
Технологии QNX и КПДА в России

Москва, 25 апреля 2018

# Практические аспекты разработки и отладки сценариев (make/ksh)

Игорь Рондарев, ООО «СВД Встраиваемые Системы»

СВД Встраиваемые Системы

SWD Software

# Содержание

- Сценарии сборки программных продуктов (**Makefiles**)
- Сценарии командного интерпретатора (т. н. «**скрипты**» или «**shell scripts**»)

# Часть 1: Сценарии сборки программных продуктов (**Makefiles**)

- Сценарии сборки программных продуктов (**Makefiles**)
  - Основные способы создания Makefile-проектов для ЗОСРВ «Нейтрино» и QNX
  - Дополнительные инструменты разработчика
    - **remake** — расширенный вариант **GNU Make**

# Сценарии сборки программных продуктов (**Makefiles**)

- Сборка приложений для **ОСРВ QNX 6.x/7.x** и **ЗОСРВ «Нейтрино»** для всех поддерживаемых целевых вычислительных архитектур осуществляется с помощью инструмента **make** (реализация - **GNU Make**)
- Поддержка разнообразных целевых платформ и опций построения проекта достигается за счёт использования собственной подсистемы расширений (т. н. *mk-файлов*)
  - Пример:
    - `make` **`CPULIST=arm VARIANTLIST=g`** `install`

# Сценарии сборки программных продуктов (**Makefiles**)

- Структура Makefile

```
цель1: [зависимости]
   [действие1]
   [действие2]

цель2: [зависимости]
   [действия]
```

- Пример:

```
all: hello

test:
   @echo This is test!
```

- **Неявные цели (т. н. Implicit Rules)**
  - Позволяют выполнять базовые действия
    - Пример: **make all → cc hello.c -o hello**
  - Отключаются (в отладочных целях) с помощью ключа **«-r»**

# Сценарии сборки программных продуктов (**Makefiles**)

- Состав сборочной среды (на примере GNU/Linux):
  - компиляторы, линковщики и другие вспомогательные инструменты (**gcc + binutils**)
    - не рассматриваем, т. к. за пределами данной темы
  - утилита **make** (файл **${QNX_HOST}**/usr/bin/make)
    - Работает по сценариям, задаваемым файлами **Makefile/GNUmakefile** и подключаемыми из них компонентами (обычно имеют расширения «**.mk**»)

# Сценарии сборки программных продуктов (**Makefiles**)

- Состав сборочной среды (продолжение):
  - **подключаемые компоненты** (**recurse.mk, qrules.mk, qtargets.mk** и т. д.)
    - Являются «ядром» сборочной системы
    - Отвечают в т.ч. за поддержку целевых архитектур
    - Расположены в каталоге **${QNX_TARGET}/usr/include/mk**
  - переменная окружения **MAKEFLAGS**
    - Указывает расположение файлов для директивы «include», используемой в Makefile`ах
      - Пример: **MAKEFLAGS=-I${QNX_TARGET}/usr/include**
    - Обычно устанавливается при загрузке инструментальной системы

# Создание проектов под ЗОСРВ «Нейтрино» и QNX

- **Типовые задачи**
  - Создание нового проекта «с нуля»
  - Создание нового проекта на базе существующего исходного кода
  - Портирование Open Source-проектов; в частности, использующих сборочную систему Autotools (**autoconf/automake**)
    - «**./configure** --target=i486-... → **make** → **make install**»
  - и т.д.
- **Возможные варианты решений**
  - Базовые утилиты командной строки
    - подготовка проекта для сборки под **QNX / ЗОСРВ «Нейтрино»** с помощью утилит «**addvariant**» и «**delvariant**»
  - Momentics IDE:
    - создание и импорт проектов/каталогов с исходными кодами
    - **Momentics 7**: импорт с поддержкой **GNU Autotools Toolchain**

