Сборка кросс-компилятора Amiga GCC 3.3.3 на Linux или Windows

от Троя Сильвея

Это руководство может помочь Вам собрать кросс-компилятор GCC версии 3.3.3 на Linux или Windows XP. Я успешно проверил все положения этого руководства на дистрибутивах Red Hat и Windows XP. Другие руководства советуют использовать GCC 2.95, но неудачная сборка с их помощью вынудила меня выработать собственные инструкции, которые я и предлагаю Вашему вниманию. Дополнительно мной была вкратце описана сборка библиотеки и освещены вопросы выделения Chip-памяти при использовании GCC.

Пожалуйста, перечитайте это руководство несколько раз перед тем как начать использовать его на практике. Это существенно сэкономит Вам время и избавит от раздражения в последствии. Считайте что перед Вами черновик настоящего полноценного руководства, причём несколько раз испытанный на разном специфическом оборудовании. Резервная копия всех Ваших данных также не будет лишней. Я рекомендую начинать с чистой только что установленной ОС, которую Вы сможете легко удалить и начать заново, если дела пойдут не так как Вы ожидаете. Но этого ведь никогда не случится, правда? :)

Моя задача в том чтобы помочь Вам собрать собственную среду разработчика на базе Linux или Windows XP. Мы будем использовать Linux-подобную среду Cygwin на Windows XP для сборки и эксплуатации кросс-компилятора GCC способного к созданию исполняемых файлов для AmigaOS 3.x и AmigaOS 4.x (а м.б. и AmigaOS 2.x). Возможно Вы нацелены на получение PPC-кода и будете указывать компилятору цель *TARGET=powerpc-amigaos*, но у меня нет компьютера с процессором PowerPC, чтобы проверить такую возможность. Использование UAE (http://www.winuae.net) или AmigaForever (http://www.amigaforever.com) позволит Вам получить среду для создания, тестирования и запуска приложений на Windows, Linux и AmigaOS. Также это руководство поможет Вам собрать среду разработчика для кросс-компиляции на базе Linux, если Вы предпочитаете этот путь. В этом случае просто игнорируйте раздел посвящённый установке Cygwin и сразу переходите к разделу посвящённому Linux.

Необходимости

Нижеизложенное написано для помощи начинающим программистам и не предназначено для лиц осведомлённых в основах программирования для различных ОС и используемых в них компиляторов. Невозможно было охватить в этих нескольких страницах всю проблематику, но если Вы — начинающий программист, то Вам следует внимательно следовать инструкциям. Вы будете должны хранить файлы в специально созданных директориях и это должны быть именно те файлы и именно те директории которые необходимы компилятору. Отсутствие необходимого файла или невозможность использования ожидаемой директории (CD) сделает сборку скорее всего невозможной. В процессе сборки будет появляться много предупреждений (warning). Такие сообщения не влияют на возможность сборки, а получаемый в результате исполняемый файл оказывается как правило работоспособным. Однако, если Вы столкнётесь с сообщением о настоящей ошибке

(error), это означает что сборка скорее всего не удастся. Существует несколько ошибок-исключений не мешающих сборке, о которых я расскажу позже.

Так что... Вы конечно эксперт в Amiga, виртуозно владеете MS Windows, немного нахватались основ по Linux и умеете пользоваться текстовыми редакторами. Даже если это не так, опыт кросс-компиляции позволит Вам прыгнуть немного выше Вашей головы. Вы всё равно можете многому научиться и многое попробовать. Всегда есть возможность попросить помощи на форумах посвящённых Amiga, поискать на сайтах в сети Internet другие руководства и инструкции. Если Вы занимаетесь сборкой на Windows Вы можете активно пользоваться буфером обмена через Блокнот и для Вас это будет более лёгким путём внесения изменений в среду Cygwin Linux, нежели использование редактора Vi (особенно если Вы никогда раньше не работали на Linux).

Для установки всех компонентов Вам потребуется высокоскоростное подключение к сети Internet. Медленное тоже подойдёт, но пройдёт несколько часов прежде чем Вы скачаете весь Cygwin. Необходим относительно производительный компьютер с предустановленной MS Windows XP, 2 Гб свободного пространства на винчестере и хотя бы 256 Мб памяти. Одно время я использовал для сборки Pentium II 350 Mhz и 128 Мб памяти. Это даже работало, но очень медленно. Особенно производительный PC рекомендуется для использования эмуляторов подобных UAE. Pentium 4 и 2 Гб памяти — очень хороший ПК для наших целей. Собранный проект легко может занимать 1,7 Гб на жёстком диске. Много пространства расходуется на среду Cygwin Linux, поэтому при сборке на Linux изначально такое кол-во дисковой памяти просто не понадобится.

