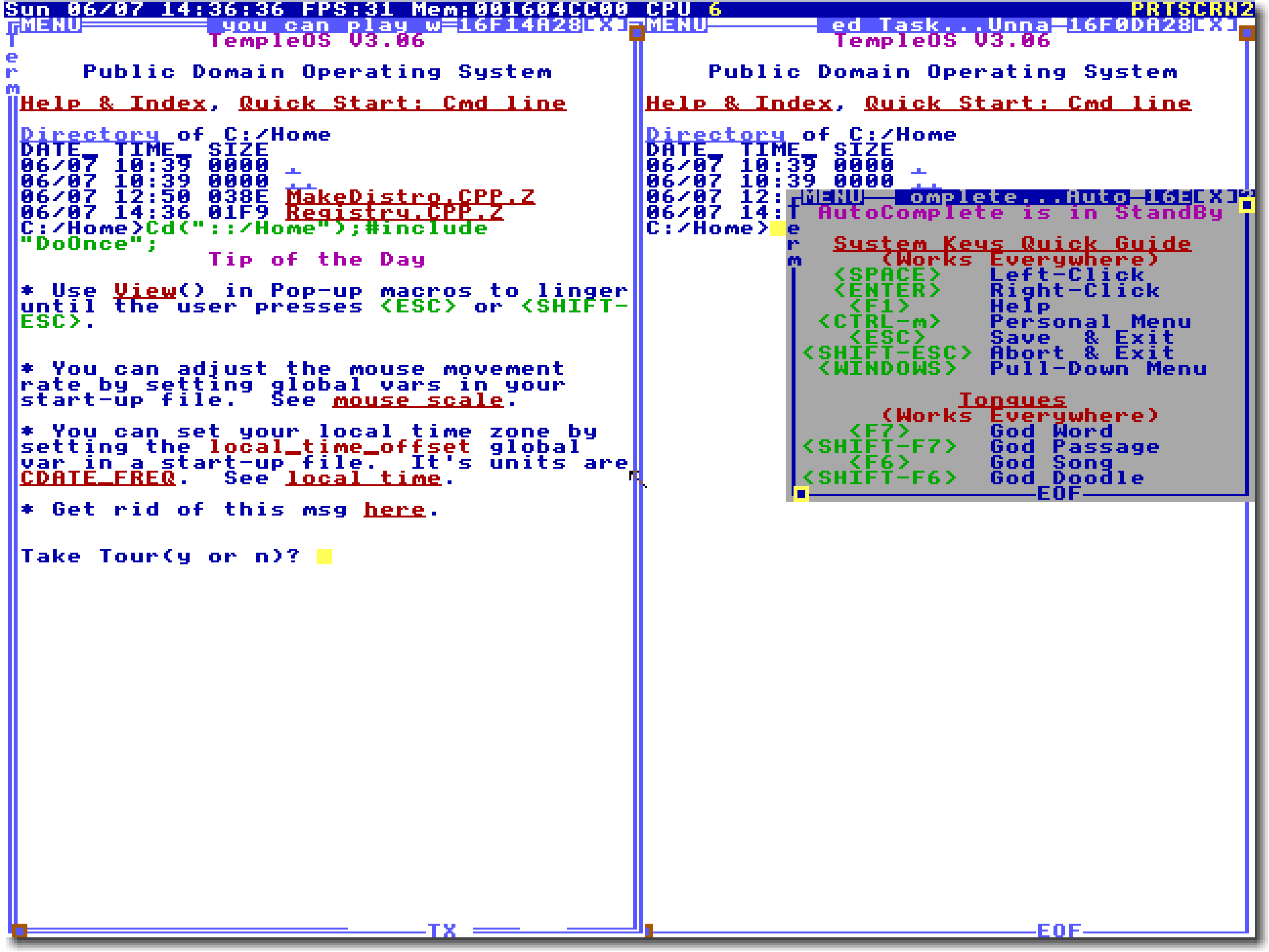
A Constructive Look At TempleOS

June 8th, 2015

TempleOS is somewhat of a legend in the operating system community. Its sole author, Terry A. Davis, has spent the past 12 years attempting to create a new operating from scratch. Terry explains that God has instructed him to construct a temple, a 640x480 covenant of perfection. Unfortunately Terry also suffers from schizophrenia, and has a tendency to appear on various programming forums with a burst of strange, paranoid, and often racist comments. He is frequently banned from most forums.

