11 October 1968, Volume 162, Number 3850 SCIENCE

# Dartmouth Time-Sharing

Development of the system by a team of faculty and undergraduates is described.

John G. Kemeny and Thomas E. Kurtz

The rapid growth of computers and their uses has been the most important scientific development of the last two decades. It is now commonly accepted that computers can carry out calculations millions of times faster than human beings. It is also becoming evident that this large quantitative increase actually brings about a qualitative change in the ways we deal with information and with calculational problems. Not so well understood is the complex programming effort needed to convert a given collection of transistors, magnetic cores, and wires into a "machine" that is capable of providing services conveniently to its clients. In short, the role of "software" in the total computer system is not well understood.

The purpose of this article is to describe the development of one particular software system, the original Dartmouth time-sharing system.

Most readers will be familiar with the notion of time-sharing, the ability to provide simultaneous service to many users equipped with typewriter-like terminal devices. These users may be typing in new programs, calling programs from a "library" (the collection of programs available to all), presenting requests for processing, or watching results being typed. Many other forms of service are available whether the user is in the computation center, in his office, or 3000 miles away. Whatever these services are, provision for them must be made in the software system.

The concept of time-sharing was first realized around 1960 on a small computer, the Digital Equipment Corporation PDP-1, and was developed jointly by personnel from M.I.T. and from Bolt Beranek and Newman Inc. In the intervening years, however, the development and replication of such systems and their descendants has been much slower than hoped, primarily because their complexity had been vastly underestimated.

These delays permitted the arguments over the merits of the time-sharing concepts to persist for many years before comparisons based on actual demonstrations could take place. Nonetheless, when these demonstrations did occur, time-sharing proved to be much more successful than even its proponents had hoped and achieved important applications in unanticipated areas.

The several dozen time-sharing systems that exist or are claimed to exist today range from significant research instruments to systems of limited usefulness. Relatively few, however, have achieved a level of technical and economic success measured by the number of replicated versions of the system in normal operation. The most widely replicated of the successful systems was the Dartmouth time-sharing system (DTSS). Designed and constructed by a team consisting of the authors and a dozen undergraduate students at Dartmouth College during the period 1963–65, it became the backbone of several commercial time-sharing services as well as making its appearance into numerous industrial and engineering organizations. Today over 50 copies are in operation. At Dartmouth College, it operated for 16 hours a day, 7 days a week, and was shown to be able to handle between 30 and 40 simultaneous users.

In complexity, DTSS may be described as a general-purpose system with a limited scope; that is, it offered a number of different types of user services, while, at the same time, placing relatively severe limitations on the size of the computing job that could be handled. However, the experiment verified the prediction that most computing jobs, especially in an educational environment, are indeed small. It also provided the type of program creation and editing services that are most often needed and used by most ordinary users running ordinary computer jobs. The DTSS proved successful enough to enable 80 percent of all Dartmouth students, and a significant fraction of the faculty, to learn to program a computer.

## Computing a Part of Liberal Arts

The primary goal motivating our development of DTSS was the conviction that knowledge about computers and computing must become an essential part of liberal education. Science and engineering students obviously need to know about computing in order to carry on their work. But we felt exposure to computing and its practice, its powers and limitations must also be extended to nonscience students, many of whom will later be in decision-making roles in business, industry, and government. The administration and the Board of Trustees of Dartmouth gave us their full support as they, too, realized and accepted the goal of "universal" computer training for liberal arts students.

Dr. Kemeny is professor of mathematics and Dr. Kurtz is director of the Kiewit Computation Center, Dartmouth College, Hanover, New Hampshire.

We realized that traditional methods of handling computer programming at the college level would not work on a broad scale. First of all, and most importantly, the majority of liberal arts students would simply not elect a full course in computer programming. Second, those who did, or those who elected to learn on their own, would probably encounter two different kinds of obstacles that would tend to obscure and delay the understanding of the nature of computing and its applications.

