Семинар
«Технологии QNX и ЗОСРВ «Нейтрино»
в России»

Москва, 19 апреля 2016

«Особенности кросс-разработки под 64-разрядную платформу Нейтрино-Эльбрус»

Докучаев Андрей, СВД Встраиваемые Системы

SWD Software

СВД Встраиваемые Системы

Содержание

- **Инструментарий разработчика**
  - **Компилятор, интеграция в IDE, средства бинарного анализа**
  - **Отладка, трассировка ядра, удаленное взаимодействие**
- **64-разрядное окружение среды исполнения**
  - **Формат исполняемых модулей и адресное пространство процесса**
  - **Переносимость кода между 32- и 64- разрядными окружениями**
  - **Перенос имеющихся наработок на платформу Нейтрино-Эльбрус**

Компилятор LCC

В состав архива с кросс-компилятором LCC *входит собственно оптимизирующий компилятор и пакет* binutils.

Компилятор/линкер доступен посредством следующих псевдонимов и скриптов из состава КСР:

```
# qcc -Vgcc_ntoe2k
# e2k-unknown-nto-gcc    → ntoe2k-gcc
# ntoe2k-gcc             → e2k-linux-gcc
# e2k-linux-lcc
```

*Компилятор языка C++ (***ntoe2k-g++ / e2k-linux-l++***) используется аналогично.*

Работа с исполняемыми файлами платформы

Штатный инструментарий пакета `binutils` *может быть вызван при помощи следующих псевдонимов:*

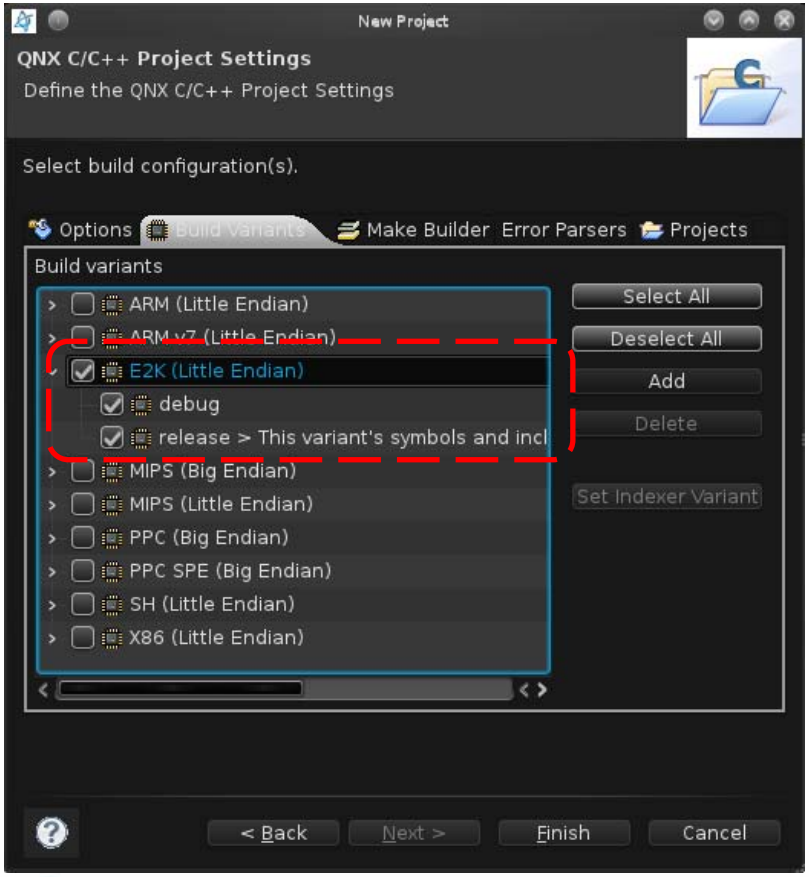
```
# e2k-unknown-nto-readelf   →   ntoe2k-readelf
# ntoe2k-readelf            →   e2k-linux-readelf
# e2k-linux-readelf
```

