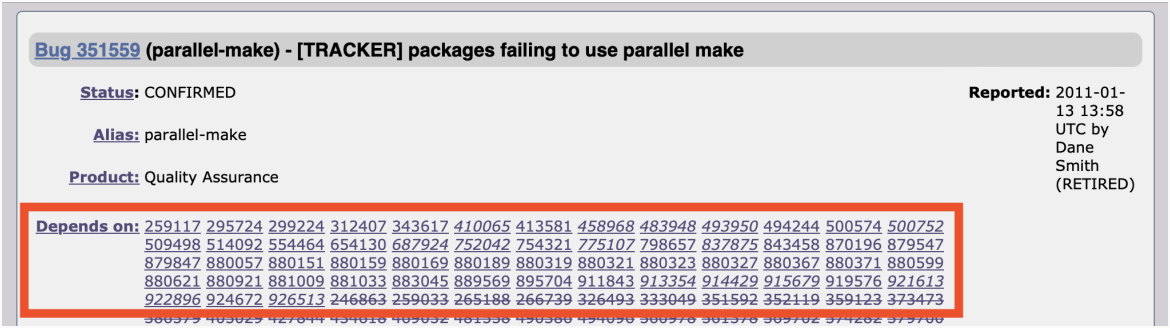
Введение: Состояния гонки в схемах сборки

- Если сборка библиотеки затянется, произойдет ошибка:



Введение: Состояния 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.

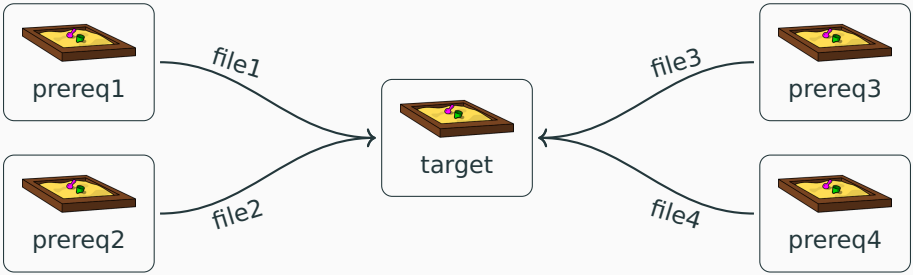
Цель исследования: Разработать инструмент для упрощения поиска и отладки состояний гонок в схемах сборки. Требования к нему:

- 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 песочнице.
- Каждая песочница имеет только файлы, собранные целями-зависимостями.



Обзор существующих решений: make --shuffle

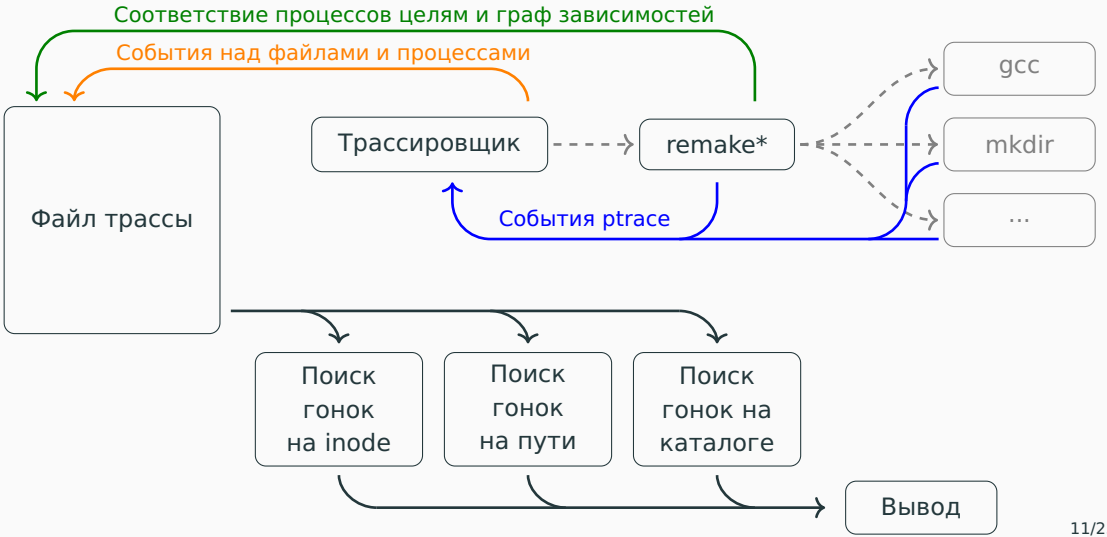
- `make --shuffle` — перемешивает список пререквизитов каждой цели.
- Это увеличивает вероятность проявления гонки, но помогает не всегда.

