M0125

April 2, 1971

COMPUTER SYSTEMS: FUTURE RESEARCH DIRECTIONS

by Jerome H, Saltzer*

I, The Methods of Information System Research

The following question is frequently asked, and I think it's a fairly good one: "What is a university doing in the business of constructing operating systems?" My answer to this question is very simple: We are doing research. I'd like to try in the next few minutes to shed some light on that answer because it takes a bit of reasoning to get there. What does it mean to do research on computer systems?

Clearly, one can hypothesize a system and construct on paper a design. One can even model the design and build simulations of it and explore the results of the simulation. But the one problem you run into, especially if the proposed system has features which are distinctly different than other systems you have experience with, is that you have no way to model the way that users are going to load it. Since there is no way to model the load, there is a very important missing piece of the simulation. In fact, it often means that simulation is a complete waste of time. To the extent that you know what the load looks like, you can simulate, and get results limited only by your ingenuity. To the extent that you don't have any way to predict what the load will look like, you are stuck. The conclusion, if you accept this premise, is that in order to do research on computer systems, you've got to have real users. You are going to have to do something to find out how real users react to your proposals; you can't have real users unless you build the system.

* Massachusetts Institute of Technology, Department of Electrical Engineering and Project MAC - This paper is an edited version of a talk given at a symposium on the Multics System. January 21-22, 1971.

This leads to the general technique that the computer system research group of Project MAC has followed in developing two major systems at MIT, the Compatible Time Sharing System and the Multics system. First, propose a system that has certain properties; then construct a prototype of the system. As you construct the prototype, a lot of engineering tradeoffs come to light; frequently these tradeoffs by themselves give much insight into the real nature of the problem you are trying to solve. After you construct the prototype, you then try it out on a live user community. The strategy is to observe the usability of the system under live, realistic conditions; that is, first get the users to adapt to it, and-then begin measuring the way they load, so you can go back and ask the question about how does a new system idea work. The next step is to iterate: redo ideas that didn't work out very well; find a more understandable interface chat users can accept; discover a more effective way of doing something so that users won't be afraid of some features, and so on.

This iteration, going around the loop of redesign and retry, while trying to discover simplification and underlying structures, represents the real research effort: you are learning something about the way systems should be organized. But inherent in this iteration is a system development phase during which you actually construct something; therefore there is much opportunity for confusion of the goals.

Of course, there are some limitations in this approach; you can't be guaranteed that everything is going to work out perfectly when you engineer a live system. For one thing, you can't propose to make changes that are so radical that no user is willing to try the result, since the whole point of building the system was to get users. So this does limit the kinds of things you can propose doing. If helps a tremendous amount if you can identify, as we have at MIT, a number of users who have run up against the stops on some other system. At least some of these users may be willing to take some risk if they think that your system has a reasonable chance of removing a barrier that's in their way.

Another obvious limitation to this approach is that it forces you to work only with strategies which are economically viable. Otherwise, your guinea pigs are either going to have to be subsidized or else they're not going to help. If your system doesn't offer features that are good enough to overcome a cost disadvantage, then it cannot attract the users that you need to test it. This is a limitation because there are ideas you would like to try which may not be economically viable today, but will be in five years -- technology that you definitely see in the lab, but just hasn't made it out to the production line. On the other hand, it does tend to assure some contact with reality. It keeps you from producing something which you think will be economically viable in five years, but which in fact will still be impractical by an order of magnitude then.

Another limitation is a need to control the rate of change. That is, the iteration process in which you go back and make fixes, and understand what's wrong, results in a desire to change the system to provide a better interface. Unfortunately, while you may be providing the user with a better interface he has adapted to the old one already, and his patience and willingness to readapt are probably limited. In other words, if you make experiments that destroy user confidence, if you make changes so rapidly that users get unhappy, you may make great local progress, while losing the global battle to attract and keep a test user load. The reason you built the system to start with was to attract users.

