Отчет по внутреннему ОКР “Защищенный режим”

Внедрение технологии защищенного режима исполнения Эльбрус в задаче разработки и отладки СПО промышленных контроллеров

Мустафин Т. Р., Алехин А. И., Черкашин С. Ю., Зубов И. Н.,
Лубинец М. И., Кравцунов Е.М.

АО «МЦСТ», Москва, 15 декабря 2017 г.

Содержание

* Постановка задачи
* Область применения защищенного режима
* Ранее реализованные программные компоненты
* Созданные программные компоненты
* Детали реализации: проблемы и их решения
* Технология в действии
* Результаты и дальнейшее развитие
* О финансировании

Защищенный режим

Указатель элемента массива представляется в виде **дескриптора** 128 бит. Дескриптор содержит адрес начала массива и размер массива. При обращении за границу массива возникает исключительная ситуация (поддержано в аппаратуре).

Данные: неинициализированные, указатель или число — различаются с помощью тэгов (внешних), поддержанных аппаратно. При обращении в память по числу или неинициализированным данным возникает исключительная ситуация.

Поддержка в аппаратуре и компиляторе и ядре ОС сделана давно, реального внедрения пока нет.

Предыстория

1) В конце 2015-го года в АО «МЦСТ» группой разработчиков ядра ОС, компилятора и библиотек была инициирована работа по созданию минималистичного дистрибутива ОС для встраиваемых систем, работающего в защищенном режиме.

2) В 2016 году началась проработка последовательности действий по реализации минималистичного дистрибутива в защищенном режиме. В результате предварительного исследования было принято решение портировать на Эльбрус библиотеку uclibc-ng, являющуюся ориентированным на встраиваемые системы минималистичным аналогом glibc.

3) В конце 2016 года был открыт внутренний ОКР «Защищенный режим». ОКР включал в себя несколько работ, описанных в техническом задании. Помимо формализации требований и определения сроков выполнения работ, ОКР позволил финансировать работу подразделения АО «МЦСТ» в ОЭЗ «Иннополис». Подразделение было создано для привлечения к работе новых талантливых специалистов, выпускников Университета Иннополис. Это позволило решить еще одну задачу проекта — нехватку специалистов по защищенному режиму.

На сегодняшний день основные задачи проекта выполнены.

Постановка задачи

Основная задача проекта: внедрение технологии защищенного режима в типовых прикладных задачах встраиваемых систем путем разработки набора системных библиотек, функционирующих в защищенном режиме.

Дополнительная задача 1: подготовить группу специалистов по защищенному программированию, способную сопровождать процессы интеграции и внедрения разработанной технологии на промышленных предприятиях и предприятиях ВПК

Дополнительная задача 2: определить целевую группу потребителей технологии защищенного режима, стратегию работы с этой группой, определить дальнейшие технические и организационные шаги по привлечению финансирования и внедрению разработанных технологий.

Область применения

Область применения технологии защищенного режима определяется следующими требованиями:

1) все системные библиотеки и прикладные программы стека ПО написаны на языке C;

2) количество и объем системных библиотек минимальны, набор библиотек сформирован исходя из назначения ПО;

3) применение защищенного режима в программах должно быть оправданным например повышенными требованиями к надежности ПО.

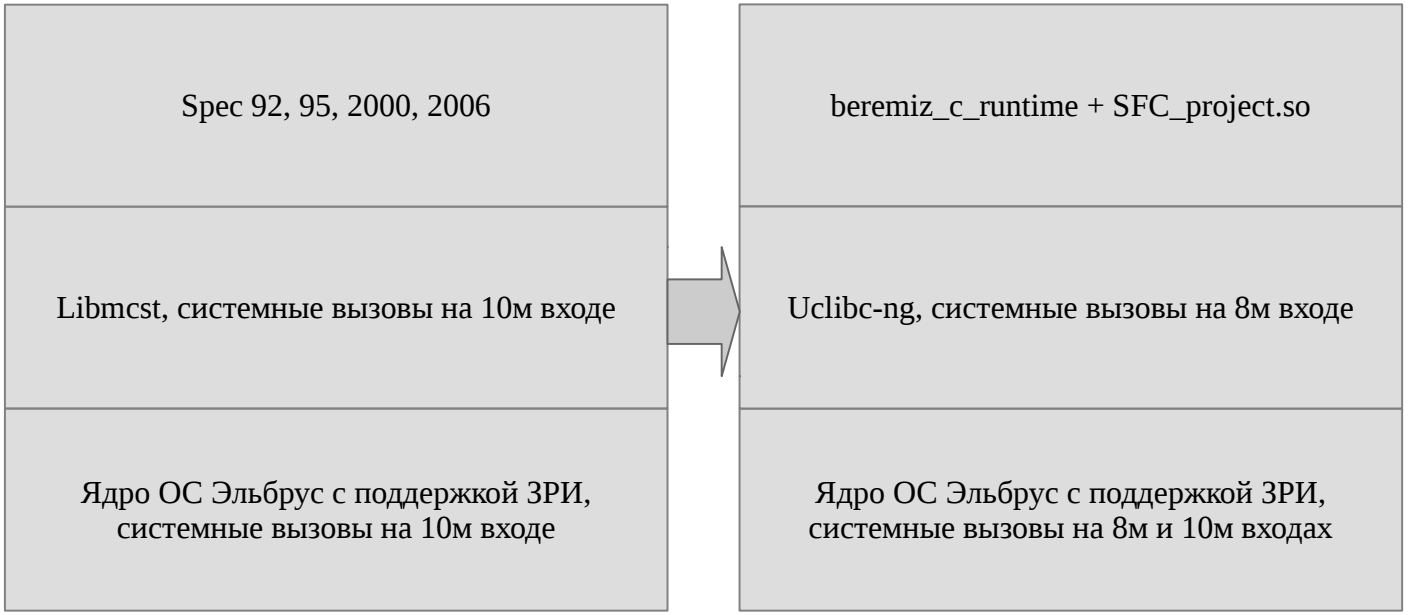
Наиболее вероятная область внедрения результата — **промышленные контроллеры**. В этой области могут быть использованы наработки ПАО «ИНЭУМ им. И.С. Брука» в части поддержки программирования промышленных контроллеров на языках стандарта IEC 61131-3 с помощью среды разработки «Beremiz».

