THE TRANSITION TO ON-LINE COMPUTING: PROBLEMS AND SOLUTIONS

edited by Fred Gruenberger

1967
THOMPSON BOOK COMPANY
Washington, D.C.

ACADEMIC PRESS
London

an informatics inc, publication

Sole agent for the entire world (except North and South America):

ACADEMIC PRESS
Berkeley Square House
Berkeley Square
London, W.1

Library of Congress Catalog Card No. 67-30073

Printed in the United States of America

WHY THIS BOOK?

DR. WALTER F. BAUER

Many believe that the development of on-line systems is part of a continuous evolution of data processing technology. Others believe that on-line computing represents a technological revolution—a singular event in the history of data processing. It is probably neither. Some may argue that the first maintenance console with its blinking lights on the computer in the early 1950's was the beginning of the on-line display. There is certainly merit to this argument. Still others may argue that, on the other hand, the sudden increase in usage and interest of this new concept constitutes more of a revolution than any other event in the development of data processing.

There is hardly any question, however, that the development of on-line systems constitutes a quantum jump in the evolving technology, if only from a standpoint of the interest which the data processing community has shown in it, for to a great extent, it is people and what they think which greatly influences the course of these events. Another apparently unimpeachable view is that on-line systems technology will be the vehicle, or at least the catalyst, to catapult the data processing industry to a level of $20 billion annual sales in the next ten years. On-line systems bring the computer to the user. This will have a profound effect in the proliferation of computer use.

But aside from the business and historical factors, on-line systems bring to the picture new problems and new factors in the design and implementation of computer systems. Because of the many new technical factors, and because there is an immediacy to the need for developing knowledge about these systems, there is in turn, a need for a book of this type. This is basically why we, at Informatics, decided to publish this book. We believe that the book can provide a much needed, useful tool to assist in disseminating knowledge about these new systems. To be sure, not all questions concerning on-line systems are answered by the papers published herein. The papers, however, cover many areas and represent a kind of snapshot of the state of this modern technology.

We, at Informatics, prefer the title "on-line" to "real-time" or "time-sharing." Real-time, we believe, is an outdated term whereas time-sharing is narrow and reflects more the computer's status than the status of the user who is *on-line* to the computer in these new systems.

This then, is a collection of papers written by Informatics technical staff members. Although the papers were written over a one or two year period of time, they were brought to focus by a symposium which was sponsored by Informatics in conjunction with UCLA in 1965, and by a series of seminars, "On-Line Computing Methodology," which were given by Informatics in 1966. The papers are written by working experts; not by observers of the technical scene and remote from it. We believe it is especially appropriate that Informatics publish this book since, from the beginning of our business activities, we have been highly oriented toward these systems, and an overwhelming percentage of the activity of the company is thus directed.

It has been estimated that, by the year 1975, nearly all computers in this country will be "on-line" in the sense of being part of an electronics system, or being attached to instrumentation such as communications and display devices. It is our hope that this book makes a contribution to the technical knowledge required.

Contents

WHAT IS ON-LINE?

FRED GRUENBERGER

THIS BOOK deals with on-line use of digital computers. On-line is not a fixed point in the world of computing; it is a continuum; you can have more or less of it. It deals with the capability of a computer system to react to external demands; the two principal variables that we deal with are time and dollars.

Figure 1 shows, in gross form, the relation of these two variables. To perform any task with a computer, some programming and systems work must be done; this is the threshold. If, now, the task is processed in the traditional batch mode, one has what might be considered the antithesis of on-line; that is, the computer system is highly unresponsive to its external demands. The situation might be characterized by the typical use of an IBM 704 in 1957—for each task, the user had control of the entire machine, and each task had its own cycle of setup, production, recovery, and degradation.

The continuum of on-line represents all attempts, by whatever means, to make the system more efficient in terms of work performed and in responsiveness. A tiny first step in that direction is the continuous flow processing scheme involving a monitor and a direct coupled computer system. Such a system isn't responsive in the sense usually used in on-line work, but it does tend to increase efficiency and it can (I'm picking my words carefully) reduce the turnaround time significantly.

