TEN COMMANDMENTS OF THE MOSES PROJECT

Dan Pezely
March 1991

I. Everything will change.

This is the golden rule. The foundation system must be modifiable at run-time by the user. Therefore, all internal system variables, temporary variables, function pointers, file descriptor tables, space-search order, etc, must all be accessible to the user. Make use of genuine dynamic linkers to replace built-in routines so upgrades may be performed at run-time without having to restart the kernel thus maintaining memory references, etc.

II. Allow anything, but more importantly, do not disallow anything.

As we take the next step toward a true virtual reality system, not enough research has been done to fully understand its potential. The difficulty is predicting what future users may want to do. Therefore, do not force anything upon the user, and do not restrict anything unnecessarily either.

III. When in doubt, put the feature in a user-accessible location.

This follows from the first two guidelines but is worth stating even if obvious. There is no reason to hide features from the user, and security is not a reason at this level.

IV. Build upon existing platforms.

We are not setting out to develop the ultimate system to end all systems, but rather, we are out to develop a platform which researchers and developers can be both functional and comfortable with. If our system is designed to work with other platforms, users will be able to exploit that and use platforms which they are comfortable with and might have an immediate application for.

V. We are in this for the long-haul.

Virtual environments for the masses is a nice idea but an unrealistic design goal at this time. More will be said when performance is evaluated; however, hardware technology will lag behind such implementation designs.

VI. The design must last.

This is a foundation, not a single application. Any feature which might cause future compatibility problems or future implementation design compromises should be moved out of the system kernel and into the user (application) layer or be implemented as an external system service.

VII. Find the re-occurring design elements.

Recursion in design reduces complexity and ideally approaches a more abstract design. Overall, abstract designs require smaller amounts of code to implement. The less code there is, the less there is to break and upgrade. The less code there is to break or upgrade, the longer the foundation should last.

VIII. Learn from experience, both our own and those of others.

Research has already been done in many, many areas which this design can benefit from. Make use of that research and experience. For our own research and to gain experience, any code developed in the design stage should be considered disposable code. Of course, never discard code, but do not be afraid to put it aside to try a new path. Much of this design seems to be combining existing technologies in a new way. That is true, so take advantage of that and see what the results are.

IX. The source code *is* the implementation document.

The technical manual is the system source code. Books and papers are being written, but slightly more than fifty-percent of the source code files are comments explaining the inner workings. Documents, such as this one, are to educate client programmers with the background which the source code comments assume.

X. Provide uniform access to all resources.

Function, storage, and communication are the three basic elements available to all entities using the system. Communication may be seen as one specific function but is mentioned separately for emphasis. All resources, actual or virtual, which the native operating system provides should be accessible through a system-independent interface. Typical operating systems provide only a hardware-independent interface which varies from platform to platform.