ПО: старые и новые компоненты

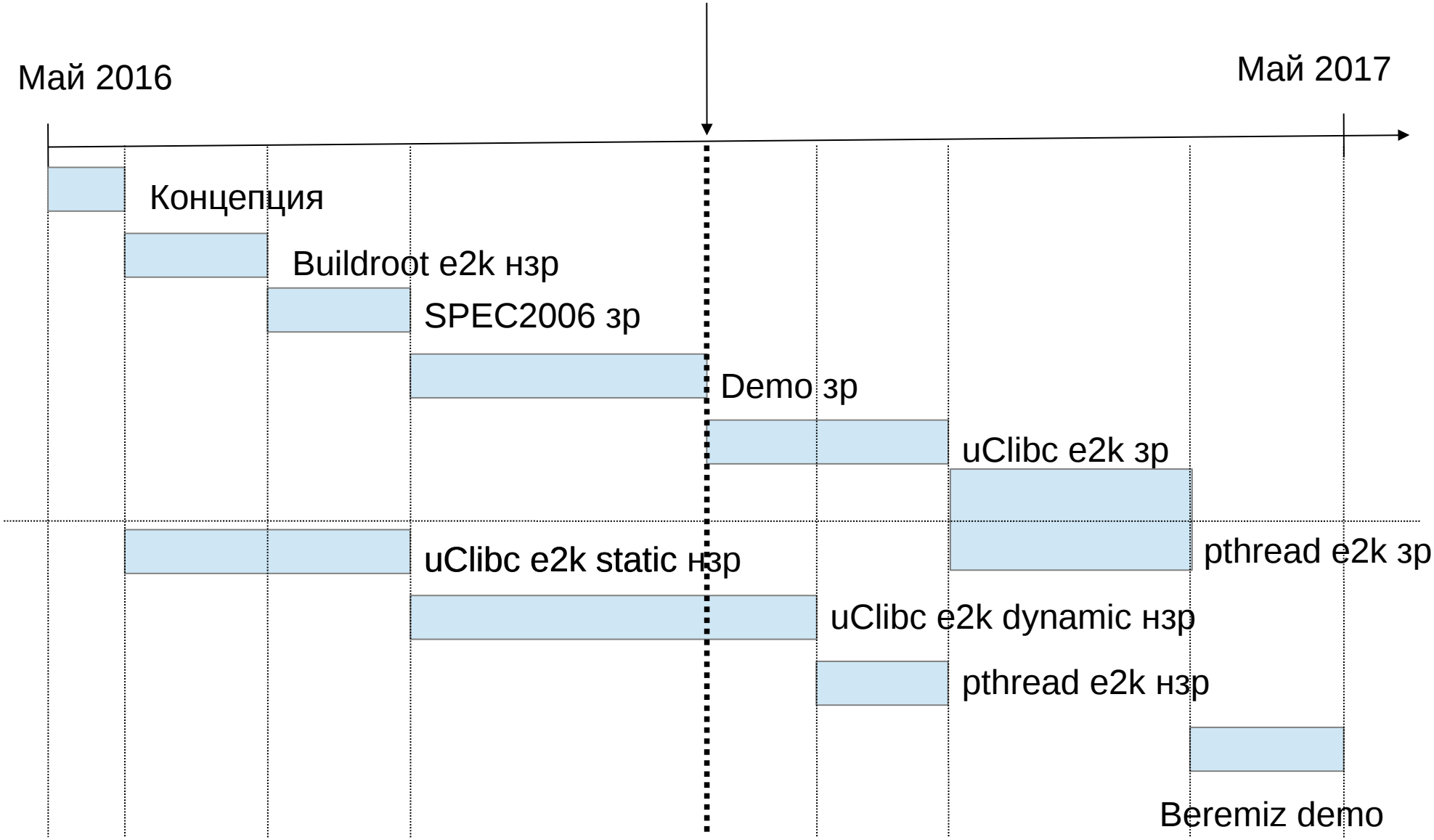
	Старые компоненты	Новые компоненты
Компилятор lcc	**-mptr128**	**-mptr128**
Ядро ОС Linux для Эльбрус	* **Вход №10** для системных вызовов * контроль работы над указателями в стеке	**Вход № 8** для системных вызовов
Библиотека языка С	Библиотека **libmcst**: * без поддержки потоков, * без поддержки сети, * без поддержки dlopen (динамическая линковка была, но не было поддержки динамических библиотек), * без реализации malloc в библиотеке (каждый malloc приводит к системному вызову), * без поддержки сигналов и таймеров.	Библиотека **uclibc-ng** в защищенном режиме: * поддержка динамического связывания (dlopen, dlsym) * поддержка многопоточности (pthread_create, pthread_exit, pthread_join), * поддержка функций синхронизации (pthread_mutex_init, pthread_mutex_destroy, pthread_mutex_lock, pthread_mutex_unlock), * поддержка функций сигналов и таймеров * поддержка простой реализации malloc (без реализации в библиотеке, каждый malloc приводит к системному вызову), * поддержка функциями работы с сетью (socket).
Дополнительные библиотеки		* Библиотека libmodbus * Библиотеки libmodbusregshelper и libmodbussrv были реализованы на языке Си (переписаны с языка C++ на C) * Графические библиотеки на языке С: mesa, x86emul, rfb (подготовлены к переносу в защищенный режим)