Finally, there is the problem of insuring that the user load which you attract is representative of a very broad class of information system user. There are no simple answers to this limitation. We are fortunate again, at MIT, in having a very broad scope of computer use, from interactive one-shot student problems to complex scientific evaluations, and to social science and administrative data base manipulation. Also fortunate is an administrative structure which permits most MIT users to choose their source of computation based on cost and technical performance for their own job. Thus, for our particular environment, we have an opportunity to attract a very broad user community.

II. The Objective of Information System Research

I have tried to capture in a few words the research objective of the computer systems research group at Project MAC as follows: we hope to turn the fabrication of large scale information processing systems into routine engineering development projects. Of course, this objective requires learning much more about the conceptual understructure of such systems. That is a much broader objective than merely inventing ingeneous techniques of fabrication. The problem is that today, systems of the size of Multics, whether they*re implemented at a university, in the Department of Defense, or at a computer company, invariably are a back breaking effort of high powered specialists. You could ask, "Well, what's wrong with doing it with high powered specialists?" and the answer of course, is that there simply aren't enough of them to do all the things we*d like to do. There are literally dozens of projects waiting to be done if it were possible to do them routinely, but if you have to do them with specialists, you can not tackle very many of them. One of the other problems with specialists is that you don't often identify the high powered specialist until after the fact, and frequently once you've identified him and you ask him if he would like to build another system, his answer is "no".

A possible question is: “You've built Multics -- isn't that sufficient?" My answer there is "no". There are a host of interesting problems which will require an order of magnitude more machinery, more software, and more organization. Some examples from the daily news include the ABM support system, and the FAA air traffic control system. I think the present status of computer systems research is such that if the FAA were to announce this morning that they had just finished their new air traffic control system and were putting it into operation this afternoon, you would probably take the train home. The point is, there is no methodical way of cranking these systems out, and that's the thing we'd like to try to change. Another good example is the National Data Bank. Even though it is being debated quite a bit, in fact I don't really think anyone knows how to implement it. One can tie all the various data storing systems together and invent facilities so that if you know what to ask for and know where to look for it you can get it. But in order to carry out the operations the supporters and distrusters of the data bank are debating, it requires a technology of network and data organization that does not exist today. I would hope that the technology of control of such systems -- privacy, authorization, and so on, matures as rapidly as does the technology of organization.

The other example of a system beyond present capabilities is something which I would call the on-line company. You don't have any real examples today of an on-line company, in which the full potential of having information storage on-line, updated by everyone in the course of his job, and used with real effectiveness by everyone from the inventory clerk to the president. You don't find that capability because you can't depend on systems, you can't automatically crank them out, and you can't routinely modify them fast enough to keep up with the needs of the company. Until we have moved down the path towards the routine project, these kinds of systems will remain beyond reach.

III <u>Some Topics of Information System Research</u>

With that overview, it is appropriate now to move on to some specific examples of research topics. I've selected a few examples of areas which seem to offer some hope of making progress: large files, memory models, networks, mutually suspicious programs, and changes of scale.

The topic of large files I might rename to <u>really</u> large files. The point is that today Multics -- and other on-line systems as well -- has mechanisms that handle very smoothly 10^{10} bits of on-line storage. By way of comparison, that is enough storage to maintain the programs and data of 1000 scientific programmers; or the inventory of a Jordan Marsh size department store. Unfortunately, there are a lot of interesting= problems which need 10^{12} bits -- that's 100 times as much. One of the reasons you can't build a national data bank is that no one knows how to keep 10^{12} bits organized. The problem is <u>not</u> the technology of getting 10^{12} bits of storage in one place. Several hardware designs of 10^{12} bit memories have been proposed and some have been implemented. The issue really is the organization of all that memory and the effective use of it by the computer system. After the national data bank come such examples as personnel and logistics files of the Defense Department, the on-line Internal Revenue Service (perish the thought), the customer account files of the Bell System or any other large corporation, the larger insurance companies, the library automation services, and the needs of the intelligence community. The Lawrence Radiation Laboratory at Livermore has a 10^{12} bit file which stores results of bubble chamber scanning, among other things, but it's used in very limited ways.

