Serie: DOSBox unter Windows kompilieren 10/10

Diese Artikelserie zeigt auf, wie man unter Windows eine Entwicklungsumgebung mit Open Source basierten Programmen erstelllt und darin Programme und deren Abhängigkeiten kompiliert. Die daraus entstehenden Dateien können unter Windows wie normale Programme verwendet werden.

Im zehnten und letzten Teil wird das selbst kompilierte DOSBox endlich ausprobiert und zum laufen gebracht.

Das kompilierte DOSBox befindet sich im Verzeichnis (D:\Entwicklung\MinGW-MSYS\MSYS\home\matse\dosbox-0.72\src\) und heisst dosbox.exe. Am einfachsten kopiert man sich die Datei in ein leeres Verzeichnis.

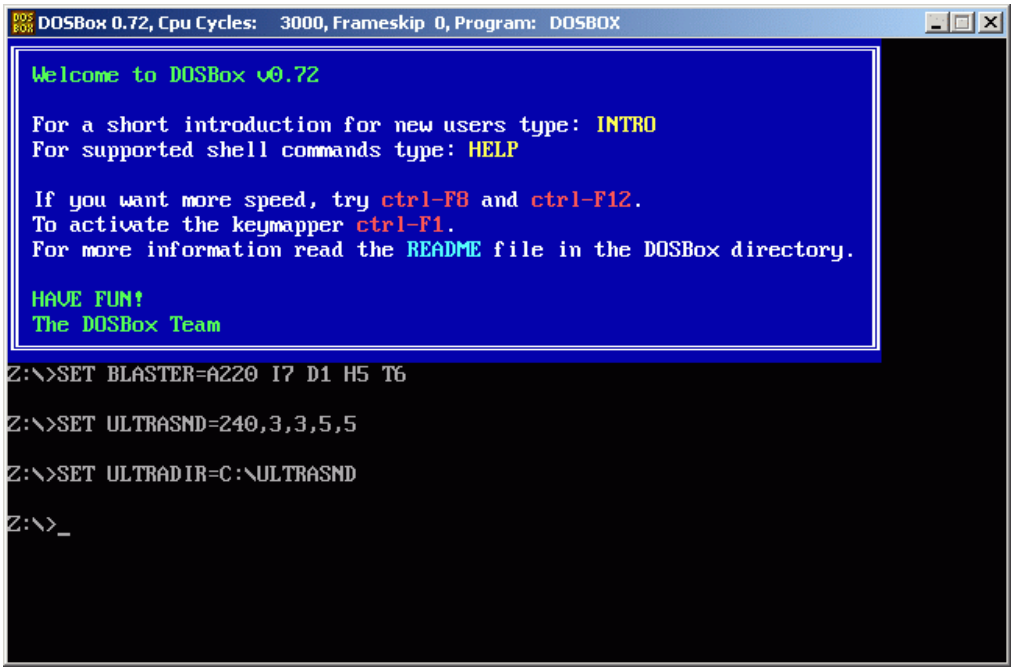
Versucht man die Datei zu starten, erhält man zunächst einmal eine Fehlermeldung:

DOSBox Fehlermeldung

Der Grund für die Meldung ist die nicht auffindbare SDL.dll Datei. Diese – und alle weiteren Abhängigkeiten – befindet sich im Verzeichnis (D:\Entwicklung\MinGW-MSYS\MinGW\bin\). Kopiert man aus diesem Verzeichnis die **SDL.dll** Bibliothek in das selbe Verzeichnis wie sich die dosbox.exe befindet, wird die Meldung bezüglich SDL.dll nicht mehr erscheinen. Nun kann man einfach für jede Fehlermeldung welche erscheint die entsprechende dll in das Verzeichnis mit der dosbox.exe kopieren.

Sind alle Abhängigkeiten aufgelöst startet DOSBox mit dem

allseits bekannten DOS Bildschirm.



Dosbox funktioniert

In der DOSBox kann man den Befehl config -writeconf default.cfg eingeben um eine Standard Konfigurationsdatei zu erzeugen.

Um nun die Umlaute nutzen zu können, muss die eben erstellte default.cfg Datei editiert werden. Hierzu beendet man zuerst DOSBox durch Eingabe von exit.

Als nächstes erstellt man im Verzeichnis, in welchem sich die dosbox.exe befindet ein Verzeichnis mit dem Namen **disk**. Dieses Verzeichnis dient später als Laufwerk C: für die DOSBox.