ПО: старые и новые компоненты

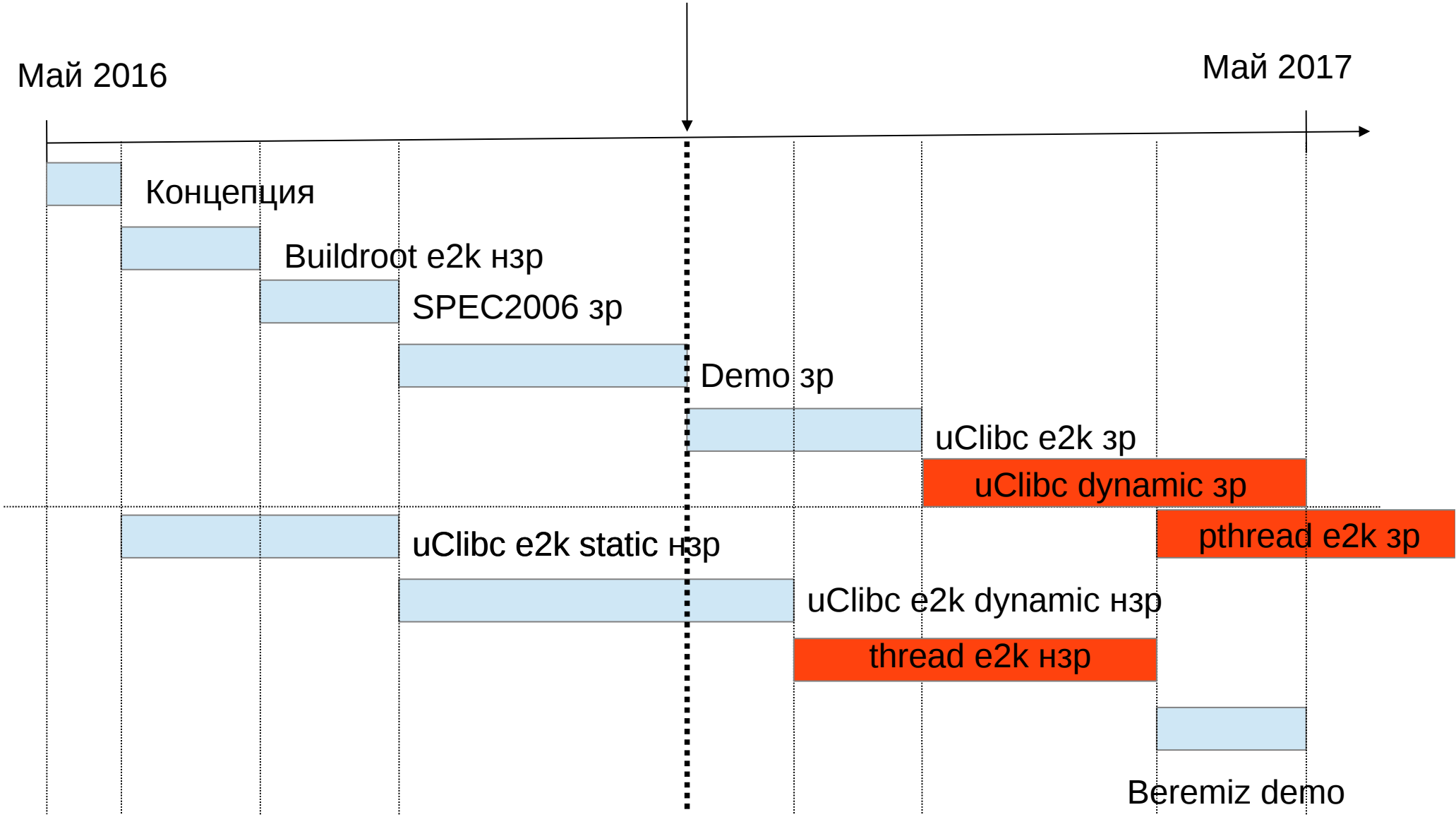
Стек ПО с libmcst (до проекта) и стек ПО с uclibc-ng (после проекта)