This combination of TempleOS's amateurish approach and Terry's unfortunate outbursts have resulted in TempleOS being often regarded as something to be mocked, ignored, or forgotten. Many people have done some or all of those things, and it's understandable why.



```
Sun 06/07 14:36:36 FPS:31 Mem:001604CC00 CPU 6                                  PRTSCRN2
MENU   you can play w  16F14A28[X]  MENU   ed Task...Unna  16F0DA28[X]
              TempleOS V3.06                          TempleOS V3.06

     Public Domain Operating System           Public Domain Operating System

Help & Index, Quick Start: Cmd line           Help & Index, Quick Start: Cmd line

Directory of C:/Home                          Directory of C:/Home
DATE_ TIME_ SIZE                              DATE_ TIME_ SIZE
06/07 10:39 0000 .                            06/07 10:39 0000 .
06/07 10:39 0000 ..                           06/07 10:39 0000 ..
06/07 12:50 038E MakeDistro.CPP.Z             06/07 12:
06/07 14:36 01F9 Registry.CPP.Z               06/07 14:
C:/Home>Cd("::/Home");#include                C:/Home>
"DoOnce";
              Tip of the Day

* Use View() in Pop-up macros 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
CDATE_FREQ.  See local time.

* Get rid of this msg here.

Take Tour(y or n)?
```

```
MENU   omplete...Auto  16E[X]
  AutoComplete is in StandBy

    System Keys Quick Guide
        (Works Everywhere)
    <SPACE>     Left-Click
    <ENTER>     Right-Click
      <F1>      Help
   <CTRL-m>     Personal Menu
     <ESC>      Save  & Exit
 <SHIFT-ESC>    Abort & Exit
   <WINDOWS>    Pull-Down Menu

            Tongues
        (Works Everywhere)
      <F7>      God Word
  <SHIFT-F7>    God Passage
      <F6>      God Song
  <SHIFT-F6>    God Doodle
                EOF
```

TempleOS after it has booted up.

I'm reminded of a movie I once saw called Lars And The Real Girl, in which a man buys a RealDoll and treats her as his real girlfriend. Rather than laughing at him, the residents of his town instead band together and treat her as if she were a real person too. When I started watching it, I expected some Will Ferrell-esque comedy where this guy would be played only for laughs. Instead, I found an incredibly compassionate story within. The writer, Nancy Oliver, got the idea after thinking:

> *"What if we didn't treat our mentally ill people like animals? What if we brought kindness and compassion to the table?"*

There are many bad things to be said about TempleOS, many aspects of it that seem poorly constructed or wouldn't work in the "real world". I'm going to ignore them here. It's very easy to be negative, but you will never learn anything new by doing so.

Many might consider TempleOS a waste of time, compared to more fully-featured OSs such as Linux, because it will never have the same success. Plan 9, developed by Bell Labs, was a research OS designed to be a successor to Unix. Despite some big names and big ideas, it was never any kind of commercial success. Was Plan 9 therefore a waste of time? Many would argue not, as some of its ideas have since found their way into other products.

Perhaps we should instead look at TempleOS as a *research operating system*: what can be accomplished if you're not locked into established thinking, backwards compatibility, and market demands.

What can we learn if we are only willing to listen?

Installation

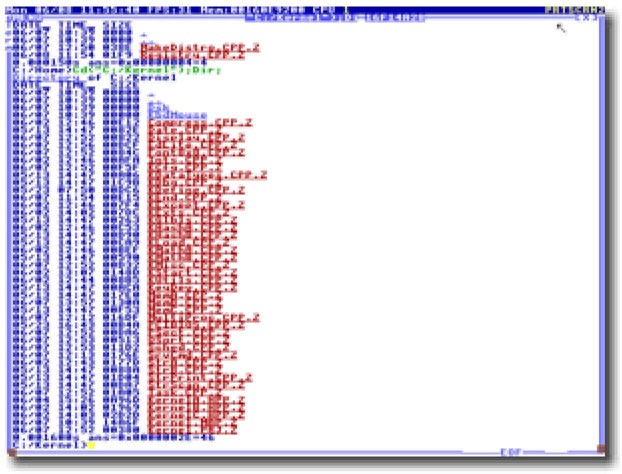
Installation was incredibly painless. The supplied distribution acts as both a LiveCD and hard-disk installer. Hit 'Y' a couple of times and you've installed it. It installed faster than Ubuntu does.