For most applications, that's the point: We seek to cut the response time of the machine, and we are willing to pay something to do it. At one time it was common to suggest cutting the response time to the ultimate, for which the term "real time" was used. That term is not much used any more. It implied a response time of nearly zero; that is, the actions of the computer were in synchronization with actual events. Such a situation is necessary, for example, in controlling the path of a satellite as it is launched. Actually, the computer and the event must be out of step by some small increment of time; computers are fast, but not infinitely so. Figure 1 suggests that the nearer one approaches to real time, the greater the cost, to the extent that true real-time response is reserved to the military or to some agency with extraordinary demands and a budget to match. (It has been pointed out that for some applications, such as the sale of a seat on an airplane, it can be considered that the event being controlled actually takes place inside the computer, in some real sense. In such a case, one has, indeed, a real-time situation.)

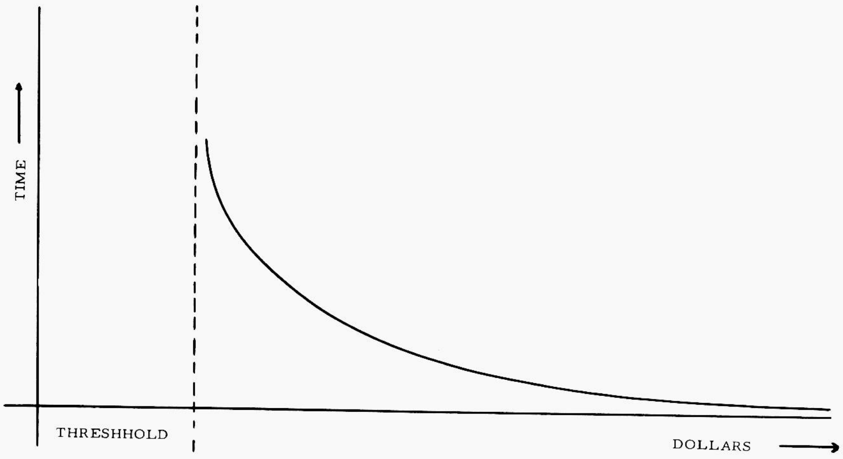

Figure 1. THE COST OF RESPONSIVENESS FOR AN ON-LINE SYSTEM.

But on-line usually demands more than simply a quick response. To be interactive to the requirements of humans or other machines, an on-line system must be able to accommodate interrupts whose arrival in time is unpredictable. The lowering of the response time will increase the costs; the additional constraints will also introduce new complexities into the system. The hardware and software problems that arise are the subjects of later chapters.

THE NATURE OF ON-LINE PROCESSING

The swing to on-line use of computers (and it is reasonable, in some sense, to predict that the great bulk of all computing work will be on-line within a few years) has already had one beneficial effect; namely, that it has forced system designers to think in terms of the users for a change, and to human engineer their product.

Let me illustrate what I mean by describing something that is common to nearly every traditional batch processing system. Suppose, in such an atmosphere, you prepare a short program in assembly language and submit it to the system. You may get back a symbolic listing, for a 10-line code, of a dozen pages, some of which bear useful, readable information, such as your original code, its binary equivalent (the machine language code), a symbol table, an octal core dump, and the like. Your job card (in a monitor system) will be repeated back to you; you may have some error messages; and there will be an accounting summary to show how you used the machine's time. You will also get, in all likelihood, a lot of printed information that, to you, is complete gibberish, usually marked by dollar signs or other strange symbols. Your error messages may be of the form "Error 342 on line 12," or, worse, "Error in phase 2," and no one in your shop can tell you what those messages mean. There are systems in operation that furnish error messages of this type on every run, including those that are in apple-pie order. Now, if you show such a listing to the in-house system programmers, you will be told the following, in order:

1. Don't worry about it.
2. We always get that.
3. We don't know what it means either.
4. We don't know how to get rid of it.