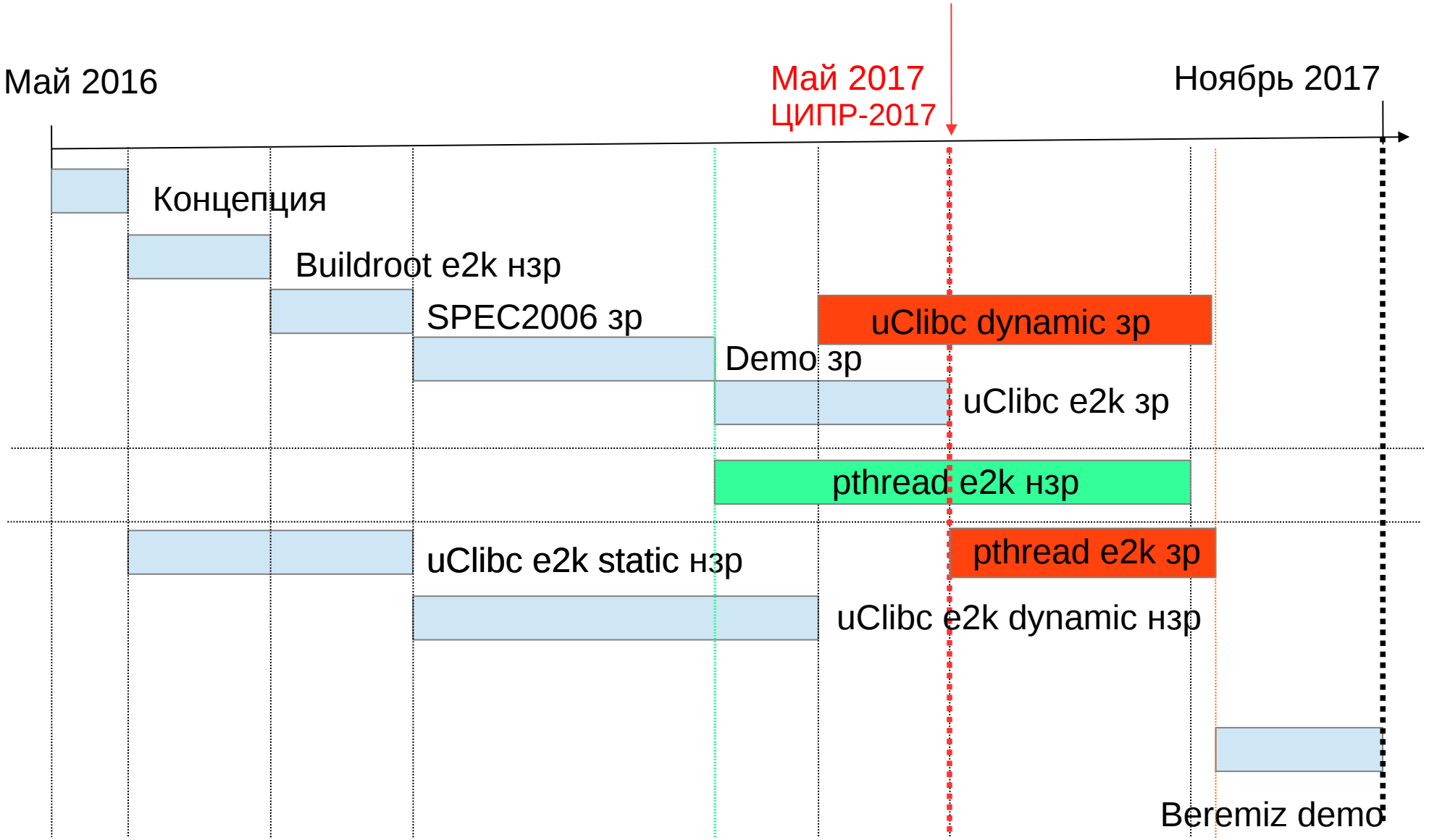
Планы и реальность: план год назад



Планы и реальность: реальность



Планы и реальность: реальность



Детали реализации

Отладка uclibc-ng в незащищенном режиме: исправлено **83 ошибки**

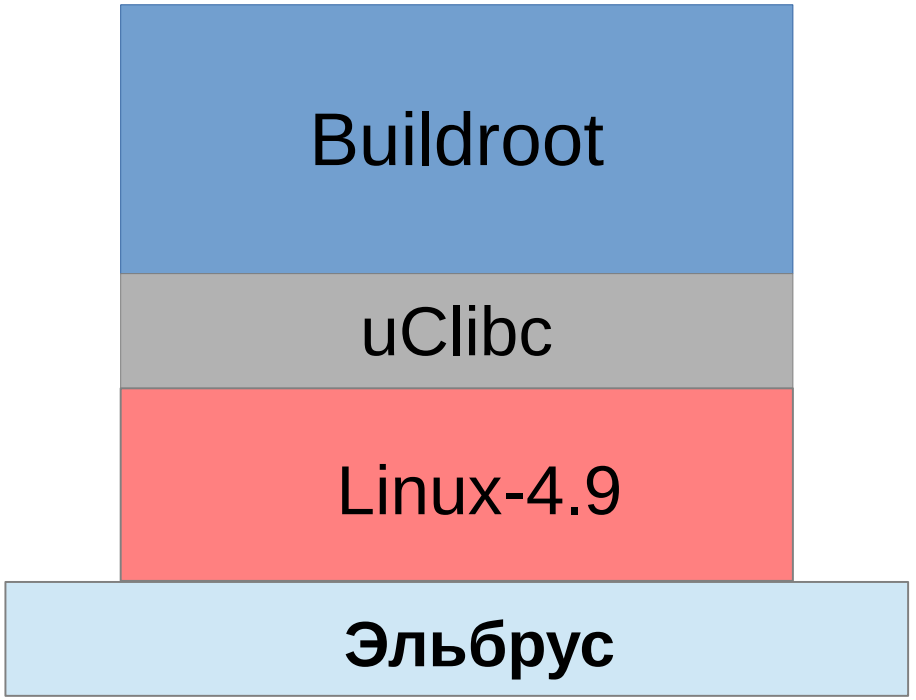
1) Отладка многопоточности в uclibc-ng в незащищенном режиме: **76 ошибок**

- реализация выделения памяти в главном потоке для TLS при использовании DTV и в других потоках при вызове clone (память выделялась некорректно)
- реализация примитивов синхронизации (отсутствовали atomic и spinlock)
- реализация корректного выхода из функции отбработки исключений

2) Перенос из glibc и отладка libm в uclibc-ng: **7 ошибок**

- Исправлен алгоритм работы с глобальной переменной signgam в функций lgamma, lgammaf, lgammal
- Исправлены реализации функций nan, nanf, nanl, strtof_nan, strtod_nan (возвращали значение nan с неверной мантиссой)
- Добавлена поддержка работы с флагом исключения FE_DENORMAL
- Был добавлен изначально отсутствующий функционал обработки исключений математических функций с помощью функции matherr. Функция-обработчик может быть определена пользователем для перехвата исключений (аналог try/catch в C++).

Детали реализации: Buildroot



- Маленький дистрибутив: 24 Мб.

- Собран полностью на библиотеке uClibc, специализарованной на использование по встраиваемых окружениях.

- Графическая подсистема, исвользующия linux framebuffer для отрисовки интерфейсов и графики.

- 7 патчей uclibc
- 4 патча пакетов buildroot

Детали реализации

uclibc-ng в минимальной конфигурации в защищённом режиме, вход в ядро № 10:

1) Реализация функции **_start, -** точки входа в библиотеку

2) Реализация механизма обращения к системным вызовам из библиотеки

3) Исправление операций, которые работают нормально в обычном режиме, но в защищённом режиме недопустимы:

- оптимизированные алгоритмы работы с памятью
- реализация специального макроса сравнения адреса функции с нулём
- замена целочисленных типов данных, используемых для хранения указателей на void*
- выравнивание всех указателей на 16 байт
- Замена некоторых ассемблерных инструкций: ldw на ldapw , stw на stapw

Детали реализации: вход №8 в ядро

Проблемы:

1) Для использования системных вызовов с указателями в качестве параметров защищенными программами необходимо проводить преобразование дескриптора в обычный указатель.

2) Системные вызовы ядра linux принимают на вход до 6 параметров. Стандартный размер окна для передачи параметров составляет 8 двойных регистров (по 64 бита). В защищенном режиме дескриптор занимает два двойных регистра — один квадрорегистр (128 бит). Поэтому не для всех системных вызовов получается разместить параметры внутри регистрового окна стандартного размера.

3) Некоторые системные вызовы принимают дескрипторы внутри структур, эти дескрипторы также необходимо преобразовывать в указатели.

4) Некоторые системные вызовы возвращают указатели. Для передачи пользователю указатели необходимо преобразовать в дескрипторы.

Детали реализации: вход №8 в ядро

Решения проблем:

1, 2) Решение во входе №8 строится на расширении регистрового окна передачи параметров: вместо 8 двойных регистров окно состоит из 14 двойных регистров (спасибо Александру Фёдорову sanekf@mcst.ru за идею и консультации). Увеличенный размер окна передачи параметров позволяет размещать каждый параметр на отдельном квадрорегистре, и проводить преобразование дескрипторов в указатели с помощью одного, общего для всех системных вызовов, участка кода. Для этого в ядре заведена таблица с масками для системных вызовов. Маска системного вызова показывает на каких местах в окне передачи параметров лежат дескрипторы. При таком подходе все аргументы всех системных вызовов умещаются в окне передачи параметров, не требуя передачи через память.

3, 4) Уникальные обёртки системных вызовов необходимы только системным вызовам, принимающим дескрипторы внутри структур, а также системным вызовам, возвращающим указатели. Уникальных оберток на 8-м входе сейчас насчитывается порядка 10.