Nun editiert man die default.cfg Datei und fügt folgende Befehle hinzu, welche das disk Verzeichnis als Laufwerk C in DOSBox einbinden, das Tastaturlayout auf Deutsch stellt und auf das Laufwerk C: wechselt.

```
[...]
[autoexec]
# Lines in this section will be run at startup.
keyb gr 850
```

```
mount C "./disk"
C:
```

Hat man die Änderungen abgespeichert, startet man DOSBox mit folgendem Befehl:

```
dosbox -conf defalt.cfg
```

Nun kann man z.B. ein Verzeichnis Namens Hülle erstellen mit dem Befehl (**mkdir Hülle**). Wenn alles gklappt hat, sieht man mit dem dir Befehl das Verzeichnis und hat somit eine DOSBox Version kompiliert, welche Umlaute unterstützt.

Serie: DOSBox unter Windows kompilieren 9/10

Diese Artikelserie zeigt auf, wie man unter Windows eine Entwicklungsumgebung mit Open Source basierten Programmen erstelllt und darin Programme und deren Abhängigkeiten kompiliert. Die daraus entstehenden Dateien können unter Windows wie normale Programme verwendet werden.

Nachdem in den vorangegangenen Teilen die MinGW/MSYS Entwicklungsumgebung erstellt und alle Abhängigkeiten kompiliert wurden, kann in diesem Teil endlich DOSBox kompiliert werden.

Das bereits in Teil 5 dieser Artikelserie heruntergeladene DOSBox Archiv wird nochmals benötigt. Hat man es nicht mehr, kann man dosbox-0.72.tar.gz in das MSYS Heimverzeichnis herunterladen.

Mein Umlaute Patch kann ebenfalls in das MSYS Heimverzeichnis

heruntergeladen werden.

Nun kann Dosbox entpackt, mein Umlautepatch eingespielt und das Programm endlich kompiliert werden.

```
tar xvzf dosbox-0.72.tar.gz
bunzip2 dosbox072matse.diff.bz2
cd dosbox-0.72
patch -p1<../dosbox072matse.diff
LIBS="-lvorbisfile" ./configure --prefix=/mingw && make
strip src/dosbox.exe
```

Im letzten Teil dieser Artikelserie wird gezeigt, wie man das selbst kompilierte dosbox verwenden kann, wo seine Abhängigkeiten zu finden sind und wie man eine Basiskonfiguration erstellt.

Serie: DOSBox unter Windows kompilieren 8/10

Diese Artikelserie zeigt auf, wie man unter Windows eine Entwicklungsumgebung mit Open Source basierten Programmen erstelllt und darin Programme und deren Abhängigkeiten kompiliert. Die daraus entstehenden Dateien können unter Windows wie normale Programme verwendet werden.

In diesem Artikel wird die auf SDL basierende Soundbibliothek SDL_sound kompiliert. Diese wird benötigt, um z.B. CD Images mit OGG Dateien als Audio Tracks mounten zu können. Die SDL_sound Bibliothek unterstützt von Haus aus das MP3 Format. Um Ogg Vorbis Dateien abspielen zu können, müssen erst einige zusätzliche Bibliotheken heruntergeladen und kompiliert werden.

Zunächst müssen die Dateien der folgenden Tabelle heruntergeladen und in das Heimvereichnis von MSYS kopiert werden:

Paket	Beschreibung	Link
libogg	Container für z.B. Vorbis Musikdateien.	libogg-1.1.3
libvorbis	Vorbis Musik-Komprimieralgorithmus	libvorbis-1.2.2rc1
SDL_sound	Auf SDL basierende Musikbibliothek.	SDL_sound-1.0.3

Die OGG Bibliothek kann ohne grössere Umstände entpackt und kompiliert werden.

```
tar xvzf libogg-1.1.3.tar.gz
mkdir libogg-build
cd libogg-build
../libogg-1.1.3/configure --prefix=/mingw && make && make install
cd ..
```

Die hier verwendete Version von libvorbis ist eigentlich ein Release Candidate. Also eine Version die zwischen Betaversion und offiziell veröffentlich ist. Der Grund für diese Wahl ist die mangelnde Unterstützung für MinGW in der offiziell verfügbaren Version.

```
tar xvjf libvorbis-1.2.2rc1.tar.bz2
mkdir libvorbis-build
cd libvorbis-build
../libvorbis-1.2.2/configure --prefix=/mingw && make && make install
cd ..
```

Versucht man das SDL_sound Paket unter MinGW zu kompilieren, funktioniert dies nicht, da gewisse Headerdateien mehrfach

referenziert werden. Um dieses Problem zu umgehen müssen mit SED Anpassungen am Sourcecode vorgenommen werden.

```
cd
tar xvzf SDL_sound-1.0.3.tar.gz
mkdir SDL_sound-build
cd SDL_sound-build
for a in mpglib/mpg123_sdlsound.h timidity/tables.h;do
sed -e "s/#include.*
//g" ../SDL_sound-1.0.3/decoders/${a}>/tmp/gaga
cat /tmp/gaga > ../SDL_sound-1.0.3/decoders/${a}
rm /tmp/gaga
done
../SDL_sound-1.0.3/configure --prefix=/mingw && make && make install
cd ..
```

Da nun alle Abhängigkeiten erzeugt wurden, wird im neuten Teil dieser Artikelserie endlich die DOSBox kompiliert.