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

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

Реализация

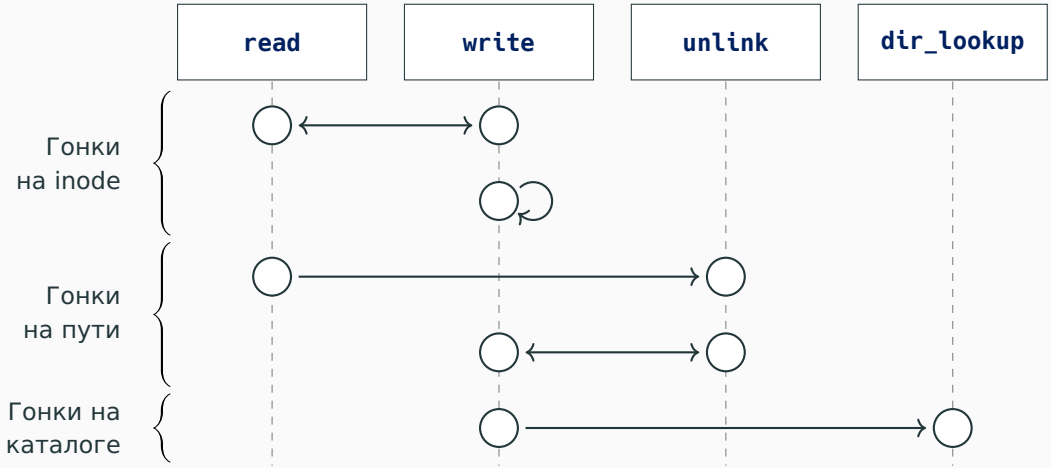
Реализация: Архитектура



Реализация: Пример трассы

```
...
1037: DEPENDENCY   9 exifprobe 10 jpegtags.h
1037: DEPENDENCY   9 exifprobe 8 global.h
1037: DEPENDENCY   9 exifprobe 8 Makefile
1036: CHILD        1157 1037 43 /home/.../make -j 8
1037: TARGET_PID   9 exifprobe 1157
1037: DEPENDENCY   3 all 9 exifprobe
1036: CHILD        1157 1157 38 /bin/sh -c ./mkcomptime > comptime.c
1036: READ         16 /etc/ld.so.cache 1157 2129 725351962 8
1036: READ         31 /lib/.../libc.so.6 1157 2129 190302230 8
1036: WRITE        55 /home/.../comptime.c 1157 2129 725310162 8
1036: CHILD        1158 1157 38 /bin/sh -c ./mkcomptime > comptime.c
1036: CHILD        1158 1158 22 /bin/sh ./mkcomptime
...
```

Реализация: Виды гонок



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.

- Дополнительно отслеживается `fork`, `clone`, `exec`, `spawn`.
- Остальные системные вызовы фильтруются через `seccomp` BPF.

Реализация: Метод критических доступов

- Санитайзер сообщает о гонке путём поиска зависимостей между парами конфликтующих доступов;
- Разработан метод, позволяющий пропускать ненужные проверки, сокращая сложность до линейной.

- Это позволяет санитайзеру работать быстро на проектах любого размера.

