2013, Москва

LinuxTesting.org

Моделирование окружения драйверов операционной системы Linux для поддержки статической верификации

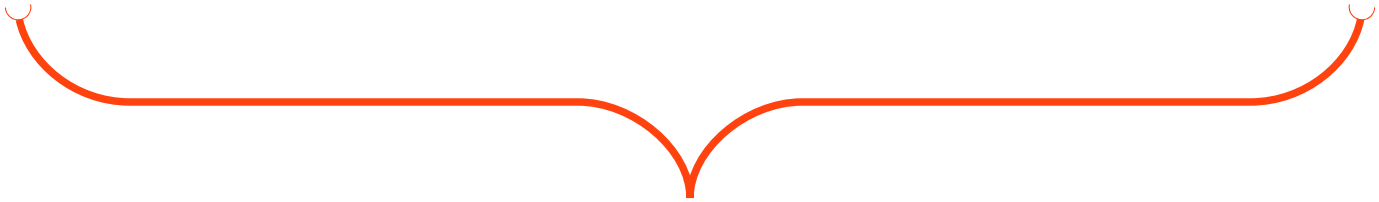
Захаров Илья, студент 5 курса ФУПМ
Ilja.zakharov@ispras.ru

ISP RAS

Institute for System Programming of the Russian Academy of Sciences

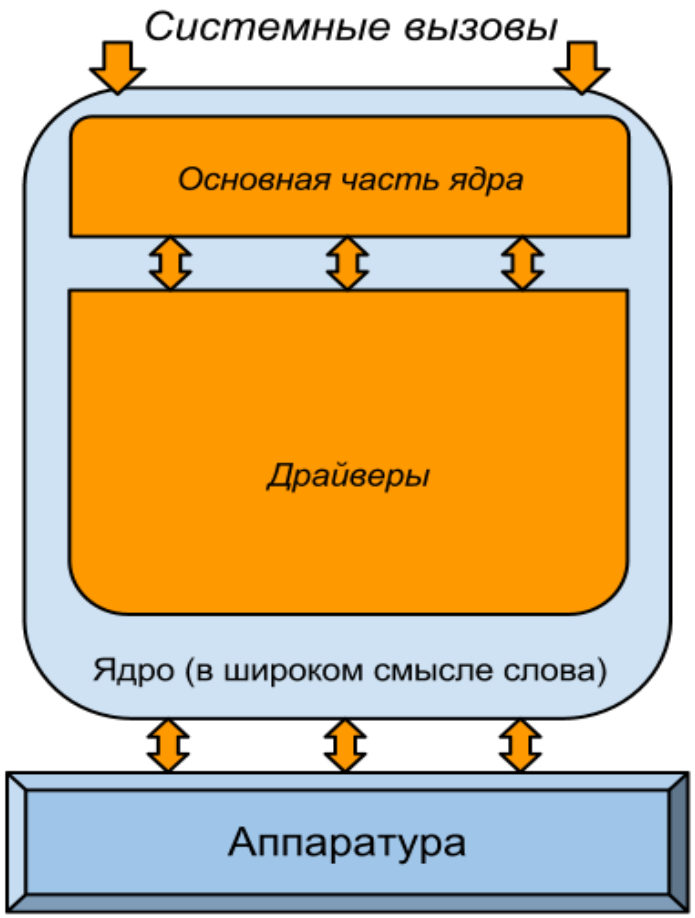
Ядро Linux

- Большое число разработчиков
- Высокая скорость разработки
- Высокая сложность кода
- Высокие требования к корректности