Последнее что Вам понадобится: это специфичные файлы для Amiga, которые Вы можете получить из архива Aminet (http://aminet.net), Geek Gadgets и др. сайтов. Важно чтобы Ваши рабочие директории и файлы имели корректные пути. Когда подойдёт время выполнить «make» и приступить к сборке, система будет рассчитывать что необходимые ей файлы находятся в положенных им местах. Если сборка не удаётся, можно попробовать восстановить состояние файловой системы из резервной копии, после чего повторно убедиться что необходимые директории были созданы, перемещены или скопированы по корректным путям.

Важно. Если по каким-то причинам Вы захотите начать всё «с нуля», Вы можете удалять временные папки при сохранении всех загруженных файлов. Т.е. удалить список директорий и начать выполнение инструкций заново, в обход инструкции по скачиванию (я проделал это 30-40 раз за время написания руководства).

```
# rm -R ~/amigaos/build-binutils
# rm -R ~/amigaos/binutils-2.14
# rm -R ~/amigaos/build-gcc
# rm -R ~/amigaos/gcc-3.3.3
# rm -R ~/usr/local/amigatools
```

Приступим

1a (начните отсюда если сборка происходит на Windows XP)

Ok. У Вас установлена Windows XP, Вы подключены к сети Internet и готовы к работе. Первой в списке на установку будет среда Cygwin. Вы можете прочитать немного о Cygwin в Википедии (чтобы хотя бы знать что Вы устанавливаете). Посетите сайт проекта http://cygwin.com Отличное руководство в разделе «Installing Cygwin» на этом сайте описывает выбор всех компонентов которые нам будут нужны, поэтому Ваша задача - выполнить его в точности, обращая особое внимание на разделы «Devel», «Doc» и «Editor». Также важно включить BZip2 из раздела «Utils» и WGet из раздела «Web». Мы будем использовать эти 2 утилиты для загрузки и установки некоторых из наших программ. После установки Cygwin на Вашем рабочем столе появится иконка для запуска среды. Если Вы не хотите производить установку на диск C:\ , Вам потребуется отредактировать файл *INSTALLDRIVE:\cygwin\cygwin.bat* и изменить букву тома на предпочитаемую Вами или соответствующий сценарий просто не сможет запускать среду. Нажмите на иконку установленной среды Cygwin и переходите к шагу 2.

1b (начните отсюда если сборка происходит на Linux)

В этом руководстве ожидается что Вы будете работать под пользователем отличным от root. Если Вы хотите работать от имени root, создайте рабочую директорию и исправляйте команды приведённые в руководстве по мере необходимости. При работе от имени обычного пользователя, время от времени Вам потребуется выполнять команды от имени root, используя su или sudo т.к. обычный пользователь не имеет разрешений для доступа к некоторым файлам и директориям. Инструкция подскажет Вам когда необходимо будет выполнить команду с повышенными правами. Будьте готовы в этом случае ввести пароль пользователя root (при использовании команды su).

Если на Вашем Linux ещё не установлены средства разработки и версии библиотек для разработчика, они Вам понадобятся. В моей системе я использовал GCC 3.3.3 и GCC tools идущие в комплекте с моим дистрибутивом Red Hat, но у Вас вероятно будут другие версии. Если Вы будете придерживаться версии 3.x это будет хорошо, потому что у меня нет возможности узнать что будет установлено у Вас. Другие пользователи проходили эту инструкцию со сборкой GCC версий 2.7, 2.95, 3.1, 3.2, 3.4 и EGCS 1.1.2, так что всё должно получиться. Основа которая должны быть установлена: binutils, gcc, gdb, make, mktemp, bison и 2 утилиты: bzip2 и wget. (*Примечание: если Вас угораздило выполнять сборку на MacOS X, можете воспользоваться wget отсюда* - http://www.statusq.org/images/wget.zip).