Serie: DOSBox unter Windows kompilieren 7/10

Diese Artikelserie zeigt auf, wie man unter Windows eine Entwicklungsumgebung mit Open Source basierten Programmen erstelllt und darin Programme und deren Abhängigkeiten kompiliert. Die daraus entstehenden Dateien können unter Windows wie normale Programme verwendet werden.

Im siebten Teil wird die auf SDL aufbauende Netzwerkbibliothek SDL_net erzeugt. Durch diese Bibliothek hat man in der DOSBox die Möglichkeit Spiele wie etwa Quake über das Netzwerk zu

spielen. Prinzipiell kann man diese Abhängigkeit auch einfach auslassen. DOSBox wird trotzdem kompiliert, bietet aber keine Netzwerkunterstützung an. Da wir jedoch eine vollständie DOSBox kompilieren wollen, wird nachfolgend erläutert wie man SDL_net kompiliert.

Von der Webseite, von welcher man bereits SDL heruntergeladen hat, lädt man sich das Paket SDL_net-1.2.7.tar.gz herunter und kopiert es in das Heimverzeichnis der MSYS Umgebung.

Das Paket wird folgendermassen kompiliert.

```
cd
tar xvzf SDL_net-1.2.7.tar.gz
mkdir SDL_net-build
cd SDL_net-build
../SDL_net-1.2.7/configure --prefix=/mingw --disable-gui &&
make && make install
cd ..
```

Im achten Teil dieser Artikelserie wird die Soundunterstützung für DOSBox kompiliert.

Serie: DOSBox unter Windows kompilieren 6/10

Diese Artikelserie zeigt auf, wie man unter Windows eine Entwicklungsumgebung mit Open Source basierten Programmen erstelllt und darin Programme und deren Abhängigkeiten kompiliert. Die daraus entstehenden Dateien können unter Windows wie normale Programme verwendet werden.

Im fünften Teil wurde die Simple DirectMedia Library

kompiliert, in diesem Teil wird die PNG Bibliothek kompiliert. Diese Bibliothek ermöglicht einem Programm die Verwendung und Erstellung von PNG Grafiken. Die PNG Bibliothek wird von DOSBox benötigt um die Screenshots – welche während einer DOSBox Session erstellt werden können – abspeichern zu können.

Zuerst müssen die Pakete der folgenden Tabelle in das Heimverzeichnis der MSYS Umgebung heruntergeladen werden.

Paket	Beschreibung	Link
Z Bibliothek	Bibliothek mit dem Komprimieralgorithmus Z.	zlib-1.2.3
libpng	PNG Bibliothek	libpng-1.2.35

Um zlib unter Windows als dll kompilieren zu können müssen einige Änderungen vorgenommen werden. Ausserdem wird die Installation (make install) Fehlermeldungen ausgeben, da einige if Abfragen ohne zugehöriges fi vorhanden sind. Die Fehlermeldungen haben jedoch keine Relevanz, da die wichtigen Dateien installiert werden.

```
cd
tar xvzf zlib-1.2.3.tar.gz
cd zlib-1.2.3
sed -e 's/INSTALL = $(CP)/INSTALL = install/g'
win32/Makefile.gcc > Makefile
make && make test testdll && INCLUDE_PATH=/mingw/include
LIBRARY_PATH=/mingw/lib make install
install zlib1.dll /mingw/bin
cd ..
```

libpng lässt sich im Gegensatz zur zlib bereits schon einfach kompilieren.

```
cd
tar xvzf libpng-1.2.35.tar.gz
mkdir libpng-build
```

```
cd libpng-build
cp ../libpng-1.2.35/scripts/makefile.mingw Makefile
SRCDIR=/home/matse/libpng-1.2.35 prefix=/mingw make
SRCDIR=/home/matse/libpng-1.2.35 prefix=/mingw make test
SRCDIR=/home/matse/libpng-1.2.35 prefix=/mingw make install
cd ..
```

Im siebten Teil dieser Artikelserie wird die SDL_net Bibliothek kompiliert.

Serie: DOSBox unter Windows kompilieren 5/10

Diese Artikelserie zeigt auf, wie man unter Windows eine Entwicklungsumgebung mit Open Source basierten Programmen erstelllt und darin Programme und deren Abhängigkeiten kompiliert. Die daraus entstehenden Dateien können unter Windows wie normale Programme verwendet werden.