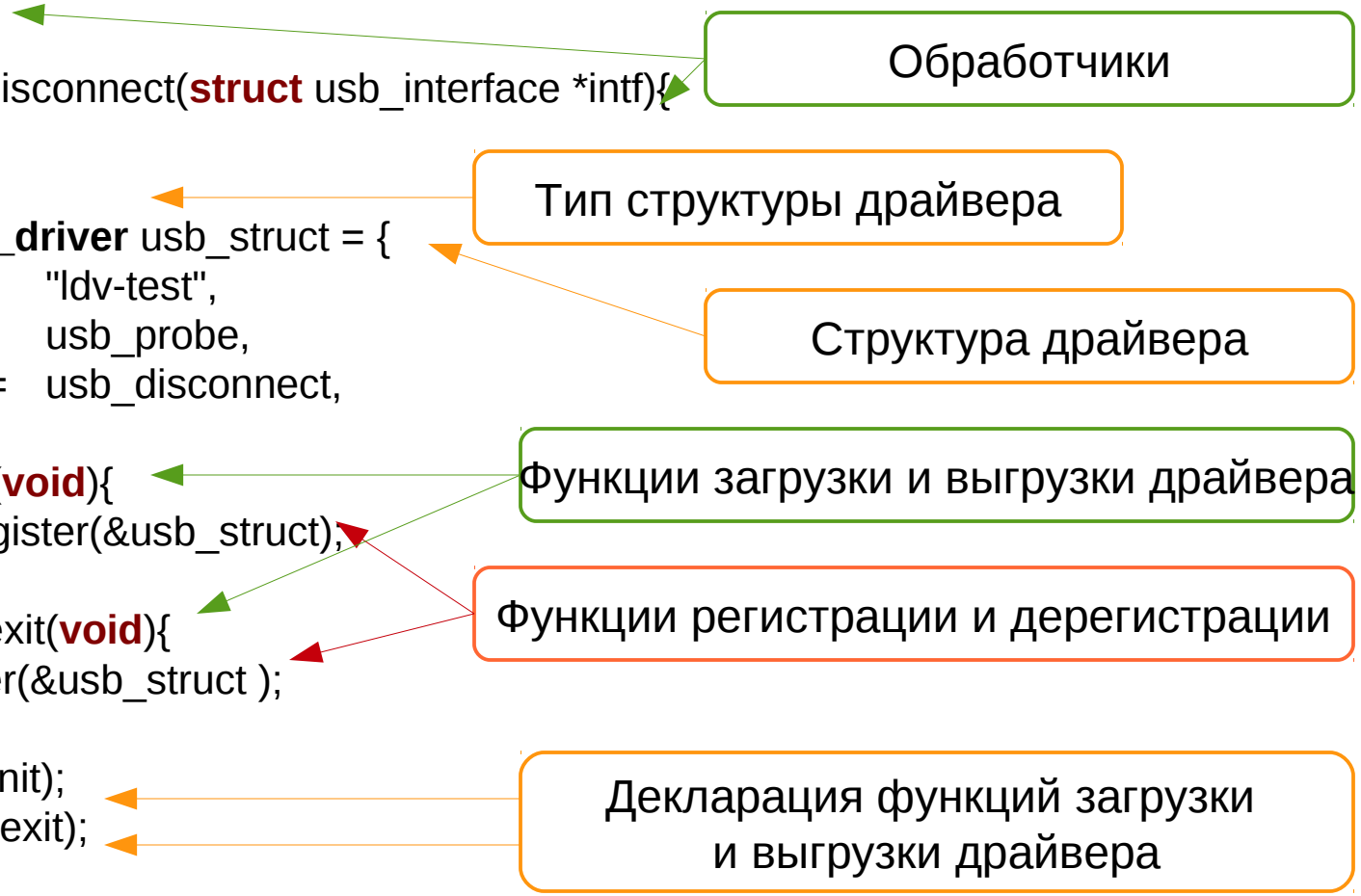
Возникает потребность в верификации

Драйверы в ядре Linux

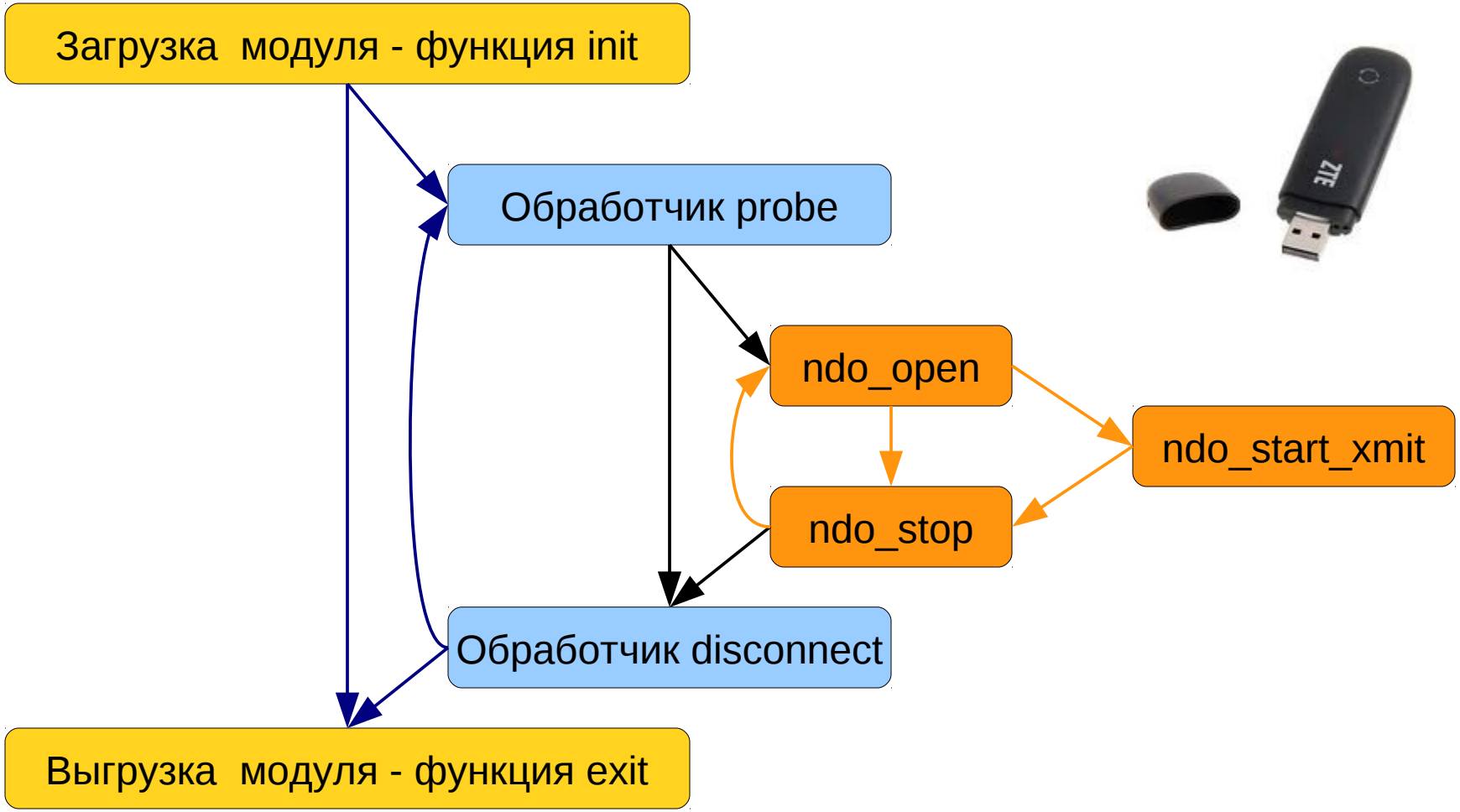


Пример драйвера

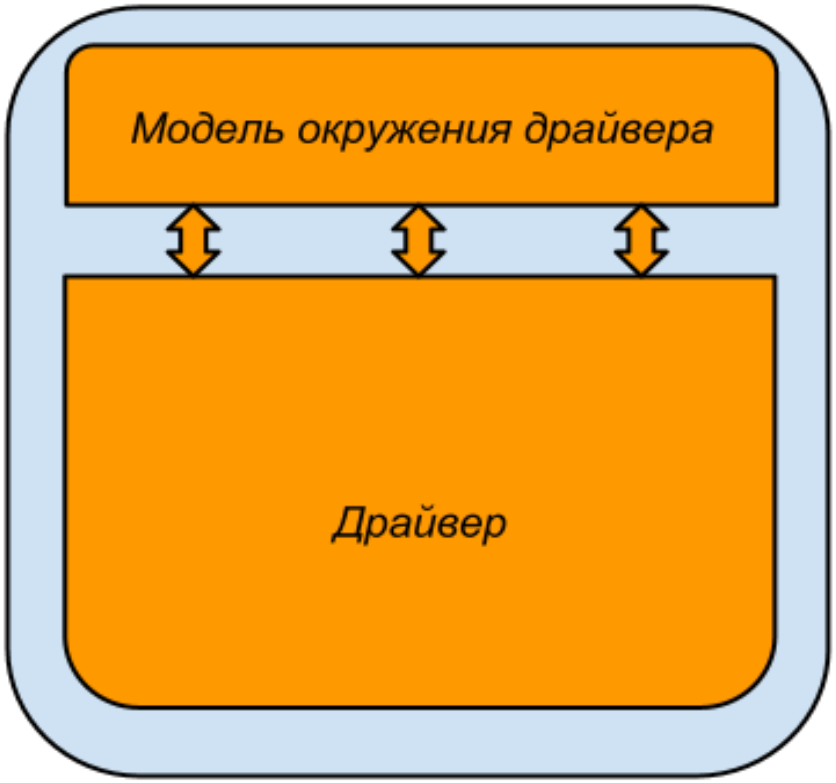
```
int usb_probe(struct usb_interface *intf, const struct usb_device_id *id){
    ...
}
static void usb_disconnect(struct usb_interface *intf){
    ...
}
static struct usb_driver usb_struct = {
    .name =         "ldv-test",
    .probe =        usb_probe,
    .disconnect =   usb_disconnect,
};
int __init usb_init(void){
    return usb_register(&usb_struct);
}
void __exit usb_exit(void){
    usb_deregister(&usb_struct );
}
module_init(usb_init);
module_exit(usb_exit);
```



Пример работы драйвера.



Модель окружения драйвера



Требования к модели окружения

- Со стороны инструмента верификации
  - Корректная Си программа
  - Одна точка входа
  - Упрощенные конструкции
- Со стороны пользователя
  - Полнота
  - Адекватность
  - Простота сопровождения

Существующие решения

	Полнота	Адекватность	Простота сопровождения представления модели
Microsoft SDV	+	+	-
DDverify	+	+	-
Avinux	+/-	-	-
LDV	+	+/-	+

Недостатки генератора моделей окружения LDV

Недостаточные возможности по заданию конфигурации:

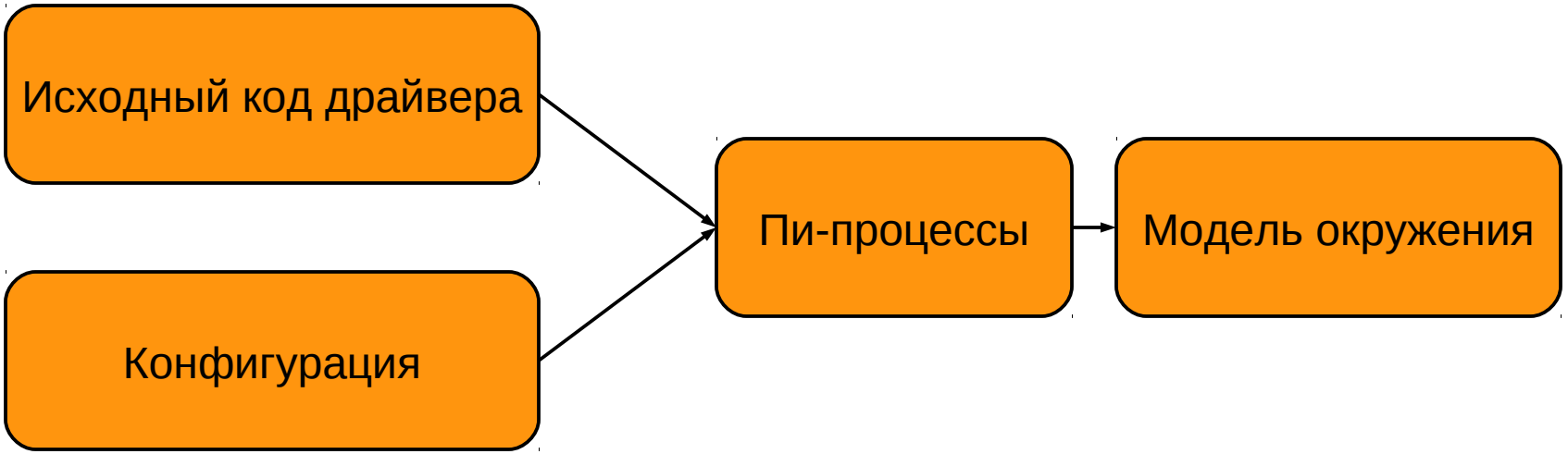
- Нельзя описать все возможные последовательности вызовов обработчиков из одной структуры.
- Нельзя описать взаимодействие обработчиков из разных структур драйвера

Сбор информации по исходному коду драйверов для моделирования окружения на основе регулярных выражений.

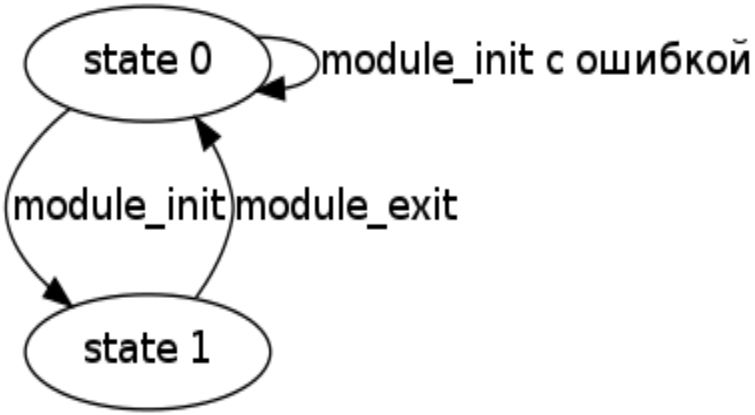
Статистика по ядру 3.0.1

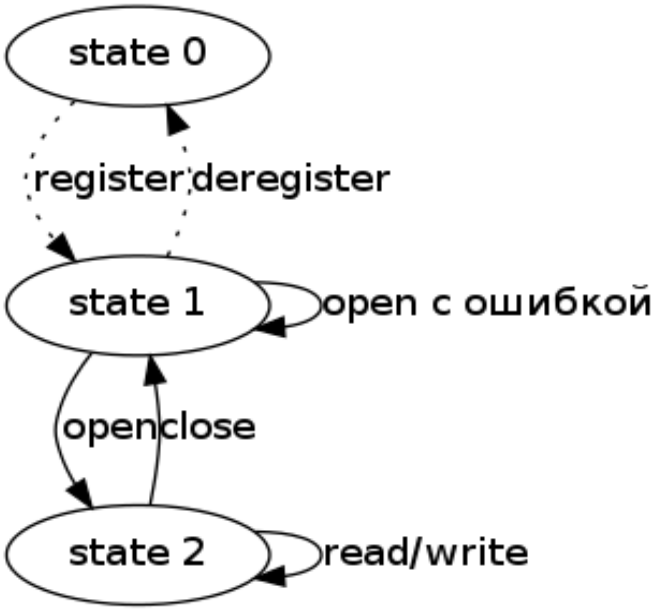
- Модулей – более 2500
  - всего драйверов с 1 структурой: 1202 - 47%
  - всего драйверов с 2 структурами: 648 - 25%
  - всего драйверов с 3 структурами: 304 - 12%
- Структуры драйверов – более 400 типов
  - 100 типов структур, использующихся в более чем 10 драйверах
  - 25 типов структур, использующихся в более чем 50 драйверах