2. (этот шаг будет идентичным для Windows и Linux)

Теперь мы создадим рабочие директории в нашей домашней директории и загрузим в них необходимые исходные файлы. Символы ~/ указывают на домашнюю директорию пользователя, а после символа # идут команды которые надо вводить в консоль (не надо набирать этот символ, он является просто подсказкой, которая в Вашей системе может отличаться).

```
# mkdir -p ~/amigaos/build-binutils/m68k-amigaos
# mkdir -p ~/amigaos/build-gcc/gcc/include
# mkdir ~/amigaos/gg-archives
# cd ~/amigaos
```

После перехода в директорию *~/amigaos* мы должны скачать необходимые файлы с исходными текстами GCC и binutils командой wget.

```
# wget http://adsl-065-006-130-241.sip.asm.bellsouth.net/files/binutils-2.14-src.tar.bz2
# wget http://adsl-065-006-130-241.sip.asm.bellsouth.net/files/gcc-3.3.3-m68k-src.tar.bz2
```

Теперь разархивируем их.

```
# bzip2 -dc binutils-2.14-src.tar.bz2 | tar -xf -
# bzip2 -dc gcc-3.3.3-m68k-src.tar.bz2 | tar -xf -
```

Загрузим следующие файлы с Geek Gadgets в директорию *gg-archive*.

```
# cd ~/amigaos/gg-archives
# wget http://gg.portail-e.com/geekgadgets/amiga/m68k/snapshots/current/bin/fd2inline-1.11-bin.tgz
# wget http://gg.portail-e.com/geekgadgets/amiga/m68k/snapshots/current/bin/ixemul-48.0-bin.tgz
# wget http://gg.portail-e.com/geekgadgets/amiga/m68k/snapshots/current/bin/ixemul-48.0-env-bin.tgz
# wget http://gg.portail-e.com/geekgadgets/amiga/m68k/snapshots/current/bin/ixemul-48.0-inc-bin.tgz
# wget http://gg.portail-e.com/geekgadgets/amiga/m68k/snapshots/current/bin/libamiga-bin.tgz
# wget http://gg.portail-e.com/geekgadgets/amiga/m68k/snapshots/current/bin/libnix-1.2-bin.tgz
```

Теперь мы можем разархивировать их в директорию *~/amigaos/build-binutils/m68k-amigaos*

```
# cd ~/amigaos/build-binutils/m68k-amigaos
# tar -xzf ../../gg-archives/fd2inline-1.11-bin.tgz
# tar -xzf ../../gg-archives/libamiga-bin.tgz
# tar -xzf ../../gg-archives/libnix-1.2-bin.tgz
# tar -xzf ../../gg-archives/ixemul-48.0-env-bin.tgz
# tar -xzf ../../gg-archives/ixemul-48.0-inc-bin.tgz
# tar -xzf ../../gg-archives/ixemul-48.0-bin.tgz
```

Создайте директорию *amigatools* для исполняемых файлов которые будут собраны в будущем. Поскольку для создания директорий в */usr/local* требуются права root, Вам потребуется ввести команду su.

```
# su – root (этот шаг пропускается при сборке на Windows/Cygwin)
# mkdir -p /usr/local/amigatools/lib
# mkdir -p /usr/local/amigatools/m68k-amigaos/sys-include
# mkdir -p /usr/local/amigatools/m68k-amigaos/include/dos
# chmod -R 755 /usr/local/amigatools
```

Если Вы всё ещё имеете права root, вернитесь к обычному пользователю.

```
# exit (этот шаг пропускается при сборке на Windows/Cygwin)
```

Теперь нам необходимо установить наши переменные *PREFIX* и *TARGET*, которые будут использованы при сборке утилит и компилятора.

```
# export PREFIX=/usr/local/amigatools
# export TARGET=m68k-amigaos
# cd ~/amigaos/build-binutils
```

Теперь мы можем приступить к сборке бинарных исполняемых утилит. Нижеследующая команда отображается в 2 строки, но на самом деле это одна строка. Символ '\' указывает что мы продолжаем набирать ту же самую команду в следующей строке.

```
# ../binutils-2.14/configure --target=$TARGET --prefix=$PREFIX --disable-nls \
--with-gnu-as --with-gnu-ld —with-headers=$PREFIX/m68k-amigaos/include
```

После завершения конфигурирования, мы можем приступать к сборке Amiga binutils из исходных кодов. Если сборка происходит на Linux, убедитесь что Вы имеете права для установки в целевую директорию (*PREFIX*). Вам может потребоваться подняться до пользователя root командой su (если Вы не имеете достаточных прав). На медленном ПК компиляция может занять 10-30 минут.

```
# make (компиляция на AMD 1.8 ГГц занимает 3 минуты)
# make install (su –c ‘make install’ если сборка происходит в Linux)
```

Теперь мы готовы для сборки компилятора GCC. Добавляем новые Amiga binutils в путь поиска и изменяем рабочую директорию сборки.

```
# export PATH=$PREFIX/bin:$PATH
# cd ~/amigaos/build-gcc
```

Запускаем конфигурацию компилятора GCC. Просто следуем той же самой схеме что и выше.

```
# ../gcc-3.3.3/configure --target=$TARGET --prefix=$PREFIX --enable-languages=c,c++ \
--with-headers=$PREFIX/m68k-amigaos/include --with-gnu-as —with-gnu-ld
```