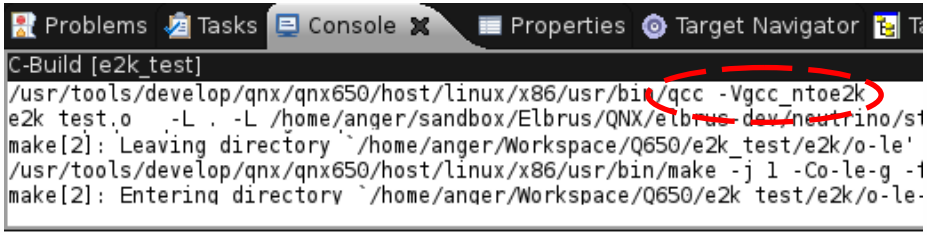
Все остальные утилиты, включая одноименные модули хост системы, дальше заголовков ELF-файлов продвинуться не смогут.

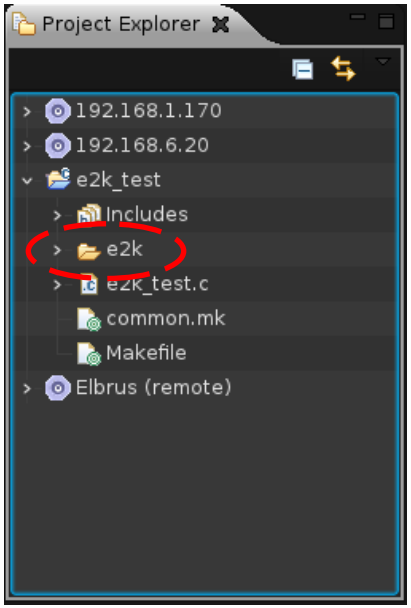
В стандартную поставку также включены QNX-специфичные компоненты, адаптированные для работы с 64-разрядными ELF-файлами:

- *утилиты* **use** *и* **usemsg** *для формирования и считывания USE-сообщений;*
- *утилиты* **ldrel**, **bindres** *для линковки в Photon-приложения ресурсов;*
- *утилиты* **mk*-64** *для создания 64-разрядных загрузочных образов QNX.*

Интеграция в QNX Momentics IDE







Интеграция в IDE осуществляется автоматически, если установка была завершена без ошибок.

SWD Software

СВД Встраиваемые Системы

Содержание

- **Инструментарий разработчика**
  - Компилятор, интеграция в IDE, средства бинарного анализа
  - **Отладка, трассировка ядра, удаленное взаимодействие**
- 64-разрядное окружение среды исполнения
  - Формат исполняемых модулей и адресное пространство процесса
  - Переносимость кода между 32- и 64- разрядными окружениями
  - Перенос имеющихся наработок на платформу Нейтрино-Эльбрус

