

# **Исследование требований к языку системного программирования высокого уровня на примере анализа кода микроядерной ОС seL4**

Учебная практика  
студента 343 группы  
Вильданова Эмира

Научный руководитель: Салищев С. И.



# Информационная безопасность

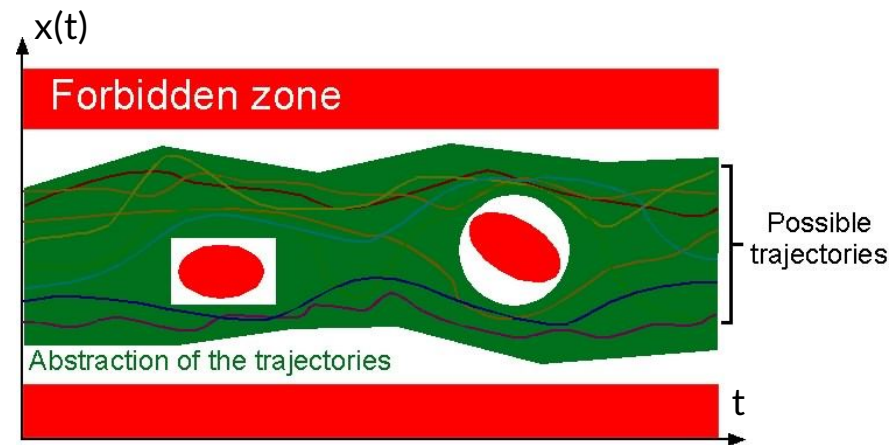
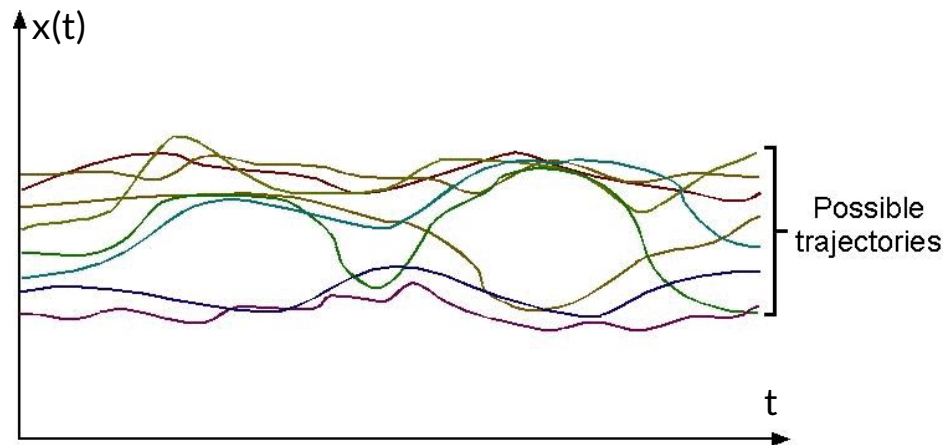
- Информационные технологии проникли во все сферы нашей жизни
- Безопасность программного обеспечения сегодня играет большую роль
- Существует необходимость в системе, делающей процесс разработки кода более надёжным
- Создание подобной системы для системы реального времени является реализуемым



# Методы проверки исходного кода

- Unit-тестирование
- Fuzzing
- DSE и BMC
- Абстрактная интерпретация

# Абстрактная интерпретация

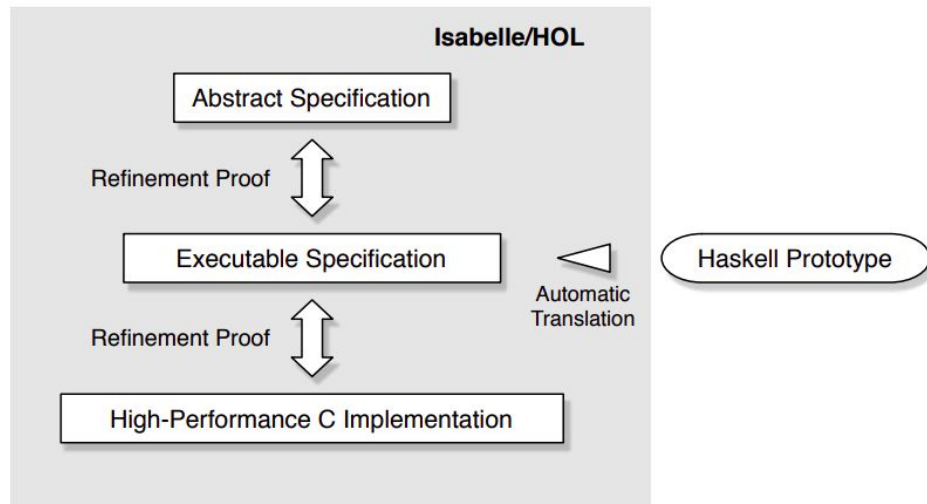


$t$  — время

$x(t)$  — состояние программы (вектор входных данных)