In diesem Teil wird die erste Abhängigkeit von DOSBox kompiliert. DOSBox bietet z.B. die Möglichkeit CD's mit Audio Tracks so einzubinden, dass statt der originalen, unkomprimierten WAV Dateien komprimierte MP3 oder OGG Dateien verwendet werden können. Um nun die Unterstützung hierzu nicht komplett neu programmieren zu müssen, haben sich die Programmierer an bestehenden Projekten bedient. Hierdurch ist die Arbeit der Programmierer zwar weniger geworden, derjenige welcher den Quelltext kompiliert, muss aber auch die benötigten Abhängigkeiten zuerst herunterladen und kompilieren.

Eine solche Abhängigkeit ist z.B. Simple DirectMedia Layer

(SDL). Simple DirectMedia Layer ist eine plattformübergreifende Multimedia Bibliothek welche low level Zugriff auf Audio, Keyboard, Maus, Joystick, 3D hardware via OpenGL, und 2D Video Framebuffer zur Verfügung stellt.

Zuerst wird das SDL-1.2.13.tar.gz Paket in das Homeverzeichnis der MSYS Umgebung heruntergeladen.

Um SDL unter Windows zusammen mit DOSBox nutzen zu könenn, müssen einige Quelltextdateien angepasst werden. Da sich hierzu im DOSBox Quellcode bereits ein Patch File findet, wir zusätzlich das Archiv dosbox-0.72.tar.gz heruntergeladen.

Beide Archive werden nun entpackt und SDL wird gepatcht.

```
tar xvzf dosbox-*.gz
tar xvzf SDL-1.*.tar.gz
sed -e "s/VERSION 1,2,12,0/VERSION 1,2,13,0/g" \
-e "s/@@ -13,7 +13,7 @@/@@ -12,7 +12,7 @@/g"
dosbox-0.72/src/platform/sdl-win32.diff >sdl-win32.diff
rm -fr dosbox-0.72
cd SDL-1.2.13
patch -p1<../sdl-win32.diff
cd ..
```

Achtung, patch wird eine Fehlermeldung bringen.

Aus irgend einem Grund kann die Datei src/main/win32/version.rc nicht gepatcht werden. Da in dieser Datei lediglich ein Zeichen verändert werden muss, öffnet man diese Datei am besten mit einem Texteditor (nicht Word!!) der UNIX Zeilenumbrüche versteht (z.B. Notepad++). In der Datei **src/main/win32/version.rc** sucht man nach **FILEFLAGS 0x0L** und ersetzt dies durch **FILEFLAGS 0x4L**

Als nächstes werden die DirectX Header Dateien heruntergeladen und entpackt, damit SDL diese nutzen kann.

Zum Schluss wird alles entpackt, SDL kompiliert und die

erzeugten Dateien installiert.

```
cd
tar xvzf directx-devel.tar.gz -C /mingw include
mkdir SDL-build
cd SDL-build
../SDL-1.2.13/configure --prefix=/mingw && make && make install
cd ..
```

Nach dem kompilieren findet man in /mingw/bin die Datei SDL.dll vor. Auf diese wird später DOSBox referenzieren.

Im sechsten Teil dieser Artikelserie wird die libpng kompiliert, welche unter anderem für die Erzeugung von Screenshots in DOSBox benötigt wird.

Serie: DOSBox unter Windows kompilieren 4/10

Diese Artikelserie zeigt auf, wie man unter Windows eine Entwicklungsumgebung mit Open Source basierten Programmen erstelllt und darin Programme und deren Abhängigkeiten kompiliert. Die daraus entstehenden Dateien können unter Windows wie normale Programme verwendet werden.

In den vorangegangenen Artikeln wurde die Entwicklungsumgebung erstellt. In diesem Teil wird zum ersten mal etwas in dieser Umgebung kompiliert. Um genau zu sein, handelt es sich dabei um die Autotools. Prinzipiell sind diese Programme in der MinGW/MSYS Umgebung bereits vorhanden. Jedoch handelt es sich um ältere Versionen, welche – gerade bei neueren Paketen – immer mal wieder zu Problemen führen können. Aus diesem Grund

werden die aktuellsten Autotools aus dem Quelltext erstellt.

Die Autotools sind mehrere Programme, welche vor dem eigentlichen kompilieren diverse Informationen des Systems sammeln. Dadurch ist schon vor dem kompilieren klar, welche Header Dateien oder welche Libraries verwendet werden sollen. Auch können die Autotools ermitteln ob benötigte Abhängigkeiten vorhanden sind.

Anstatt wie bisher die herunter zu ladenden Dateien im Basis-Installationsverzeichnis (D:\Entwicklung\MinGW-MSYS) zu platzieren, sollten diese ab sofort im Home-Verzeichnis der MinGW/MSYS Umgebung gespeichert werden. Die hat zwei Gründe:

1. Man befindet sich nach dem ausführen von **msys.bat** direkt in diesem Verzeichnis.
2. Das Basis-Installationsverzeichnis befindet sich prinzipiell ausserhalb der MinGW/MSYS Umgebung.