What's the problem here? Why doesn't memory usage just scale up? Multics certainly doesn't scale up, and it is fruitful to see why not. One of the things we found necessary to provide absolute storage reliability in Multics is to copy, once a week, all of the on-line information storage onto tape. We do it on Sunday night because it takes a few hours. The trouble is that if we had 100 times as much, we would find that it took two weeks to make the copy. Which means that clearly the technique doesn't scale up well.

What is happening is that there seem to be some real issues in getting performance, reliability, and cross reference all in one scaleable design. When you've got a file of even 10^{10} bits, it is certain to have structure. It isn't just a collection of 10^{10} bits that some guy handed you, and said "Here, remember these and I'll come back and get them later." There is some structure inside, and the structure implies that some of those bits are going to have to be names of other objects in the file. In other words, there has to be cross reference from one address to another in the file.

Now, the best kind of addressing for performance is the absolute physical addresses. On the other hand, if you begin using absolute physical addresses, multiple copies intended for reliability are difficult to work in. Worse, if you have a single small area of the memory system fail, you want to move that information to some other part of the system, and continue operation. But now its physical address has changed, and there are all those references to the old address. The textbook approach to this problem is to introduce maps, but for a 10^{12} bit memory the maps get to be very large and the extra mapping references begin to eat into the performance. Thus, it is the combination of objectives which so far has limited our ability to construct really large quantities of memory which is reliable, randomly-accessible, and effective in performance.

A second exciting research topic is memory modeling. For analysis as well as synthesis we need good models of the way users load the system. The difficulty here is simply that there are many different kinds of devices with a variety of properties and users do various kinds of different things, but there is no good way to predict what kind of performance is going to result. In other words, you can't say "Here's what my users are doing, would you synthesize for me a storage system that will give me the following performance?" The synthesis ability isn't there. We have a lot of results on apparently independent topics. You can look in the research journals and pick out papers telling you what happens if you have interleaved memory, and what happens if you use a cache memory on certain kinds of loads, and you find another result that tells you what is the effect of disk arms moving in certain ways for certain assumptions. Other papers report on demand paged virtual memories. I'm very much reminded of the situation that existed some years ago in filter theory in which one could look in one place and find a solution for π-type filters and elsewhere in another paper, a solution for T-type filters but no-one as yet had seen the under-structure that allows a common view of the situation: the graph theory, linear differential equations, and complex variable theory which allow most such problems to be solved by inspection.

Such a coherent view of memory, device properties, user loads, and sharing simply doesn't exist, but it is essential for progress, and likely to be developed soon. Another form of this question comes up in the commonly debated topic of space-time tradeoffs. Most algorithms have alternative implementations. Frequently one implementation is faster, but takes more memory space, while another is slower but more compact. Actually, there's a third dimension added in a big system with a variety of memory devices. Just because you use more space it doesn't necessarily mean that you use more of the fastest memory. The traditional example of tradeoff between running time and space is the use of hash coding as a way of storing things in tables. The more densely packed you make a hash table, the slower the lookup operation. This effect was first exploited long ago and is now the basis for the symbol tables in most language translators. One can compute formulas for this particular space-time tradeoff, but when you put a hash table into a virtual memory, something new happens. The densely packed table, which requires multiple lookups, may require that the entire table be in the highest performance memory. On the other hand, if you use a longer, less dense cable, it takes up more virtual memory, but you may be able to look things up in one or two references and thus use only a small portion of the highest performance memory. Thus a new dimension has come into this tradeoff which really is going to require some exploration to understand.

My third example of a research area is that of networks -- information being shared at a distance. If one has two processors which aren't located at exactly the same point in space you develop a problem, because when the two processors try to share information they begin making copies of it, and copies arc deadly things in a system with sharing. Copies lead to an updating problem, and the question "which copy is the 'correct' one?" This problem comes up when one processor is in Los Angeles and the other is in Boston, where the separations are measured in tens and hundreds of milliseconds. But it also comes up if you have two nanosecond processors in the same machine room which are separated by ten feet. Ten feet means ten nanoseconds, or ten instruction times and to keep the instruction rate up there will have to be per-cpu copies of information in process.

