Porting CircleMUD to New Platforms

Jeremy Elson

November 17, 2002

Abstract

CircleMUD is a very portable program by design, but is not guaranteed to run on every platform that exists. This document is for experienced programmers trying to make CircleMUD work on their platform.

1 Introduction

CircleMUD should work on most UNIX platforms without any modifications; simply run the "`configure`" script and it should automatically detect what type of system you have and anything that may be strange about it. These findings are all stored in a header file called `conf.h` which is created in the src directory from a template called `conf.h.in`. A `Makefile` is also created from the template `Makefile.in`.

Non-UNIX platforms are a problem. Some can't run CircleMUD at all. However, any multitasking OS that has an ANSI C compiler, and supports non-blocking I/O and socket-based TCP/IP networking, should theoretically be able to run CircleMUD; for example, OS/2, AmigaOS, Mac OS (Classic versions; Mac OS X supports CircleMUD's configure script from the command line), Windows 3.11/NT/9*.

The port can be very easy or very difficult, depending mainly on whether or nor your OS supports the Berkeley socket API. Windows 95, for example, supports Berkeley sockets with a few modifications using the WinSock library, and OS/2 supports Berkeley sockets with no source changes at all.

The general steps for porting CircleMUD to a non-UNIX platform are listed below. A number of tips for porting can be found after the porting steps. Note that we have already ported Circle to Windows 95, so if you're confused as to how to perform some of these steps, you can look at what we have done as an example (see the files `conf.h.win`, `Makefile.win` and `README.WIN`).

Note that you should not try to do this unless you are an experienced C programmer and have a solid, working knowledge of the system to which you are trying to port the code.

2 Porting the Code

Step 1. Create a "`conf.h`" file for your system.

Copy the template "`conf.h.in`" to "`conf.h`", and then define or undefine each item as directed by the comments and based on the characteristics of your system. To write the `conf.h` file, you'll need to know which header files are included with your system, the return type of signals, whether or not your compiler supports the '`const`' keyword, and whether or not you have various functions such as `crypt()` and `random()`.

Also, you can ignore the `HAVE_LIBxxx` and `HAVE_xxx_PROTO` constants at the end of `conf.h.in`; they are not used in the code (they are part of UNIX `autoconf`).

Step 2. Create a `Makefile`.

Again, copy the template `Makefile.in` and make any changes which may be appropriate for your system. Make sure to remove the `@xxx@` variables such as `@LIBS@`, `@CC@`, `@NETLIB@`, etc., and replace them with the appropriate values if necessary.

Step 3. Make the appropriate patches to the code so that the TCP/IP reads and writes and signal handling are compatible with your system. This is the hardest part of porting CircleMUD. All of the changes you will need to make will probably be in the source file `comm.c`.

Step 4. Test your changes! Make sure that multiple people can log in simultaneously and that they can all type commands at the same time. No player should ever have a "frozen" screen just because another is waiting at a prompt. Leave the MUD up for at least 24 hours, preferably with people playing it, to make sure that your changes are stable. Make sure that automatic events such as zone resets, point regeneration, and corpse decomposition are being timed correctly (a tick should be about 75 seconds). Try resetting all the zones repeatedly by typing "`zr *`" many times. Play the MUD and make sure that the basic commands (killing mobs as a mortal, casting spells, etc.) work correctly.

Step 5. If you are satisfied that your changes work correctly, you are encouraged to submit them to be included as part of the stock CircleMUD distribution so that future releases of Circle will support your platform. This prevents you from re-porting the code every time a new version is released and allows other people who use your platform to enjoy CircleMUD as well.