Unter Windows befindet sich dieses Verzeichnis im Pfad (D:\Entwicklung\MinGW-MSYS\MSYS\home\<ANMELDENAME>\). In dieses Verzeichnis lädt man nun alle Pakete der nachfolgenden Tabelle herunter:

Paket	Beschreibung	Link
autoconf	Autoconf ist ein erweiterbares Paket von M4 Makros welche Shell Scripts erstellen um automatisch Quelltextpakete zu konfigurieren.	autoconf-2.63
automake	Automake erstellt automatisiert Makefile.in Dateien.	automake-1.10.2
libtool	Libtool bietet Unterstützung bei der Verwendung von dynamischen Bibliotheken auf unterschiedlichen Systemplattformen.	libtool-2.2.6a

Die Pakete werden entpackt und pro Paket wird ein Build Verzeichnis erstellt in welchem die Pakete kompiliert werden.

Hierdurch wird im Original Sourcenpfad nichts verändert.

```
for a in autoconf- automake- libtool-;do tar xvzf ${a}*;mkdir ${a}build;done
cd autoconf-build
../autoconf-2.63/configure --prefix=/mingw && make && make install
cd ../automake-build
../automake-1.10.2/configure --prefix=/mingw && csmake && csmake install
cd ../libtool-build
../libtool-2.2.6/configure --prefix=/mingw && make && make install
cd ..
```

Auf der MinGW Webseite wird empfohlen, selbstkompilierte Bibliotheken in /mingw zu installieren. Damit zukünftig erstellte Pakete wissen, dass die Bibliotheken hier zu suchen sind, müssen noch einige Einstellungen an der Entwicklungsumgebung vorgenommen werden.

```
sed -e 's#export HOME LOGNAME MSYSTEM HISTFILE#PKG_CONFIG_PATH="/mingw/lib/pkgconfig"\
CFLAGS="-pipe -O2 -mms-bitfields -march=i686"\
export HOME LOGNAME MSYSTEM HISTFILE PKG_CONFIG_PATH CFLAGS#g' /etc/profile>/tmp/profile
cat /tmp/profile > /etc/profile
rm /tmp/profile
```

Wenn man nun die MSYS Umgebung mittels exit verlässt und danach nochmals die **msys.bat** aufruft, ist alles so eingestellt, dass zukünftige Kompilationen die Bibliotheken ebenfalls in der /mingw Struktur suchen.

Im fünften Teil dieser Artikelserie wird die erste Abhängigkeit von DOSBox kompiliert.

Serie: DOSBox unter Windows kompilieren 3/10

Diese Artikelserie zeigt auf, wie man unter Windows eine Entwicklungsumgebung mit Open Source basierten Programmen erstelllt und darin Programme und deren Abhängigkeiten kompiliert. Die daraus entstehenden Dateien können unter Windows wie normale Programme verwendet werden.

Während im zweiten Teil dieser Artikelserie beschrieben wurde wie man die MinGW Entwicklungsumgebung erstellt, wird in diesem dritten Teil beschrieben, wie man die MSYS Umgebung einrichtet.

Wikipedia schreibt zu MSYS

> *MSYS (Minimal SYStem) ist eine Softwareportierung der Unix-Shell auf die Windows-Plattform. Es dient MinGW-Entwicklern als ein minimales System, welches unter anderem configure-Skripte ausführen kann.*

MSYS installiert zum Beispiel eine bash Shell, Komprimiertools wie bzip2 oder lzma und verschiedene andere Programme, welche für eine Entwicklungsumgebung benötigt werden.

Die Pakete der nachfolgenden Tabelle können alle in das Basis-Installationsverzeichnis (D:\Entwicklung\MinGW-MSYS) heruntergeladen werden.

Paket	Beschreibung	Link
bash	Die Bourne Again Shell für Windows.	bash-3.1
bzip2	Das Komprimierprogramm bzip2.	bzip2-1.0.3

coreutils	Diverse Basisprogramme wie cat, echo, tail etc.	coreuitls-5.97
cpmake	Eine Make Variante für Windows.	cpmake-3.81
csmake	Eine Make Variante für Windows.	csmake-3.81
diffutils	Programme um Dateien zu vergleichen.	diffutils-2.8.7
findutils	Programme um Dateien zu finden.	findutils-4.3.0
gawk	Die Gnu AWK Programme.	gawk-3.1.5
lzma	Das lzma Komprimierungsprogramm.	lzma-4.43
make	Das originale make Programm.	make-3.81
MSYS-dll	Die MSYS Bibliothek.	MSYS-dll-1.0.11
msysCORE	Basiskomponenten von MSYS.	msysCORE-1.0.11
tar	Tape Archiver Programm.	tar-1.0.11-2
texinfo	Das info Paket.	texinfo-4.11
perl	Die PERL Programmiersprache.	perl-5.6.1
crypt	Die GNU Kryptographie Bibliothek	crypt-1.1-1