I'm not sure that there are fundamental new issues exposed by a network that aren't already in ordinary operating systems, but the net-work places a premium on certain issues. For example, the problems of protocol, and communication discipline, and accounting cannot be finessed as they often can in a single operating system. Of course, the problem of load modeling comes up if you try to ask how big the channels have to be between the various pieces of the network. The prediction of bandwidth requirements is a non-trivial thing which requires knowing how users will load the network. Again we have the problem that I mentioned at the very beginning -- until you build one you don't know. Even though you see some obvious uses, you are not quite sure exactly what proportions they are going to take.

On the topic of networks, there is some research going on in Japan on the problem of putting together a multi-processor nanosecond computer which has local memory in each of the processors. They are asking questions about traffic requirements. If one of the processors writes something in its local memory, all the others which potentially are sharing that item have got to be told about it, which means that cross traffic is going on among the processors. They are trying to predict the amount of that cross traffic, and coming up with alarming results. The point is that there are things here which require much modeling and that are not yet understood.

There are a variety of other research topics I will just briefly touch on. Probably the one topic which has the biggest immediate academic payoff is what I would term "simplification of mechanism." As we look at a piece of our present system, we frequently discover a way with half the code to do all the same things, do some of them better, and add a couple of other features. That can only happen when one has acquired additional insight into what it was he was doing. There are several examples in Multics experience in which we have rewritten modules two or three or four times and each time gained a factor of two or four or sometimes ten in operating effectiveness. You just don't get factors of ten by noting that a test can be done a little faster in a different order. You get factors of ten by understanding the strategy better. And it's the understanding of strategy which goes into the simplification of mechanism that we think is one of the chief results we can hope to produce.

Another good research topic is one I would name "implementation of protection between mutually suspicious programs." Multics contains a protection mechanism based on concentric rings of greater and greater authority. This seems to be an adequate technique for a class of problems in which program A is suspicious of program B, but B is willing to trust A. For example, the instructor doesn't trust the student, but the student trusts the instructor. That kind of arrangement implements very smoothly with rings of protection, since they are purely hierarchical, totally ordered. But as soon as you get to a mutually suspicious case, you begin to run into some problems.

A good example of the mutually suspicious situation is the following: suppose we have an entrepreneur who has just written a brand new PL/I compiler he claims is better than any other PL/I compiler anywhere and he wants to market it using system facilities to protect his interests. He wants to make sure you can't make a copy of it when you run it, and that he can get a record of the fact that you used it. In other words, he is suspicious of his users. So far, so good: the ring mechanism of Multics works beautifully. But you begin to get into trouble if a customer of the entrepreneur is constructing a brand new linear programming program, written in PL/I which he considers proprietary. Now, the customer begins to worry that the private PL/I compiler may steal a copy of his LP program while compiling it. Now you have a mutually suspicious situation. I want to use your program but I don't want you to see the data I'm allowing your program to massage. That's the situation that no one knows quite how to handle, and yet there are certainly some applications that are quite interesting.

Still another, and today my last, example of a research topic is "what do you learn when you scale a system up and down in size?" Probably the most immediate reason for building Multics is that its predecessor system, CTSS on the 7094, did not scale up or down at all. Both hardware and software were "special cased" for the size of the problem being solved there. In changing the scale, you discover bottlenecks and tradeoffs you hadn't realized were there, so it becomes a very fruitful exercise in understanding. A good example of a particular scaling problem is what happens if you attach 25 processors. When you have more than one processor you run into interference problems: two of the processors may happen to reach for the same piece of data at the same time, so one has to wait for the other. Unfortunately, we don't have a model of the data sharing. When you put on 25 processors, is 3 percent of their time lost in interference? or 90 percent?

In closing, I'd like to review again the overall research objective in this area: we hope to turn the fabrication of large scale information processing systems into routine engineering development projects. I hope that my comments this afternoon have given you some insight into what this objective means and why I consider it important.