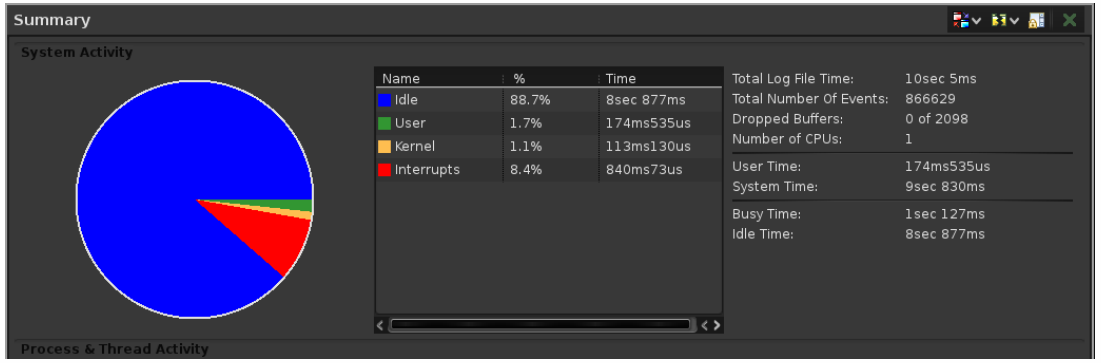
Отладка и трассировка

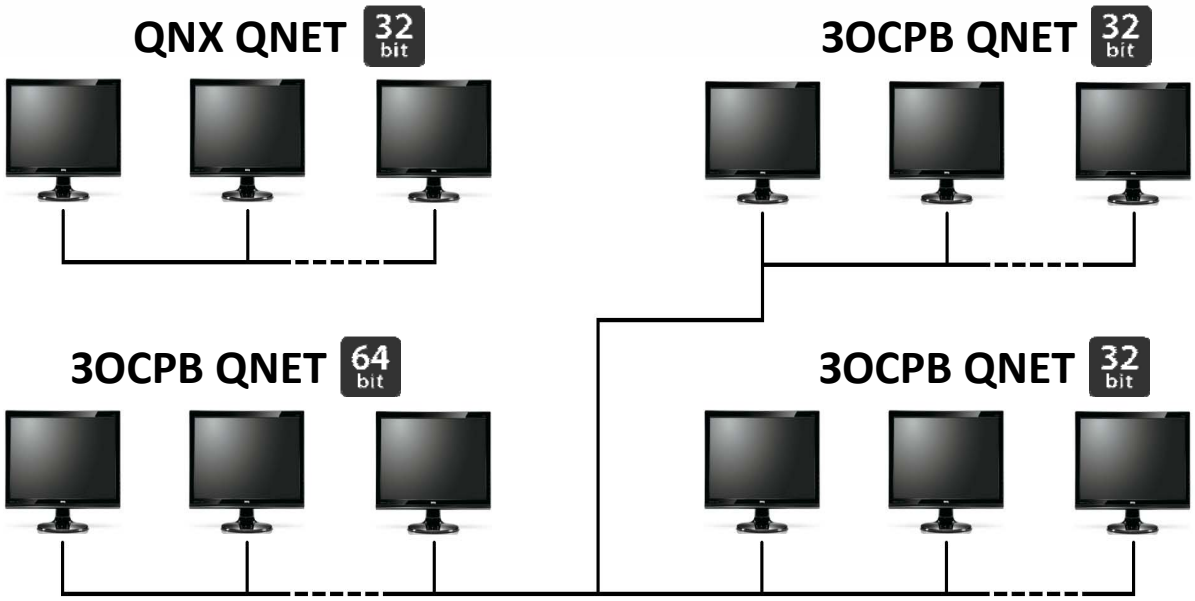
В настоящий момент возможна отладка только по посмертным дампам. В QNX за этот сервис отвечает сервер **`dumper`**. *Пример запуска:*

```
# dumper -d /tmp &
# slay -sSIGSEGV slogger    →      /tmp/slogger.core
```

Завершение портирования отладчика GDB 7.2 ожидается в 2016 году.

Трассировка полнофункциональна: как через IDE, так и вручную при запуске сервера **`tracelogger`**:





Средствами защищенной сети QNET достигается взаимодействие систем с различной разрядностью без дополнительной настройки.

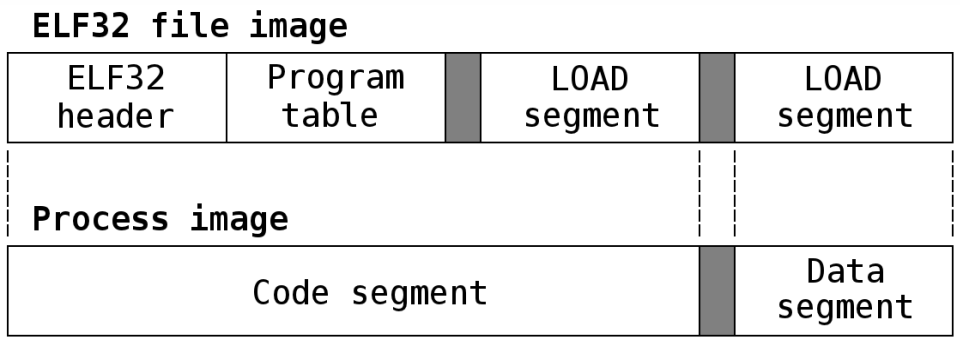
Для удаленного взаимодействия также доступны:

- *Сервер **qconn** для поддержки базового функционала IDE;*
- *Защищенный стек сетевых протоколов TCP/IP;*
- *Консоль на последовательном порту;*
- *VNC / phinx / phindows.*