Детали реализации: malloc

В uclibc-ng есть три варианта функции malloc:

Malloc-simple

- каждое выделение памяти приводит к mmap, каждое освобождение памяти приводит к munmap
- не использует системный вызов brk

Malloc

- выделение памяти через mmap или brk, определяется сборкой
- переиспользование освобожденной памяти
- free вызывает munmap/brk если есть большая свободная непрерывная область памяти

Malloc-standard (dlmalloc)

- переиспользование освобожденной памяти
- выделение небольших (<64 страниц) участков памяти через brk
- выделение больших участков памяти через mmap

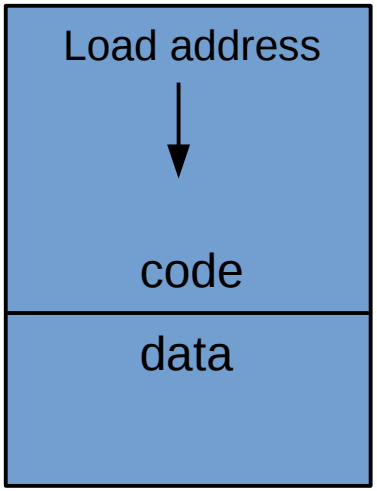
Для ЗРИ выбран **malloc-simple**, в дальнейшем — malloc-protected

Детали реализации: динамическое связывание

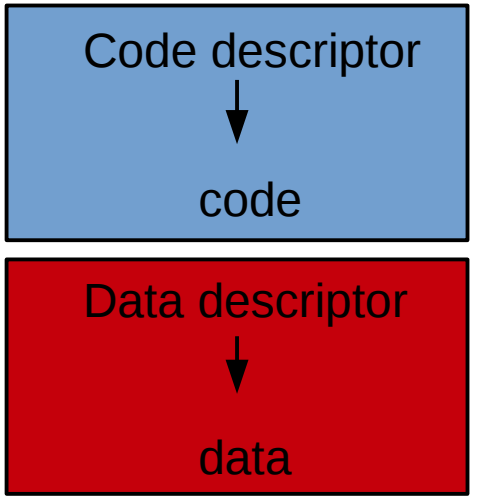
Трудности реализации динамического загрузчика:

- Хранение адресов в виде целочисленных значениях
- Использование недопустимой адресной арифметики
- Особенности загрузки программы в память в защищённом режиме:

Обычный режим



Защищённый режим



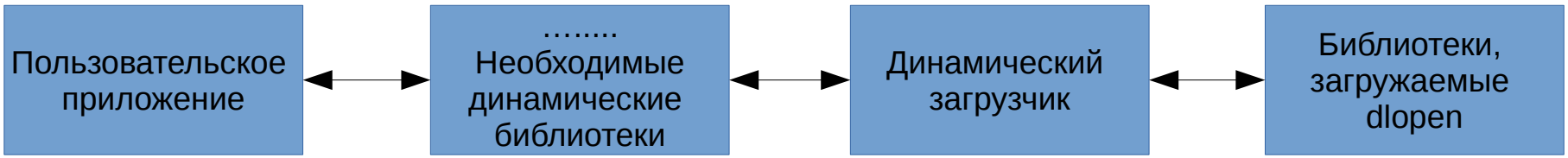
Было принято решение перенести реализацию динамического загрузчика из libmcst с доработками

Детали реализации: dlopen

Трудности реализации:

- В libmcst динамический компоновщик загружается в память как статическая программа, в цепочку загруженных модулей не включается, использование его функций и структур невозможно.
- Необходимо передать из ядра на пользовательском стеке дескрипторы на области кода и данных динамического загрузчика.
- Используя переданные дескрипторы включить динамический компоновщик в цепочку модулей и произвести связывание.

Общий вид цепочки загруженных модулей:

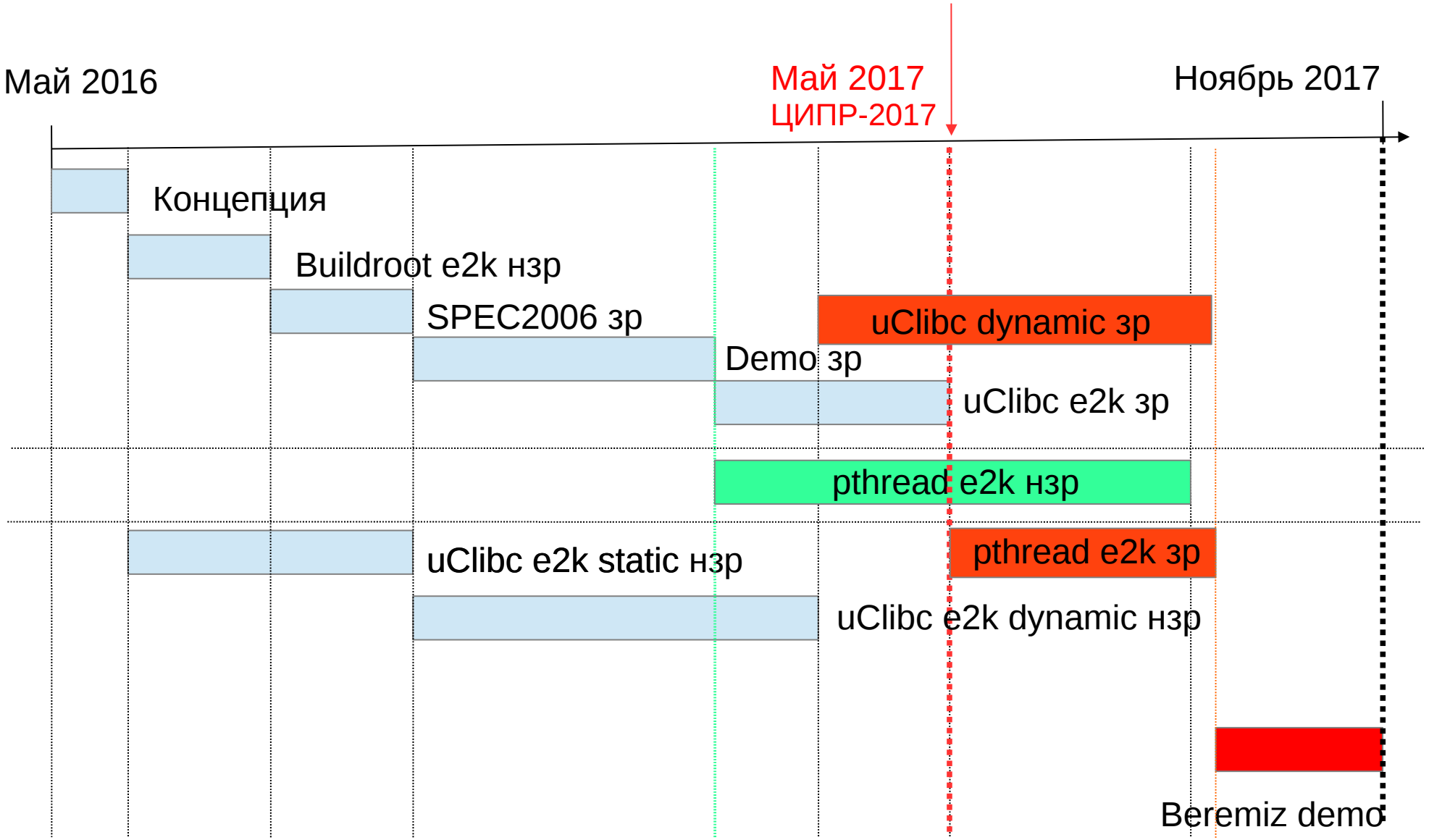


Детали реализации: pthread

Задача beremiz_runtime требует работы следующих функций nptl:

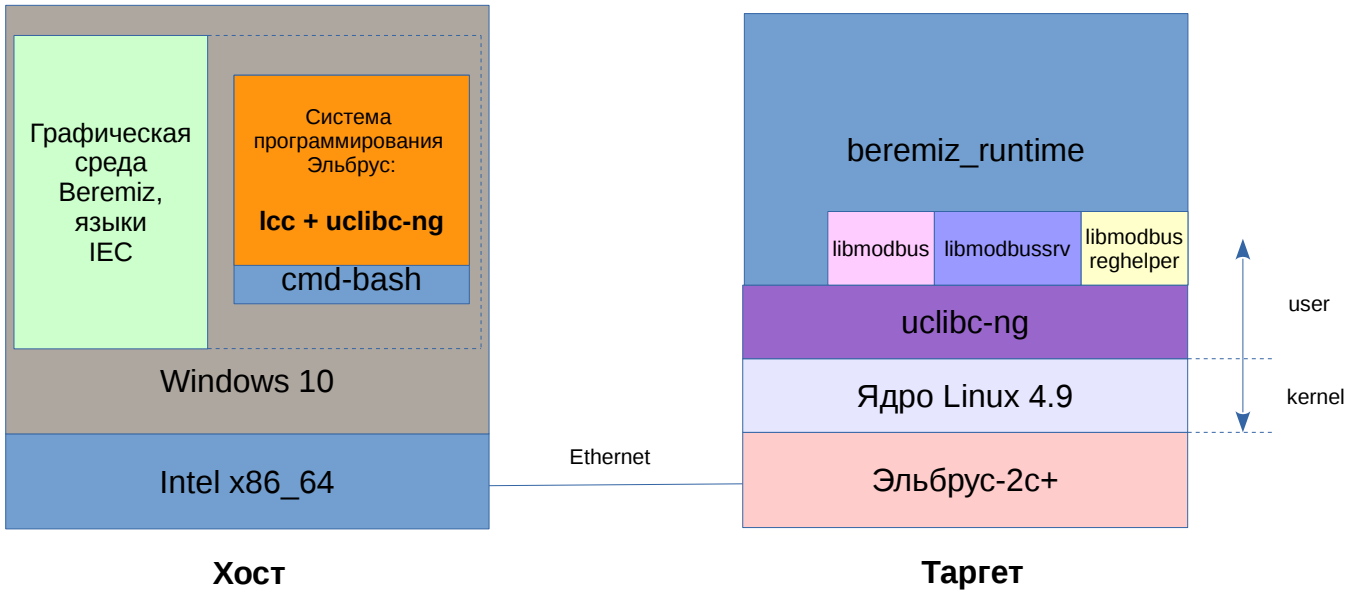
- **pthread_create**: функция clone из libmcst с передачей дополнительных параметров на регистрах. sys_clone из входа №10 с приемом параметров из увеличенного регистрового окна.
- **pthread_join, pthread_mutex_init, pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock, pthread_mutex_destroy** опираются на системный вызов futex. futex на 8м входе через маску.
- **setjmp** и **longjmp** взяты из libmcst. Исправили ошибку longjmp в ядре 4.9.

Планы и реальность: реальность



Детали реализации: реальная задача

“Деритесь настоящими мечами” К. Мацусита



Детали реализации: реальная задача

beremiz_runtime — многопоточное приложение. При запуске **beremiz_runtime** создаются следующие основные потоки, у каждого из которых есть своё предназначение:

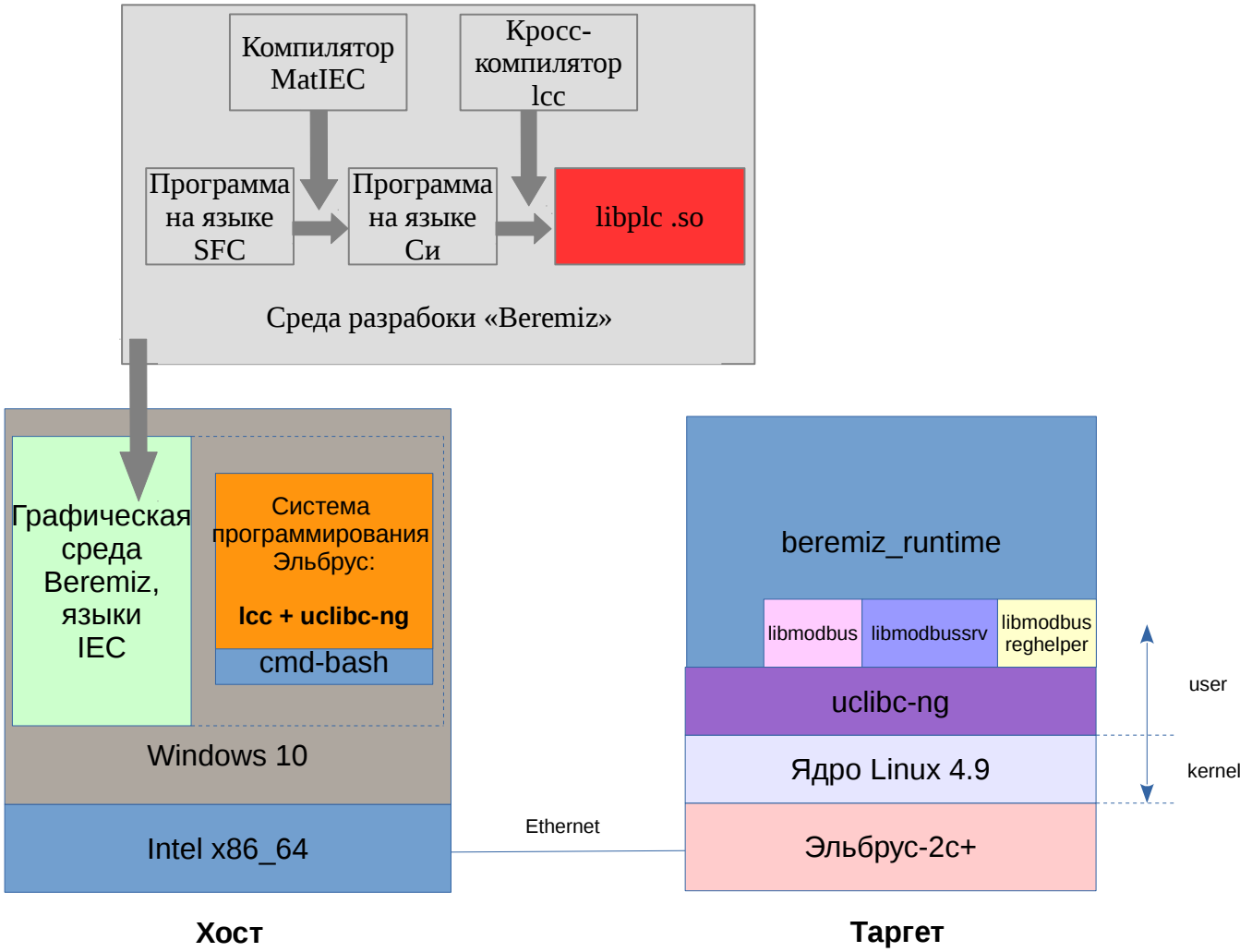
* status_thread: периодической отправка статуса устройства на управляющую машину.
* control_thread: приёма и обработка управляющих команд от среды Beremiz
* set_trace_thread/get_trace_thread: установку/получения состояния переменных
* retain_thread: работа с сохраняемыми (retain) переменными, которые записываются в энергонезависимую память (на носитель информации) целевого устройства. Данный поток, например, отвечает за приём и сохранение библиотеки с отладочным кодом от среды Beremiz.