Booting Windows this morning, I can't help but notice how long it takes. And even when Windows 'appears' to have booted, it doesn't actually become properly usable for perhaps another minute. If you listen, you can spot the point at which usability is declared because Windows plays the startup sound to indicate it.

TempleOS boots off hard disk in 1 second. There is no paging, so it is instantly usable.

Shell

TempleOS has its own programming language, HolyC. The whole operating system is written in it, except for x64 assembly in the lower-level parts. Perhaps unexpectedly, the same language is also used for the shell. That's right, you execute shell commands using a C-like language, and they go directly into the compiler.



Every filename printed in the shell is a hyperlink. You can right-click to get a context menu on each one.

There is no built-in calculator application, because the shell itself is one. Just enter `5+7` on the command line and you'll get the answer. You can even go further than this and use the shell as a REPL to build entire programs in.

You have a menu file which sits in your home directory, and it accessible at any time by hitting Ctrl-M. By editing this file, you can create any kind of launcher you wish. Most files auto-save on exit, but this one does not. This means you can also use it as a popup scratchpad.

TempleOS has system-wide autocomplete. You can hit Ctrl-F1 at any point and get a list of completable words. Not just filenames, but also symbol names. All source code is indexed and you can jump to any function from anywhere, even from the shell. The same system works in any program throughout the OS.

TempleOS's unified hypertext really shines when presented in the shell. From the command-line, you can call `Uf("Foo")` to disassemble a function, and each symbol printed will be hyperlinked in the shell window. Click on it to go to the source. objdump can't do that.

The `Type()` function is used to display files, like DOS's type or Unix's cat. Of course, hypertext is respected. You can even use Type to show .BMP files directly in the shell. It raises an interesting challenge for other OSs - why do shells have to be pure text? Why can't we have a multimedia shell?

File Explorer

Most operating systems have something like Explorer, Nautilus or File Manager to let you browse around a directory tree just by clicking. TempleOS does have a File Manager program (Ctrl-D), but it's kinda just an extension of the shell, and surprisingly you don't need it for most operations. By using the hyperlink system that permeates the operating system, the *shell itself* can act as an explorer. Type `Dir;` for a listing, then you can simply click on any directory hyperlink to change to that directory and get a new listing, all within the same shell. Or click on ".." to go up. It takes a little getting used to, but after having used it for a while I have to admit growing quite attached to it.

HyperText (DolDoc)