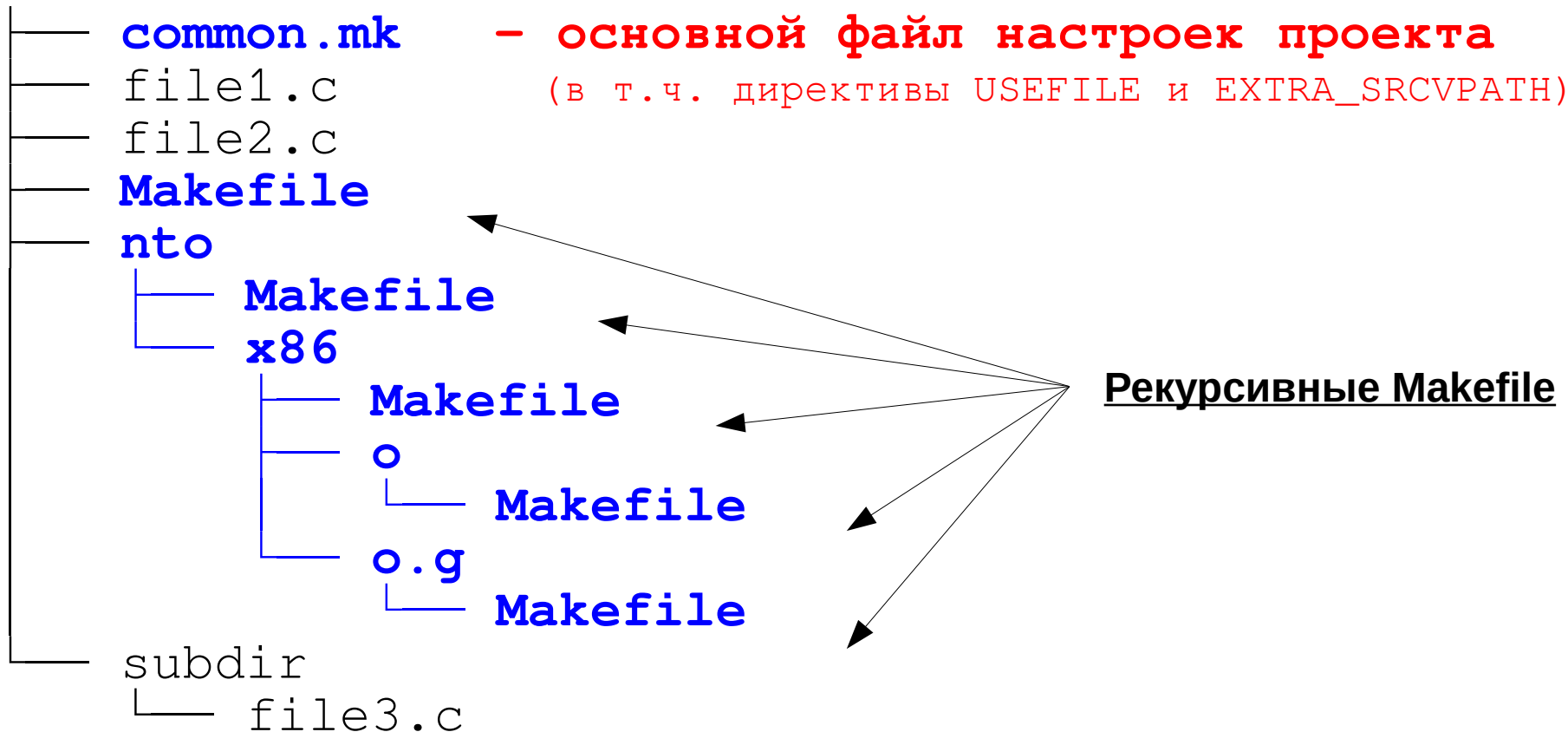
# Структура проекта QNX / ЗОСРВ «Нейтрино» (вариант)