## Особенность верификации seL4

- Абстрактная спецификация
- Функциональная модель реализации
- Реализация на языке C





## Пример 1

Абстрактная спецификация  
на языке Isabelle/HOL

```
schedule = do
  change_current_domain;
  next_domain <- gets cdl_current_domain;
  threads      <- gets (active_tcbcs_in_domain next_domain);
  next_thread <- select threads;
  switch_to_thread (Some next_thread)
od
```

```
schedule :: Kernel ()
schedule = do
  curThread <- getCurThread
  action <- getSchedulerAction
  case action of
    ResumeCurrentThread -> return ()
    ...
```

Формальная  
реализация на  
языке Haskell

## Пример 2

```
lemma ccorres_return_cte_cteCap [corres]:  
  fixes ptr' :: "cstate  $\Rightarrow$  cte_C ptr"  
  assumes r1: " $\wedge s\ s'.\ g.\ (s,\ s') \in \text{rf\_sr} \implies (s,\ \text{xfu}\ g\ s') \in \text{rf\_sr}$ "  
  and xf_xfu: " $\wedge s\ g.\ \text{xf}\ (\text{xfu}\ g\ s) = g\ s$ "  
  shows "ccorres ccap_relation xf  
    (λs. ctes_of s ptr = Some cte) {s. ptr_val (ptr' s) = ptr} hs  
    (return (cteCap cte))  
    (Basic (λs. xfu (λ_. h_val (hrs_mem (t_hrs_' (globals s)))  
      (Ptr &(ptr' s  $\rightarrow$  ['cap_C']))) s))"  
  apply (rule ccorres_return)  
  apply (rule conseqPre)  
  apply vcg  
  apply (clarsimp simp: xf_xfu ccap_relation_def)  
  apply rule  
  apply (erule r1)  
  apply (drule (1) rf_sr_ctes_of_clift)  
  apply (clarsimp simp: typ_heap_simps)  
  apply (simp add: c_valid_cte_def)  
done
```



## Цель работы

Изучить требования к системному языку программирования высокого уровня, удовлетворяющего требованиям полноты возможных к реализации алгоритмов, разумного времени проверки и эстетичности кода.

## Задачи

- Ознакомиться с существующими аналогами систем, проверяющими код на безопасность
- Изучить архитектуру и исходный код микроядра seL4
- Изучить синтаксис функционального языка программирования Isabelle/HOL
- Исследовать блок верификации seL4:
  - Выявить требования, налагаемые на код микроядра
  - Проанализировать свободу написания кода, удовлетворяющего выявленным требованиям
  - Проанализировать время выполнения проверок
  - Проанализировать эстетический аспект кода, удовлетворяющего выявленным требованиям





## Выявленные требования

Архитектурные	Общие
<ul style="list-style-type: none"><li>● Разрешение глобальных переменных для системных структур данных с отслеживанием их инвариантов</li><li>● Ограничение работы с памятью через Capability blocks</li><li>● Ограничение на исполнение в одном потоке</li></ul>	<ul style="list-style-type: none"><li>● Недопущение выхода за пределы массива</li><li>● Недопущение входа ядра в неактивное состояние</li><li>● Недопущение арифметических ошибок и ошибок переполнения</li><li>● Проверка правильной организованности списков</li><li>● Отсутствие пересечений по памяти</li></ul>



# MIRAI

- Абстрактный интерпретатор
- Написан на языке Rust
- Использует SMT Solver Z3
- Более популярен, чем аналогичные системы
- До сих пор ведётся разработка

## Пример кода MIRAI

```
let layouts_match = old_length
    .equals(new_length.clone())
    .and(old_alignment.equals(new_alignment.clone()));
let (layouts_match as bool, entry_cond as bool) =
    self.check_condition_value_and_reachability(& layouts_match);
if entry_cond_as_bool.unwrap_or(true) && !layouts_match_as_bool.unwrap_or(false)
    ...
```

# Проверка требований

```
pub fn foo(arr: &mut [i32], i: usize) { arr[i] = 123; }
fn main() {
    let mut arr = [0, 3];
    foo(&mut arr, 6);
}
```



```
arr[i] = 123; //~ possible index out of bounds
^^^^^^
```

```
fn big_number() -> i32 { return 1000000000; }
fn main() {
    let mut x = 1000000000;
    x *= big_number();
}
```



```
warning: attempt to multiply with overflow
--> emir_example/src/main.rs:35:5
   |
35 |     x *= big_number();
   |     ^^^^^^^^^^^^^^^^^^
```



## Результаты

- Проведено ознакомление с существующими аналогами систем, проверяющих код на безопасность:
  - Изучены языки программирования, осуществляющие статическую проверку кода на этапе компиляции
  - Изучен аналог системы проверки кода системы реального времени
  - Выявлены требования, налагаемые на проверяемый код
- Изучена архитектура и исходный код микроядра seL4
- Поверхностно изучены функциональные языки программирования Haskell и Isabelle
- Изучен блок верификации seL4 и выявлены требования, предъявляемые системой проверки
- Найден инструмент, на основе которого может быть реализована более понятная система проверки кода на безопасность