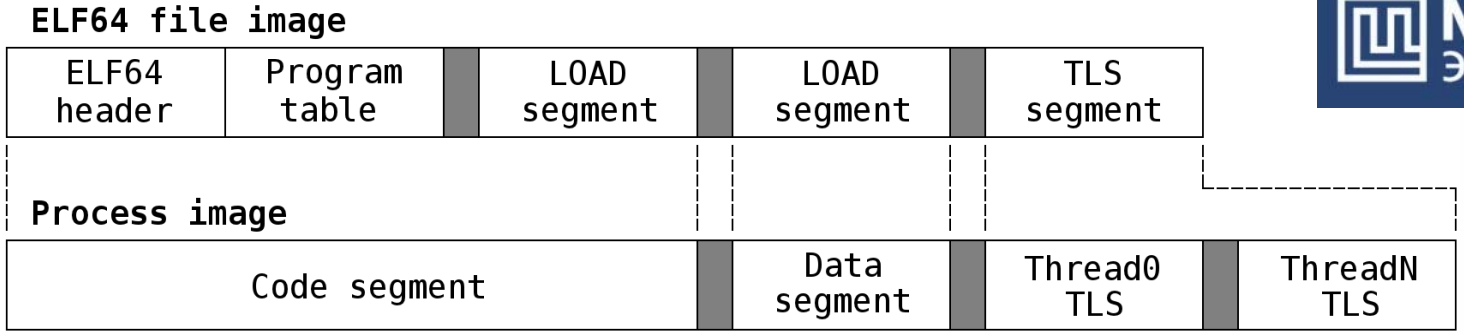
Содержание

- Инструментарий разработчика
  - Компилятор, интеграция в IDE, средства бинарного анализа
  - Отладка, трассировка ядра, удаленное взаимодействие
- **64-разрядное окружение среды исполнения**
  - **Формат исполняемых модулей и адресное пространство процесса**
  - Переносимость кода между 32- и 64- разрядными окружениями
  - Перенос имеющихся наработок на платформу Нейтрино-Эльбрус

64-разрядное окружение среды исполнения
Формат исполняемых модулей и ABI



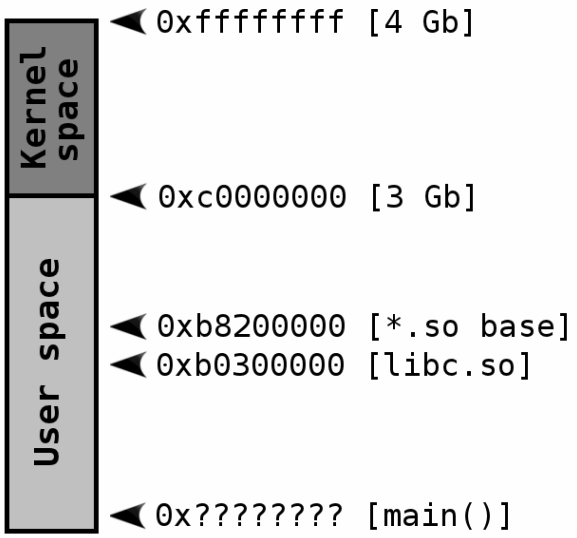
Различия исполняемых модулей проявляются как на этапе линковки, так и на этапе исполнения. Реализации Application Binary Interface (ABI) 32-разрядного и 64-разрядного окружения компиляторов GCC и LCC не совместимы.



32-разрядное адресное пространство непривилегированного процесса

– *адресное пространство ограничено 32 битами (**адресуется до 4 Gb**);*

– *непривилегированному коду **из них доступно 3 Gb независимо от архитектуры**.*



