Hosts/W32 - QEMU

wiki.qemu.org/Hosts/W32

QEMU on W32 and W64 hosts

This documentation is work in progress - more information will be added as needed.

While QEMU's main host platform is Linux, it is sometimes also useful to build or run it on members of the W32 / W64 family of operating systems (MS Windows 8, Windows 10, Windows 11, ...) or on ReactOS (a W32 clone). Support for W64 was added in QEMU 1.1. See https://www.qemu.org/docs/master/about/build-platforms.html#windows for information about which versions of Windows are currently supported.

Please note that less developers work on QEMU for W32 / W64 hosts, so it might be less stable.

Also note that the building process of QEMU involves some Python scripts that call os.symlink() which needs special attention for the build process to successfully complete. On newer versions of Windows 10, unprivileged accounts can create symlinks if Developer Mode is enabled. When Developer Mode is not available/enabled, the SeCreateSymbolicLinkPrivilege privilege is required, or the process must be run as an administrator.

Building QEMU for W32

QEMU for W32 needs a fairly complete Mingw-w64 based development environment with tools (make, compiler, linker, ...) and some additional libraries. Building with the older MinGW does not work!

Cross builds

Compilation of QEMU for W32 on non-W32 hosts (e.g. Linux hosts) is called cross compilation. Some Linux distributions (Debian, Ubuntu, Fedora and maybe others) already include packages needed for cross compilation, so the installation of these packages is the first step.

Debian squeeze based cross builds

Note: Building on Debian squeeze is no longer supported, so the following commands won't work with latest QEMU.

```
# Debian squeeze for W32:
apt-get install gcc-mingw32 mingw32-binutils mingw32-runtime

# Debian (squeeze?) for W64:
apt-get install gcc-mingw32 mingw32-binutils mingw-w64
```

SDL support is not included in standard MinGW, but packages for MinGW are available on the SDL homepage. POSIX thread support is not included in Debian or Ubuntu. Latest QEMU will need it, so you have to get it from MinGW (see links below).

Cross compilers usually are installed in /usr/bin with a prefix. For Debian, the cross gcc is called i586-mingw32msvc-gcc. This cross prefix must be passed to QEMU's configure.

```
# Debian cross configuration for W32:
configure --cross-prefix=i586-mingw32msvc- [--extra-cflags=-mthreads]
```

Compiler option is needed for gcc versions which don't support TLS (thread local storage) without it (version 4.4 which is Debian's default needs it!).

Debian does not include a cross pkg-config, but it is required for cross builds. The following script can be saved as /usr/bin/i586-mingw32msvc-pkg-config and optionally be linked to /usr/bin/amd64-mingw32msvc-pkg-config.

```
#!/bin/sh
basename=`basename $0`
prefix=/usr/`echo $basename|sed s/-pkg-config//`
PKG_CONFIG_LIBDIR=$prefix/lib/pkgconfig
export PKG_CONFIG_LIBDIR
pkg-config --define-variable=prefix=$prefix $@
```

Debian stretch based cross builds

```
# Debian stretch for W32 and W64:
apt-get install g++-mingw-w64 mingw-w64 mingw-w64-tool mingw-w64-i686-dev mingw-w64-x86-64-dev nsis
```

In addition, several packages from Cygwin are needed. Add https://qemu.weilnetz.de/debian/ as a package source (see instructions there) and install the required cross packages.

Linux Mint based cross builds

These instructions were tested with the Linux Mint Debian Edition on 2012-06-02.

```
# Linux Mint for W32 and W64 (about 463 MiB):
apt-get install mingw-w64
```

