Future Trends in Time Sharing Systems*

Jack B. Dennis
Massachusetts Institute of Technology

Project MAC

January 13, 1969

Computation Structures Group Memo No. 36

*Edited transcription of a talk prepared for the Operation Research/ Time Sharing Symposium, National Bureau of Standards, Gaithersburg, Md., October 29-31, 1968.

Future Trends in Time Sharing Systems

Jack B. Dennis

There are three difficulties with my task. In the first place I have chosen to speculate on the future and that is frequently hazardous. A second difficulty is following the great show provided by the previous speakers. Finally I am sure everyone here would just as soon discuss the issues already raised over lunch.

What levels of capability can we expect in future Time Sharing systems? How long will it be before these levels of capability are generally available? To discuss these questions I shall indulge in a classification of Time Sharing systems. This brings us back to the question: What is Time Sharing? Two views have been prevalent around Project MAC at M.I.T. in our speculations on the ultimate objectives of time sharing. The first view is conveyed by a phrase Professor Licklider is fond of - man/computer symbiosis. This is the idea that when a person is working with a computer they are pursuing together a common intellectual goal: The computer is acting as an intelligence amplifier for the person. In this concept there is no reference to the interaction of a person with fellow intellectuals that share a common interest with him. In recognition of the dependence of creative pursuits on the base of knowledge developed by other individuals, a second view of Time Sharing has evolved. This is the thought that a Time Sharing system is serving an intellectual community of users for which the computer is the medium through which they share their knowledge. These visions of Time Sharing are still far from being realized. I hope to give you some indication of why this is so.

A more prosaic definition of Time Sharing undoubtably represents the original intent of the term. According to this defintion Time Sharing is the multiplexing of limited resources (e.g. the processor and memory of a computer) among many activities. This concept of Time Sharing has been employed in computation a very long time - so long as computer memory and processing capacity have been applied concurrently to several independent processing activities.

My classification of Time Sharing Systems is according to the distance they have logged along the road from the latter primitive meaning of Time Sharing towards the more general concept of a computer utility serving as the medium for exchange of knowledge within a community.

Those acquainted with the history of computer systems often hold that the first Time Sharing systems were what I call transaction systems. The Sage air defense system and a sophisticated airlines reservation systems such as the Sabre System originated by American Airlines, are well-known examples of transaction systems. These systems include a central computer that multiplexes its processing capacity among a large number of agents interacting with it via remote terminals. The operations performed by these systems are principally the accessir and updating of common centralized data bases relating to the mission of the system. One property of transaction systems is that all of the processing activity performed by the system is foreseen by the designers. The users transaction system are not performing individual independent activities, but are performing activities, entirely anticipated by the system designer, that collectively accomplish the mission of the system.

The techniques developed for transaction systems are directly applicable to what I call dedicated information systems. These systems provide widely distributed users remote access to data bases maintained at a central installation. A good example is a retrieval system for bibliographic information in some specialized field - medical diagnosis, for example. The hardware and programming

technologies developed for transaction systems are well known and the implementation of dedicated information systems is straightfoward.

The second level of Time Sharing system is the dedicated interactive system. Such a system makes a particular programming language, usually one of limited capability, available to users at interactive terminals. Perhaps the best known example is the Johniac Open Shop System(JOSS) originally set up at the Rand Corporation. With a dedicated interactive system, users may write, edit, test, revise and run small programs expressed in the single language implemented by the system.

The third classification is the general purpose interactive system with which users may develop and run from remote consoles programs expressed in any one of several high level languages - for example, Fortran, Basic, Algol, etc. Two of the most popular commercial Time Sharing computers support software systems that are representative of this class - the General Electric model 265 and the Scientific Data Systems model 940.

The fourth classification is the extensible systems. The differentiating characteristic of an extensible system is that a user is not restricted to the programming languages supplied as part of the system. The user may prepare files of data and/or programs and make them available to other members of the community having access to the system. Furthermore a user is able to construct from his remote terminal programs that implement a new programming language, and he is able to authorize their use by others. The best known extensible system is the Compatible Time Sharing System (CTSS) developed at M.I.T. Systems that achieve this level of capability are rare, but systems similar in many ways to CTSS have been implemented at several institutions and commercial systems of comparable compatibility are becoming available.

At present, the available system technology - hardware and software- is not able to offer to the market a Time Sharing system of greater capability than the extensible variety. Furthermore, it is not foreseeable when more advanced conceptions of Time Sharing systems will be generally available. This is unfortunate because there are important directions in which the capabilities of computer systems need to be extended. These essential directions of evolution serve to define further classifications of Time Sharing systems.

For the fifth level I again use a term from Professor Licklider: coherent systems. These are characterized by the following property: Program modules written by various authors may be used in the construction of new programs without need of knowing the internal mechanism of the modules. The easiest way to create a new program is to assemble it from already existing pieces. I like to call the ability to combine arbitrary programs into larger units programming generality. Later in this talk, I will explain in more detail the far-reaching implications of programming generality regarding the design of computer systems. While no existing systems qualify perfectly as coherent systems, a number of programming languages and computer systems have achieved unusual degrees of programming generality. An early example is the Genie system built on the Rice University Computer, in which it is possible for program modules to acquire and release storage as required in the course of a computation. Other examples are certain programming subsystems implemented under CTSS at M.I.T. for applications in civil engineering and industrial management. In these two cases, a specialized programming system was developed within which the coupling of program modules is relatively easy. Some programming languages have characteristics important to programming generality: These include the list processing language Lisp, and languages such as Landin's Iswim based on the Church lambda calculus.

When multi-access computer systems are linked through communication channels, further qualitative levels of performance may be distinguished. An information network is an interconnected set of computer installations in which data bases created at any installation are (with the owner's permission) available for use at any point in the network.