Использование доступных процессу 3 Gb адресного пространства

– ***сегменты кода и данных** самого процесса и всех разделяемых библиотек;*

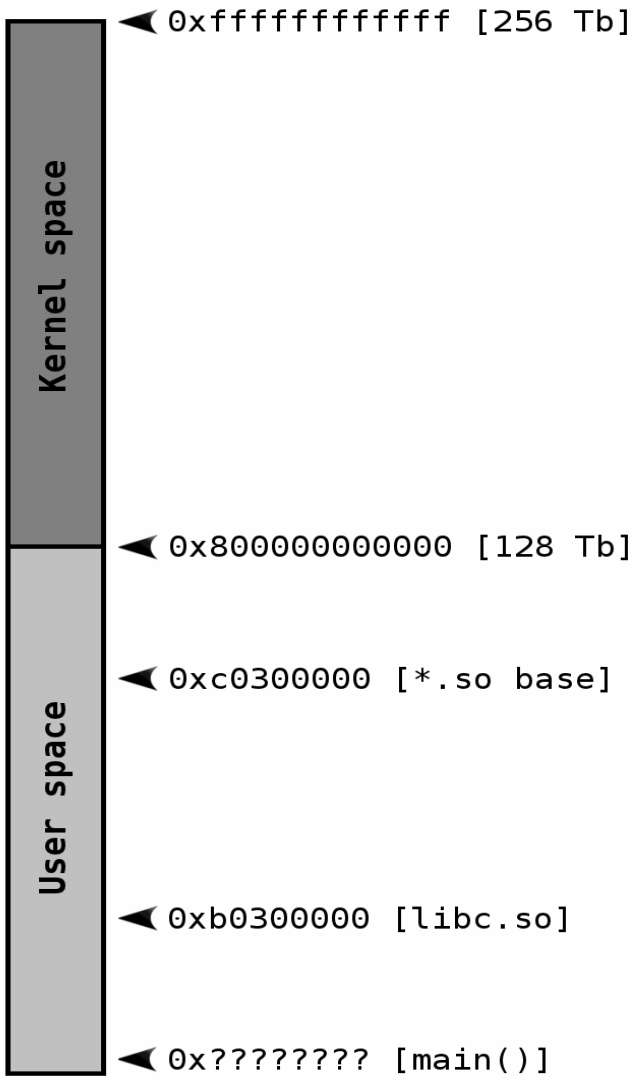
– ***стеки** потоков;*

– ***TLS** (Thread Local Storage) всех потоков;*

– ***динамически выделенная**/разделяемая память;*

– *подключенная через* `mmap()` *физическая память.*

64-разрядное окружение среды исполнения
Адресное пространство процесса



Адресное пространство пользовательского процесса в 64-разрядном окружении

- *виртуальный адрес ограничен 48 битами (**адресуется до 256 Tb**);*
- *физический адрес ограничен 40 битами (1 Tb);*
- *непривилегированный код может адресовать нижнюю половину адресного пространства (**доступно до 128 Tb**);*
- *отказ от автоматического управления адресным пространством (управление виртуальной адресацией при использовании* `mmap()`*) **позволяет обеспечивать обратную совместимость** с 32-разрядными системами.*

Выравнивание данных на границе страниц виртуальной памяти

- *отсутствие выравнивания может вызывать **исключение процессора** и приводить к отмене части уже исполненной текущей широкой команды;*
- *ядро ОС прерывает работу, выполняет логику обработки исключения, включая **длительное восстановление контекста** работы с памятью.*

Диагностика и оптимизация

- ***трассировка ядра**: анализ процессов, чье исполнение генерирует большое число исключений процессора;*
- ***временный механизм**: диагностические сообщения с опцией ядра “-vvvv”.*

```
#
 exception = exc_data_page ip=c067cc98 thp 80000f002020 CPU 0 inkernel 0
trap_cellar(): cnt=0 addr=000b4ffc data=b4da800000000 condition=560001110023000
trap_cellar(): fault type=00000002 - page_bound inkernel c00
trap_cellar(): cnt=1 addr=0006f060 data=00000000 condition=5a000031030332d
```

“page_bound”

Содержание

- Инструментарий разработчика
  - Компилятор, интеграция в IDE, средства бинарного анализа
  - Отладка, трассировка ядра, удаленное взаимодействие
- **64-разрядное окружение среды исполнения**
  - Формат исполняемых модулей и адресное пространство процесса
  - **Переносимость кода между 32- и 64- разрядными окружениями**
  - Перенос имеющихся наработок на платформу Нейтрино-Эльбрус