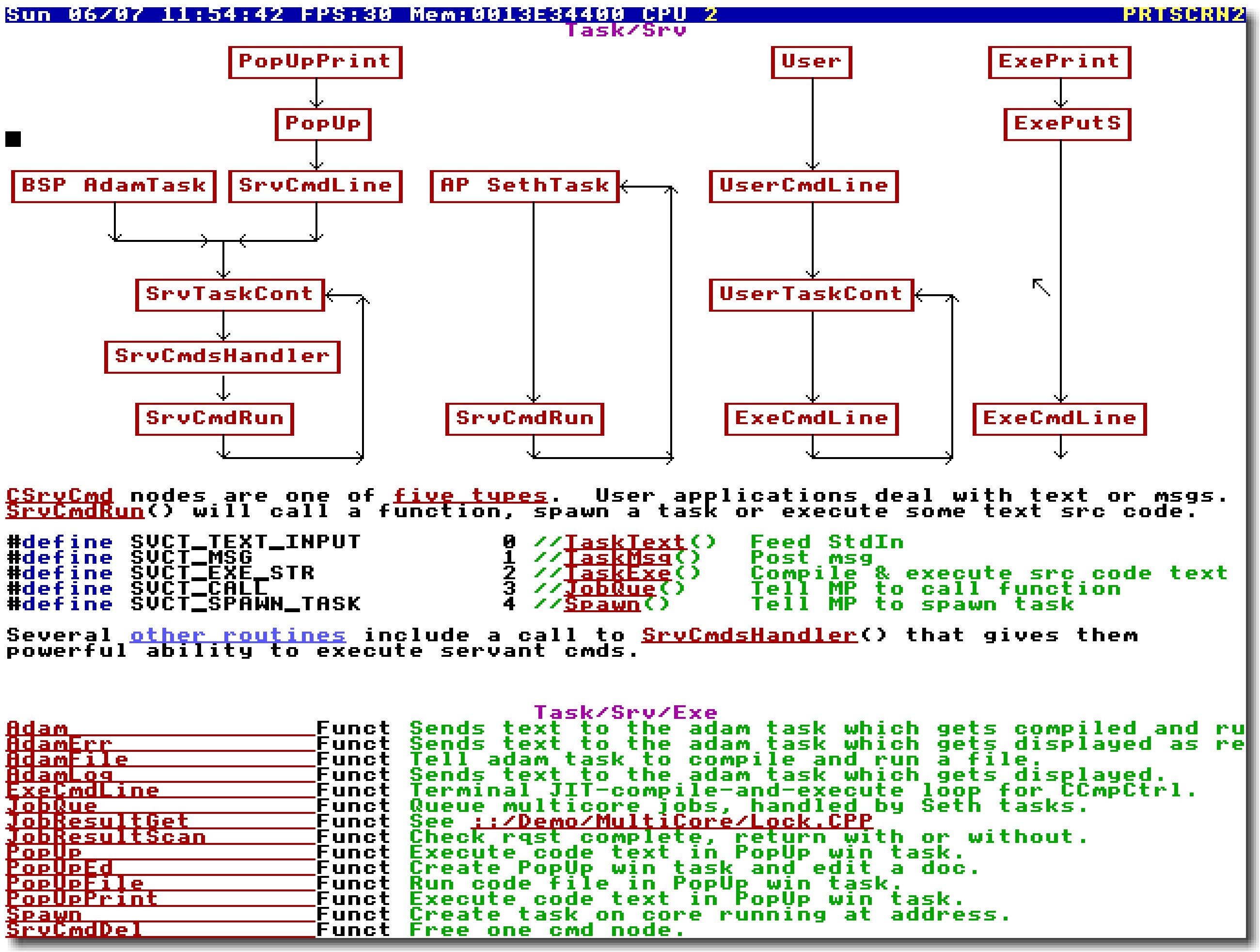
The most notable feature of TempleOS is its ubiquitous hypertext system, DolDoc. This is the foundation for the both the shell and the text editor. Unlike Unix which represents everything via plain-text, everything in Temple is stored in DolDoc format. The format itself is somewhat akin to RTF, and you can hit Ctrl-T at any point to inspect the raw text directly.

But DolDoc isn't just for text. You can store images (and even 3D meshes) directly into documents. You can put macros in there: hyperlink commands that run when you click on them. So if you want to build a menu or launcher, you just make a new text document and put links in it.

All of this allows something similar to the Oberon system developed at ETH Zurich, where the distinction of text, programs, menus and forms all blurs together into one.

HTML, JSON, XML, shell scripts, source files, text files - TempleOS replaces all of these via one unified hypertext representation.

In a file from the TempleOS source code, one line contains the passage "Several other routines include a ...", where the "other routines" part is a hyperlink. Unlike in HTML, where that perhaps may lead to a page listing those other routines, here a DolDoc macro is used so that a grep is actually performed when you click on it. While the HTML version could become stale if no-one updated it, this is always up-to-date.



```
CSrvCmd nodes are one of five types.  User applications deal with text or msgs.
SrvCmdRun() will call a function, spawn a task or execute some text src code.

#define SVCT_TEXT_INPUT          0 //TaskText()   Feed StdIn
#define SVCT_MSG                 1 //TaskMsg()    Post msg
#define SVCT_EXE_STR             2 //TaskExe()    Compile & execute src code text
#define SVCT_CALL                3 //JobQue()     Tell MP to call function
#define SVCT_SPAWN_TASK          4 //Spawn()      Tell MP to spawn task

Several other routines include a call to SrvCmdsHandler() that gives them
powerful ability to execute servant cmds.


                              Task/Srv/Exe
Adam             Funct Sends text to the adam task which gets compiled and ru
AdamErr          Funct Sends text to the adam task which gets displayed as re
AdamFile         Funct Tell adam task to compile and run a file.
AdamLog          Funct Sends text to the adam task which gets displayed.
ExeCmdLine       Funct Terminal JIT-compile-and-execute loop for CCmpCtrl.
JobQue           Funct Queue multicore jobs, handled by Seth tasks.
JobResultGet     Funct See ::/Demo/MultiCore/Lock.CPP
JobResultScan    Funct Check rqst complete, return with or without.
PopUp            Funct Execute code text in PopUp win task.
PopUpEd          Funct Create PopUp win task and edit a doc.
PopUpFile        Funct Run code file in PopUp win task.
PopUpPrint       Funct Execute code text in PopUp win task.
Spawn            Funct Create task on core running at address.
SrvCmdDel        Funct Free one cmd node.
```