Danach wird MSYS soweit vorbereitet, dass man später die restlichen Tools installieren kann. Zuerst muss hierzu ein MSYS Unterverzeichnis erstellt werden, dann werden die MSYS Kernkomponenten entpackt und zum Schluss ein Installationsscript gestartet. Hierzu gibt man folgende Befehle in der Eingabeaufforderung von Windows ein:

```
cd /D D:\Entwicklung\MinGW-MSYS\
mkdir MSYS
tar -xvzf msysCORE* -C ./MSYS/
tar -xvzf MSYS-* -C ./MSYS/
cd MSYS\postinstall\
pi
### Nachfolgend die Fragen und Antworten ###
Do you wish to continue with the post install? [yn ] y
Do you have MinGW installed? [yn ] y
Where is your MinGW installation? d:/Entwicklung/MinGW-
```

```
MSYS/MinGW
cd ..\..\
```

Wenn alles funktioniert hat, sollte man nun im MSYS Verzeichnis **msys.bat** starten können und eine Shell erhalten. Ein **gcc -v** (welches von /mingw/bin geholt wird) sollte genau so funktionieren wie ein mount. Hat alles geklappt kann man die Shell mit **exit** wieder beenden.

Jetzt kann man die MSYS Umgebung mit den heruntergeladenen Tools Erweitern, bzw. updaten.

```
cd /D D:\Entwicklung\MinGW-MSYS\
tar -xvzf bash-* -C ./MSYS/
tar -xvzf bzip2-* -C ./MSYS/
tar -xvzf coreutils-* -C ./MSYS/
tar -xvzf cpmake-* -C ./MSYS/
tar -xvzf csmake-* -C ./MSYS/
tar -xvzf diffutils-* -C ./MSYS/
tar -xvzf findutils-* -C ./MSYS/
tar -xvzf gawk-* -C ./MSYS/
tar -xvzf lzma-* -C ./MSYS/
tar -xvzf make-* -C ./MSYS/
tar -xvzf tar-* -C ./MSYS/
tar -xvzf texinfo-* -C ./MSYS/
tar -xvzf crypt-* -C ./MSYS/
tar -xvzf perl-* -C ./MSYS/
```

Leider wurden die Coreutils in der Version 5.97 falsch gepackt, so dass im MSYS Verzeichnis ein coreutils-5.97 Verzeichnis erzeugt wurde. Darin befinden sich die bin, lib und anderen Verzeichnisse welche eigentlich in das MSYS Verzeichnis installiert werden sollten. Diesen Umstand kann man mit den folgenden Befehlen (welche innerhalb der ausgeführten msys.bat eingegeben werden) beheben:

```
cd /D D:\Entwicklung\MinGW-MSYS\MSYS\
```

```
msys.bat
### Alle folgenden Befehle in der msys Shell eingeben!!! ###
cd /
mv coreutils-5.97/bin/* bin/
### da wir mit mv mv.exe nicht verschieben können, muss
### es kopiert und separat gelöscht werden.
cp coreutils-5.97/bin/mv.exe bin/
mv coreutils-5.97/info/* info/
mv coreutils-5.97/lib/* lib/
mv coreutils-5.97/man/man1/* man/man1/
rm -r coreutils-5.97
```

Im vierten Teil dieser Artikelserie werden die Autotools installiert. Da diese direkt aus dem Quelltext erstellt werden, wird zum ersten mal die Entwicklungsumgebung ‚live' eingesetzt.

Serie: DOSBox unter Windows kompilieren 2/10

Diese Artikelserie zeigt auf, wie man unter Windows eine Entwicklungsumgebung mit Open Source basierten Programmen erstelllt und darin Programme und deren Abhängigkeiten kompiliert. Die daraus entstehenden Dateien können unter Windows wie normale Programme verwendet werden.

Im zweiten Teil dieser Artikelserie beschreibe ich die Installation einer Entwicklungsumgebung basierend auf dem MinGW Projekt. Eine Entwicklungsumgebung besteht aus einer Zusammenstellung von Programmen und Werkzeugen, welche zur Erstellung beliebiger Software verwendet werden kann. In solch einer Zusammenstellung können z.B. ein Texteditor zum

bearbeiten der Quelltexte, ein Dateiverwaltungssystem, ein Kompiler und ein Linker vorhanden sein.