The first obstacle was that only the most hardy student could persevere through the actual experience of running programs on the typical batch-processing system then in vogue. The debugging (error-correcting) process usually requires a large number of tries before the problem is successfully solved. If it takes on the order of 1 day for one try, the student will either lose interest or forget what the problem was. At best, he will waste time standing around waiting for the day's results to appear. (Some educational institutions had developed outstanding batch-processing systems that provided reasonably good computer service for student use, but it is our opinion that even the most successful of these falls far short of a time-sharing system such as DTSS in terms of convenience and pedagogical advantage.)

The second obstacle for our typical liberal arts student was the unnecessary complexity presented by the computer languages he had to use. Both Fortran and Algol, the two most popular languages at that time, contained serious pedagogical disadvantages. Fortran contains a number of grammatical rules, the need for which is not intuitively obvious to the student. As a result, certain common situations require puzzling constructions. While Algol was admittedly more elegant than Fortran, this very elegance posed its own pedagogical problems. Similar concerns would have applied to most of the other languages then in use. It should be noted that so-called "machine languages" or "assembly languages" present far greater pedagogical problems and are completely unsuitable for liberal arts use.

Therefore in planning an approach suitable for a moderate-size liberal arts college, such as Dartmouth, we decided that two radical changes were needed in the way computing was available. First, the student had to be able to debug his program and get it running in a relatively short period of time, with very little inconvenience to himself. Second, the programming language taught should be so simple that its details would not be an obstacle to even the most nonscientific student. Instituting these changes would allow the teaching effort to be concentrated on the application of computing to other subjects, such as mathematics, rather than on the acquisition of skill in the use of some obscure and complex language or with the detailed requirements of the computer system.

The first needed change led to the decision to design and construct a time-sharing computer service for student use. The second needed change led to the decision to design a new, simple, yet flexible, language. This new language was called Basic.

We attribute the quite unexpected degree of success of DTSS in part to the ease of use and efficiency of the time-sharing system, and in part to the simplicity of the language Basic. We have been most pleased to note how widely our philosophy of time-sharing has been accepted, and how rapidly Basic is being adopted by many computation centers of varying types.

## Historical Chronology

Dartmouth's experience in computing had begun in 1959 with a very small LGP-30. This experience revealed that bright undergraduate college students were extremely adept at computer programming, and that with little training they could produce major programs and software systems better than those professionally produced and commercially available.

During the year 1962 plans began to be formulated as to what type of computer system would be constructed. At the same time, the National Science Foundation was approached for support of the project to construct a time-sharing system and to develop its use in teaching large numbers of students. Without the generosity of NSF the project could not have succeeded.

At one point during the summer of 1962 a visit was made to a computer-manufacturing plant by one of the authors and a bright undergraduate student programmer. After spending less than 1 day obtaining a few facts about the hardware components, this student produced, within a week, a rough prototype design for a way that a time-sharing system could be made to operate with available components. Although this design was never implemented, it did convince us that a time-sharing system, built around a small, medium-speed computer and requiring no specially constructed hardware, was feasible. (We note in passing that this undergraduate is today a member of the faculty of an Ivy League university.)

During the year 1963 plans were developed further, and the computer hardware was selected. The hardware was manufactured by the General Electric Company, and consisted of a GE-235 computer system, a Datanet-30 communications computer, and a disk file that could be used by either computer. In the fall of 1963 our student programmers (together with the two of us) began learning how to program the new computer. We alternated between high-level system design and elementary exercises in code writing. In January of 1964 we received notification from NSF of their approval of our two-pronged project. We were officially on our way!

For a hectic month or two emergency alterations were made in the basement of an old college building—the only available site for the computer. The alterations included the installation of new electrical power lines and an air-conditioning system that later proved to be less than adequate for the job. The total floor space available at that time was about 1700 square feet, including staff offices.

The computer equipment itself arrived on schedule during the final week of February 1964, and was rendered electrically operational around mid-March. At this point, the student programmers occupied themselves for the better part of the day and night determining how the computer really operated under actual conditions (which was never exactly as described in the manuals), preparing programs, testing their theories of design for a computer system involving two processing units, and attempting to debug the programs they had prepared. Most of the dozen or so students were undergraduate mathematics majors. Some were sophomores who had not yet selected a major subject; these younger students in many cases did some of the best work.