You are told, in other words, to live with this nonsense and not to make a fuss over it.

Generally, on-line work, particularly when it involves graphic displays, can't tolerate this sloppiness. If nothing else, every extraneous piece of information represents a waste of time, and time is just what on-line systems try to conserve. But, more importantly, the useless information is more glaringly apparent in an on-line system, and the designers are motivated to engineer it out. The output of on-line systems tends to get nice and clean and more useful to the user.

WHAT ON-LINE IS

If it is not entirely clear what on-line use of a computer is, it must be even less clear what new problems it generates and to what extent our industry has been able to live with and handle those problems. Clear-cut lines between hardware and software tend to blur, and cost analysis methods of where to put the dollars for an optimum solution are still lacking. As always, the pioneers have paid a large price to find out, the hard way, a few answers to guide those who follow.

Some of the flavor (and jargon) of on-line can be gained by a homely analogy. Imagine that you are attending a series of talks about on-line work. If each speaker had his material carefully prepared and delivered it as prepared, we would have a situation analogous to *batch processing* (and the talk could be given via a videotape player). But a good talk is given in an on-line atmosphere; there is interaction between the speaker and his audience. The speaker may think of something and comment on it, which is akin to *real time*. He may have *interrupts* and may, in fact, trigger them by *polling* the audience. It is not uncommon to experience an interrupt during an interrupt, leading to *second level* and third level interrupts and requiring a *scheduling algorithm* and *priorities*. (The analogy is quite good here. Do you answer questions in the order in which they're asked, or in some priority order, or according to a scheme that disposes of the trivial questions first?) The speaker must be able to *detect errors* in the questions and to *correct errors*. If a second expert is on hand to field some of the questions, we'd have multiprocessing or parallel processing of information. And please note: it's all *nonreproducible*.

THE SHAPE OF THE ON-LINE WORLD

On-line computing has countless aspects, with every promise of more to come. It includes conversational computing as a subset, and that rich field called computer graphics. Its chief tool is time-sharing, which is almost a field in itself. On-line work expands the area of usefulness of the computer manyfold; the new problems it brings are exciting and challenging. It may be that on-line work offers the key to the solution of its own (and other) problems through the concept of on-line programming. It would be worth considering that new area a bit.

As the programming profession raised itself from the morass of absolute octal, through symbolic assemblers, on up to compilers and problem-oriented languages, each new advance was heralded as the ultimate answer: Now *anyone* could program. As a measure of the strength of that claim, recall, for each level of language, the extent to which the programmer could devote himself to his problem, as opposed to the attention he had to devote to the language and the computer system. In the early language/computer combinations (say, the SOAP days on the 650), this ratio was around 40% for the problem and 60% for the system. More recent programming systems may have succeeded in reversing the ratio, so that the user devotes only 40% of his time to the constraints of the system, and has 60% left to use on his problem.

On-line work has two areas in which this ratio improves significantly: namely, on-line programming (in which interactive use of the computer aids the normal programming process) and conversational computing (in which the interaction between man and machine becomes almost totally directed toward problem solution, and programming as such fades into the background). It may well be that in these two areas, the ratio approaches 90:10, with the 10% devoted to the details and constraints of the system. It has been said that in conversational computing the user, for the first time, is allowed—and encouraged—to explore the solution space directly, rather than having to guess where it lies and explore large areas around it. Putting it in cruder terms, the man with the problem can drive for the number he wants, instead of demanding 10,000 numbers on either side of it.

Usually, the virtues of on-line work (and particularly time-sharing) stress the economics of the total system: the improved balance between efficiencies of man and machine—and these virtues will all be real, as we learn how to handle all the new system problems that arise. But it seems to me that the greatest virtue of on-line work is the emphasis it has placed on neat, clean interaction with human beings. It holds a promise of helping, more than any other development in the field, to fulfill Hamming's dictum: "The purpose of computing is insight, not numbers."