Wikipedia schreibt zu MinGW

> ***MinGW*** *oder* ***Mingw32*** *(Minimalist GNU for Windows) ist eine Softwareportierung der GNU-Entwicklerwerkzeuge (GCC, GDB) auf die Windows-Plattform, mit der man Programme für Windows entwickeln kann. MinGW entstand aus dem Cygwin Projekt heraus. Es wird, anders als bei Cygwin, keine Kompatibilitätsschicht in Form einer DLL benötigt. Auch kann hiermit entwickelte Software unter Lizenzen veröffentlicht werden, die nicht mit der GNU GPL kompatibel sind.*

Prinzipiell kann man einfach den Installer von der MinGW Webseite herunterladen, anklicken und abwarten. Es hat sich jedoch bei mir gezeigt, dass diese Installationsart im späteren Verlauf zu Problemen führt. So wird z.B. MinGW wie auch MSYS in ein und das selbe Verzeichnis installiert. Das sollte bei neueren Versionen eigentlich kein Problem sein. Sollte… Meine Erfahrungen haben gezeigt, dass es besser ist alles von Hand zu installieren.

Vorinformationen

- Da eine Entwicklungsumgebung naturgemäss relativ viel Platz belegt, sollte man sich für die Installation eine Disk aussuchen, welche mindestens noch 4-5GB übrig hat.
- Für alle Installationen in dieser Artikelserie wird von folgendem Basis-Installationspfad ausgegangen: D:\Entwicklung\MinGW-MSYS
- Wer sich bereits ein wenig mit der Thematik des kompilierens auseinander gesetzt hat, kann getrost die neusten Versionen aller in dieser Artikelserie beschriebenen Pakete herunterladen. Man muss aber damit rechnen, dass unerwartete Fehler auftreten können. Wer auf Nummer sicher gehen will, sollte genau die in diesem

Artikel angegebenen Versionen herunterladen.

Installation und Konfiguration

1. In dem oben angegebenen Basis-Installationspfad werden als erstes alle Pakete aus der **unten stehenden Tabelle** heruntergeladen.
2. Im Basis-Installationspfad wird vom gzip Archiv **bin/gzip** entpackt, vom libarchiv die Dateien **bin/{bsdtar,libarchive2.dll}** und vom Dependencie Paket die beide Bibliotheken **bin/{bzip2,zlib1}.dll**.
3. **bsdtar.exe** in **tar.exe** umbenennen.
4. Innerhalb des Basis-Installationspfades muss ein **MinGW** Verzeichnis angelegt werden in welchem alle heruntergeladenen Pakete entpackt werden. Hierzu öffnet mein eine Eingabeaufforderung (Start/Ausführen/cmd [Enter]) und gibt darin die folgenden Befehle ein.

   ```
   cd /D D:\Entwicklung\MinGW-MSYS\
   mkdir MinGW
   tar -xvzf binutils-* -C ./MinGW/
   tar -xvzf gcc-core-* -C ./MinGW/
   tar -xvzf gcc-g++-* -C ./MinGW/
   tar -xvzf gcc-ada-* -C ./MinGW/
   tar -xvzf gcc-g77-* -C ./MinGW/
   tar -xvzf gcc-java-* -C ./MinGW/
   tar -xvzf gcc-objc-* -C ./MinGW/
   tar -xvzf mingwrt-*-mingw32-dev* -C ./MinGW/
   tar -xvzf mingwrt-*-mingw32-dll* -C ./MinGW/
   tar -xvzf w32api-* -C ./MinGW/
   ```

Paket	Beschreibung	Link
binutils	Enthält Programme wie den Assembler, den Linker und viele andere.	binutils-2.19.1

gcc-core	Grundlegende Kompiliertools sowie der C-Compiler aus der Gnu Compiler Collection.	gcc-core-3.4.5
gcc-g++	Der C++ Compiler aus der Gnu Compiler Collection.	gcc-g++-3.4.5
gcc-ada	Der ada Compiler aus der Gnu Compiler Collection.	gcc-ada-3.4.5
gcc-g77	Der g77 Compiler aus der Gnu Compiler Collection.	gcc-g77-3.4.5
gcc-java	Der java Compiler aus der Gnu Compiler Collection.	gcc-java-3.4.5
gcc-objc	Der Obective C Compiler aus der Gnu Compiler Collection.	gcc-objc-3.4.5
mingw32-dev	Grundlegende Dateien der MinGW Runtime Umgebung.	mingwrt-3.15.2-dev
mingw32-dll	Basisbibliotheken der MinGW Runtime Umgebung.	mingwrt-3.15.2-dll
mingw32-API	Essenzielle Header Dateien sowie Bibliotheken der MinGW Runtime Umgebung.	w32api-3.13
gzip	gzip ent-/packer um die .gz Dateien entpacken zu können	gzip.zip
tar	tar ent-/packer um die .tar Dateien entpacken zu können	tar.zip
dependencies	Bibliotheksabhängigkeiten zu gzip und tar.	deps.zip

Im dritten Teil dieser Artikelserie wird die MSYS Umgebung installiert, welche erst den sinnvollen Umgang mit den Entwicklungswerkzeugen ermöglicht.