During the month of April 1964 some of the students worked 50-hour weeks in addition to carrying the normal course load. Needless to say, some of their grades suffered as a result. However, the student programmers have never been permitted to disregard their studies completely and in no case has a student programmer "flunked out" because of time spent on the computer project.

At the same time, the two of us worked on the design of the language Basic. It had to be simple enough to allow the complete novice to program and run problems after only several hours of lessons. It had to respond as expected to most common formulas and constructions. It had to provide simple yet complete error messages to allow the nonexpert to correct his program quickly without consulting a manual. It also had to permit extensions to carry on the most sophisticated tasks required by experts, but these extensions could not impinge upon novice use; higher levels of power could not be bought at the cost of greater complexity for the beginner. Our solution was a statement-oriented language in which the more elementary statements (nine in number) were easily within the reach of the most unscientific layman. Other statements provided additional capabilities. All the statements are describable with an absolute minimum of explanation about function, limitations, and special cases.

One of us decided to write the first compiler for Basic himself. (A compiler is the piece of software that allows the computer to speak another language—Basic in this case.) He begged small amounts of computer time at nearby installations to make sure that, by the time DTSS was operational, we would have available a language that could work well within a time-sharing framework and take full advantage of the conversations possible under time-sharing.

## Some Design Considerations

To understand some of the problems of developing a complete software system, a diagram of the hardware components and their relationship to each other will be useful (Fig. 1).

While this two-computer configuration is only one of several ways in which a time-sharing system can be organized, its use proved extremely fortunate to us from several points of view.

First, all computing for users takes place in the slave computer, while the executive program (the "brains" of the system) resides in the master computer. It is thus impossible for an erroneous or runaway user program in the slave computer to "damage" the executive program and thereby bring the whole system to a halt. Second, the executive program, by periodic sensing, can ascertain if all is well in the slave computer, and can initiate corrective action if it is not. Third, having two computers simplified the organization and design of the system—the executive program was fairly small and was quickly brought to a state where it could be "locked-in" and ignored. On new computer hardware these needs are met within the context of a single computer, but these new capabilities were not available to us—perhaps fortunately so.

One problem that the two-computer approach did pose was communication between two machines. The master computer was equipped with a clock and could "interrupt" the slave computer. The master computer could also enter control commands into the memory of the slave computer and could read the responses to these commands. This master-slave design was a relatively new concept in computer organization, but our students were able to devise special techniques for handling the communication and control problems in a simple way.

The slave computer was programmed so that certain of its memory cells were, by agreement, used as "mailboxes" for transferring information to and from the master computer. Upon receiving an interrupt from the master computer, the slave computer would first save the crucial facts about its current work, and then scan the mailboxes for new instructions. It would then enter a code into one of the mailboxes to indicate receipt. If no tasks were called for, the slave computer would return to its current work. If assigned a task to carry out, it would do so, place a "task-completed" code in a mailbox, and return either to its current work or to an "idle" loop awaiting the next interrupt. The design of this mailbox-like communication mechanism was carried out by two students working together vociferously as each accused the other of being the cause of flaws in the design or its execution. When completed, it worked successfully without further need for modification. (Today both students are highly regarded computer professionals.)

The effect of the two-computer system is quite dramatic. Most of a given user's time is spent in typing, looking at answers being typed out, and thinking of what to do next. During these periods, which take minutes, he is talking only to the master computer, which is relatively inexpensive and can carry on conversations with all users at the