Процесс построения модели окружения



Представление о порядке вызова





Конфигурация

- Спецификации для типов структур
  - Описание регистрации
  - Описание всп. сигналов, функций, переменных
  - Описание порядка вызова обработчиков
- Шаблоны для неизвестных типов структур драйверов

Регистрация структуры драйвера

```
after: call (int usb_register_driver(struct usb_driver
    *, struct module *, const char *))
{
        $ACTIVATE;
}
```

Вектор состояния

```
<start_state>
   <hnd>probe=1</hnd>
   <hnd>disconnect=0</hnd>
   <hnd>suspend=0</hnd>
   <hnd>resume=0</hnd>
</start_state>
```

Сигналы

```
<signals>
    <signal name="usb_get">
        usb_counter++;
    </signal>
    <signal name="usb_put">
        usb_counter--;
    </signal>
</signals>
```

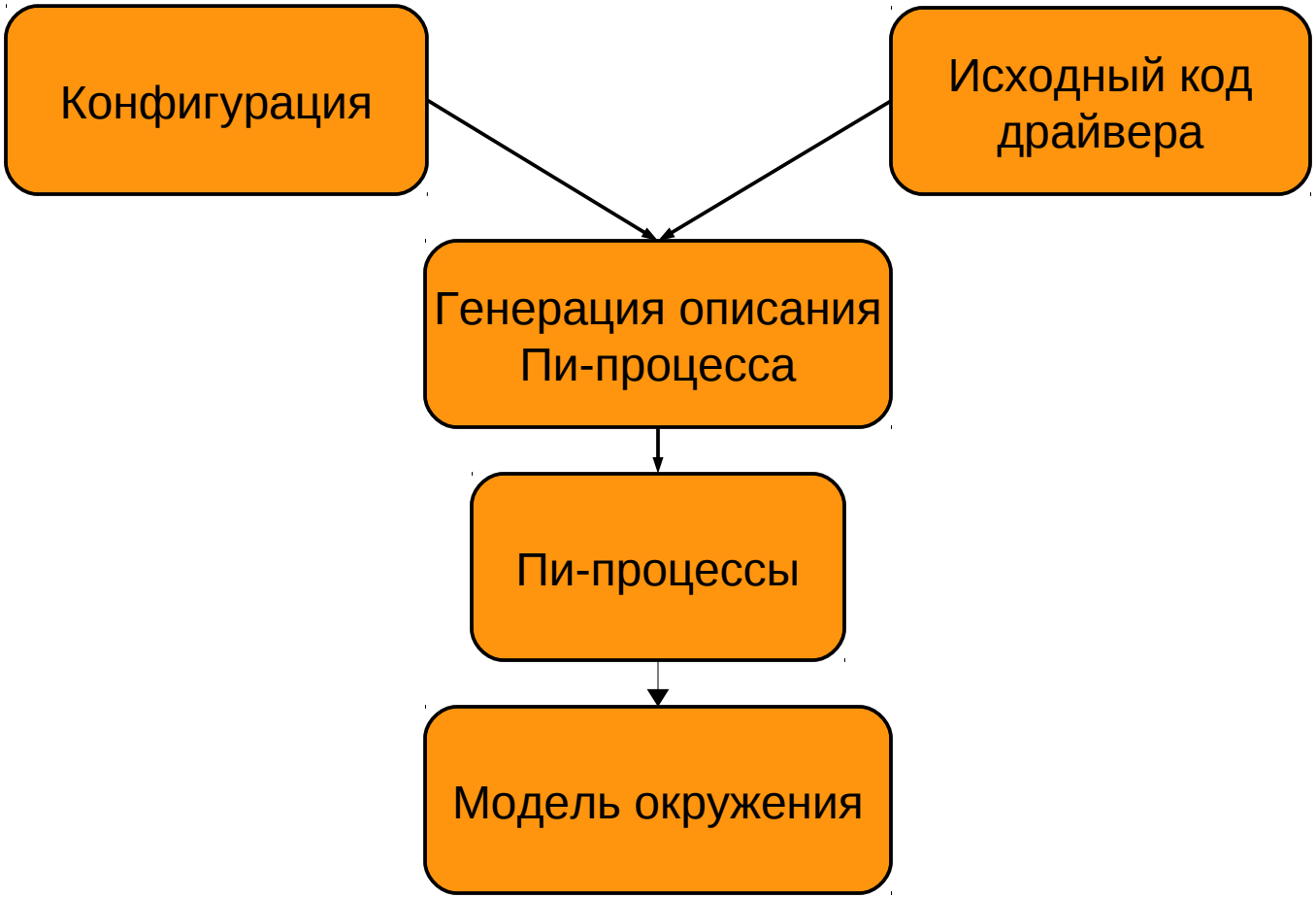
События для обработчиков

```
<- - Проверка возвращаемого значения обработчика  probe - ->
<event>
   <condition>$RET==0</condition>
   <execute>$send(module_get);</execute>
   <new_state>
        <hnd>probe=0</hnd>
        <hnd>disconnect=1</hnd>
        <hnd>suspend=1</hnd>
   </new_state>
</event>
```