Совместимость на уровне исходного кода

```
printf( "%d ",  sizeof( char ) );
printf( "%d ",  sizeof( int ) );
printf( "%d ",  sizeof( long ) );
printf( "%d ",  sizeof( long long ) );
printf( "%d ",  sizeof( float ) );
printf( "%d ",  sizeof( double ) );
printf( "%d ",  sizeof( void * ) );
printf( "%d ",  sizeof( int8_t ) );          /* stdint.h */
printf( "%d ",  sizeof( int16_t ) );         /* stdint.h */
printf( "%d ",  sizeof( int32_t ) );         /* stdint.h */
printf( "%d ",  sizeof( int64_t ) );         /* stdint.h */
```

32 bit: **1 4 4 8 4 8 4 1 2 4 8**

64 bit: **1 4 8 8 4 8 8 1 2 4 8**

Количество памяти, требуемое для хранения данных и особенно указателей необходимо учитывать при любых преобразованиях типов.

Совместимость на уровне исходного кода

```
malloc( size );                    /* Максимальное значение для size? */
```

32 bit **0x80000000 (2 Gb)**

64 bit **Не ограничено**

```
addr = malloc();                   /* Тип данных для хранения адреса? */
```

32 bit
- **int (не переносимо)**
- **uint32_t (не переносимо)**
- **void ***

64 bit
- **long**
- **uint64_t**
- **void ***

Совместимость на уровне данных

Файлы, сформированные следующим образом:

```
struct db_item { long b; void *c; int d; };
...
write( fd, &variable, sizeof( struct db_item ) );
read(  fd, &variable, sizeof( struct db_item ) );
```

не пригодны к переносу между системами с различной разрядностью.

```
int  addr = (int)&buffer;              /* &buffer → 0x80000000
*/
void *ptr = addr;
```

32 bit — **ptr → 0x80000000**

64 bit — **ptr → 0xffffffff80000000**

Спекулятивный режим выполнения команд процессора Эльбрус

*Данный режим **требует в обязательном порядке** доступности данных в кэше процессора. Режим отключается опцией компилятора* **`-fcontrol-spec`**.

Поскольку ОС Эльбрус не предоставляет механизма управления кэшированием данных для прикладного кода, компилятор по умолчанию формирует код для спекулятивного выполнения.

Доступ к памяти, выделенной следующим образом:

```
void *ptr = (void *)mmap( ... PROT_NOCACHE ... );
```

с большой вероятностью приведет к попытке обращения не к памяти, а к кэшу, если не был отключен спекулятивный режим.

Физическая адресация

```
int mem_offset64( ..., off64_t * offset, ... );
```

При инициализации DMA контроллеров внешней периферии имеет смысл обращать внимание на способность устройства адресоваться ко всему диапазону физических адресов, поддерживаемому архитектурой. Эльбрус (архитектура E2K) имеет аппаратное ***ограничение в 40 бит*** *при физической адресации.*

Для корректной работы драйверов реализовано ***расширение для функций mmap*(), позволяющее выделять физические адреса в пределах 32 бит****:*

```
mmap64( 0, 0x1000, PROT_READ | PROT_WRITE | PROT_NOCACHE,
        MAP_PHYS | MAP_ANON | MAP_BELOW4G, NOFD, 0 );
```

Константа объявлена в <sys/mman.h> ***только для*** 64 bit ***систем****.*

Физическая адресация

Рекомендуемым способом *выделения физической памяти, адресуемой 32 битами является использование типизированной памяти. Перенос системного ПО между системами с различной разрядностью в этом случае обеспечит функциональную совместимость системной библиотеки. Тем не менее, типизированная память должна быть определена модулем* **`startup`**.