Дано: каталог с файлами «исходников»

```
$ tree
.
├── file1.cc
├── file2.cc
└── subdir
    └── file3.cc
$ i486-pc-nto-qnx6.5.0 -c file1.c -o file1.o
$ i486-pc-nto-qnx6.5.0 -c file2.c -o file2.o
$ i486-pc-nto-qnx6.5.0 -c subdir/file3.c -o
subdir/file3.o
$ i486-pc-nto-qnx6.5.0-ld -o myApp \
<длинная_последовательность_параметров_и_файлов>
```

# Структура проекта QNX / ЗОСРВ «Нейтрино» (вариант)

```
$ addvariant -i OS nto x86 o     - Release-версия
$ addvariant -i OS nto x86 o.g   - Debug-версия
```



# Сборка проекта с помощью make

```
$ make
make -j 1 -Cnto -fMakefile
make[1]: Entering directory '/home/kpda/mk_msk2018/nto'
make -j 1 -Cx86 -fMakefile
make[2]: Entering directory '/home/kpda/mk_msk2018/nto/x86'
make -j 1 -Co -fMakefile
make[3]: Entering directory '/home/kpda/mk_msk2018/nto/x86/o'
/opt/qnx650/host/linux/x86/usr/bin/qcc -Vgcc_ntox86 -c -Wc,-Wall  -O -DNDEBUG   -I.
-I/home/kpda/mk_msk2018/nto/x86/o -I/home/kpda/mk_msk2018/nto/x86
-I/home/kpda/mk_msk2018/nto -I/home/kpda/mk_msk2018 -I/home/kpda/mk_msk2018/subdir1
-I/opt/qnx650/target/qnx6/usr/include     -DBUILDENV_qss  /home/kpda/mk_msk2018/file1.c
/opt/qnx650/host/linux/x86/usr/bin/qcc -Vgcc_ntox86 -c -Wc,-Wall  -O -DNDEBUG   -I.
-I/home/kpda/mk_msk2018/nto/x86/o -I/home/kpda/mk_msk2018/nto/x86
-I/home/kpda/mk_msk2018/nto -I/home/kpda/mk_msk2018 -I/home/kpda/mk_msk2018/subdir1
-I/opt/qnx650/target/qnx6/usr/include     -DBUILDENV_qss  /home/kpda/mk_msk2018/file2.c
/opt/qnx650/host/linux/x86/usr/bin/qcc -Vgcc_ntox86 -c -Wc,-Wall  -O -DNDEBUG   -I.
-I/home/kpda/mk_msk2018/nto/x86/o -I/home/kpda/mk_msk2018/nto/x86
-I/home/kpda/mk_msk2018/nto -I/home/kpda/mk_msk2018 -I/home/kpda/mk_msk2018/subdir1
-I/opt/qnx650/target/qnx6/usr/include     -DBUILDENV_qss
/home/kpda/mk_msk2018/subdir1/file3.c
/bin/rm -f  /home/kpda/mk_msk2018/nto/x86/o/mk_msk2018
/opt/qnx650/host/linux/x86/usr/bin/qcc -Vgcc_ntox86
-o/home/kpda/mk_msk2018/nto/x86/o/mk_msk2018    file1.o    file2.o    file3.o   -L . -L
/opt/qnx650/target/qnx6/x86/lib -L /opt/qnx650/target/qnx6/x86/usr/lib  -Wl,--rpath-
link . -Wl,--rpath-link /opt/qnx650/target/qnx6/x86/lib -Wl,--rpath-link
/opt/qnx650/target/qnx6/x86/usr/lib
make[3]: Leaving directory '/home/kpda/mk_msk2018/nto/x86/o'
make[2]: Leaving directory '/home/kpda/mk_msk2018/nto/x86'
make[1]: Leaving directory '/home/kpda/mk_msk2018/nto'
```

- **Основные недостатки GNU Make**
  - малая информативность вывода по умолчанию, в т.ч. информации об ошибках
    - Пример:
      - `Makefile:1: *** [all] Error 127`
    - Ситуация частично исправлена в **GNU Make 4.x**
  - перегруженность вывода диагностического режима (**make -d / make --debug=v**)
  - средства отладки сценариев не предусмотрены
    - **target1:**

```
@echo [DEBUG] Start of target1
…
@echo [DEBUG] End of target1
```