Спецификация для типа структуры драйвера

- Общие сведения (Имя, тип и т.д.)
- Регистрация и дерегистрация
- Сигналы
- Стартовое состояние (вектор состояния)
- Вспомогательные переменные и функции
- Операция инициализации
- Сведения об обработчиках
  - Общие сведения (аргументы, возвр. значение)
  - События на вызов обработчиков

Случай неизвестного типа структуры драйвера



Пример модели

```
void  entry_point(void){
ldv_state_variable_1=0;  ...
while(1){
   switch(nondet_int()){
      case 0:{...
      }
      case 1:{
        if(ldv_state_1 != 0){
            switch(nondet_int()){
                case 0:{...} break;
                case 1:{...} break;
                default: break;
            }
        }
      }
      break;
   }
}
ldv_final: ldv_check_final_state();
return;
}
```

```
case 0:{
    if(ldv_state_0 != 0){
     switch(nondet_int()){
       case 0:{
        if(ldv_state_0 == 1){
          retval_init=usb_init();
          if(retval_init==0){...}
        }
       }break;
       case 1:{
        if(ldv_state_0 == 3 && ref_cnt==0){
          usb_exit();
          goto ldv_final;
        }
       }break;
       default: break;
     }
    }
}
break;
```

Преимущества подхода

- Улучшение точности модели
  - Можно описать все возможные последовательности вызовов обработчиков из одной структуры.
  - Можно описать взаимодействие обработчиков из разных структур драйвера
  - улучшение качества анализа исходного кода
- Дополнительные средства отладки позволяют упростить сопровождение модели

Результаты применения

В среднем на 30 % лучшие показатели по сравнению со старым генератором моделей окружения LDV.

- Считалось количество ложных вердиктов инструмента верификации из-за некорректной модели
- Использовался инструмент статической верификации BLAST
- ядра Linux 3.0.1 и 3.5.3
- Проверялось 5 видов ошибок: инициализация спинлоков, освобождение памяти с флагом ATOMIC, парное использование module_put и module_get и т.д.
- Количество ложных вердиктов из-за некорректной модели может варьироваться от нескольких до нескольких сотен

Направления развития

- Построение модели для драйверов, состоящих из нескольких модулей
- Добавление в модель новых элементов (прерывание, отложенные задачи)
- Улучшение анализа исходного кода драйвера
- Моделирование более сложных ситуаций

LinuxTesting.org

Спасибо!

Захаров Илья
Ilja.zakharov@ispras.ru
http://linuxtesting.org/project/ldv

ISP RAS
Institute for System Programming of the Russian Academy of Sciences