Теперь нам необходимо отредактировать файл *~/amigaos/gcc-3.3.3/gcc/config/m68k/amigaos.h* Это просто замечательно если Вы знаете редактор Vi в Linux, если же нет — воспользуйтесь WordPad или Блокнот. Соответствующий путь к файлу в Windows будет: *C:\Cygwin\home\<имя пользователя>\AmigaOS\GCC-3.3.3\GCC\Config\m68k\amigaos.h*

```
# vi ~/amigaos/gcc-3.3.3/gcc/config/m68k/amigaos.h
```

В файле *amigaos.h* необходимо найти раздел *STARTFILE_SPEC* (можно использовать функцию поиска Vi: нажать Esc /STARTFILE_SPEC и нажать Enter, напоминаю выход без сохранения через Esc q! и Enter).

```
#define STARTFILE_SPEC
"%{!noixemul:"
"%{fbaserel:%{!resident:bcrt0.o%s}}"
"%{resident:rcrt0.o%s}"
"%{fbaserel32:%{!resident32:lcrt0.o%s}}"
"%{resident32:scrt0.o%s}"
"%{!resident:%{!fbaserel:%{!resident32:%{!fbaserel32:"
"%{pg:gcrt0.o%s}%{!pg:%{p:mcrt0.o%s}%{!p:/crt0.o%s}}}}}}}"
"%{noixemul:"
"%{resident:libnix/nrcrt0.o%s} "
"%{!resident:%{fbaserel:libnix/nbcrt0.o%s}%{!fbaserel:libnix/ncrt0.o%s}}}"
```

Нам необходимо добавить наш путь PREFIX/lib в строку содержащую crt0.o. Находим и изменяем эту строку:

```
"%{pg:gcrt0.o%s}%{!pg:%{p:mcrt0.o%s}%{!p:/crt0.o%s}}}}}}}"
```

на следующее:

```
"%{pg:gcrt0.o%s}%{!pg:%{p:mcrt0.o%s}%{!p:/usr/local/amigatools/lib/crt0.o%s}}}}}}}"
```

Теперь мы скопируем файл crt0.o в нашу директорию.

```
# cp ~/amigaos/build-binutils/m68k-amigaos/lib/crt0.o /usr/local/amigatools/lib
```

Теперь мы займёмся компиляцией Amiga GCC и эта часть наиболее сложна т.к. сборка командой make будет обрываться ошибками. Дело в том что в процессе сборки будут искаться файлы которых ещё не существует. Как только мы получим сообщение об ошибке по причине отсутствия файла, мы возьмём собранный файл, скопируем его по указанному пути и запустим сборку сначала. Даже если бы мы положили необходимые файлы заранее, перед первым проходом компиляции, мы всё равно получили бы в результате ошибку по причине перезаписи и более новых версий файлов. Поэтому нам придётся действовать подобным образом.

И несколько минут спустя (5 минут на моих 1,8 ГГц) Вы получите сообщение подобное нижеследующему.

```
/home/tsilvey/amigaos/build-gcc/gcc/include/stddef.h:57:26: machine/ansi.h: No such file or
directory
make[2]: *** [libgcc/./_muldi3.o] Error 1
make[2]: Leaving directory `/home/user1/amigaos/build-gcc/gcc'
make[1]: *** [libgcc.a] Error 2
make[1]: Leaving directory `/home/user1/amigaos/build-gcc/gcc'
make: *** [all-gcc] Error 2
```

Это значит что необходимо скопировать требуемый файл *ansi.h* и запустить сборку заново. На наше счастье она продолжится с того момента когда была прервана, поэтому нам не придётся ждать второй раз. Если же необходимый файл уже существует, Вам будет предложено его перезаписать новой версией. Отвечайте N (НЕТ)!! на все запросы о перезаписи. Там будет что-то около 8 файлов которые потребуется таким образом пропустить.

```
# cp -ir ~/amigaos/build-binutils/m68k-amigaos/include ~/amigaos/build-gcc/gcc/
# cp ~/amigaos/build-binutils/m68k-amigaos/lib/libamiga.a ~/amigaos/build-gcc/m68k-amigaos/lib
# make
```

Компиляция всё равно закончится сообщением об ошибке, но поскольку компилятор и утилиты всё равно будут собраны нам это ничем не повредит (это кажется странным, но я не гуру чтобы ломать голову над подобными материями). Вот такая ошибка:

```
configure: error: installation or configuration problem: C
compiler cannot create executables.
make: *** [configure-target-libiberty] Error 1
```