OpenSUSE based cross builds

Add http://download.opensuse.org/repositories/windows:/mingw:/win32/openSUSE_11.4 (update with your release version) to the list of software repositories. Then install at least the following packets (most of them are pulled via dependencies):

```
mingw32-binutils
mingw32-cpp
mingw32-cross-binutils
mingw32-cross-cpp
mingw32-cross-gcc
mingw32-cross-pkg-config
mingw32-filesystem
mingw32-gcc
mingw32-glib2
mingw32-glib2-devel
mingw32-glib2-lang
mingw32-headers
mingw32-libgcc
mingw32-libgmp
mingw32-libintl
mingw32-libintl-devel
mingw32-libmpc
mingw32-libmpfr
mingw32-libSDL
mingw32-libSDL-devel
mingw32-libssp
mingw32-runtime
mingw32-zlib
mingw32-zlib-devel
```

This toolchain does not include libiberty.a in its binutils package, but it also does not need to. If building against a QEMU version that still pulls this in unconditionally, simply drop the `-liberty` from configure.

For W64 use the corresponding win64 repository and mingw64- packages.

Fedora based cross builds

Fedora supports both W64 and W32 cross builds.

```
# Fedora for W32 cross build:
dnf install mingw32-pixman mingw32-glib2 mingw32-gmp mingw32-SDL2 mingw32-pkg-config

# Fedora for W64 cross build:
dnf install mingw64-pixman mingw64-glib2 mingw64-gmp mingw64-SDL2 mingw64-pkg-config
```

Cross compilers usually are installed in /usr/bin with a prefix. This cross prefix must be passed to QEMU's configure. The prefix depends on your target platform.

For Fedora W64 builds, the cross gcc is called x86_64-w64-mingw32-gcc.

```
# Fedora cross configuration for W64:
./configure --cross-prefix=x86_64-w64-mingw32-
```

For Fedora W32 builds, the cross gcc is called i686-w64-mingw32-gcc.

```
# Fedora cross configuration for W32:
./configure --cross-prefix=i686-w64-mingw32-
```

Note that "-mingw32-w64" appears in prefix for both W32 and W64 builds.

Docker based cross builds

As of June 2016, the master tree supports "docker based compiling", which can be used for convenient windows cross build.

Make sure your docker command works ("docker ps" or "sudo docker ps" reports no error), then cd into the root of QEMU source tree and run

```
make docker-test-mingw@fedora V=1 DEBUG=1 J=4
```

, it will download and initialize the needed docker image for you, and drop you into a shell in the started container. Run

```
cd $QEMU_SRC
```

then

```
./configure --cross-prefix=x86_64-w64-mingw32-
```

or

```
./configure --cross-prefix=i686-w64-mingw32-
```

for 64bit/32bit builds respectively.

Native builds with Mingw-w64

Get and install Mingw-w64. In addition, some more packages are needed:

Libraries (also needed for cross builds)

- GLib Run-time (http://ftp.gnome.org/pub/gnome/binaries/win32/glib/2.28/glib_2.28.1-1_win32.zip)
- GLib Development (http://ftp.gnome.org/pub/gnome/binaries/win32/glib/2.28/glib-dev_2.28.1-1_win32.zip)
- gettext-runtime Development (http://ftp.gnome.org/pub/gnome/binaries/win32/dependencies/gettext-runtime-dev_0.18.1.1-2_win32.zip)

Tools (only needed for native builds)

pkg-config (http://ftp.gnome.org/pub/gnome/binaries/win32/dependencies/pkg-config_0.23-3_win32.zip)

Get the QEMU source code (git or tarball), then run configure and make.

Native builds with Cygwin

Builds with the normal Cygwin compiler are not supported. Nevertheless, cygwin can be used as a build environment because it also contains all necessary Mingw-w64 packages.

```
# Don't build in the QEMU source directory. Using a subdirectory is better.
# Here is an example of a debug build.
SRC_PATH=$PWD
BUILD_DIR=$PWD/bin/debug/i686-w64-mingw32
mkdir -p $BUILD_DIR
cd $BUILD_DIR
$SRC_PATH/configure' '--enable-debug' '--cross-prefix=i686-w64-mingw32-'
make
```

For 32 bit builds, these packages should be installed:

Required packages

- mingw64-i686-gcc-g++
- mingw64-i686-glib2.0
- mingw64-i686-pixman
- mingw64-i686-pkg-config