Serie: DOSBox unter Windows kompilieren 1/10

1987 habe ich als 10 Jähriger Knirps mit meinem C64 herumgespielt. Eigentlich war zu der Zeit der 386er gerade aktuell, doch von solch einem Rechner konnte ich nicht einmal träumen Jedenfalls erschien im Jahre 1987 das Spiel **Police Quest: In Pursuit of the Death Angel**. Ich weiss noch, wie ich damals bei uns im Ort an die Herbstschau ging und da zum ersten mal dieses Spiel erblickt hatte. Das Spiel hatte mich sofort gefesselt. Man durfte selber steuern wohin man den Protagonisten schicken wollte und wenn man z.B. eine Tür aufmachen wollte, stellte man die Figur vor selbige und gab **open door** ein. Ich war sofort Feuer und Flamme und gab mein ganzes Taschengeld dafür aus, jeden Tag einen Messeeintritt zu erwerben um mich bei diesem Stand aufhalten zu können. Irgendwann liessen mich die Leute da sogar selber spielen! Und hätte es nicht so etwas wie Hunger, Durst, Schlaf oder Hausaufgaben gegeben, ich würde wohl heute noch dort sitzen

Jedenfalls sind es solche Erinnerungen, welche mich persönlich immer wieder die alten Spiele hervorkramen lassen. Doch leider wird es immer schwieriger solche Spiele unter einem heutigen System überhaupt zum laufen zu bekommen. Die Spiele welche sehr Hardwarenah programmiert wurden, rasen bei den heutigen Gigahertzen nur so über den Bildschirm. Die ISA Soundkarten gibt es schon lange nicht mehr und von onboard PCI Karten wussten die damaligen Spiele noch nichts. Und von heutigen Monitorauflösungen jenseits von 640×480 will ich gar nicht erst anfangen zu sprechen.

Um also all diese Hürden überwinden zu können, bietet sich die Verwendung eines Emulators an. Und genau für diesen Zweck gibt es einen Emulator mit Namen DOSBox.

Zitat Wikipedia:

> *DOSBox ist ein freier x86-Emulator, der das Betriebssystem DOS und die in dessen Ära gebräuchliche Hardware nachbildet. Ziel ist das Ausführen älterer, DOS-basierter Software, die mit modernen Computersystemen nur eingeschränkt oder gar nicht kompatibel ist.*

Da der Emulator auf den unterschiedlichsten Systemen wie etwa Linux, BSD, Mac OS X und vielen anderen läuft, ist er natürlich sehr beliebt und verbreitet. Mit Hilfe dieses Emulators habe ich schon so manches Kleinod aus früheren Kindertagen wieder in altem Glanz erstrahlen lassen.

Titelbild von Bazooka Sue

Doch wie das nunmal im Leben so ist, irgendwann gelangt man an einen Punkt wo selbst die tollsten Tricks das alte Spiel nicht mehr zum laufen bringen. Bei mir war dieser Punkt beim Spiel Bazooka Sue erreicht.

Das Problem bei diesem Spiel ist die Verwendung von Umlauten in den Dateinamen. So gibt es z.B. eine Datei GLÜCK.WAV oder LÖWE.WAV welche in der entsprechenden Situation abgespielt wird.

Da DOSBox ein internationales Projekt ist und die

Programmierer englischsprachig sind, gibt es für Umlaute zuerst einmal keine Unterstützung. Das merkt man auch daran, dass das Spiel an den jeweiligen Stellen mit einer Fehlermeldung abstürzt.

Doch Open Source wäre nicht Open Source, wenn man nicht einfach selber Hand am Code anlegen könnte. Also habe ich mir den Quellcode heruntergeladen und die besagte Stelle relativ schnell identifiziert und gepatch.

Viel schwieriger war dann jedoch die Frage zu beantworten, wie man denn dieses Paket – inklusive all seiner Abhängigkeiten – unter Windows XP zum kompilieren bringt. Nach einer Woche intensiven ausprobierens, Foren scannens, Bugreports lesen und Anwendung von Vodoo Praktiken habe ich es tatsächlich geschafft eine DOSBox Version auf meinen Rechner zu zaubern, welche Umlaute in Dateinamen versteht.

Da dieses Unterfangen doch einige Stolperfallen beherbergen kann, habe ich mich entschieden eine Artikelserie zu veröffentlichen, welche aufzeigt, wie man das kompilieren einer derart komplexen Software mit Hilfe von Open Source Tools unter Windows bewerkstelligen kann. Obwohl es in diesem Artikel explizit um DOSBox geht, kann diese Artikelserie für jedermann interessant sein, der selber das eine oder andere Projekt unter Windows kompilieren möchte.

Im zweiten Teil der Serie werde ich mich der Erstellung und Installaiton der MinGW/MSYS Entwicklungsumgebung widmen.