Реализация: Особенности реализации

1. Фильтр BPF и метод критических доступов увеличивает скорость работы санитайзера;
2. Санитайзер поддерживает три вида гонок, полагается на номера inode, чтобы находить гонки между жёсткими ссылками;
3. Санитайзер поддерживает установку точек останова и отладку с помощью интерактивной консоли;
4. Отладка всех обнаруженных состояний гонок может производиться на одной и той же записанной трассе, без пересборки проекта.

Тестирование

Тестирование: Пример

```
all: test1 test2 test3 test4 test5 test6 test7 test8

test%:
        mkdir -p tests
        echo $@ > tests/$@

test8:
        # mkdir -p tests - забыт
        echo 'override' > tests/$@
```

- С **`make --shuffle`** гонка проявляется в **12.5% случаев**.
- **Санитайзер** обнаруживает гонку в **100% случаев**.

Тестирование: Список проектов

Сергей Трофимович с помощью `make --shuffle` обнаружил состояния 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

- Ошибка при сборке 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` не существует, хотя требуется целью **`installtutorbin`**.
- Санитайзер сообщил об этой гонке:

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

- Цели **`inst_dir/bin`** и **`installtutorbin`** являются **неупорядоченными**

Тестирование: Другие гонки в Vim

- Санитайзер также сообщил о ранее неизвестных гонках:

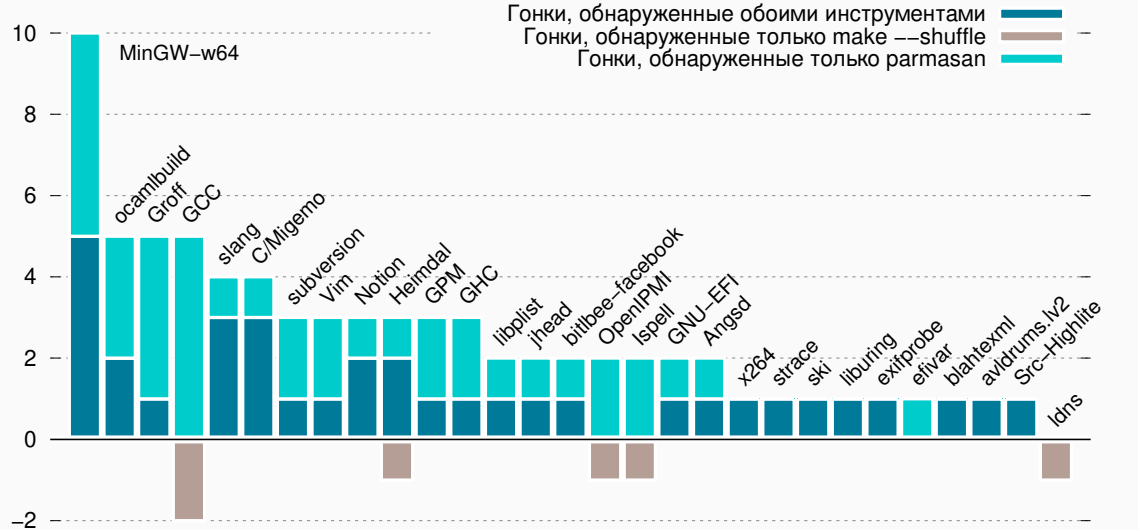
```
race found at file 'src/po/LINGUAS':
 - write at target gvim.desktop
 - write at target vim.desktop
```

- LINGUAS — временный файл, используемый 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
```

- Эту гонку нельзя обнаружить с помощью `make --shuffle`

Тестирование: Результаты



Тестирование: Замедление

- Санитайзер замедляет сборку проекта на 12-20% вне зависимости от его размера.

Программа	Время сборки	Время сборки с санитайзером	Замедление
vim	0:18, 3	0:20, 7	13, 1%
x264	0:29, 6	0:33, 0	11, 5%
exifprobe	0:01, 36	0:01, 53	12, 6%
angsd	0:21, 3	0:23, 8	11, 7%
gcc	50:28, 0	60:53, 5	20, 6%

Тестирование: Выводы

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

Дальнейшие улучшения:

- Улучшение учета символьных 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`