Суть работы **beremiz_runtime** состоит в том, чтобы запустить на целевом устройстве отлаживаемый код из динамической библиотеки, полученной из программы, созданной в Beremiz, а также обеспечить интерфейс для взаимодействия со средой Beremiz и управления процессом отладки. При этом логика работы программы написанной в Beremiz будет содержаться в динамической библиотеке, передаваемой на целевое устройство.

Детали реализации: реальная задача

Процесс отладки в режиме таргет-хост.

Шаг 1: Хост: компиляция программы на языке SFC в графической среде «Beremiz»

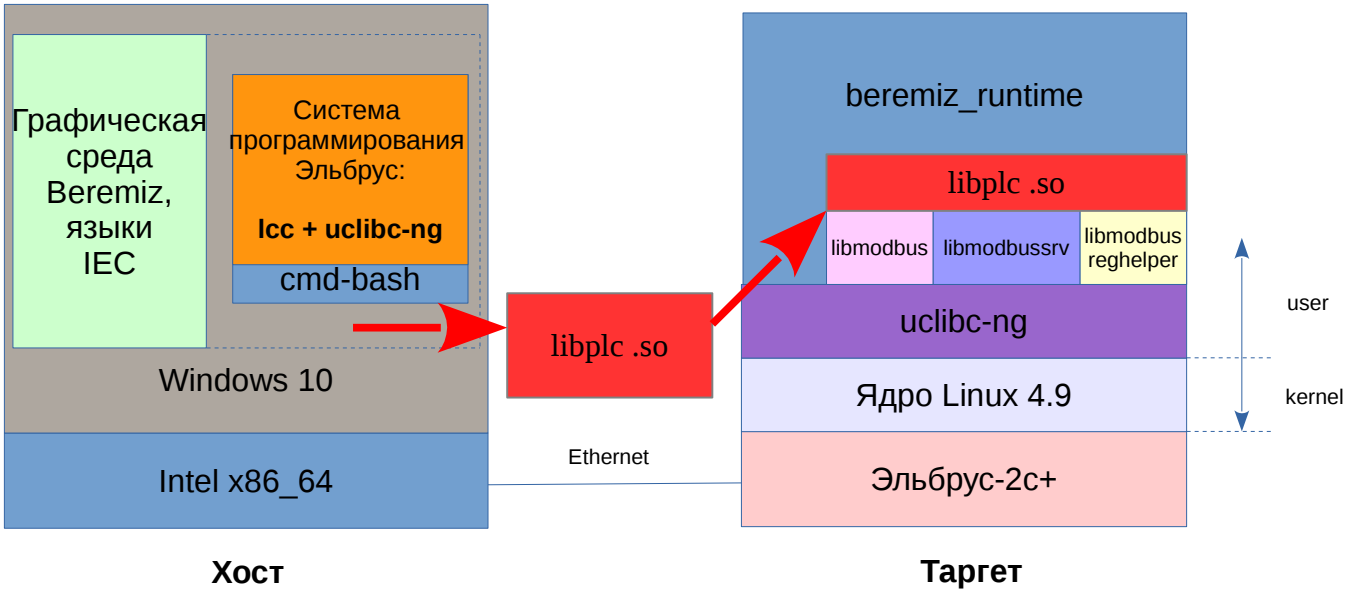


Детали реализации: реальная задача

Процесс отладки в режиме таргет-хост.

Шаг 2: Хост: установка соединения с target и передача libplc.so на таргет;

Таргет: получение libplc.so, открытие его с помощью dlopen и запуск на исполнение



