

Jack Whitham

About


Blog Archive

Software

Monday, 6 July 2015

Porting third-party programs to TempleOS

I recently read about TempleOS and thought it might be interesting to try to port some software to it.



```
Public Domain Operating System
Help & Index, Quick Start: Cod line
Directory of C:\Home
DATE TIME 0120
06/07 18:39 0000
06/07 18:39 0000 MakeDistra.CPP.Z
06/07 18:39 0000 MakeDistra.CPP.Z
C:\Home\cd\ : /Home >> RunIndex
00000000: Tip of the Day
* Use View() in Pop-up menus to linger
until the user presses <ESC> or <SHIFT-
ESC>.
* You can adjust the mouse movement
rate by setting global vars in your
start-up file. See mouse_scale.
* You can set your local time zone by
setting the local_time_offset global
var in a start-up file. It's units are
CONF=HR. See local_time.
* Get rid of this msg here.
Take Tour(y or n)?
```

I first heard of TempleOS when it was called "Losethos". It was found by some of the people I knew at [the University](#), where it was generally regarded as an absurd project. Since that time, the OS has been renamed, many people have heard of it, there are [positive reviews of it](#), and its lone creator Terry Davis has become famous, being [profiled by Vice magazine](#). The project is eccentric and absurd, but it is also [admired by programmers as a substantial achievement](#): all of this written by one man, working almost from scratch, building his own software tools, a kernel, a user interface, applications. It must be a decade's work.

An "alien" operating system

The "alien" nature of the project makes it interesting. Most OS designers would not start by developing their own programming language - most likely, they would write their kernel in C, with some assembly code. This was the approach used for Linux, Windows and DOS. OSs written in other languages are curiosities which don't tend to make it far outside of research labs - Microsoft's [Singularity](#) would be an example. But even these unusual OSs are still based on existing languages. Using an existing language makes everything easier. You can reuse development tools, and once the kernel is working, you can reuse applications and user interface software.

So it was with [Linux](#). When Linus Torvalds wrote Linux, he didn't need to write development tools, applications and a user interface, because those components already existed. Most were provided by [the GNU project](#), which had spent much of the 1980s gradually reimplementing essential software tools. Torvalds only needed to write the Linux kernel. The Unix shell, the shell applications, the X Window System and even some classic games would work on Linux as soon as they were recompiled. C was the common language.

HolyC

Terry Davis defied convention by creating his own language, HolyC. HolyC is superficially

Dr Jack Whitham

Software Engineer, [Rapita Systems](#), York



Blog Archive

- 2017
 - 22 October [Modified Unix Tools...](#)
 - 15 October [Notifications, from...](#)
 - 08 October [Differences between...](#)
 - 08 October [At the Yorkshire Mar...](#)
 - 01 October [unwind: target platf...](#)
 - 24 July [Updating a Raspberry...](#)
 - 26 March [Encrypted password s...](#)
- 2016
 - 29 December [Site updates](#)
 - 17 June [Supporting Duff's De...](#)
 - 29 March [A detailed timing tr...](#)
 - 28 February [Profiling versus tra...](#)

similar to C, but the similarities are not deep. It's roughly on the same level as the similarities between Java and C: the two share some syntax, but they're fundamentally different languages.

This difference means that TempleOS exists in an extremely separated software ecosystem. At the time of writing, you can't compile a C program on TempleOS. There is no C compiler. There is no C library. You would have to manually translate the C program to HolyC.

I am sure this choice was deliberate. In [his review of TempleOS](#), Richard Mitton noted that HolyC enables some things that you cannot do with C. For instance, every program is JIT-compiled from source, and the shell is an interpreter for HolyC code. You load a program by "#include"-ing it into the shell, and then you can call its functions from the command line. If you change it, you can usually "#include" it again.

My impression is that C just did not fit into Terry Davis' vision of how his OS should work. Part of the point of the project was to completely avoid it. C's many detractors might agree with the wisdom of this!

How to port a C program to TempleOS

Fortunately, TempleOS is not entirely disconnected from all convention. It runs on a PC. Therefore, TempleOS does have a language in common with Linux, namely machine code.

I decided that the quickest way to port a C program to TempleOS was to compile it on Linux, but arrange for all of the Linux system calls to be translated to TempleOS system calls.

