I Was Blind, But Now I Can C

by Robert W. Dray

Have you ever felt that the devil sent us computers to punish us for all the times we did the right and honourable thing?

I own a SuperPET, but at work I use the new ICON computer with its Unix-like operating system. Recently, I was informed that I was to teach the C programming language beginning in January 1986. I think my superiors selected C because it was not one of the many languages I could practice at home on the SuperPET. Never fear, TPUG to the rescue with Super-OS/9, another Unix-like system, for the SuperPET.

When I first heard that TPUG was offering OS-9 for the SuperPET, I was elated. I would now have an operating system similar to the one on the ICON, and I could get a C compiler for it. With no hesitation, I sent the cheque, and some time later I received a phone call to come into town and pick up a parcel.

With great excitement I opened the package containing several books, some disks and a cute little circuit board. On reading the instructions, I learned to my horror that I would have to take a soldering iron to my SuperPET. How could I violate a long-time friend that way? Nevertheless, after 24 hours of studying the diagrams and wondering whether or not I was capable of such delicate surgery, I opened the lid and started.

The instructions were fairly clear, and I eventually reached the point where they said to turn on the computer, and run the test program. The test didn't work.

You can imagine the sinking feeling in the pit of my stomach. Had I killed the patient? What was this act of foolishness going to cost me to have repaired?

I pulled the parts back out and checked all the pins and solder joints; everything looked okay to me, so I put it back together. This time it passed the test, and I had OS-9 running on my own computer.

Unlike Commodore's own operating system, which is burned into ROM chips, OS-9 is disk-based. If you wish to change Commodore's BASIC 2.0 to BASIC 4.0, for example, you have to remove some chips and replace them with new ones. To modify a disk-based operating system, you simply put the new information on the disk.

Disk-based systems are easily personalized. If you don't like the opening message on power up, you can easily change it. If you are very weird, you even can change the names of the commands so that **dog** instead of **dir** will produce a directory of the disk. You could fix it so that nobody would be able to use your system because only you know the commands.

With Super-OS/9 up, my next task was to get C, so I ordered it from TPUG. After some initial problems (my order got misplaced), I finally received the package of two disks and a book.

The book, like other computer-related books, assumes you know far more than you actually do. There I was with two C disks plus one operating system disk. Now, my 4040 disk drive has only two slots and, any way you figure it, three disks can't fit into two slots! After some reading and a lot of frustration, I noticed that there were two versions of the C compiler. The one you use depends on which version of OS-9 you are running. I could put aside the disk for Level 2 OS-9 systems, and use the one with the program **cc1**. Now I was down to two disks and two drive slots. But which goes where?

OS-9 was meant for very large disk-based systems. A single Commodore 173K diskette can have a very long directory if the individual programs are short. You can imagine how long the directory would be if the disk could hold 10 megabytes. To get around this problem, Unix-like systems create a tree structure of directories and subdirectories. Each directory or subdirectory can contain files or subdirectories. This enables you to organize the contents of your disk so that, for example, all the files related to one job are in the same directory. This system makes makes it much simpler to deal with crowded disks.

One of these directories is called **cmds**, and this is where the OS-9 system goes to find out how to perform any of the commands you give it. Well, each of the two disks, the C compiler and the operating system disk, had a **cmds** directory. With a flash of insight, I figured that when using the compiler, I would not need the OS/9 disk, since the compiler disk had its own **cmds** directory. Thus, the compiler disk goes into drive 0.

The problem of where to stuff these disks required only three days to solve. (Nearing the third day, my guesses as to where to stuff them were becoming increasingly imaginative.) The next problem was to determine where to place the C program I wanted to compile. Since I didn't need drive 1 for anything else, I decided to create a program and store it there.

When using the tree structure of directories, the directory in which you are located is called your working or data directory. You move from one (sub)directory to another with the command **chd xxxx**, where 'xxxx' is the name of the directory you wish to enter. If the directory is many layers down in this tree structure, you can specify the complete path, starting with the drive number. For example, you may wish to go from a directory on drive 0 to one called **sam** on drive 1. You would use the command:

```
chd /d1/school/chemistry/sam.
```

In addition to the working directory, there is another directory called the execution directory. This is the directory you tell the operating system to search to find out what a given command means. When you first power up, this execution directory is automatically set as the **cmds** directory on drive 0. Now, wouldn't you think that placing the compiler disk in drive 0 with a **cmds** directory on it, would enable the system to find the commands. No way, Jose! You've no idea how I have come to hate the message **error #216**.

