Десятая независимая научно-практическая конференция
«Разработка ПО 2014»
23 - 25 октября, Москва

Визуализация динамики параллельных программ для анализа поведения и поиска ошибок

Половцев А.М., Крикун Т.С., Верт Т.С., Зозуля А.В., Ицыксон В.М.

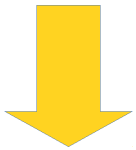
СПбПУ

Актуальность решаемой задачи

- Рост размера программ
- Программы состоят из множества процессов и/или потоков

Актуальность решаемой задачи

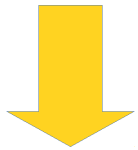
- Рост размера программ
- Программы состоят из множества процессов и/или потоков



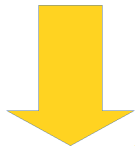
- Увеличение количества ошибок (утечки памяти и ресурсов, deadlock, race condition)
- Увеличение сложности поиска ошибок и отладки

Актуальность решаемой задачи

- Рост размера программ
- Программы состоят из множества процессов и/или потоков



- Увеличение количества ошибок (утечки памяти и ресурсов, deadlock, race condition)
- Увеличение сложности поиска ошибок и отладки



- Необходимо средство для обнаружения причин возникновения ошибок и их устранения

Способы решения

- Тестирование

Способы решения

- Тестирование
  - Слишком сложно выявить ошибки из-за недетерминированности поведения

Способы решения

- Тестирование
  - Слишком сложно выявить ошибки из-за недетерминированности поведения
- Верификация и статический анализ

Способы решения

- Тестирование
  - Слишком сложно выявить ошибки из-за недетерминированности поведения
- Верификация и статический анализ
  - Слишком ресурсоемко из-за огромного количества состояний

Способы решения

- Тестирование
  - Слишком сложно выявить ошибки из-за недетерминированности поведения
- Верификация и статический анализ
  - Слишком ресурсоемко из-за огромного количества состояний

Данные методы ограничены в использовании

Динамическая визуализация

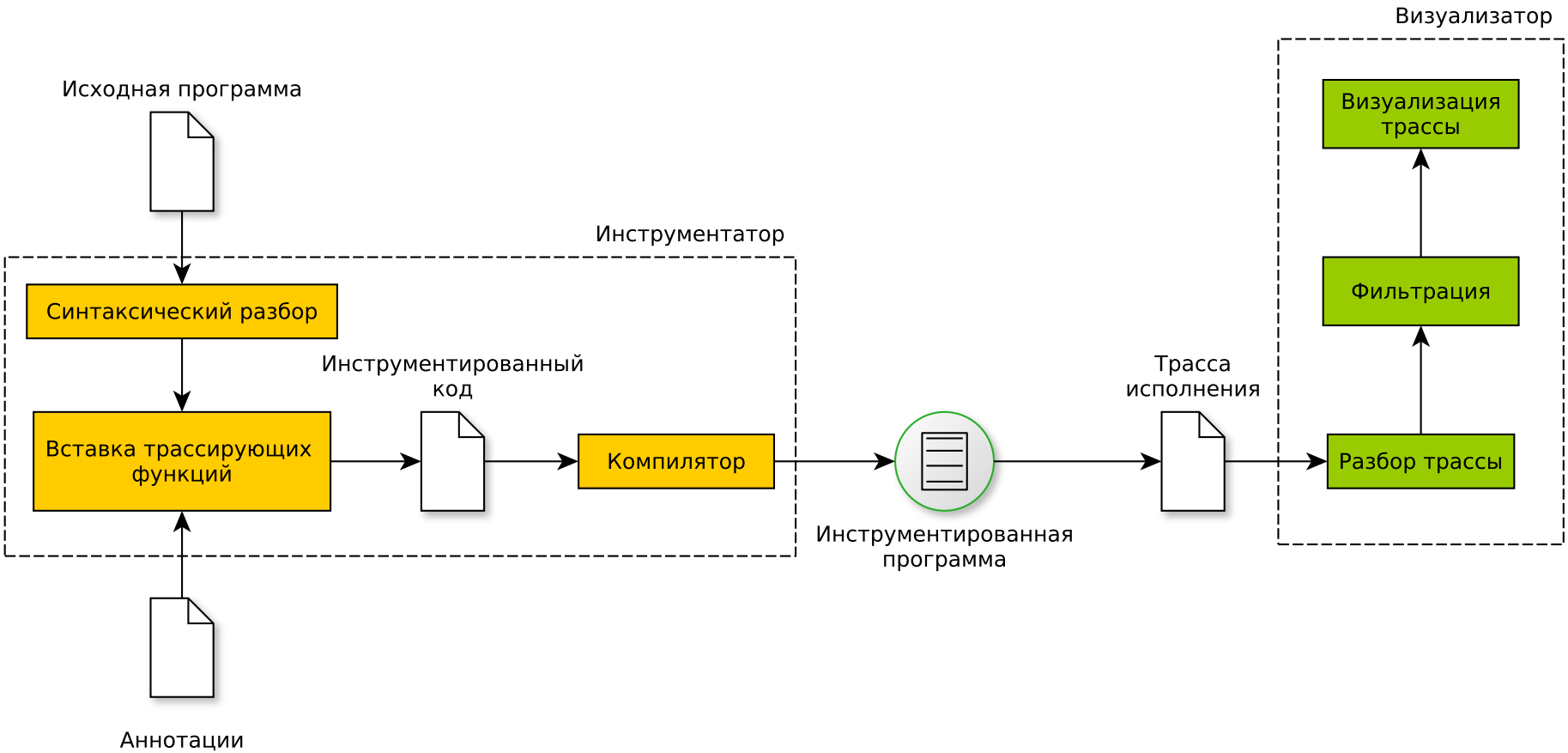
- Динамический анализ на основе трасс исполнения
- Модификация исходного кода для вставки трассирующих функций
- Трасса может быть очень большой $\rightarrow$ необходима интерпретация