To do this, I compiled a C program against the embedded C library [uClibc](#), generating a position-independent binary file. uClibc provides all the runtime services that C programs expect, implementing functions like "printf" and "malloc".

I modified uClibc so that each system call would be directed to a particular location within the header of the binary. Each instance of the "syscall" instruction instead became an indirect "jmp" instruction.

Then I wrote a loader program in HolyC which was able to start programs compiled in this special way. The loader's functions are as follows:

1. copy the binary into memory within TempleOS,
2. modify it so that all system calls are directed to a HolyC function,
3. set up the environment and program arguments (argc, argv, etc.),
4. start the program,
5. clean up afterwards e.g. freeing memory.

The loader is a 500 line program written in HolyC and x86_64 assembly code.

This approach enables a C program - and, in principle, any program that can be compiled on Linux - to be executed on TempleOS.

There are restrictions. For instance, the program can only use the system calls that I have implemented in HolyC. This means that, for now, programs are limited to very simple file and screen I/O. There is no graphics support. Many of the capabilities available on Linux, and indeed on TempleOS, are just not available to programs loaded in this way.

Frotz - the first C program for TempleOS?

Without graphics support, the proof-of-concept application is quite limited in what it can do. I therefore chose to port "[frotz](#)". This is an interpreter for Z-machine games. The system requirements are minimal, but it can be used to play perhaps thousands of "interactive fiction" (text adventure) games. They include the original Infocom games and plenty of newer games, mostly found on [IFdb](#).

Here it is, running a Z-machine port of "[Adventure](#)".



You can load other Z-machine games into Frotz:



I put together a demo, in the form of a CD image, which includes Adventure and two other (free) Z-machine games. You can download the demo from my Github page:

<https://github.com/jwhitham/frotz/>

That page also has some setup instructions, all of the source code, and some greater detail about the techniques used.

Working Process

I did most of the development work in Linux. I first wrote the loader on Linux itself, using C. Then I ported it to Windows, which meant changing a few system calls, but was relatively easy. When I was able to load the same "frotz" binary on both Windows and Linux, I ported the loader to HolyC on TempleOS.

This was challenging. Debugging was hard, and on several occasions I made TempleOS crash completely by doing dumb things that would merely have crashed a single process on Linux or Windows, e.g. freeing the same memory twice. TempleOS has no memory protection, so mistakes of that sort are not forgiven. However, I was impressed by its ability to recover from other sorts of bug. When all else fails, it drops to a debugging monitor and prints a backtrace.

In order to keep using git, vim, a disassembler and other dev tools, I dual-booted my virtual machine between TempleOS and Linux, using a FAT32 partition to exchange files.

Why?

I recall seeing [this comic](#) while working as a researcher. Recently, TempleOS reminded me of it. (Many of my colleagues were the guy in the first picture. I'm the other guy.)

For me, the fun - the challenge - is to make something work within difficult constraints. Like making a C program work on a system that's deliberately not based on C. Or building an OS from scratch in the Terry Davis way, where "from scratch" means "ex nihilo".

Porting software to TempleOS is a challenge that goes beyond the usual approaches of recompiling the program and translating some platform-dependent functions.

This small project was filled with reminders of the extent to which we depend on software infrastructure, i.e. tools. Not just the C compiler, but also effective debuggers, editors and version control systems. TempleOS hits the reset button on all of these things: programmers' tools are all different. I depend so heavily on the tools that I normally use. It's a real culture shock to work in an environment where none of them are available. But that's why it's a challenge.

Further work

The next step would probably be to get some sort of Curses emulation running, so that Nethack can be ported. Support for bitmap graphics and raw keyboard I/O would be enough for Doom and Quake. Though these games would be hampered by the 4-bit colour used by TempleOS, positional colour dithering can be used to simulate full colour, and this would probably work well enough to be playable. (I once did [an FPGA port of Doom](#) with only 3-bit colour hardware - the dithering was done in hardware, but PCs are fast enough to do it all in software.)


Porting something like Dosbox would also unlock a lot of possibilities. And porting a C compiler, e.g. GCC, might actually approach *usefulness*...!

Posted by Jack Whitham at [21:40](#)

[Newer Post](#)

[Home](#)

[Older Post](#)

 [Subscribe to my RSS feed](#)

Simple template. Template images by [merrymoonmary](#).