Return of the Necromancer

Adam Thornton

September 17, 2025

Software Necromancy Part II

Return of the Necromancer

Adam Thornton, May 20201

`https://athornton.github.io/return-of-the-necromancer` (PDF)

Two main topics for today

Porting sort-of-modern software to ancient operating systems

Unix v7 as a daily driver

First port: Frotz for TOPS-20

Frotz is an interpreter for Infocom-format Z-machine games. This includes the released Infocom games–Zorks 1-3, Planetfall, The Hitchhiker's Guide to the Galaxy, Trinity, et cetera.

It also includes the post-Infocom Z-machine games: Curses, Anchorhead, et al. Z5 existed in the real Infocom world (Trinity, Beyond Zork, Solid Gold). Z8 is purely post-commercial.

Why Frotz?

I maintain the Linux CLI port of Inform 7; dumb-frotz is the test harness there, so I knew it reasonably well. The dumb-frotz port does not require a cursor-addressible terminal, which meant I wasn't going to have to find or write a curses layer for my target systems.

It's ANSI C (almost), and pretty straightforward.

Why TOPS-20?

Infocom used a TOPS-20 system as their development system once they went commercial. Mark Crispin (RIP) created a lovely TOPS-20 distribution, Pandas, which includes an ANSI C compiler and a lot of Unix accommodation support. This plus the KLH-10 emulator made getting a development environment pretty easy, and setting up TCP/IP on TOPS-20 wasn't too difficult; thus getting files in and out was almost straightforward.

Why not ITS?

Because ITS sucks.

Sure, it'd be nice to bring the Z-machine games all the way back home. But the C compiler is slightly pre-K&R and the overall environment is something I find just excruciating.

Lars Brinkhoff and David Griffith may be working on this.

So, what's hard here?

9-bit bytes

The Z-machine looks a lot like a PDP-11: 16-bit word, 64K addressible memory (z5 and z8 basically just stretch the alignment, allowing more ROM text, but not doing a lot with the writeable memory of the Z-machine). The host machine is 36-bit with 9-bit chars.

The linker

You've only got six characters, effectively, for a symbol. Modern applications do not typically have only-six-character function names.

We'll start with the linker

This is the easier problem. If the function names are too long…make them shorter. Write a source code preprocessor:

`https://github.com/athornton/gnusto-frotz-tops20`

I used Perl, because for line-at-a-time text parsing and regex substitution, it still beats the hell out of Python.

Once you've done that....

`https://github.com/athornton/tops20-frotz`

urbzig.sed contains the symbol remappings.

Then you get to tackle the hard part.

Nine-bit bytes

From the Dumb-Frotz README, in CAVEATS:

lack of 8-bit char and 16-bit short. I didn't bother to think much about this. If you're using a 36-bit Honeywell or something, let me know.

In practice, not that bad

Turn macros back into functions for easier debugging.

Do a lot of &0xFF and &0xFFFF.

There are a few places with pointer arithmetic that you need to be careful with.

Frotz has a decent test suite, so it was an iterative process to see which opcodes were failing and then a matter of tracing the flow and figuring out where the failure in the opcode is.

And now it works

All the V3-V8 games run. Those that need fancy display capabilities (Jigsaw, Beyond Zork, some of the stunt games) don't work so great, of course.

Second port: ZIP for 2.11BSD

Another Z-machine interpreter, put together for my annual Elvis party.

Zork I was actually released for RSX-11, and I think there's a reasonable RSX-11M+ interpreter for most games, but I wanted it to be Unix.

What's hard here?

The hard part is that the PDP-11 is not actually bigger than the Z-Machine. This isn't a problem per se. The whole point of the Z-machine was to let you implement a 16-bit virtual machine on an 8-bit micro.

But I didn't have time or desire to write a tight little Z-code terp in PDP-11 assembly. And anyway those were generally only v3 interpreters, and I wanted at least v5 games to work.

Where to start?

Modern Z-terps assume that there's plenty of system memory available: no need to do paging and memory is at least 32 addressible bits. So I had to go back to a DOS-era interpreter to find something that didn't make those incorrect assumptions.

ZIP. No, not that one.

I ended with the Zmachine InterPreter, ZIP (named before the archive format), now called "zterp" for obvious reasons

`https://github.com/athornton/pdp11-zterp`

The only changes I remember needing were some linker flags to use split instruction and data spaces.

A few of the bigger Z5 games still don't run, but most games do.

Third port: Forth for v7

I started with Leif Bruder's Forth: `https://gist.github.com/lbruder/10007431`

It's a single-file C program, ANSI C. It builds cleanly on 2.11 BSD.

Ah, but v7

So, there are a few problems. The six-character symbol thing is back in effect. That's tractable.

The C compiler is K&R, not ANSI. That too is tractable; mostly unprotoizing functions, and there was a concatenation macro that I needed to fiddle with.

But still not success

So now I can get all the functions to compile, but the linker is angry because either (if I leave it as one file) there are too many symbols, or (if I break the functions into individual files) there are too many files. I'm fairly confident there is a happy medium here somewhere, but I haven't put in the time to find it. The v7 linker is fragile and does not have good error messages.

v7 as a Daily Driver

I'm not **really** suggesting doing this, but…how close to usable is a v7 system?

Let's assume that ”editing, compiling, and running C files and shell scripts” is much of what you need on a day-to-day basis.

There are two things obviously wrong with v7 for that out of the box: no screen editor, and no networking, so no easy way to get a file into the v7 environment for editing.

Screen Editor

Webb Miller wrote A Software Tools Sampler in 1987. It includes a little vi-sorta-alike called s. This can be found at `https://github.com/udo-munk/s/`.

It took a little hacking to make work under v7, but not much (I don't remember the details, alas). It still doesn't work reliably on large files, and the screen handling can be a little funny, but…basically it works like vi. Which is no emacs, but is better than ed, for sure.

Getting files into v7

There's a trivial and lame way. I use iTerm2 on the Mac. It has a ”paste slowly” option.

cat > filename … and then paste slowly into it from the Mac's effectively-infinite pasteboard. Doesn't work well for non-text, but that's not much of a problem, honestly.

I wanted something slightly less janky.

UUCP

Before there was universal TCP/IP, there was UUCP. Designed for dialup, batch file transfer. This is kind of perfect for what I needed.

I followed the guide at UUCP for v7 and now have a reasonable way to move files from the Pi host (which of course is a completely-modern Linux system) to v7, and vice versa. It's extremely slow, but what do I care?

Verdict

You wouldn't WANT to use it, but…v7 is still much more pleasant than DOS or Windows 3.x. If you really want to use a Unix on a PDP-11, though, NetBSD 2.11 is pretty much a fully-baked BSD, with a TCP/IP stack and vi and everything.

Other stuff

I'm working on getting a DECNet Phase IV (over IP) cluster going between a real VAX, a real Alpha, an emulated VAX, and an emulated Alpha.

Also trying to get a real Unix installed on a real VAXStation 4000vlc. It will probably be NetBSD.