An operating non-system

G. Yuval
Department of Computer Science
Carnegie-Mellon University
Pittsburg, PA 15213

Most multi-user operating systems are designed around the following

Syllogism a:

(1) the customer will make mistakes;
(2) the customer can write and run machine language;

Therefore

(3) we need a system which no machine language instruction, executed by a user, can crash.

The result of this syllogism is the need for a very fancy 'machine language', usually implemented by interpreting every instruction on an underlying micro-machine.

Now the only real reason for premise (2) is the alleged great efficiency of machine code, and any such efficiency is liable to get lost in the interpretation process.

We might try to design an operating system around a different

Syllogism b:

(1) the customer will make mistakes;
(2) mistakes at the machine-language level are expensive to protect against, and even more expensive not to;

Therefore

(3) the customer must not get down to machine language.

If we define "machine language' to be the micro-language executed by the real hardware, most computers run this way -- a fact everybody tries to ignore.

If we dethrone syllogism a, and try to design around syllogism b, we get the following system:

(1) TC -- 'the compiler' -- a safe compiler for a safe language, that will write its signature on every object-file it produces.

(2) TL -- 'the loader' -- which will only load files signed by the 'the compiler' TC.

(3) L -- an I/O library, for the compiled code, that does NOT enable its user to forge signatures.

(4) CSI -- a command string interpreter that can invoke TL and TC.

and nothing else.

A simple way to implement a 'signature' is to make the first and last 50 characters of a file invisible to anyone but 'the loader', and to make the I/O routines write them and skip over them automatically.

If we put (1)-(4) on a disk, and start up CSI manually, we seem to have all the protection traditionally expected from an operating system, without the traditional cost of making everything slow and expensive at run time.

The language TC accepts does not have to be a particularly nice one; if some users don't like it, they can get a compiler written to translate their own favorite language into TC's language (possible by changing DO; into BEGIN); TC's language only has to be very safe; in particular, it must be impossible to write a loader in it, because this would enable us to circumvent TL.

It is sometimes said (e.g. L. Smith, "architectures for secure operating system", MITRE, 1975, AD-A 009 221) that, before we can trust the compiler that far, we have to prove its correctness. This is not so: we only have to prove that it generates safe code. In most compilers, the actual code generation is done by a few fairly small procedures, and we therefore only have to prove some simple properties of small chunks of code. The compiler can still generate (e.g.) divide instructions to perform an add, but as non-operating system programmers, we couldn't care less - it is only hurting those who use it to add.