Поэтому мы завершаем нашу долгую сборку.

```
# make install (su –c ‘make install’ если вы работаете в Linux)
```

Вы можете увидеть несколько новых компиляторов находящихся здесь:

```
# ls -l /usr/local/amigatools/m68k-amigaos/bin
```

Запустим gcc, чтобы удостовериться что он работает и имеет корректную версию.

```
# /usr/local/amigatools/m68k-amigaos/bin/gcc –v
```

Среди разной выдаваемой информации Вы должны увидеть такое:

```
../gcc-3.3.3/configure —target=m68k-amigaos
```

Теперь нам необходимо скопировать новые собранные библиотеки в такое место где компилятор сможет их найти. Если мы выполняли сборку под обычным пользователем, нам придётся исправить права на файлы чтобы обычный пользователь смог получить к ним доступ.

```
# su - root (этот шаг пропускается при сборке на Windows/Cygwin)
# chmod -R 755 /usr/local/amigatools (этот шаг пропускается при сборке на Windows/Cygwin)
# exit (этот шаг пропускается при сборке на Windows/Cygwin)

# cp ~/amigaos/build-binutils/m68k-amigaos/lib/libc.a  /usr/local/amigatools/lib/gcc-lib/m68k-amigaos/3.3.3
# cp ~/amigaos/build-binutils/m68k-amigaos/lib/libamiga.a  /usr/local/amigatools/lib/gcc-lib/m68k-amigaos/3.3.3
```

Если мы собрали Amiga GCC несмотря на все возникавшие ошибки, будем считать что теперь мы готовы его использовать. Новый компилятор находится по пути */usr/local/amigatools/bin/m68k-amigaos-gcc.exe* Для удобства я всегда создаю символическую ссылку.

```
# cd /usr/local/amigatools/m68k-amigaos/bin
# ln -s ./gcc ./gcc68
```

Имя gcc68 отделяет кросс-компилятор Amiga GCC от локальной версии Linux GCC (и позволяет избежать мне путаницы в дальнейшем тексте). Теперь мы добавляем путь к нему в свою переменную *PATH*.

```
# export PATH=$PATH:/usr/local/amigatools/m68k-amigaos/bin
```

Чтобы не выполнять эту команду каждый раз когда понадобится gcc68, можно занести её в файл .bashrc из директории Вашего обычного пользователя.

После того как компилятор gcc68 был собран, пути к его библиотекам и заголовочным файлам жёстко заданы. Поэтому Вы не сможете поменять путь */usr/local/amigatools* на какой-нибудь другой. Вы можете увидеть настройки по умолчанию выполнив команду:

```
# gcc68 –v
```

Хотя мы закончили со сборкой исполняемых файлов, есть ещё несколько моментов которые необходимо проверить в нашей среде разработчика. Давайте протестируем наш компилятор на знаменитом *helloworld.c* Я написал её в Vi, но Вы вполне можете использовать для этой цели Блокнот. Для нашего первого теста, Вам потребуется поместить копию заголовочного файла *stdio.h* в ту же директорию, что и программу.

```
# cp ~/amigaos/build-binutils/m68k-amigaos/include/stdio.h /usr/local/amigatools/m68k-amigaos/include/dos
```

Теперь давайте вернёмся в домашнюю директорию обычного пользователя создадим тестовую программу и скомпилируем её.

```
# cd ~/
# vi helloworld.c
```

Чтобы не «изобретать велосипед», вот текст helloworld.c

```
#include <dos/stdio.h>
int main(void)
{
  printf("Hello World n");
  return 0;
}
```

Давайте посмотрим работает ли компиляция. Мы не только скомпилируем эту маленькую программу, но и оптимизируем её. Конечно попытка оптимизации в данном случае не является практичной, но мы просто проверяем функционал компилятора.

```
# gcc68 -O -o helloworld -m68020 ./helloworld.c
```

Если же компиляция прервалась по любой причине, Вы можете использовать нижеследующие команды с подробным выводом ошибок для нахождения пути который был пропущен или файла который был собран с ошибками. Я использовал её для определения некорректно указанного пути.

```
# gcc68 -v ./helloworld.c –m68020 -Wl,--verbose 2> ldlog.txt
# vi ./ldlog.txt
```