To submit your changes you must make a patch file using the GNU '`diff`' program which can be downloaded by anonymous FTP from `ftp://ftp.gnu.org:/pub/gnu/diffutils-x.x.tar.gz`. `diff` will create a patch file which can be later used with the '`patch`' utility to incorporate your changes into the stock CircleMUD distribution. For example, if you have a copy of stock (plain) CircleMUD in the "`stock-circle`" directory, and your changes are in "`my-circle`", you can create a patch file like this:

```
diff -u --new-file --recursive stock-circle/src my-circle/src > patch
```

This will create a file called 'patch' with your patches. You should then try to use the 'patch' program (the inverse of 'diff') on a copy of stock circle to make sure that Circle is correctly changed to incorporate your patches.
This step is very important: if you don't create these patches correctly, your work will be useless because no one will be able to figure out what you did! Make sure to read the documentation to 'diff' and 'patch' if you don't understand how to use them.
If your patches work, CELEBRATE!!

Step 6. Write a README file for your operating system that describes everything that has to be done by another user of your operating system to get CircleMUD to compile from scratch. You should include a section on required hardware, software, compilers, libraries, etc. Also include detailed, step-by-step instructions on how to compile and run everything. You can look at the other README files in the distribution (README.WIN, README.OS2, etc.) for examples of what your README file should include.

Step 7. You are done! Congratulations! Mail your conf.h, Makefile, patches, and README file to the CircleMUD Group <cdev@circlemud.org> so that they can be included in future releases of CircleMUD. Please share your work so that other users of your OS can use Circle, too.

3 Porting Tips

Some tips about porting:

3.1 Making your own CIRCLE_system constant

Each system to which Circle is already ported has a CIRCLE_xx constant associated with it: CIRCLE_UNIX for plain vanilla UNIX CircleMUD, CIRCLE_WINDOWS for MS Win95/NT, CIRCLE_OS2 for IBM OS/2, and CIRCLE_AMIGA for the Amiga. You must use a similar constant for your system. At the top of your conf.h, make sure to comment out "#define CIRCLE_UNIX" and add "#define CIRCLE_YOUR_SYSTEM".

3.2 ANSI C and GCC

As long as your system has an ANSI C compiler, all of the code (except for comm.c) should compile with no major complaints. However, Circle was written using gcc, and some compilers are nitpicky about things that gcc does not care about (and the other way around). Therefore, you are *highly* encouraged to use gcc if at all possible. gcc has been ported to a very large number of platforms, possibly including yours, and your port will be made much easier if you use gcc. You can download gcc via anonymous FTP from ftp://ftp.gnu.org:/pub/gnu/.

3.3 Non-Blocking I/O

Make absolutely sure to use non-blocking I/O; i.e. make sure to enable the option so that the `read()` system call will immediately return with an error if there is no data available. If you do *not* use non-blocking I/O, `read()` will "block," meaning it will wait infinitely for one particular player to type something even if other players are trying to enter commands. If your system does not implement non-blocking I/O correctly, try using the `POSIX_NONBLOCK_BROKEN` constant in `sysdep.h`.

3.4 Timing

CircleMUD needs a fairly precise (on the order of 5 or 10 ms) timer in order to correctly schedule events such as zone resets, point regeneration ("ticks"), corpse decomposition, and other automatic tasks. If your system supports the `select()` system call with sufficient precision, the default timing code should work correctly. If not, you'll have to find out which system calls your system supports for determining how much time has passed and replace the `select()` timing method.

3.5 Signals and Signal Handlers

A note about signals: Most systems don't support the concept of signals in the same way that UNIX does. Since signals are not a critical part of how Circle works anyway (they are only used for updating the wizlist and some other trivial things), all signal handling is turned off by default when compiling under any non-UNIX platform (i.e. the Windows 95 and Amiga ports do not use signals at all.) Simply make sure that `CIRCLE_UNIX` is *not* defined in your `conf.h` file and all signal code will be ignored automatically.

4 Final Note

IMPORTANT: Remember to keep any changes you make surrounded by `#ifdef` statements (i.e. "`#ifdef CIRCLE_WINDOWS` ... `#endif`"). If you make absolutely sure to mark all of your changes with `#ifdef` statements, then your patches (once you get them to work) will be suitable for incorporation into the CircleMUD distribution, meaning that CircleMUD will officially support your platform.