- **«Remake – GNU Make with comprehensible tracing and a debugger»**
  - Независимый проект (http://bashdb.sourceforge.net/remake/)
    - Основан на исходном коде **GNU Make**, активно развивается
  - Открытый исходный код + готовые сборки для различных дистрибутивов **GNU/Linux** (также возможная сборка под **Win32**):
    - Пример (для **Debian GNU/Linux**)
      - **`sudo apt install remake`**

# Remake

- **«Remake – GNU Make with comprehensible tracing and a debugger»***
  - Возможности:
    - Более **подробный вывод** информации об ошибках и целях (**targets**) по сравнению с обычным **GNU Make**
    - Отображение стека вызовов
    - **Встроенный профилирощик**, в т.ч. с возможностью построения графиков
      - Используется формат «callgrind» (см. **Valgrind**)
    - **Встроенный пошаговый отладчик**
      - точки останова (**breakpoints**), исследование значений переменных и т. д.
  - Область применения
    - Изучение, отладка и оптимизация имеющихся проектов (в т.ч. с использованием механизмов профилирования)

# Remake — основные «ключи»

- **remake --tasks / remake --targets**
  - Позволяет «изучать» возможные «цели» (**targets**) в текущем каталоге (или явно задаваемом с помощью ключа **«-C»**)
  - Пример:
    - «Общеупотребительные»: **make all, make clean, make.**
    - Вопрос: какими ещё возможностями обладает проект?
      - Что даёт: эффективное использование существующих и отладка создаваемых «целей»
- **remake -x,--trace**
  - Расширенная информация о процессе выполнения сборки проектов
- **remake -X,--debugger**
  - **Отладчик……**

# Remake — режим пошаговой отладки

- **remake -X**
  - GNU Debugger (**gdb**)-подобный интерфейс
  - Команды:
    - **break, delete** — работа с точкими останова (**breakpoints**)
    - **step, continue** — перемещение по Makefile
    - **info** — информация о текущем состоянии
      - Пример:
        - **info variables; print <var>** - просмотр переменных и их значений
    - **bt** — стек
    - **pwd** — текущая директория
    - и т. д.
- **remake -!, --post-mortem**
  - остановка и переход в отладчик сразу после ошибки

# Документация

- **Remake**
  - Официальная документация (Wiki, видеопримеры):
    - http://bashdb.sourceforge.net/remake/remake.html/index.html
- **Сборочная система QNX / ЗОСРВ «Нейтрино»**
  - Раздел «**Conventions for Recursive Makefiles and Directories**» документации по **QNX SDP 6.5.0**
    - для QNX SDP 7.0 также актуален с некоторыми изменениями
  - «Встроенная» справочная информация:
    - `use addvariant; less $(which addvariant)`

# Часть 2: сценарии командного интерпретатора

- Сценарии командного интерпретатора (т. н. «**скрипты**» или «**shell scripts**»)
  - Области применения и основные задачи
  - Дополнительные инструменты разработчика
    - **shellcheck** — статический анализатор

# Сценарии командного интерпретатора

- **Командный интерпретатор** — инструмент, предназначенный для организации взаимодействия с операционной системой
  - Предназначен для выполнения как индивидуальных команд, так и заранее записанных последовательностей (т. н. сценариев или «**скриптов**»)
  - Примеры интерпретаторов: **sh, bash, csh, fesh и т.д.**
- Пример «скрипта» (файл ***delfile.sh***):

```
#!/bin/ksh
SOMEFILE=$1
echo "Have a nice Day! Let's delete some file.."
rm -rf /${SOMEFILE}
```