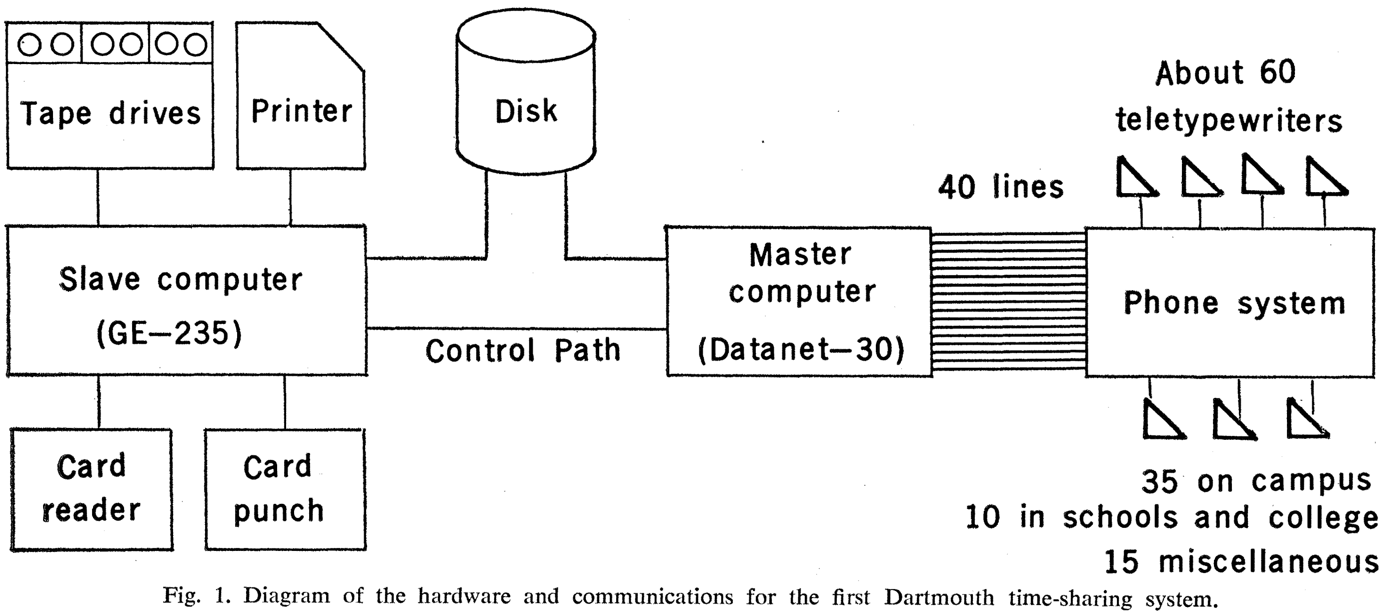

Fig. 1. Diagram of the hardware and communications for the first Dartmouth time-sharing system.

same time. The more powerful slave computer works on only one problem at a time, for a few seconds, providing computer services in the narrower sense of the word. It is the high ratio of typing time to computing time that makes time-sharing both possible and practical. The system is efficient from the user's point of view if he can get small amounts of computing time frequently and as soon as he needs them. Only a small minority of users ever need long periods of computing, and these periods are handled while the slave computer has nothing more urgent to do. From a theoretical point of view, this system anticipated the current understanding that communication, not calculation, is the primary activity in a time-sharing system.

## System Begins Operating

Barely a month and a half after the equipment became available for use by Dartmouth College, perhaps the most significant event in this whole history took place. On 1 May 1964, at 4 a.m., the Dartmouth time-sharing system was born as it successfully executed its first problem, one that had been supplied to it from a teletype, in Basic, with the answers being returned to the same teletypewriter.

During the subsequent month the system was "shaken down" by students in a numerical analysis course. These students took turns attempting to run programs on the three teletypewriters that were available. One measure of the quality of time-sharing systems in a state of development is the average period of operation between failures. During those early weeks the mean time to failure was 5 minutes, and these hardy numerical-analysis guinea pigs performed a service above and beyond the call of duty.

The quality of the system rapidly improved, and by mid-June of 1964 it was operating well enough to warrant increasing from 3 to 11 the number of teletypewriters, and to offer an indoctrination program for faculty members at Dartmouth College. Of approximately 500 faculty and staff members, nearly 200 attended three lectures by one of us on the use of the time-sharing system. Many of them followed the suggestion to practice on the teletypewriters, which were conveniently available since the undergraduate student body was on vacation. An extremely important effect of the training exercise was that most of the faculty and administration at Dartmouth College quickly became aware of the value and uses of the time-sharing system. Their enthusiasm and unqualified support has been an invaluable asset.

