DRAFT II
5/16/72

Properties of the Multics Environment*

by J. H. Saltzer

The topic that I'd like to move to is some interesting properties of the Multics environment. One of the most useful aspects of Multics is that it provides a base on which one can construct or operate higher level subsystems. We shall call the base that is provided the Multics environment.

The chief property that this environment has is that the subsystem writer finds that several of the problems he is normally faced with have already been solved in a way that he can depend on. In other words, he can devote his attention to working on his particular problem and he doesn't really have to worry about things such as input and output, storage management, device independent multiprogramming, and various issues of this type. This Multics environment is fairly general, and at the same time fairly efficient, which is what makes it valuable. The key here is that by providing an environment that solves some of these problems effectively, we hope that we can give the subsystem designer better support for his subsystem than he would have time or resources to provide for himself. The point is that when one is dealing with a subsystem that is trying to solve some fairly significant problem, he can't afford to spend much time providing the support machinery underneath the subsystem. Although in principle, he might be able to develop an underpinning, that was exactly right for his system, it may be actually very difficult for him to spend the time. In this case we can provide him with some machinery which, while

---

* from a talk given in January, 1971, at an M.I.T. Symposium on the Multics System.

maybe not exactly the same interface that he would have designed, it's probably better overall than what he would have wound up with. The kinds of subsystems that are of special interest from this point of view are those for which there is an online information base which may be looked at and updated by a variety of different people, possibly at the same time. Some examples are department administrative systems or library catalogue systems, simulation and modeling systems, management game systems, airline reservation systems, inventory and billing systems, airline equipment location systems and so on. All of these require an online remotely accessible, shared data base. Clearly, if one does not have sophisticated support, then one can still construct such systems. Most of these applications I have just described do use computer today, but often with much less than optimal results, because of the difficulty of programming them with a primitive base.

Let's look to see then what this environment of Multics consists of. There are a half a dozen key aspects which seem to make a difference when one is trying to develop a subsystem. The first key aspect is that the system, from the point of view of the subsystem constructor, is configuration independent. He does not concern himself with how many central processors happen to be attached; he doesn't really have to know how many thousands of words of core memory are attached to the system; the particular disk and drum configuration are of no concern; even I/O device types, if he wishes, can be masked off from his area of concern. The value of this independence is that one can construct a subsystem which simply does not have to change when one acquires a faster central processor, or more memory, or one discovers that there is a cheaper typewriter console available from another manufacturer.

In all of these cases, it is necessary to make some changes in Multics, but then all of the subsystems which are built on it can use the new facility or configuration.

The second key aspect is that Input/Output operations are not necessarily for online storage. Instead, one can write programs that directly address all of the online storage of the system as though it is located in the virtual memory of the program. The details of the meaning of this will come out later, but first we should notice the value to the programmer. Storage management is being automatically managed behind the scenes. The programmer simply does not perform storage management or address allocation while he constructs his subsystem thus he has more time to spend on his own problem.

A typical example arises if you ask a team which is constructing a modern compiler, for example for PL/I, on a typical non-Multics machine. You will find that they have spent much time organizing ten or twelve overlays and making absolutely sure that the right piece of information is in the right place at the right time. Upon examining the overall effort, one may conclude that the complexity of the compiler may have doubled because of the explicitly programmed overlays. A factor of two in complexity, of course, may be the trigger for slipped schedules, poor performance, and an unmaintainable piece of software. The key here is that if we can get that half which is storage management out of the way by using an efficient system-provided alternative then the speed with which the superstructure can be developed will increase, and more ambitious superstructures become feasible.