Finally, I reserve the term information utility for the ultimate evolutionary form of Time Sharing that I can presently envision. An information utility is distinguished from an information network by the ability of a user to program a transaction system within the computer utility. The achievement of this objective still seems to be ten to fifteen years in the future. I cannot give here all the arguments required to substantiate this opinion; however there is one major obstacle I wish to point out. The approach toward ensuring the integrity of information in a transaction system and in a general purpose Time Sharing system are rather different. In a general purpose system the technique used is to provide backup files which can be used to restore the system information base in the event of a failure. The backup files are updated at the end of each user's console session. In a transaction system the transaction routines are designed to complete a definite piece of processing on each activation. Hence the only record that may be inconsistent at an unscheduled shutdown is the one being processed at that instant. Obtaining this property in a multiplexed computer system requires adherence to programming conventions that effect the detailed coding of the transaction routines. This is in direct conflict with a basic assumption of general purpose Time Sharing - that the user should not have to be concerned with the problems of data integrity in designing his programs. The resolution of these two viewpoints on the integrity of the information base of a computer system is perhaps the most serious unsolved problem in computer system design.

In the future computer programming costs will increasingly dominate hardware costs. Therefore the most important avenue of progress is to develop means for reducing the human activity required to produce programs to a small fraction of what it is today. Figure 1 lists four means of reducing the cost of programming. The first two are already in wide use. There is no longer any serious argument that high level programming languages do not reduce the human effort required to specify a program. Also it is generally appreciated that on-line debugging and editing considerably lessen the work of putting a program into useful operating form.

One way to acquire a program (number 3) is to borrow it from someone else who as already made the investment of designing and checking its operation. This should be a very effective method, for presumably, it should require no programming at all. However, the problems of program compatibility between similar systems and different systems rudely interfere with success. The elimination of these difficulties is through the standardization of programming languages.

A most important potential means of creating a program is (number 4) to construct it from existing programs. Very often the exact program for an application does not exist but major components of it have been expressed as programs, very likely in different languages. Programming generality is the property of a computer system that enables users to combine existing program modules independently expressed in possibly different languages.

In current practice, a programmer expresses his program in terms of the basic primitive operations implemented by the computer installation (Figure 2). These primitives include the arithmetic and logical operations, operations for manipulating character strings, the use of subscripting to select an element of an information structure, the basic algorithms implemented by routines in the subroutine library, and operations on files. The majority of programs written

1. High level programming language
2. On-line program development
3. Interchange of programs among users
4. Use of existing programs as components in building new programs.

Figure 1. Means of reducing programming costs.

today are expressed in terms of these primitive functions with the aid of a high level programming language.

Rather, what should be commonplace is illustrated by Figure 3. Once some program modules have been expressed in terms of the primitive operations at an installation, further program modules that perform more complex functions are created by combining modules at the first level. Still other programs are constructed from these resulting in a hierarchy of arbitrary depth.

What are the requirements for a computer system to offer programming generality? It must be possible for two program modules to operate together although they are written by different programmers or in different languages. Neither program author should be required to know the internal details of the other module, its memory requirements or what other program modules it uses to perform its function. The concern of a programmer using a module should only be that it performs the desired transformation of input values into output values, and operates with adequate efficiency. Four requirements are implied by these objectives: Since the modules are independently specified, all information exchanged between any pair of modules must be passed as parameters - there can be no global information. Secondly, any module must be able to create arbitrary information structures during its operation. One module must not be required to foresee the amount of storage a second module will need to carry out a requested computation. A module used in the construction of a larger module may depend for its operation on yet other program modules. The author of a new program should not have to be aware of this dependence. Finally, it must be possible to transmit an arbitrary information structure to another module as an argument.

These characteristics are not provided by conventinal computer systems. One implication is that all storage allocation decisions made to fill a module's

need for memory must be made by the computer system: The computer hardware and operating system must manage the assignment of information within the hierarchy of memory media used in a modern computer installation. These decisions cannot be made by the program modules because each module is presumed unaware of the storage requirements of the other modules operating with it.

In the majority of present day computer systems, a program can reference information via two mechanisms. On one hand is information represented in main (core) memory, made up of simple variables, arrays and linked structures. These data are accessed directly by programs through the addressing facilities of the processing unit. The second mechanism is used for information represented as files, and is implemented by the operating system and a file directory or catalog. The use of two separate addressing mechanisms creates a severe problem in realizing programming generality because there is no rational general rule for deciding which method of access should be used for the different components of a computation. Also, in most contemporary systems it is not possible for information represented as files to be passed as an argument of a procedure call, since the mechanisms used to communicate arguments assume their values reside in main memory. These are basic arguments that programming generality requires that a computer system implement a virtual memory or address space of sufficient size to contain all files pertinent to a user's application. The Multics system under development at M.I.T. is unique in abolishing the distinction between "file" and "array" by implementing an unusually large virtual memory.

Returning to the subject of compatibility, it is important to note that the primitive operations of a computer system for file manipulation and console communication are largely determined by the operating system, and not by programming languages or their compilers. As Time Sharing becomes widely available, more and more programming systems will be implemented that depend intimately

on the file processing and console communication primitive in use at a particular installation. This is as if the job control cards became dispersed throughout a program instead of being something simply added as a prelude. Fruitful exchange of programs within the future computer utility will only be possible if these aspects of operating systems become standardized. The small effort being devoted to this question indicates we are destined to endure much frustration arising from these sources of incompatibility.

I have explained several basic problems in computer system design which must be attacked and solved along the path toward the computer utility. It is difficult to see how Time Sharing systems beyond the power of the extensible systems can come into general use within the next four or five years.