Актуальный пример для архитектуры **`e2k`**:

```
fd = posix_typed_mem_open( “/below4G/ram”, O_RDWR, ... );

addr = mmap64( NULL, size, PROT_READ | PROT_WRITE,
               MAP_SHARED, fd, 0 );
```

Встроенные функции компилятора (C extensions)

Реализация **`__builtin*()`** *расширений в компиляторе LCC для архитектуры Эльбрус может существенно отличаться от аналогичного функционала GCC для 32-разрядных архитектур.*

*Например, представленные ниже расширения в реализации LCC допускают обращение только к предыдущему уровню стека (*level = 0*):*

```
void * __builtin_return_address( unsigned int level );
void * __builtin_frame_address(  unsigned int level );
```

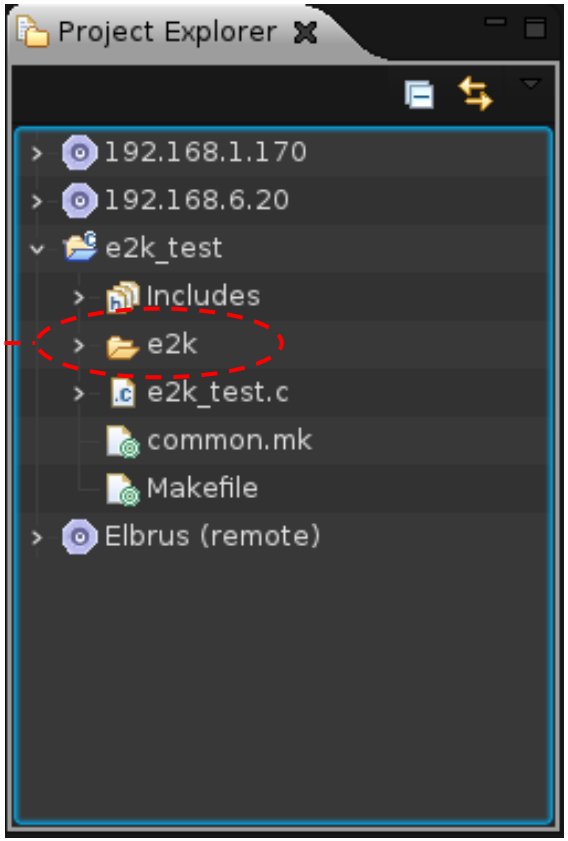
Содержание

- Инструментарий разработчика
  - Компилятор, интеграция в IDE, средства бинарного анализа
  - Отладка, трассировка ядра, удаленное взаимодействие
- **64-разрядное окружение среды исполнения**
  - Формат исполняемых модулей и адресное пространство процесса
  - Переносимость кода между 32- и 64- разрядными окружениями
  - **Перенос имеющихся наработок на платформу Нейтрино-Эльбрус**

Условная компиляция (в зависимости от разрядности адреса)

```
struct db_item {
#ifndef __64__
    long                data32;
#else
    int                 data32;
#endif
};
```

```
#ifndef __64__
typedef long            s32;
typedef long            pointer;
#else
typedef int             s32;
typedef uintptr_t       pointer;
#endif
```



Условная компиляция (в зависимости от архитектуры)

```
#include <sys/elf.h>

void                    *dma_base = NULL;
#ifdef __E2K__
Elf64_Ehdr              ehdr = NULL;

dma_base = mmap64( 0, 0x1000, PROT_READ | PROT_WRITE |
                   PROT_NOCACHE, MAP_PHYS | MAP_ANON |
                   MAP_BELOW4G, NOFD, 0 );
#else
Elf32_Ehdr              ehdr = NULL;

dma_base = mmap64( 0, 0x1000, PROT_READ | PROT_WRITE |
                   PROT_NOCACHE, MAP_PHYS | MAP_ANON,
                   NOFD, 0 );
#endif
```

Ограничения TLS при использовании вызовов ядра Thread*()

Использование явно вызовов ядра `Thread*()` вместо функций `pthread*()` не рекомендуется. *В некоторых случаях это может приводить к неработоспособности ряда механизмов TLS-адресации. Исключением являются TLS-объекты, организованные на базе* **`pthread_key_create()`**.

Замечание справедливо для C++ программ, а также для кода на Си, вида: ----------→

```
__thread int v;

void * thread( ... )
{
    v = 1;
    return (0);
}

int main()
{
    ...
    ThreadCreate( ... thread ... );
    ...
}
```

Спасибо за внимание

Андрей Докучаев
Ведущий инженер-программист

(812) 346-89-56 доп. 104
support@kpda.ru

www.kpda.ru

SWD Software

СВД Встраиваемые Системы