Динамическая визуализация

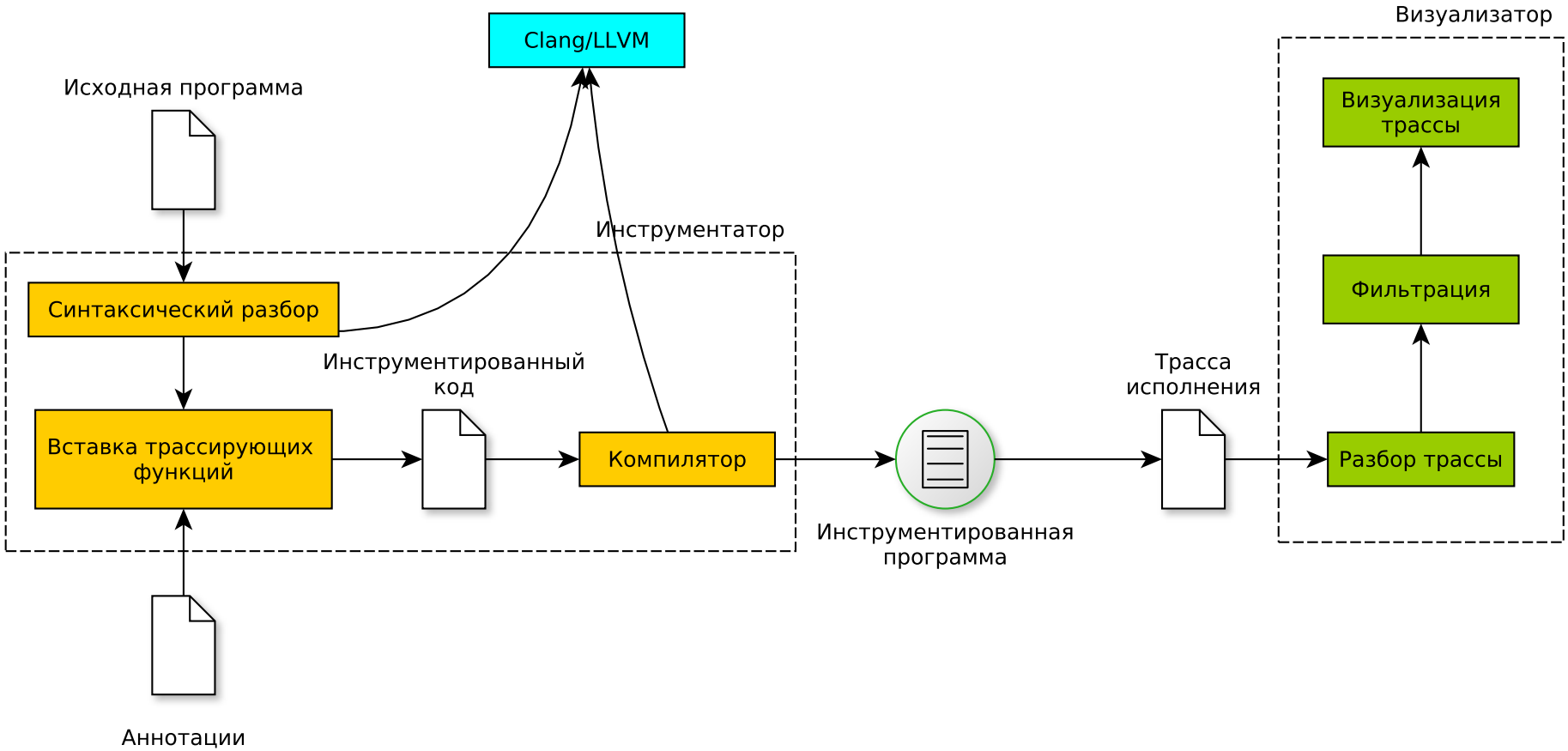
- Динамический анализ на основе трасс исполнения
- Модификация исходного кода для вставки трассирующих функций
- Трасса может быть очень большой $\rightarrow$ необходима интерпретация