Table 1. Use of the Dartmouth system (approximate).

| User group | Size of group | Number of different users | |
|---|---|---|---|
| | | May 1968 | Year 1967–1968 |
| Dartmouth | | | |
| Undergrad | 3000 | 1000 | 2000 |
| Graduate | 600 | 250 | 450 |
| Faculty | 400 | 100 | 150 |
| Other colleges (4) | | 550 | 950 |
| Secondary schools (23) | | 2200 | 4600 |
| Total | | 4100 | 8150 |

By the fall of 1964 we were ready to start the freshman training program. Since 80 percent of all Dartmouth students complete a year of college mathematics (though there is no mathematics requirement), the second semester of freshman mathematics was an ideal course for the introduction of computer programming. In the science sequence this course introduces several major techniques of the calculus, while in the nonscience sequence the course covers finite mathematics; both are well suited for computer applications.

The academic year at Dartmouth is divided into three terms, and each student takes only three courses during a 10-week term. This calendar allows a great deal of freedom for experimentation—it is very useful to have one-third of the student's attention for a 10-week period. We designed a series of three lectures to introduce the student to programming in Basic, to be delivered during the first week of the term. We then reserved a teletype for each student for ½ hour a week for the remaining 9 weeks. He was required to program and debug four problems on his own. These problems were coordinated with the subject matter of the course and ranged from a trivial exercise to a quite substantial program.

After the evaluation of the first course we made two changes. We increased the reserved teletype time to ¾ hour a week—we had underestimated how badly our students type. On the other hand, the lectures were reduced to two. We have found that two 1-hour lectures are entirely adequate to introduce the novice to Basic. By the end of the second hour he is raring to write his first program.

We next had to face the problem of who would check the thousands of Basic programs written by our freshmen annually. Clearly, only the computer could do an adequate job. Following our usual philosophy, we did *not* try to devise a general program-checker but settled for the simplest solution that would be adequate for our purposes. We added to Basic a small software package that allows a test-program to take control from the student program. When the freshman thinks that he has debugged his problem, he types "TEST," and receives either an official approval or hints as to how his program is failing. These test programs are also written in Basic, and once such a routine exists for a given exercise, it may be used for years. The system has the very great advantage that only the student knows how many stupid mistakes he made before his program was accepted.

We have now trained four freshman classes, so that by June 1968 over 80 percent of all our students knew how to write a computer program. The typical training session results in about 95 percent completion of all assigned problems, a figure so high that it testifies more to the universal enthusiasm of the students than to the excellence of the instruction. We have found it very interesting to note how little help the students demand. Although a graduate student is available throughout the year to answer computer questions, he is not at all busy. A freshman has no trouble finding another freshman to answer his computer questions.

## Use In and Out of the Curriculum

The effect of almost universal knowledge in the use of computers and universal availability of the computer has been overwhelming. While precise statistics are difficult to obtain, the approximate data shown in Table 1 suggest a very high degree of infusion of computing into the life of the college, its associated professional schools, and colleges and secondary schools in the region. The undergraduate figure is especially notable since only about 25 percent of the undergraduates are actual or potential science majors.

Much of the use by students is incidental to actual course work, or represents homework-type calculations undertaken on the initiative of the students themselves. Nonetheless, a recently conducted survey shows that almost half of the 40 or so academic departments report "official" computer use in one or more courses; these 80 or so courses are listed below by title.

*Business School*
- Managerial Economics
- Environmental Economics
- Accounting and Finance
- Auditing

*Medical School*
- Pharmacology (several courses)
- Physiology
- Medical Physiology

*Engineering School*
- Introduction to Engineering
- Statics and Dynamics
- Solid Mechanics
- Active and Nonlinear Circuits
- Thermodynamics
- Computational Methods in Engineering
- Electricity and Magnetism
- Thesis, Honors Seminar
- Transportation Planning
- Combined Digital and Analog Controls
- Finite State Machines
- Heat, Mass and Momentum
- Nuclear Reactor Theory
- Engineering Economy
- Optimization Theory
- Decision Theory in Design
- Wave Propagation
- Tutorial