It's not every IDE that lets you embed images and flowcharts directly into your source code, that kinda makes you sit up and take notice. And yes, those flowchart boxes are hotlinked, so you can click on them to go directly to the source code that implements them.

You can press Ctrl-R at any point to bring up the resource editor, which lets you draw things. The sprites you draw are embedded directly in the document, and you can refer to them using numbered tags. There's no standalone paint program supplied with the OS because you already have one accessible at any time, from within any program. If you want to sketch a doodle, just open a new document, draw things into it and save it out.

HolyC

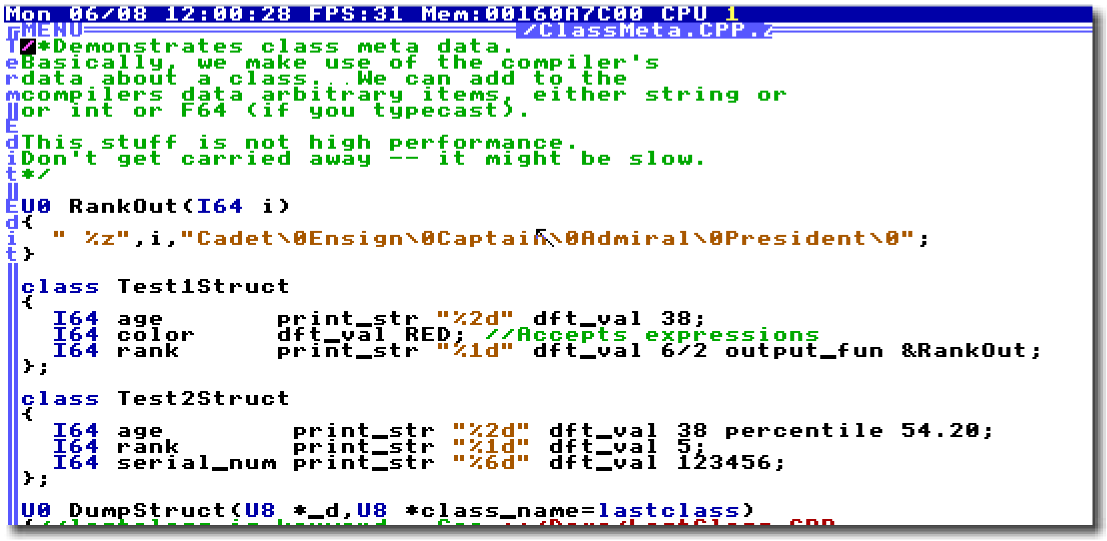
The language provided, HolyC, is at its heart a reasonably complete version of C but with some notable extensions.

There is no main() function in TempleOS programs. Anything you write at top-level scope is executed as it runs through the compiler. In C++ you can do something like `int a = myfunction();`, but you can't just write `myfunction();` and just run it. Why not?

Every piece of code in TempleOS (except the initial kernel/compiler) is JIT compiled on demand. Yes that's right - you can run a program without compiling it, simply by using an `#include` statement from the command line. The program is then brought into the shell's current namespace, and from there you can execute individual functions directly just by issuing commands.

You can tag a function with the `#help_index` compiler directive, and it'll automatically appear in the documentation at the right place. And yes that's fully dynamic. You don't need to run a rebuild process, just compile the file and the documentation updates. Hit F1 and you can see your changes reflected in the help system.