TIME-SHARING AND MULTIPROCESSING TERMINOLOGY

ROBERT A. COLILLA

IN THE GENERAL AREAS of time-sharing, multiprogramming, and multiprocessing, some terms in current use are very imprecise. It is hoped that as a result of this paper, some of the meanings will be clarified and, perhaps, a step gained in the direction of standardized usage.

When one thinks of time-sharing, some well-known systems immediately come to mind. Among these are MAC at MIT, the SDC Command Research Laboratory system, the ATLAS time-sharing system at Cambridge, and the RAND JOSS system. The MAC and SDC systems are similar in overall approach, but JOSS and the ATLAS system differ enough from them and from each other to merit special attention. All four systems are characterized generally by the fact that the central processor is not required to complete one job before starting another.

Three of the above systems have an on-line, perhaps real-time, character in that communication between users and machines is made through on-line consoles. Since this is significant to the notion of time-sharing, the character of on-line systems is discussed first.

In an on-line system, a man or some device is either supplying information to a computer and/or waiting for the computer to supply him or it with information. There is always the implication that time constraints are important in the sense that either communication must occur within very precise intervals or, at least, that frequently delayed responses will defeat the purpose of the system. One also speaks of peripheral devices as being on-line to a computer if there are channels connecting the peripherals to the computer. Clearly the existence of a computer with an on-line device, alone, is not sufficient to make an on-line system. If such were the case, what system would not be on-line?

The characteristic of systems that really differentiates between those that are on-line and those that are not is the extent to which the computer may regulate the rate at which it accepts input data and transmits output data, and the extent to which lack of regulatory control influences the design of the system. In general, the less regulatory control a computer has over input/output rates and the more influence this lack of control has on the system design, the more on-line a system becomes.

In particular, when a system is such that the computer has practically no control over its I/O rates and where system design is entirely oriented around ensuring that the computer is ready to receive and transmit in time, then that system is usually called real-time. Real-time systems are, therefore, those on-line systems that manifest the extreme aspects of on-lineness.

It should be clear from the preceding that neither on-line nor real-time necessarily suggests any of the salient features of time-sharing. On-line systems require neither many users nor human users to be qualified as on-line. As was stated earlier, however, some time-sharing systems do have an on-line character. Many users supply information to a computer, and it is required that a time-sharing system be designed to respond to the many users at frequent intervals. To get a better look at the term time-sharing and its companion term multiprogramming, it is good to recall some recent computer history.

TIME-SHARING AND MULTIPROGRAMMING

With the advent of asynchronous input/output operations in the late fifties, it became possible to perform simultaneous operation of a computer's central processor and its I/O processors. Programmers immediately set themselves to the task of using this new hardware capability to maximum advantage. Initially this meant better organization of individual programs. Input/output operations were strategically placed to achieve maximum use of the central processor. It did not take long, however, for the idea to be extended to that of operating two programs "simultaneously" so that one performed input/output operations while the other used the central processor. The ATLAS scheduling system carried the idea to its greatest extent. In ATLAS, all jobs are fed into the system as soon as they appear. A scheduling program selects those jobs having input/output characteristics which will tend to put as many of the computer components into motion as is possible. The objective of the ATLAS scheduling system is to "maintain the fullest possible useful activity in those parts of the computing system which can function simultaneously; that is, to reduce to a minimum periods of idleness in any part of the system which is required for further use."[1]

Since, in the implementation, the central processor is required to transfer control frequently from one program to another without necessarily waiting for any of the programs to terminate, the terms multiprogramming and parallel programming arose. The latter is rarely used anymore. Also since portions of programs are sharing the central processor sequentially in time with all the other program portions, the term time-sharing arose.

As the above techniques were being developed, people were experimenting with the idea of connecting many electric typewriters to a computer and using them as on-line I/O devices. It was recognized at the outset that if men were going to use these typewriters as on-line devices, there would be a lot of very slow input/output operations. Not only would programs be delayed because of the slowness of the transfer rates of these devices, but they would be further delayed by the users' far slower rates of typing. Time-sharing is the logical technique to employ for this situation.