- Объекты программы (ресурсы, потоки) $\rightarrow$ графические объекты
- Взаимодействие между объектами $\rightarrow$ анимация

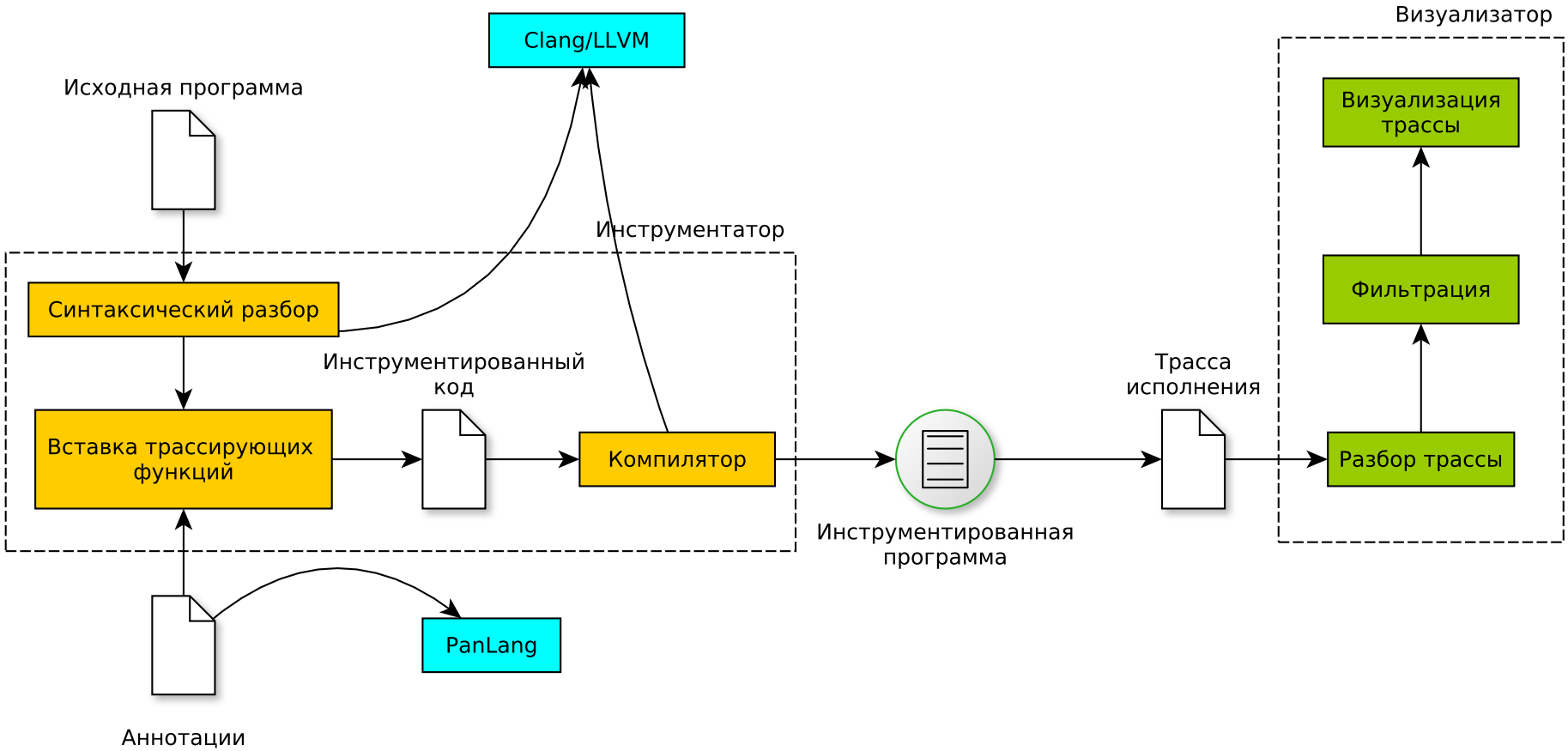
Структура разработанного средства



Структура разработанного средства



Структура разработанного средства



Отображаемые объекты

- Функции для работы с системными ресурсами
  - Создание/удаление потоков
  - Работа с мьютексами/семафорами
  - Работа с файлами
- Вызовы функций
- Точки входа/выхода из программы
- Начало/конец циклов

Аннотации

- Как описать вид воздействия вызова функции на системный ресурс?