HolyC provides an `#exe` compiler directive, which can be used to shell out to external commands and include their output back into the source code. This provides a way for the user to implement a certain set of functionality that would otherwise require macros or specialized compiler support.



```
/*Demonstrates class meta data.
Basically, we make use of the compiler's
data about a class...We can add to the
compilers data arbitrary items, either string or
or int or F64 (if you typecast).

This stuff is not high performance.
Don't get carried away -- it might be slow.
*/

U0 RankOut(I64 i)
{
  " %z",i,"Cadet\0Ensign\0Captain\0Admiral\0President\0";
}

class Test1Struct
{
  I64 age        print_str "%2d" dft_val 38;
  I64 color      dft_val RED; //Accepts expressions
  I64 rank       print_str "%1d" dft_val 6/2 output_fun &RankOut;
};

class Test2Struct
{
  I64 age         print_str "%2d" dft_val 38 percentile 54.20;
  I64 rank        print_str "%1d" dft_val 5;
  I64 serial_num print_str "%6d" dft_val 123456;
};

U0 DumpStruct(U8 *_d,U8 *class_name=lastclass)
```

You can attach any metadata to any class member.

HolyC's class system implements full metadata and reflection support. Given a class, you can enumerate every member to get its name, offset, etc. What's surprising is that you can also attach any custom metadata to any class member at compile time. Example uses for this might include storing its default value, min/max range, printf format string. Does your language support this?

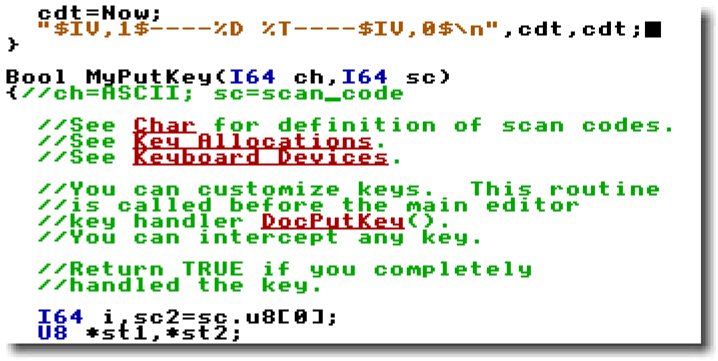
The special `lastclass` keyword can be used as a default argument for functions. It causes the compiler to supply the name of the previous argument's type as a string, allowing you to then do metadata lookups from it.

There is no ahead-of-time linker, nor object files. The dynamic linker is exclusively responsible for binding symbols together at load time. The symbol table remains accessible at runtime, and can be used for other purposes. TempleOS has no environment variables - you just use regular variables.

Programming Environment

HolyC has no formal build system. You just compile a file and you're done. If your project spans more than one file, you just #include all the files into one and compile that. The compiler can compile 50000 lines of code in less than a second.

When you hit F5 in the editor, the program is JIT compiled and run. The top-level statements are executed in turn, and your task is now loaded and ready. If your top-level statements included any sort of loop, it'll stay running there. Otherwise you'll be dropped back into a new shell. However, the shell exists within your task, so you can interactively use it as a REPL and start calling functions inside your program.



```
  cdt=Now;
  "$IV,1$----%D %T----$IV,0$\n",cdt,cdt;
}

Bool MyPutKey(I64 ch,I64 sc)
{//ch=ASCII; sc=scan_code

  //See Char for definition of scan codes.
  //See Key Allocations.
  //See Keyboard Devices.

  //You can customize keys.  This routine
  //is called before the main editor
  //key handler DocPutKey().
  //You can intercept any key.

  //Return TRUE if you completely
  //handled the key.

  I64 i,sc2=sc.u8[0];
  U8 *st1,*st2;
```

Does your current IDE support links to documentation?

Or, perhaps you might place a call to `Dbg()` somewhere in your program, or hit Ctrl-Alt-D, and be dropped directly into the debugger at that point. The debugger of course, still functions as an interactive REPL. Does your IDE support a drop-in REPL?

How much support code does it take to open a window and draw graphics into it on your operating system? In TempleOS, there is a one-to-one correspondence between tasks and windows, so a single call to `DCAlias` is enough to return the device context for your window. You can just do this as soon as your main function is called and draw into it. Making a window is surely the most important thing for a windowed operating system - why does it have to be hard? GDI, X11, DirectX and OpenGL could all learn something here.