Если компиляция прошла без ошибок, это означает что мы почти готовы к запуску нашей программы Hello World на Amiga для окончательного тестирования. Но для этого на Вашей Amiga должна присутствовать библиотека *ixemul.library* версии 48.0. Вам необходимо скачать её с aminet и положить в директорию *LIBS:* на настоящей Amiga или использовать эмулятор UAE который имеет её встроенной. Библиотека ixemul обеспечивает POSIX-совместимость и содержит дополнительные функции необходимые POSIX-совместимому компилятору Amiga GCC который мы использовали. В архиве Aminet (http://aminet.net) можно найти версии библиотеки оптимизированные для разных процессоров. Поскольку мы компилировали программу для процессора MC68020, нам необходима соответствующая библиотека под этот процессор. Вам потребуется переименовать файл *ixemul-020-fpu.library* в *ixemul.library* перед тем как положить его в директорию *LIBS:*. fpu в имени библиотеки обозначает математический сопроцессор который также должен присутствовать на настоящей Amiga. Вот несколько мест откуда Вы можете получить библиотеку:

- http://main.aminet.net/util/libs/ixemul-48.0.lha
- http://ftp.plig.org/pub/aminet/util/libs/ixemul-48.0.lha

Итак, мы копируем ранее скомпилированный *helloworld.exe* в одну из директорий нашего UAE или с использованием сетевого соединения/дискет «забрасываем» на настоящую Amiga. Запустите программу из командной строки или просто дважды щёлкнув по её иконке. Должно появиться небольшое окно со строчкой «Hello World». Если так и произошло — отлично! Теперь мы можем перемещать оставшиеся библиотеки и заголовочные файлы в область видимости нашего кросс-компилятора с тем чтобы начать работу над реальными проектами.

Нам необходимо скопировать все специфичные для Amiga заголовочные файлы в свою среду разработчика. Их можно получить в составе Amiga CD 2.1 Также они известны как NDK и их когда-то их можно было скачать с сайтов amiga.org и amiga.com, но в последствии они были оттуда удалены. Можно поискать в Google, сейчас архив NDK стал доступным на многих сайтах в сети Internet (например, http://opi.kilu.de/AWeb/files/dev/NDK3.9.lzx). Получив копию NDK Вам будет необходимо скопировать заголовочные файлы из директории */include_h* в Вашу рабочую директорию. Вы по прежнему будете нуждаться в файле *stdio.h* находящемся в директории */include/dos* куда мы скопировали его ранее.

```
# cp -R /YOUR_NDK_3.9_FOLDER/Include/include_h/* /usr/local/amigatools/m68k-amigaos/include
# mv /usr/local/amigatools/lib/gcc-lib/m68k-amigaos/3.3.3/include /usr/local/amigatools/lib/gcc-lib/m68k-amigaos/3.3.3/include.not
```

Теперь протестируем новые библиотеки компиляцией программы специфичной для настоящей Amiga. Пример такой программы будет приведён ниже. Для её сборки компилятору потребуется подключать подходящие заголовки *proto* к любой библиотеке специфичной для Amiga. Для этого такие библиотеки должны быть корректно подключены. Образцы программ из книг, журналов и сайтов в сети Internet посвящённых Amiga вероятнее всего не смогут собраться без изменений. Вам необходимо внимательно обращать внимание на те их строки которые содержат директиву Protos, например:

```
#include <proto/dos.h>
#include <proto/exec.h>
#include <proto/intuition.h>
#include <proto/graphics.h>
```

Также необходимо особо отметить обязательную процедуру открытия и закрытия библиотек.

```
IntuitionBase =(struct IntuitionBase *)OpenLibrary("intuition.library",37L);
GfxBase=(struct GfxBase *)OpenLibrary("graphics.library",33L);
CloseLibrary((struct Library *)GfxBase);
CloseLibrary((struct Library *)IntuitionBase);
```

Вам может потребоваться изменить образец кода который Вы получили снаружи в соответствии с этими знаниями или он не будет компилироваться с выдачей ошибок «missing file», «undefined functiton» или «type mismatch».

Программа которая будет дана ниже содержит многие библиотечные вызовы из числа обычно используемых в приложениях на Amiga и является хорошей проверкой компилятора. Если Вы сможете скомпилировать и запустить этот пример, Вы должны быть на пути к разработке более серьёзных проектов на новой системе. Для примера, я могу скачать исходники любого свободного проекта с архива Aminet, изменить несколько строчек (чтобы «подружить» исходники с GCC), скомпилировать и запустить в эмуляторе UAE. Поэтому я считаю что если Вы зашли столь же далеко, то Ваша среда разработчика должна быть надёжной.