Аннотации

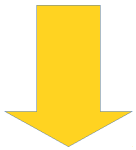
- Как описать вид воздействия вызова функции на системный ресурс?
- Все ли функции необходимо инструментировать?

Аннотации

- Как описать вид воздействия вызова функции на системный ресурс?
- Все ли функции необходимо инструментировать?

Аннотации

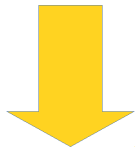
- Как описать вид воздействия вызова функции на системный ресурс?
- Все ли функции необходимо инструментировать?



- Можно задавать семантику в исходном коде инструментатора

Аннотации

- Как описать вид воздействия вызова функции на системный ресурс?
- Все ли функции необходимо инструментировать?



- Можно задавать семантику в исходном коде инструментатора
- Параметризируемый подход - язык аннотаций PanLang

PanLang

```
function FILE* fopen(const char* path, const char* mode) {
    action openFile(Result, path);
};

function int flock(int fd, int operation) {
    int op = operation & (~4);
    if (op == 1 || op == 2) {
        action lockFile(fd);
    } else if (op == 8) {
        action unlockFile(fd);
    }
};
```

Пример 1

Было:

```
int main () {
  FILE* pFile = fopen("myfile.bin", "rb");
  return 0;
}
```

Пример 1

Стало:

```
int main () {
  ...
  FILE* pFile = fopen_log("myfile.bin", "rb");
  ...
}
FILE* fopen_log(const char* arg0, const char* arg1) {
  int event_id = getEventUid();
  loginfo(event_id, FILE_OPEN_START, 0, 0, "fopen");
  FILE* result = fopen(arg0, arg1);
  loginfo(event_id, FILE_OPEN_END,
    result ? fileno(result) : 0, 0, "fopen");
  return result;
}
```

Пример 2. Инструментирование variadic-функций

Было:

```
void test(int p, ...) {}

int main () {
    test(1, 2, 3, 4);
    test(1, "str") ;
}
```

Пример 2. Инструментирование variadic-функций

Стало:

```
void test_log_1(int arg0, int arg1, int arg2, int arg3) {
    ...
}
void test_log_2(int arg0, const char* arg1) {
    ...
}

int main () {
    test_log_1(1, 2, 3, 4);
    test_log_2(1, "str") ;
}
```

Пример 3. Инструментирование вызовов по указателю

Было:

```
int foo(int a, int b) {
    return a + b;
}
int bar(int (*fun)(int, int), int a, int b) {
    return fun(a, b);
}

int main() {
    int a = 5, b = 8;
    int (*ptr)(int, int) = &foo;
    int sum = bar(ptr, a, b);
    return sum;
}
```

Пример 3. Инструментирование вызовов по указателю

Стало:

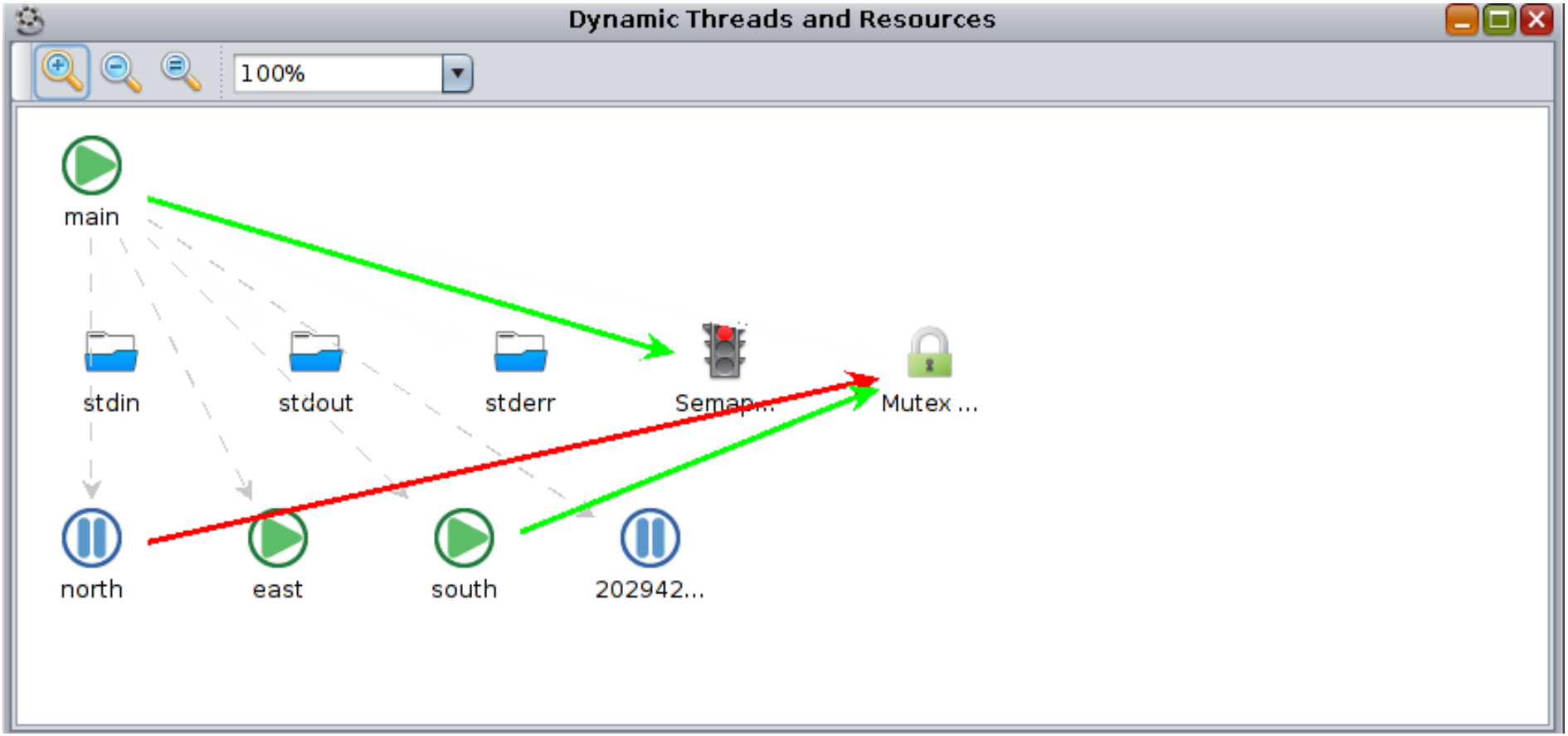
```
int foo_log(int arg0, int arg1) {
    ...
}
int bar_log(int (*arg0)(int, int), int arg1, int arg2) {
    ...
}
int main() {
    int a = 5, b = 8;