Hardware & Security

There is no hardware support. By that I mean that TempleOS does not support any hardware other than the minimal core system that makes a PC. There is no support for any graphics card other than VGA, there is no support for any soundcard other than the PC speaker, and there is no support for networking.

TempleOS does not use memory protection. All code in the system runs at ring 0, the highest privilege level, meaning that a stray pointer write could easily crash the entire system. This is a very *deliberate design choice*:

> ***http://www.templeos.org/TempleOS.html***
>
> *"It's fun having access to everything. When I was a teenager, I had a book, "Mapping the Commodore 64", that told what every location in memory did. I liked copying the ROM to RAM and poking around at the ROM BASIC's variables. Everybody directly poked the hardware ports."*

Terry's philosophy is that growing up in the 1980s, successful machines like the C64 had the same approach. A standardized platform, where you just boot it up and do things on that machine locally. The C64 was a very hands-on machine where the user was all-powerful.

Terry uses the following analogy:

- Linux is a semi-truck with 20 gears to operate.
- Windows is more like a car.
- TempleOS is a motorbike. If you lean over too far, you'll fall off. Don't do that.

He argues that Linux is *designed for a use case that most people don't have*. Linux, he says, aims to be a 1970s mainframe, with 100 users connected at once. If a crash in one users' programs could take down all the others, then obviously that would be bad. But for a personal computer, with just one user, this makes no sense. Instead the OS should empower the single user and not get in their way.

TempleOS has no file permissions. After all, if there's only one user, who else could you give permission to? To be honest, I've often wondered if Unix wouldn't be better if we just made all security happen at the mount level, instead of micromanaging it per file.

There are no such things as threads in TempleOS, as it doesn't need them. Processes and threads are the same thing, because there's no memory protection. If you need something in parallel, just spawn another process and let it share data with your own.

While I can appreciate Terry's philosophy, it's probably fair to point out that the C64 did come with one other thing - an ecosystem where it was commonplace to exchange and distribute programs on physical media, via stores and magazines. As much as I hate to say it, that bird has flown.

Conclusions

In many ways TempleOS seems similar to systems such as the Xerox Alto, Oberon, and Plan 9; an all-inclusive system that blurs the lines between programs and documents.

In this video Terry gives a brief tour of some of the more interesting features of TempleOS. At 5:50, he shows how to build a small graphical application from scratch. Now let's just think about how you'd do this in Windows for a second. Consider for a minute how much code would be needed to register a windowclass, create a window, do some GDI commands, run a message pump, etc. You'd need to set up a Visual Studio project perhaps, and either use the resource editor to embed a bitmap, or try and load it from disk somehow. Now compare it to the tiny snippet of code that Terry writes to accomplish the same task. It certainly makes you wonder where we went so wrong.

Watching TempleOS execute its built-in test suite is a jaw-dropping experience. I can't help but be impressed as a vast number of demos, games, graphing calculators, debuggers, and compilers all fly before your eyes. To see the sheer amount of content that's been written here over the years, to see such effort expended on a labor of love, is wonderfully heart-warming.

Now I don't claim that you should all immediately dump Windows/Linux/OSX and start using TempleOS as your day-to-day operating system. However, you might find that if you take the time to open your mind to new ideas, you might learn something from the most unexpected places.

So what can we learn from this and apply to our own endeavors?

- **There's more than one way things can work.**
  Just because your current toolsets all do things one way, doesn't mean that's necessarily the best way to do it.
- **There's something worthwhile to be found in everything.**
  It's incredibly easy to just assume that because something isn't successful, or is developed by just one guy, that there's nothing to discover there. But if someone's invested 12 years of work into something, is it not possible that you might find even just one interesting idea in there?
- **Don't write things off just because they have big flaws.**
  OK, so it's 16 colors, it's 640x480, and the author is his own worst enemy. But if you dismiss everything in life because it's not what you're used to, you'll never expand outside what you already know.

If GNU is the cathedral, and Linux is the bazaar, perhaps there is a place for the temple somewhere too.

Written by Richard Mitton,
software engineer and travelling wizard.
Follow me on twitter: http://twitter.com/grumpygiant