Eventually I realized that my normally intelligent machine might not be so gifted after the radical brain surgery I had performed, and I decided to tell it to change its execution directory to **cmds** on drive 0, by using the command **chx /d0/cmds**. It worked! Once you have changed the disk in drive 0, OS-9 apparently can't find the new one until you tell it where to look.

The time had come: I moved to the directory called **c.prgs** in drive 1 that contained my C program (with **chd /d1/c.prgs**). The compiler was in drive 0, so I used **chx /d0/cmds** to inform the operating system where the the commands were to be found. I then gave the command **cc1 test.c** to start compiling my program. The disk drive started to whir, and a message appeared indicating that the compiler had started. Slowly, other messages appeared on the screen as various parts of the compilation process were completed. Finally the last step was under way as the link message appeared.

This compiling process was *slow* — ten minutes or so — but it was working! Then, suddenly, a new message: **linker fatal... unable to produce output file... error #004.** I quickly grabbed my list of error messages, only to find that there *was* no error #004... I had had better moments in my life.

C source programs always end with the suffix **.c**. The compiled program has the same, but without the suffix. Looking around, I noticed a program called **test** in the **cmds** directory on drive 0, but there was nothing in it. For the next few days, I tried every thing I could think of, and the only thing I noticed was that the computer was trying to put the final compiled program in the **cmds** directory on drive 0, rather than in the directory containing the original program on drive 1.

Eventually, after several calls to TPUG, I reached Gerry Gold, who suggested I come out to a SuperPET meeting. Reluctantly admitting defeat, I made the journey.

At the meeting, Avy Moise told me that the compiler disk was full and that there was no room on it for the output file, hence the error message. The secret is to redirect the final output from its normal default destination of **/d0/cmds** to drive 1 (in a directory called **c.prgs**, in this case) with the command:

```
cc1 test.c -f=/d1/c.prgs/test
```

The gods smiled on me: the compile worked. I had written and compiled my first C program on my own computer, and it took less than six months.

At the SuperPet meeting, someone suggested a way to speed up the process by creating and using a ramdisk. In many computers, you can tell the computer that a portion of its RAM (random access memory) is a disk, which can be formatted and used just like any other disk. When you use the ramdisk, the data transfer is internal to the computer, and so is much faster. In the course of compiling a C program, many temporary files are created as the compiler gradually changes your source code into machine language. If it could write these files internally on a ramdisk, the compiling process would be much faster.

To create the ramdisk, you first ask for a directory of the ramdisk with the command **dir /dram**. This produces an error message, since the disk doesn't yet exist. You then format the ramdisk with **format /dram**. This prints some data on the screen and asks a question. Answer "y", and when it asks for the name of the disk, you simply give any name that you might give for any other disk.

At this point, I moved to the directory containing my C program, and copied the program to the ramdisk. I then used **chd /dram** to move into the ramdisk as my working directory and gave the command to compile the program. This time the compiling process went much faster, requiring only two or three minutes. I directed the final output back to the **c.prgs** directory in drive 1.

It has been a long and frustrating trip, but I try to tell myself that it was just one of life's little tests to allow me to prove once again that people can be the masters of their machines — if they are not driven insane first. □

OS/9 Software NOW AVAILABLE!

Word Processor — Stylograph III

Mail merge, Spell checker. Menu driven, fast. Extensive features. Formats to screen as you work .... US $149 Cdn $210

C Compiler

A full implementation of K & R standard C, with double precision variables. .... US $138 Cdn $170

Also Pascal Compiler .... US $160 Cdn $225

BASIC09

A fully structured, sophisticated semi-compiled (I code) BASIC. See Byte magazine April 1984 .... US $110 Cdn $155

DYNACALC

A full featured spreadsheet .... US $99 Cdn $139

DynaStar

A full screen (similar to microEDIT) editor, text formatter and spell checker. .... US $90 Cdn $127

Also available DYNAFORM (Text Formatter), DYNASPELL .... US $110 Cdn $150

To Order: Send a cheque or money order (add $10 for shipping/handling and 7% PST where applicable) to:

TPUG,
101 Duncan Mill Rd., Suite G-7,
Don Mills, Ontario, Canada M3B 1Z3

Order Now!
— price increases August 1, 1986