*Astronomy*
- Evolution of the Universe

*Biology*
- Life Science
- Genetics
- Aquatic Ecology

*Chemistry*
- General Chemistry
- Physical Chemistry

*Earth Sciences*
- Geomorphology
- Hydrology
- Geophysics
- Research

*Mathematics*
- Techniques of Calculus
- Finite Mathematics
- Computer Outside the Sciences
- Statistics
- Advanced Calculus
- Logic
- Linear Algebra
- Probability Theory
- Mathematical Statistics
- Number Theory
- Theory of Computing
- Numerical Analysis

*Physics*
- General Physics
- Classical Physics
- Contemporary Physics
- Electricity and Magnetism
- Advanced Laboratory

*Economics*
- Statistics
- Risk and the Corporation
- Econometrics
- Readings and Thesis
- Money and Banking

*Geography*
- Introduction
- Physical Geography
- Advanced Study
- Thesis

*Psychology*
- Complex Organizations
- Human Relations
- Statistics
- Laboratory-Behavior
- Mathematical Psychology
- Research

*Sociology*
- Human Society
- Comparative Social Institutions
- Methods
- Simulations of Social Processes
- Futurism

*Classics*
- Latin 1 Elementary
- Latin 3 Vergil
- Latin 80 Cato
- Greek and Roman Studies 5 (Greek Mathematics)

*Language*
- Spanish 1, 2, and 3

*Literature*
- Senior Thesis

## What Was Different about DTSS

We would now like to consider, in retrospect, what we did at Dartmouth that was different. First of all, our time-sharing system was designed for the novice. We were not worried that the expert would be able to make good use of the resulting system, as indeed he was. We went to great lengths to make the conversations with the computer simple and natural, but we never went to the point where the services were so time-consuming that the system could serve only a small elite group.

Perhaps a concrete example will explain how such fine distinctions were made. The overwhelming advantage of time-sharing is in debugging, the process most time-consuming for the human being—though not for the computer. The ability to receive diagnostics immediately and to obtain the results of test runs within seconds enables one to debug a fairly complex program in one teletype session. But just what should "immediately" mean? We have found that any response time which averages more than 10 seconds destroys the illusion of having one's own computer, and we have always kept the number of users to the point where responses of more than 10 seconds were rare.

However, many computer experts conjectured that users would want to have error diagnostics immediately after each line is typed in. Such an approach is naturally costly in terms of computer resources and reduces significantly the number of users the system can handle. We refused to adopt this extreme form of "controversial mode." Our users get diagnostics only when their entire program is compiled (translated from Basic into the language of the computer). We have found that this compromise is not only acceptable, but preferable. We find the instantaneous error messages of other systems annoying and a waste of time. It is very much like having an English teacher looking over your shoulder as you are writing a rough draft of a composition. You would like a chance to clean up the draft before it is criticized.

Perhaps our most radical departure from tradition was not to save the compiled version of a program but only its Basic version. We were seriously criticized for the "unnecessary overhead" in having to recompile every time the program is used. We claim that this piece of unorthodoxy is one of the real secrets of our success. It means that as far as the user is concerned the machine talks Basic. He may be aware of some other language in the background, but machine-language is as irrelevant to the average user as the peculiarities of the hardware. And, ironically, we have gone a long way toward turning this "inefficiency" into a source of savings.

Basic was designed not only to make it easy to use but to make its compilation significantly faster than that of other languages. With modern software technology, our original Basic compiler has been turned into a highly efficient instrument by our student programmers. Even on the medium-speed slave computer of DTSS only a small fraction of programs required more than a second to compile. (On our much faster system today compilation times below 1/10 of a second are common.) One must add that a typical user hardly ever runs the same program twice. He will make a small change, or a correction, or change his data. Once any change is made, it is much more efficient to recompile than to try to patch a program in machine language. We also gain a great deal in storage capacity by saving only the compact Basic program rather than the fully expanded version in machine language. Nor does the user save several slightly different versions of his program; it is too easy to make small changes to his Basic program as needed.