Вы можете просто вырезать, вставить и сохранить текст нижеследующей программы (также как мы это делали с Hello World). Например, сохраните файл как *windowtest.c* в новую рабочую директорию, перейдите в неё и запустите компиляцию. Помните, что Ваши пути к gcc68 должны быть корректными для использования без прямого и явного указания пути к рабочей папке. Если Вы всё ещё не добавили этот путь в Ваш .bashrc, то Вам придётся выполнять export всякий раз когда Вы приступите к компиляции, что согласитесь не очень удобно.

```
# export PATH=$PATH:/usr/local/amigatools/bin (если необходимо)
# cd (переход в Вашу рабочую директорию проекта)
# gcc68 -o ./windowtest -m68020 ./windowtest.c –lamiga (запомните –lamiga - только здесь)
```

```
//Amiga Cross-Compiler sample code windowtest.c

#include <dos/stdio.h>
#include <exec/types.h>
#include <exec/exec.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <intuition/screens.h>
#include <graphics/gfxmacros.h>
#include <graphics/gfxbase.h>
#include <proto/dos.h>
#include <proto/exec.h>
#include <proto/intuition.h>
#include <proto/graphics.h>

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct TagItem win_tags[] = {
    {WA_Left,        20},
    {WA_Top,         20},
    {WA_Width,       200},
    {WA_Height,      160},
    {WA_CloseGadget, TRUE},
    {WA_IDCMP,       IDCMP_CLOSEWINDOW},
};

int main(void)
{
    struct Window *win;
    struct RastPort *rastport;
    int x;
    IntuitionBase =(struct IntuitionBase *)OpenLibrary("intuition.library",37L);
    if (IntuitionBase!=NULL)
    {
       GfxBase=(struct GfxBase *)OpenLibrary("graphics.library",33L);
        if (GfxBase!=NULL)
        {
            win=OpenWindowTagList(NULL,win_tags);
            if (win!=NULL)
            {
                rastport=win->RPort;
                for (x=20;x<200;x+=2)
                {
                    Move(rastport,x,150);
                    Draw(rastport,20,20);
                }
                WaitPort(win->UserPort);
                CloseWindow(win);
            }
            CloseLibrary((struct Library *)GfxBase);
        }
        CloseLibrary((struct Library *)IntuitionBase);
    }
    return 0;
}
```

Опять проверяем на ошибки компиляцию.

```
# gcc68 -v ./windowtest.c –m68020 -lamiga -Wl,--verbose 2> ldlog.txt
# vi ./ldlog.txt
```

Если компиляция прошла без ошибок, самое время скопировать полученный исполняемый файл на Amiga или в одну из директорий эмулятора UAE. Если он заработает Вы должны увидеть простое окно с диагональной линией. Вы можете закрыть его нажатием на гаджет закрытия в заголовке окна слева. Вы прошли ещё один сложный этап и я могу Вас поздравить с началом Вашего следующего проекта для Amiga.

Ниже я приведу несколько тем который в той или иной мере важны. Они помогут когда придёт время для написания собственных библиотек и экспериментов со старыми программами использующими директивы работы с Chip-памятью. Кстати, рекомендую хороший редактор для программиста SciTE. Он очень хорошо работает, существует на Linux, на Windows и совершенно бесплатен!

Дополнительно

Сборка собственного файла *libamiga.a*. Сначала Вам потребуется утилита hunk2gcc. Это утилита для Amiga которая должна использоваться на настоящей Amiga или UAE. Её не существует для Linux или Cygwin, а взять её можно в архиве Aminet (http://main.aminet.net/dev/gg/hunk2aout-bin.lha).

Создайте новую директорию RAM:amigalib на Вашей Amiga. Разархивируйте в неё исполняемый файл *hunk2aout*. В ту же самую директорию скопируйте файл *amiga.lib* из NDK 3.9 По желанию туда же можно скопировать также файл *reaction.lib*, но документация утверждает что это сработает только с компилятором SAS/C для Amiga. В планах — изучение, какие другие библиотеки кроме *reaction.lib* туда можно было бы добавить (возьмите себе на заметку). Набирайте в консоли AmigaDOS следующую команду:

```
> hunk2aout amiga.lib reaction.lib (reaction.lib можно не набирать)
```

Это создаст объектный файл *a.out* для каждой библиотечной функции в hunk-файле *amiga.lib* (и любых других файлов lib которые Вы положите в RAM: и укажите в командной строке). Вы получите множество файлов с именами начинающимися на obj*.