# Теория 1

- Формальная спецификация (formal specification) — требования к вычислениям программы, записанные на каком-либо формальном языке.
- Формальная верификация (formal verification) — проверка соблюдения этих требований.
- Точность (correctness) — удовлетворение алгоритма (программы) спецификации.
- Полнота (completeness) — доказуемость в данной теории синтаксически корректных замкнутых формул или их отрицаний.
- Вычислимость (computability) функции — возможность существования алгоритма, вычисляющего данную функцию



## Теория 2

- Операционная семантика — определение значения программы, с помощью установления правил для абстрактной вычисляющей машины, набор правил вычисления.
- Денотационная семантика — сопоставление выражениям языка программирования математических объектов.




## Теория 3

- Соответствие Карри-Ховарда — наблюдаемая структурная эквивалентность между математическими доказательствами и программами, которая может быть формализована в виде изоморфизма между логическими системами и типизированными исчислениями.


**Важно:** логика высказываний соответствует простому типизированному  $\lambda$ -исчислению.

- Вывод типов в программе = запуск программы в денотационной семантике.
- Доказательство корректности программы = проверка логических предикатов.
- Просто типизированное лямбда-исчисление завершаются за конечное число шагов  $\rightarrow$  все программы либо типизируемы, либо не типизируемы.


```
/// Calls a specialized visitor for each kind of statement.  
#[logfn_inputs(DEBUG)]  
fn visit_statement(&mut self, location: mir::Location, statement: &mir::Statement<'tcx>) {
```



```
/// Write the RHS Rvalue to the LHS Place.  
#[logfn_inputs	TRACE)]  
fn visit_assign(&mut self, place: &mir::Place<'tcx>, rvalue: &mir::Rvalue<'tcx>) {
```



```
/// Calls a specialized visitor for each kind of Rvalue  
#[logfn_inputs	TRACE)]  
fn visit_rvalue(&mut self, path: Rc<Path>, rvalue: &mir::Rvalue<'tcx>) {
```



```
/// path = &x or &mut x or &raw const x  
#[logfn_inputs	TRACE)]  
fn visit_address_of(&mut self, path: Rc<Path>, place: &mir::Place<'tcx>) {
```



```

impl Library {
    fn book_exists(&self, book_id: &str) -> bool {
        self.available.contains(book_id)
        || self.lent.contains(book_id)
    }

    #[debug_requires(!self.book_exists(book_id), "Book IDs are unique")]
    #[debug Ensures(self.available.contains(book_id), "Book now available")]
    #[ensures(self.available.len() == old(self.available.len()) + 1)]
    #[ensures(self.lent.len() == old(self.lent.len()), "No lent change")]
    pub fn add_book(&mut self, book_id: &str) {
        self.available.insert(book_id.to_string());
    }

    #[debug_requires(self.book_exists(book_id))]
    #[ensures(ret -> self.available.len() == old(self.available.len()) - 1)]
    #[ensures(ret -> self.lent.len() == old(self.lent.len()) + 1)]
    #[debug Ensures(ret -> self.lent.contains(book_id))]
    #[debug Ensures(!ret -> self.lent.contains(book_id), "Book already
lent")]
    pub fn lend(&mut self, book_id: &str) -> bool {
        if self.available.contains(book_id) {
            self.available.remove(book_id);
            self.lent.insert(book_id.to_string());
            true
        } else {
            false
        }
    }
}
...

```

```

pub struct Library {
    available: HashSet<String>,
    lent: HashSet<String>,
}

```

```
fn from_bytes_unchecked(bytes: &[u8]) -> PublicKey {
    let public_key = ...;
    add_tag!(&public_key, Tainted);
    public_key
}

fn try_from(bytes: &[u8]) -> Result<PublicKey> {
    let public_key = from_bytes_unchecked(bytes);
    // Perform validity checks. Return Err if the key is invalid.
    ...
    add_tag!(&public_key, Sanitized);
    Ok(public_key)
}

// Check that `sig` is valid for `message` using `public_key`.

fn verify_msg(sig: &Signature, message: &Message, public_key: &PublicKey) -> Result<()> {
    precondition!(does_not_have_tag!(public_key, Tainted) || has_tag!(public_key, Sanitized));
    ...
}
```

```
fn from_bytes_unchecked(bytes: &[u8]) -> PublicKey {
    let public_key = ...;
    add_tag!(&public_key, Tainted);
    public_key
}

fn try_from(bytes: &[u8]) -> Result<PublicKey, E> {
    let public_key = from_bytes_unchecked(bytes);

    // Perform validity checks. Return Err if the key is invalid.
    ...

    add_tag!(&public_key, Sanitized);
    Ok(public_key)
}

// Check that `sig` is valid for `message` using `public_key`.
```