Of course, over the years the number of services under DTSS has vastly increased. We have added a variety of editorial services. We have added Algol (for the purist) and Fortran (for the old-fashioned). Our library contains some 500 programs, including many games. We have found that a computer is an efficient and inexpensive source of entertainment. We have lost many a distinguished visitor for several hours while he quarterbacked the Dartmouth football team in a highly realistic simulated game. And Basic itself has grown and matured. While we can still introduce the novice to Basic in 2 hours, today we also develop major new systems and large computer-assisted-instruction applications entirely in Basic.

## Summary

We have learned that success can result in an entirely new set of problems. Our original DTSS, which seemed much too large for a small campus with very few computer users, soon proved unable to handle the demands of the same small campus where everyone seemed to be clamoring for computer services. By 1966 we were planning the second DTSS. With the cooperation of General Electric, we opened in the fall of 1967 a time-sharing system, based on GE-635 hardware, which can handle over 100 users. The NSF has helped us provide computer services to 23 secondary schools and 10 colleges, as well as expanding the local capabilities. This fall we will launch phase II, a fully general-purpose, large-scale, time-sharing system for the GE-635. Like the first DTSS, phase II is again being written by a faculty-student coalition at Dartmouth. It will serve everyone, from the novice to the research worker who needs large production runs. We hope that with our much more powerful hardware we will be able to provide these extended services to some 150 users simultaneously without compromising our basic philosophy of making the system as easy to use for the inexperienced as for our original DTSS.

The real test comes this fall. We are confident that the expert faculty user will be very happy. But will our students after a football game still take their dates to the Kiewit Computation Center to show off their prowess with computers?

# University Integrity

Kenneth S. Pitzer

At the moment, there seems to be special need to discuss the internal logic of the university—the relations between its students, faculty, governing board, and administrative officers, and especially the factors which are essential to the university's integrity as an institution. The trials of Columbia University have been all too prominent in recent weeks. But many other American universities have suffered. And, as we look around the world, we note the troubles of one of the oldest and most prominent universities, the University of Paris. While the pressures leading toward disruption are not the same everywhere, it is true that some universities have been able to contend with these factors much better than others have. The problems here in Houston seem not to have been as severe as those in some other locations, but anyone who is sensitive to the thinking of various individuals can detect the presence of the same ideas, objectives, and frustrations.

In commenting on these problems I want to distinguish carefully between those cases where the institution suffered a real breakdown—where the educational activities were substantially disrupted—and those in which an expression of student opinion got slightly out of hand. So long as students respect the rights and privileges of others who may hold differing views or who may merely be uninterested in a particular topic, they certainly have the right to express their views on the public issues of the day. In some cases overenthusiastic picketing has been conducted in a manner that has somewhat infringed upon the rights of others, but the institution has been able to handle the situation with an appropriate firmness and compassion and then has been able to continue with no loss of integrity.

## The Pressures

What are the pressures that are especially great today? What do the activists want? Some of you undoubtedly know better than I, but I hope you will accept the following brief summary. There is deep student concern over certain issues confronting our society, especially race relations and the war in Vietnam. This concern is combined with knowledge on the part of certain older students who have seen the technique of civil rights demonstrations yield the fruit of favorable congressional action. Recently the population in general and the governmental leadership have found these techniques less convincing. As a result there is, in these active student groups, a sense of frustration. Many students have shifted their activities to the political sphere by supporting their favorite candidate for the Presidency; this is most commendable. But a small hard core of extremists—those with the greatest arrogance and the least faith in their country—have escalated their demonstrations from the legal range to the level of kidnap and blackmail. Unfortunately, in a few cases substantial numbers of other students and faculty have supported these extremists or have opposed the use of feasible methods of dealing with them.

Joseph Shoben of the American Council on Education puts it in these terms (*1*):

(1) Like a great many other citizens of our republic, students in large numbers are sufficiently frustrated and distraught by the nature and entailments of the war and by the unhappy state of our race relations to act on their discontent. (2) Because they are primarily in contact with colleges and universities as institutional

The author is president of Rice University, Houston, Texas, and president-elect of Stanford University, Stanford, California. This article is adapted from a commencement address delivered 17 May 1968 at the University of St. Thomas, Houston.