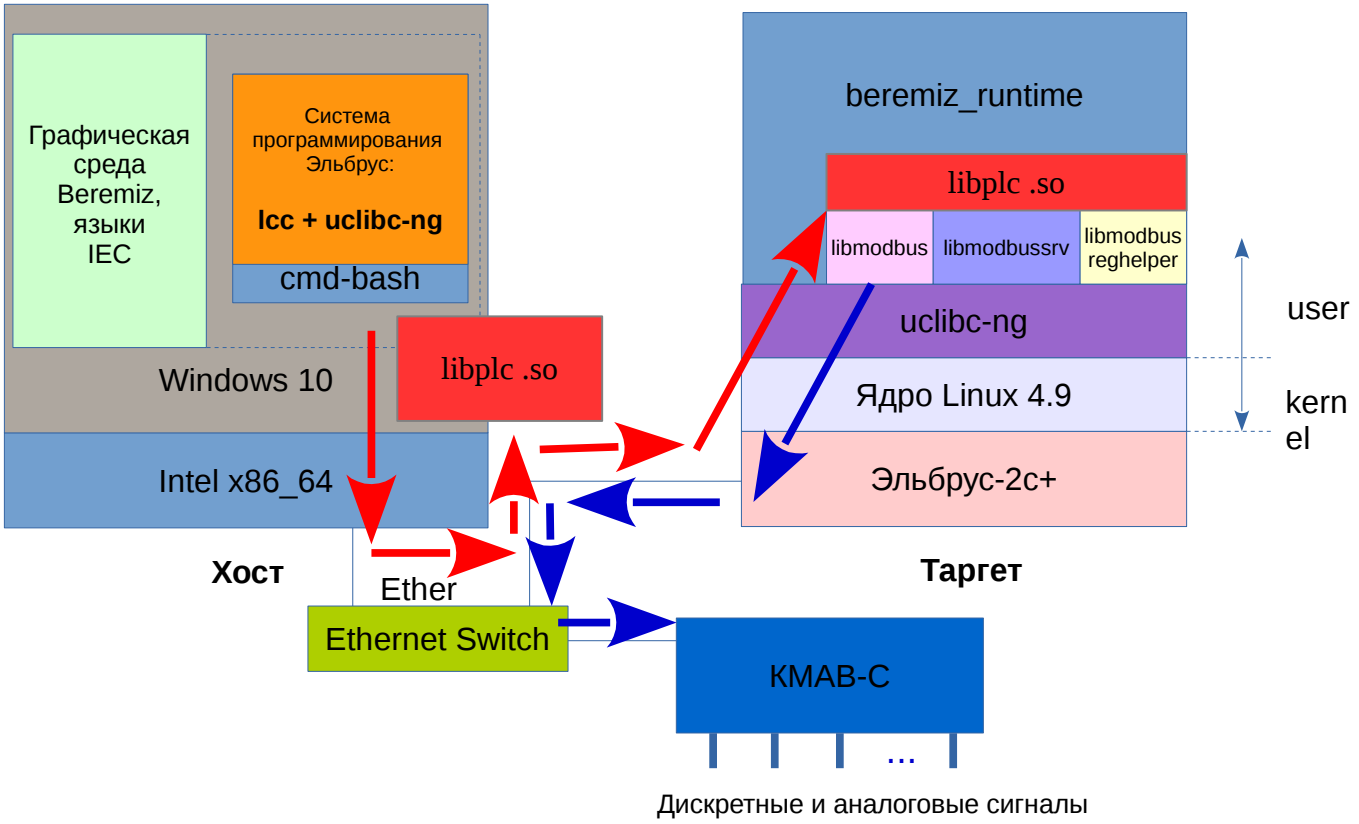
Детали реализации: реальная задача

Ошибки возникали на каждом шаге отладки реальной задачи

1. Потеря тегов дескриптора при записи дескриптора в структуру
2. Запись за границу описателя потока
3. Неинициализированные данные в longjmp
4. Ошибки в системных вызовах access, futex
5. Ошибки в системном вызове rt_sigtimedwait

Детали реализации: реальная задача

Библиотека libmodbus в защищенном режиме, ошибки в системном вызове setsockopt



Технология в действии

Результаты и дальнейшее развитие

1) Получена полнофункциональная библиотека **uclibc-ng для Эльбрус в незащищенном режиме**;

2) Разработан минималистичный дистрибутив **buildroot** для Эльбрус, ориентированный на использование во встраиваемыъ системах, использующий библиотеку **uclibc-ng**. Этот дистрибутив может использоваться в качестве вспомогательного варианта при реализации сложных внедрений (как в случае с проектом Татнефть);

3) **В защищенном режиме отлажена библиотека uclibc-ng с поддержкой многопоточности и динамического связывания**;

4) В защищенном режиме отлажена библиотека **libmodbus** и сопутствующие библиотеки **libmodbussrv, libmodbusregshelper**;

5) С помощью полученного набора библиотек выполнена интеграционная работа по внедрению результата в задачу программирования промышленных контроллеров. Результат пригоден для использования в промышленных контроллерах, построенных на «Эльбрус-1с+», разрабатываемых в ИНЭУМ (образцы – середина 2018 года);

6) Получен новый набор тестов для защищенного режима;

7) Создан задел для развития минималистичного дистрибутива в защищенном режиме части поддержки видео режима и графического режима.

Результаты и дальнейшее развитие

Первое направление: создание автоматизированных средств подготовки исходных текстов универсального дистрибутива к компиляции в защищенном режиме.

В процессе работы были выявлены типичные случаи применения опасных конструкций:
1) использование выравнивания указателей с помощью приведения указателей к типу int при оптимизации работы с массивами (примером является реализация стандартной функции strlen в uclibc-ng);
2) использование чтения из-за границы массива с целью оптимизации (такие оптимиизации считаются нормальными если разработчик уверен, что чтение не пересекает границу страницы);
3) использование глобальной переменной для хранения указателя на стек.
Выявление таких конструкций – это наиболее затратное по времени действие при переносе ПО в защищенный режим. Возможным решением является создание встроенного в фронтенд компилятора анализатора перечисленных шаблонов программирования. Такой анализатор сможет при компиляции программы в защищенном режиме выдавать предупрежденния с указанием заведомо неработающих фрагментов.

Второе направление: дополнение минималистичного дистрибутива библиотеками отображения видео, набором математических библиотек. В параллельной работе, связанной с портированием авиационной ОС4000 на Эльбрус, где разработчики отдела АЗК ОССН принимают участие, был получен интересный результат — реализация минимального набора библиотек, написанных на языке С для поддержки видео-библиотеки OpenGL. Полученный набор библиотек при переносе их в зашщищенный режим, позволит обеспечить решение всех типовых задач бортового ПО в защищенном режиме.

О финансировании

На выполнение внутреннего ОКР «Защищенный режим» было потрачено **3,9 миллиона рублей**. Поиск финансирования для продолжния работ по развитию технологии защищенного режима производился в следующих направлениях:

1) Поиск средств среди целевой аудитории «авиационный клуб». Результат — договор ЦНПО «Ленинец» - ПАО «ИНЭУМ им. И.С. Брука», в стадии подготовки — договоры ПАО «Компания Сухой» - ПАО «ИНЭУМ им. И.С. Брука», АО «РПКБ» - ПАО «ИНЭУМ им. И.С. Брука». Работы предусмотренные договорами относятся к смежной теме, особенно в части формирования набора библиотек на языке С, необходимых для задач бортовых приложений.

2) Подготовка проекта «Процессор», финансируемого из Фонда Перспективных Исследований. В проекте предусмотрено финансирование темы «Защищенный режим» в размере 40 миллионов рублей в течение 2 лет. Проект готов к защите на НТС ФПИ. В случае положительного решения по финансированию средства ФПИ поступят в апреле 2018 года.

Для продолжения работы с января 2018 необходимо открытие нового внутреннего ОКР «Защищенный режим-2» с бюджетом **4 миллиона рублей**.

Спасибо за внимание