# Сценарии командного интерпретатора

- Применение в **QNX** и **ЗОСРВ «Нейтрино»**
  - В **ЗОСРВ «Нейтрино»**, а также в **QNX 6.5 и 7.0** основным является интерпретатор **ksh** (Korn Shell, вариант **PD KSH v5.2.14**)
  - Начиная с редакции 2018 года, в состав операционной системы **ЗОСРВ «Нейтрино»** включён и установлен в качестве основного командный интерпретатор **mksh** (**Mir BSD ksh**), являющийся прямым последователем **pdksh.**
    - Интерпретатор актуализирован, внесён ряд улучшений и исправлений

- **Создание сценариев**
  - Пример: автоматизация различных операций (сборка ФПО, создание образов целевых систем, резервное копирование данных и т. д. и т. п.)
  - **Возможные проблемы:**
    - Неработоспособность или некорректное функционирование сценариев при запуске на целевой системе

- **Адаптация существующих сценариев**
  - Пример: перенос программного обеспечения из других Unix-подобных операционных систем (напр., **Linux**)
  - **Возможные проблемы:**
    - «Диалекты» командных интерпретаторов могуть быть несовместимы между собой
    - Вопрос стандартизации
      - Например, по стандарту «**POSIX.2: Shell And Utilities**»

# Варианты решения возможных проблем

- **Варианты решений**
  - **Штатные**
    - Контрольный вывод (**echo**, **printf**)
    - Различные отладочные опции командных интерпретаторов (напр. «**set posix**», «**set verbose**», «**set xtrace**»)
      - см. **QNX Neutrino Realtime Operating System > Utilities Reference > K > ksh**
    - Редактирование и отладка на целевой или инструментальной системе
      - Linux: **${QNX_HOST}/usr/bin/ksh**
      - Windows: **%QNX_HOST%\usr\bin\ksh.exe**
  - **Дополнительные**
    - Использование статического анализатора
      - **shellcheck**
    - Использование внешнего отладчика
      - **bashdb**

# shellcheck — статический анализатор «скриптов»

- **«ShellCheck, a static analysis tool for shell scripts»**
  - Независимый проект (https://github.com/koalaman/shellcheck)
  - Открытый исходный код + готовые сборки для различных дистрибутивов **GNU/Linux**
    - Debian GNU/Linux: **`sudo apt install shellcheck`**
  - Возможности:
    - Обнаружение ошибок, потенциальных ошибок и узких мест
    - Проверка на соответствие стандартам
    - Пополняемая база данных (коды ошибок **SCxxxx**)

# Shellcheck — основные опции

- **shellcheck -s,--shell**
  - Выбор «диалекта» командного интерпретатора (**sh/bash/ksh**)
- **shellcheck -e CODE1[,CODE2], --exclude**
  - Отключить срабатывание определённых правил
  - Также возможно явное указание исключений в теле скриптов (используются комментарии)
    - Пример: **`#shellcheck disable=SC2035`**
- **shellcheck -e**
  - Поддержка различных форматов вывода (для интеграции в среды и инструменты разработки)

# Привер вывода

- Пример вывода (команда: $ **shellcheck *delfile.sh***)

```
rm -rf /${SOMEFILE}
```

```
^-- SC2115: Use "${var:?}" to ensure this never expands to / .
^-- SC2086: Double quote to prevent globbing and word splitting.
```

  - т. е. «сработали» два правила:
    - **SC2115**: рекомендация дополнить обращение к переменной **SOMEFILE** конструкцией «**:?**» во избежание получения конструкции «**rm -rf /**» в случае использования пустого или неинициализированного значения
    - **SC2086:** рекомендация заключать обращение к переменной в кавычки (например, для корректной обработки значений переменных, содержащих пробелы)

# Вопросы?

- Вопросы?

# Спасибо за внимание

**Игорь Рондарев**

инженер-программист

ООО «СВД Встраиваемые Системы»

тел.: +7 (812) 346-8956
факс: +7 (812) 346-8953

http://www.kpda.ru | http://forum.kpda.ru |
http://www.swd.ru

СВД Встраиваемые Системы

SWD Software