    int (*ptr)(int, int) = &foo_log;
    int sum = bar_log(ptr, a, b);
    return sum;
}
```

Визуализация

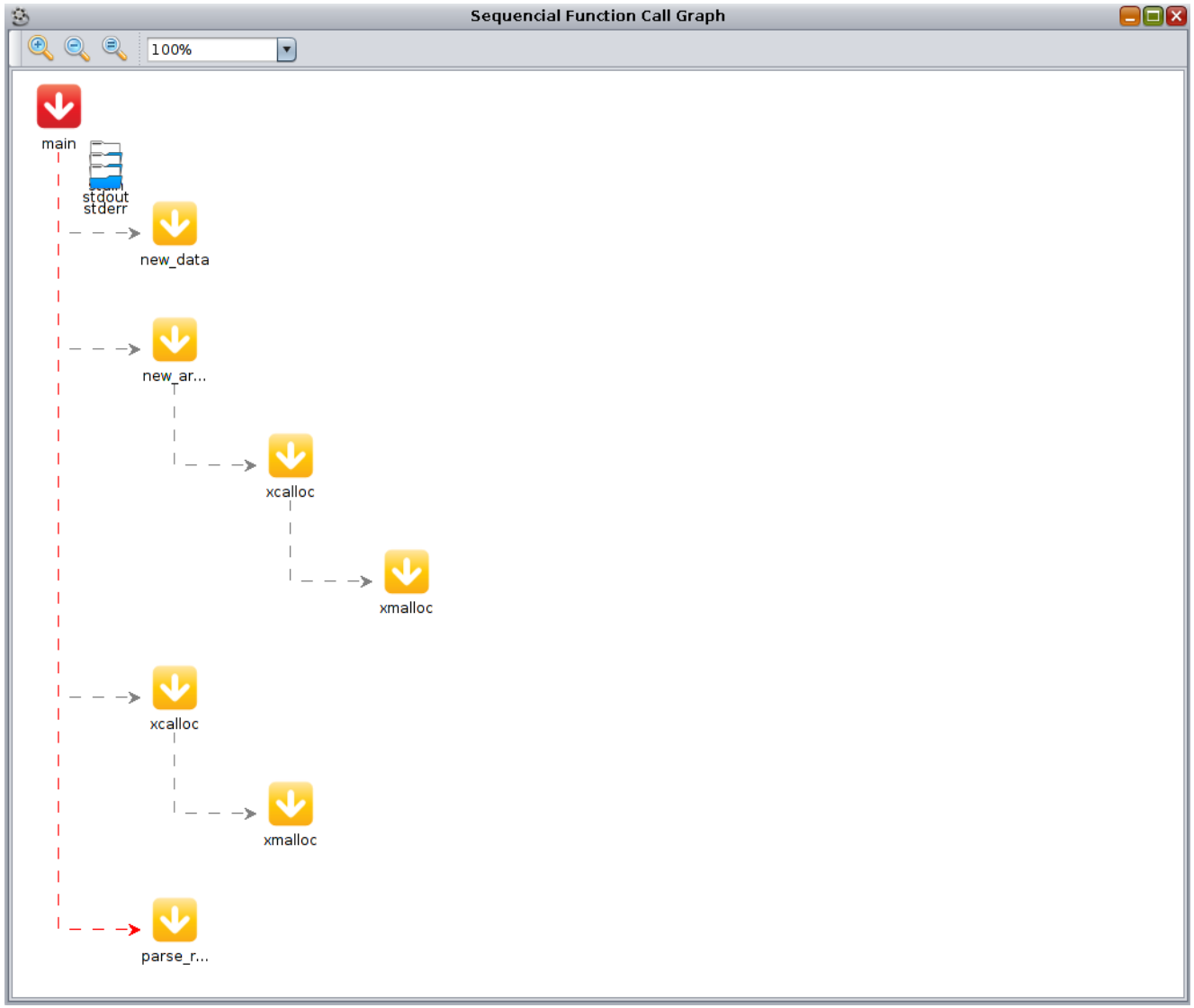
Поддерживаются следующие виды визуализации:

- Анимация создания/удаления потоков
- Анимация создания/освобождения ресурсов и потока, который ими владеет
- Визуализация стека вызовов функций
- Фильтрация отображаемых событий по идентификатору потока или ресурса

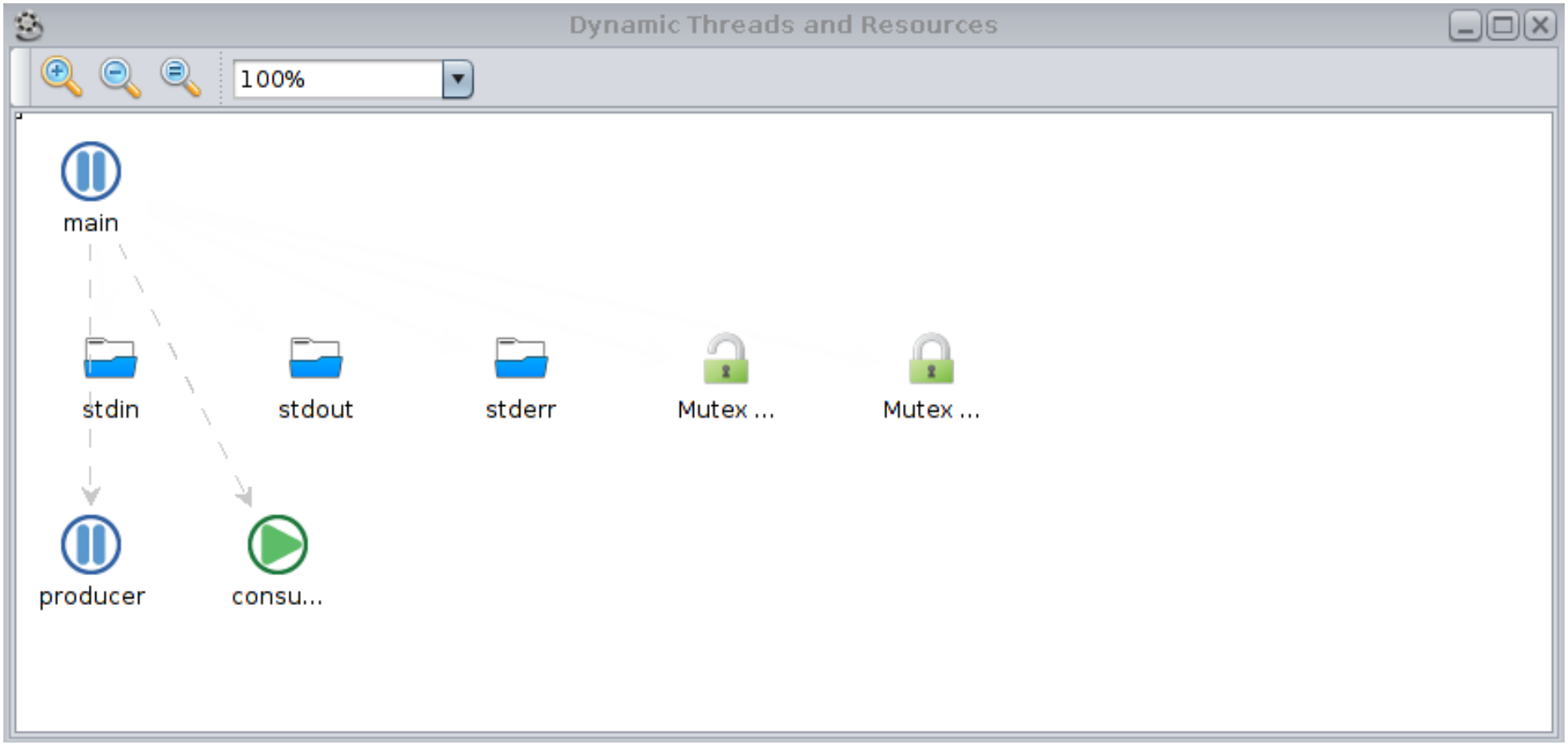
Взаимодействие потоков и ресурсов



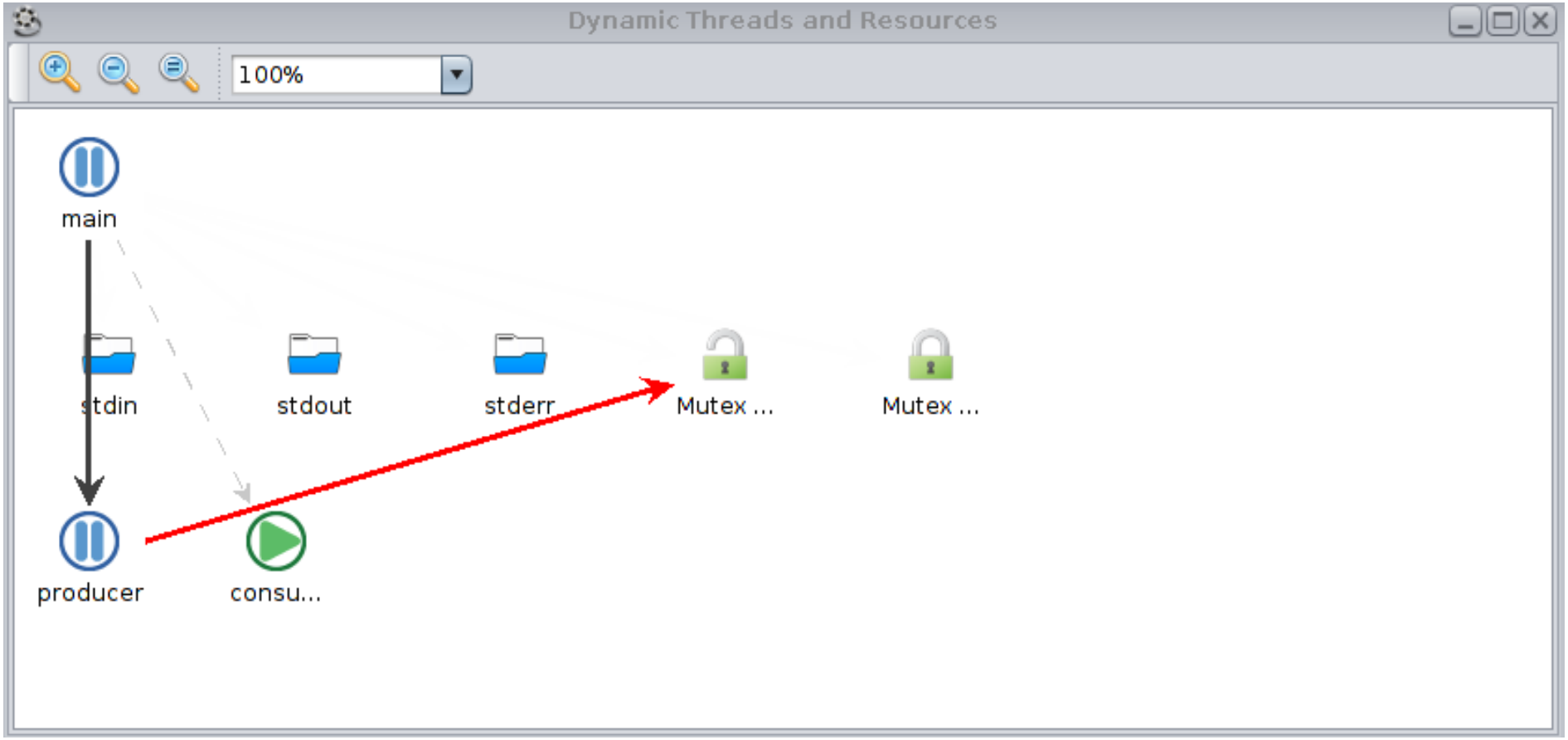
Визуализация стека вызовов функций



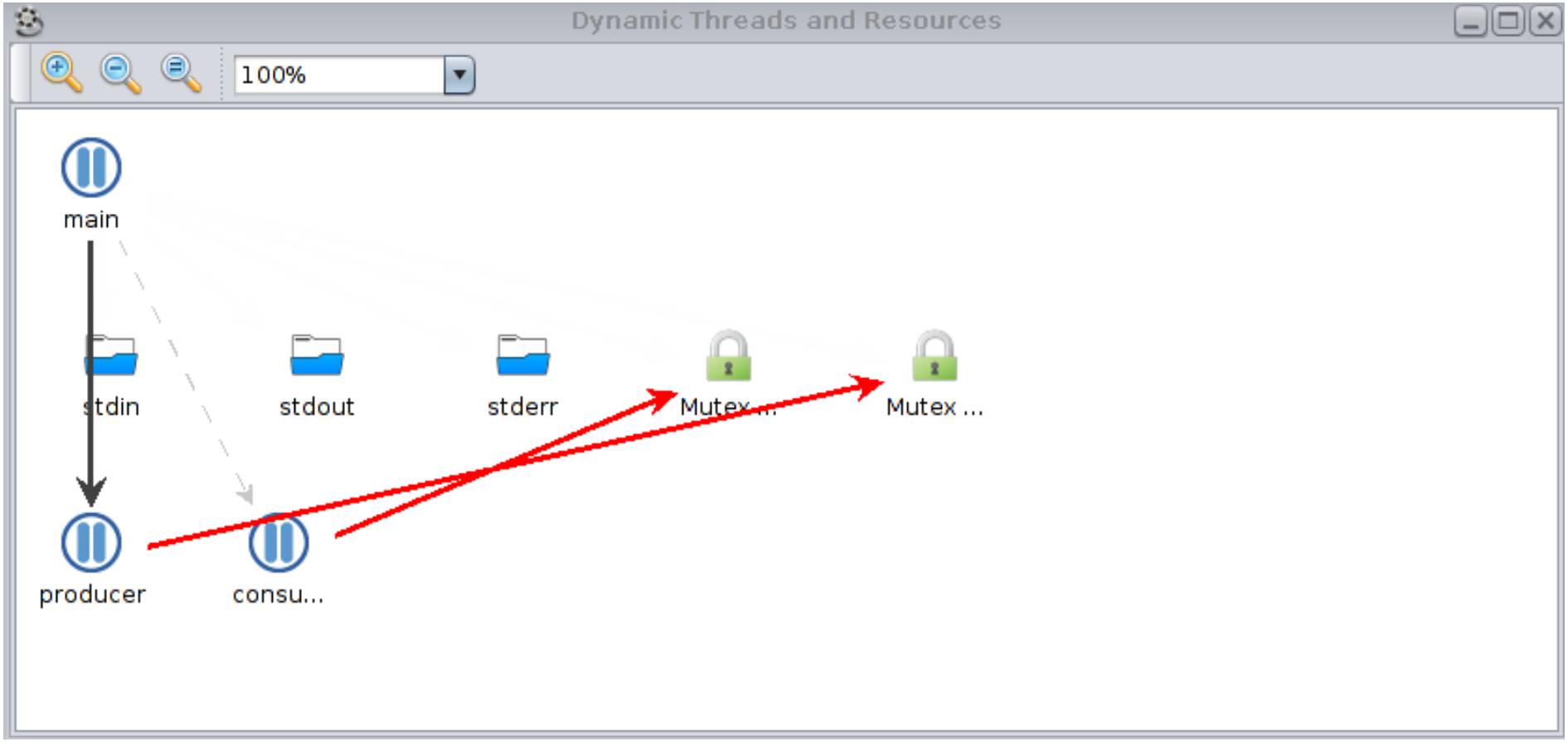
Пример


Dynamic Threads and Resources
100%
main
stdin
stdout
stderr
Mutex ...
Mutex ...
producer
consu...

Пример


Dynamic Threads and Resources
100%
main
stdin
stdout
stderr
Mutex ...
Mutex ...
producer
consu...

Пример


Dynamic Threads and Resources
100%
main
stdin
stdout
stderr
Mutex ...
Mutex ...
producer
consu...

Заключение

- Предложен подход к поиску ошибок в параллельных программах
- Реализовано инструментальное средство и визуализатор трасс исполнения программ
- Реализован метод параметризации процесса инструментирования

Спасибо за внимание

?

Эксперименты

	Объем лога	LOC
ls	3682	699
gzip	589895	8634
wget	5985	39362
mc	1343788	237050
git clone	119772	528657
postgresql		727978