Recommended packages

- mingw64-i686-curl
- mingw64-i686-gtk3
- mingw64-i686-libssh2
- mingw64-i686-libtasn1
- mingw64-i686-nettle
- mingw64-i686-ncurses
- mingw64-i686-gnutls

Optional packages

- mingw64-i686-SDL2
- mingw64-i686-libgcrypt
- mingw64-i686-libusb1.0
- mingw64-i686-usbredir

Building QEMU for W64

QEMU for W64 needs a fairly complete MinGW-w64 based development environment with tools (make, compiler, linker, ...) and some additional libraries.

Cross builds

Compilation of QEMU for W64 on non-W64 hosts (e.g. Linux hosts) is called cross compilation. Some Linux distributions (Debian, Ubuntu, Fedora and maybe others) already include packages needed for cross compilation, so the installation of these packages is the first step.

Debian based cross builds

```
# Debian cross configuration for W64:
configure --cross-prefix=amd64-mingw32msvc-
```

Fedora based cross builds

```
# Fedora cross configuration for W64:
./configure --cross-prefix=x86_64-w64-mingw32-
```

Libraries (also needed for cross builds)

- GLib Run-time (http://ftp.gnome.org/pub/gnome/binaries/win64/glib/2.22/glib_2.22.4-1_win64.zip)
- GLib Development (http://ftp.gnome.org/pub/gnome/binaries/win64/glib/2.22/glib-dev_2.22.4-1_win64.zip) - newer versions don't work because leading underscores for global symbols are missing
- gettext-runtime Development (http://ftp.gnome.org/pub/gnome/binaries/win64/dependencies/gettext-runtime-dev_0.18.1.1-2_win64.zip)

Native builds with MSYS2

MSYS2 provides a convenient environment to produce native builds for W64.

Download and run the MSYS2 installer from [1].

As per the MSYS2 documentation, download the latest repository updates with:

```
pacman -Syu
```

If required, restart the MSYS2 console. Then update the remaining packages with:

```
pacman -Su
```

Next install the basic set of developer tools:

```
pacman -S base-devel mingw-w64-x86_64-toolchain git python ninja
```

Then install any required QEMU-specific packages. For a basic setup you can use:

```
pacman -S mingw-w64-x86_64-glib2 mingw-w64-x86_64-pixman python-setuptools
```

Additional optional features require more packages. For example, to enable GTK+ and SDL user interface and user networking you can use:

```
pacman -S mingw-w64-x86_64-gtk3 mingw-w64-x86_64-SDL2 mingw-w64-x86_64-libslirp
```

- Close the MSYS2 console.
- Start mingw64.exe.

Download the QEMU source code:

```
git clone https://gitlab.com/qemu-project/qemu.git
```

Finally build QEMU with:

```
cd qemu
./configure --enable-sdl --enable-gtk
make
```

Installation

Installation is easy with the installers from https://qemu.weilnetz.de/.

Running QEMU for W32

User mode emulation is unsupported: it only works on BSD and Linux.

System emulation

All QEMU system emulation should be working (that simply means I don't know of emulations which don't work, and those which I tried, namely x86 and mips, work well).

Hardware acceleration for x86 can be enabled with the command line option `--enable-whpx`.

Special W32 devices

QEMU is based on Mingw-w64, so some commonly used UNIX device names like /dev/null or /dev/zero can be used. W32 device names also work, especially names like //./PhysicalDrive0 for the first hard disk of the host (this name must be used with extreme care or you will likely crash your system).

Text which is normally printed by QEMU to the console output channels (normally known as standard output = stdout and standard error output = stderr) might be written to files called stdout.txt and stderr.txt if QEMU was linked with SDL 1.2. If you want to see QEMU's help messages or if it does not work as expected, you should look for these files in the directory where your exe file is installed.

Links

Mingw-w64 Website (supports both 32 and 64 bit builds)
http://mingw-w64.sourceforge.net/