Time-sharing, however, is now embedded in an on-line environment and, as in all on-line environments, there are special time constraints imposed on the design. Whereas previously it was only necessary to switch from program to program to maximize utilization of all the components of the hardware, in the on-line environment there is the additional requirement to pass control to the jobs of the different users at frequent intervals so as not to ignore any user for any substantial length of time lest he become disenchanted with the system. With this additional demand, even if a single job program were exercising the components of the computer to their greatest extent with maximum efficiency, that program would have to be suspended periodically in order to make the system available to the other users.

This additional aspect to the notion of time-sharing is quite significant. It is so significant, in fact, that the requirement to respond to all users of the system has become the major characteristic of time-sharing systems and the term has become identified with this characteristic almost to the exclusion of the previous notion—that of maximizing utilization of all the components of the hardware. The term time-sharing, therefore, is used in both of these two senses.

The meaning of multiprogramming has also changed somewhat. This change is a direct result of the changing meaning of time-sharing. Whereas previously one might have described multiprogramming as the operation of a central processor that executes a number of programs in fractured fashion for the purpose of maximizing the use of the components of hardware, it would now be described without the last motive-supplying phrase. That is, the reason for performing the fractured operation is no longer part of the description.

MULTIPROGRAMMING AND MULTIPROCESSING

As is usual in developing a concept, it is as important to say what the concept is not, as it is to say what it is. The term multiprogramming is often coupled with the term multiprocessing, with the implication that the two terms mean related but different things. It should be stated first that the processing part of multiprocessing refers to processor and not to process. Multiprocessing suggests the simultaneous operation of a number of processors. In distinction, multiprogramming is confined to the operation of a single processor. In fact, it makes much more sense to talk about a multiprogrammed processor than it does to talk about a multiprogrammed computer or system.

The notion of a processor is basic enough to multiprogramming and multiprocessing to merit special attention. Despite any realization in hardware, a processor is conceptually a device that operates serially. It is this character of a processor that gives birth to the notion of multiprogramming. Multiprogramming can be said to be that operation of a serial processor which permits the execution of a number of programs in such a way that none of the programs need be completed before another is started or continued.

As was said before, multiprocessing implies more than one processor; yet it has to imply more than just this for otherwise any group of simultaneously operating computers would qualify as a multiprocessing system. Bright[2] has added the requirement that all the processors must have a common, jointly addressable memory. It also seems necessary to require that no processor be dependent on another processor in order to operate. Another distinction can be noted here. Multiprogramming is, at least at present a software notion. Multiprocessing, on the other hand, so far is exclusively a hardware notion. In searching for a possible software meaning for multiprocessing, one finds the following: Suppose one has a single job and a number of processors. Suppose, further, that the job can be divided into a number of parts, some of which can or have to be performed simultaneously on different processors, but some of which also must wait for the completion of other parts before they can be executed. Multiprocessing can be defined to be the ability to execute this divided job successfully.

To look at this in a slightly different way, multiprogramming can be said to be the task of fitting a single serial processor for many jobs, and multiprocessing is the task of fitting many serial processors for a single job. Such a definition, however, is more than most people have in mind when speaking of multiprocessing. This software notion could be called divided job processing. It may be added that divided job processing does not really require the involvement of more than one processor to have meaning. There are people who indeed define multiprocessing as being divided job processing irrespective of the number of processors involved. At this time, however, it is perhaps best to leave multiprocessing a hardware notion and wait to see if any software notion develops.

EXTENDED NOTIONS

It was said earlier that a processor is a device that performs instructions serially. Although it was implied that the instructions of concern were machine instructions, that does not have to be the case exclusively. One may think in terms of an extended machine just as well. If, for example, one thinks in terms of FORTRAN instructions, one can define a FORTRAN processor as a device for performing FORTRAN instructions serially. One can then multiprogram this processor by fitting it for a number of FORTRAN jobs. Similarly one may think in terms of many FORTRAN processors operating simultaneously to achieve a multiprocessing system.