Если Вы работаете с UAE и Cygwin переместите все полученные файлы из RAM: в следующую директорию */usr/local/crosstools/newlibamiga*. Если же нет, Вам придётся переносить их при помощи дискетки или передавать по сети и в случае Linux создавать папку вручную.

```
# mkdir /usr/local/crosstools/newlibamiga
# cd /usr/local/crosstools/newlibamiga
```

Теперь Вы можете конвертировать все полученные объектные файлы в стиле *a.out* в библиотку *libamiga.a*.

```
# ../m68k-amigaos/bin/m68k-amigaos-ar.exe qc ./libamiga.a obj.*
# ../m68k-amigaos/bin/m68k-amigos-ranlib.exe ./libamiga.a
```

Теперь необходимо найти и удалить все старые файлы *libamiga.a* (или переименуйте их, если хотите). Эта команда может найти и удалить их все:

```
find /usr/local/crosstools -name libamiga.a –exec rm '{}' \;
```

(ниже вывод директорий для моей системы)
/usr/local/crosstools/lib/libamiga.a
/usr/local/crosstools/lib/libb/libamiga.a
/usr/local/crosstools/m68k-amigaos/sys-include/lib/libamiga.a
/usr/local/crosstools/m68k-amigaos/sys-include/lib/libb/libamiga.a
/usr/local/crosstools/m68k-amigaos/lib/libamiga.a

Скопируйте новые файлы *libamiga.a* в среду GCC.

```
# cp /usr/local/crosstools/newlibamiga/libamiga.a /usr/local/crosstools/m68k-amigaos/lib/
# ../../bin/m68k-amigaos-ranlib.exe ./libamiga.a
# cd /usr/local/crosstools/bin
```

Немного о выделении Chip-памяти в GCC

Если Вы собираетесь работать с существующим кодом, или Вы привыкли работать с компиляторами для Amiga которые резервируют Chip-память метками вида *__chip*, то Вам придётся несколько изменить Ваш стиль программирования. Ниже Вы увидите пример который я написал во время работы над этим руководством.

Исходный код выглядел так:

```
__chip UWORD Data[] = {
  0x0, 0x0, 0x0, 0x0, 0x3, 0x8000, 0x7, 0xc000, 0xf,
  0xe000, 0x1f, 0xf000, 0x7f, 0xfc00, 0x1ff, 0xff00, 0x7ff, 0xffc0,
  0x3fff, 0xfff8, 0x7ff, 0xffc0, 0x1ff, 0xff00, 0x7f, 0xfc00, 0x1f,
  0xf000, 0xf, 0xe000, 0x7, 0xc000, 0x3, 0x8000, 0x0, 0x0,
  0x0,  0x0, };
```

Компилятор на настоящей Amiga знает где и как в Chip-памяти Amiga расположить данные когда читает метки вида *__chip*. Компилятор GCC в оригинале не понимает что такое *__chip* и зачем ему это нужно. Компилятор Amiga GCC мог бы выделить Chip-память и разобраться с этими метками, если бы программа для него была написана верно. Ниже будет приведён пример, как правильно запросить Chip-память. Функция не включает в себя необходимую проверку ошибок, но Вы сможете добавить её позже.

Первым делом необходимо изменить структуру данных и добавить новый указатель.

```
UWORD *Data;
UWORD chipmem_Data[] = {
  0x0, 0x0, 0x0, 0x0, 0x3, 0x8000, 0x7, 0xc000, 0xf,
  0xe000, 0x1f, 0xf000, 0x7f, 0xfc00, 0x1ff, 0xff00, 0x7ff,0xffc0,
  0x3fff, 0xfff8, 0x7ff, 0xffc0, 0x1ff, 0xff00, 0x7f, 0xfc00, 0x1f,
  0xf000, 0xf, 0xe000, 0x7, 0xc000, 0x3, 0x8000, 0x0, 0x0,
  0x0, 0x0, };
```

После этого можно создавать новый proto и функцию.

```
int LoadData(void);
int LoadData(void){
  int j;
  Data = (UWORD *)AllocMem(sizeof(chipmem_Data), MEMF_CHIP);
  for (j=0; j<37;j++)
  Data[j]=chipmem_Data[j];
  return 0;
}
```

И только после этого можно поименовать нашу новую функцию для загрузки наших данных в Chip-память.

```
int main(int argc, char *argv[]) {
  ****
  (Загрузка после того как все библиотеки будут открыты)
   LoadData();
   ****
}
```

Благодарности автора

- Мачек Плева (http://www.mil-sim.net), Par Taz (http://www.guru-meditation.net);
- Николас Мендоса за помощь в решении проблем с библиотеками и Chip-RAM;
- Ник Гэмон (http://www.gammon.com.au);
- Курт Вэлл и Вильям фон Хаген, авторы «Полного руководства по GCC» (http://www.apress.com/DefinitiveGCC);
- За знания полученные при чтении книг и руководств в сети Internet.

Перевод