Compatible Time-Sharing System (1961-1973)

Fiftieth Anniversary Commemorative Overview

IEEE computer society

The Compatible Time Sharing System (1961–1973)

Fiftieth Anniversary Commemorative Overview

The design of the cover, half-title page (reverse side of this page), and main title page mimics the design of the 1963 book *The Compatible Time-Sharing System: A Programmer's Guide* from the MIT Press.

The Compatible Time Sharing System (1961–1973)

Fiftieth Anniversary Commemorative Overview

Edited by
David Walden and Tom Van Vleck

With contributions by
Fernando Corbató
Marjorie Daggett
Robert Daley
Peter Denning
David Alan Grier
Richard Mills
Roger Roach
Allan Scherr

IEEE computer society

IEEE Computer Society
2001 L Street N.W., Suite 700
Washington, DC 20036-4928
USA

First print edition June 2011 (web revision 05)
The web edition at http://www.computer.org/portal/web/cshc is being updated from the print edition to correct mistakes and add new information. The change history is on page 50.

To Fernando Corbató
and his MIT collaborators
in creating CTSS

Contents

Preface

Time-sharing was in the air in 1961. John McCarthy had been thinking about it since 1955 and in 1959 wrote a memo proposing a time-sharing system for the IBM 709 computer at MIT. Fernando Corbató and two of his MIT Computer Center staff members, Robert Daley and Marjorie Merwin, starting working on the design of the Compatible Time Sharing System (CTSS) in the spring of 1961. The system was first demonstrated in November of 1961. (By 1963 CTSS was stable, large scale, operational system — proof positive of the feasibility of time sharing.) Thus, 2011 is the Fiftieth Anniversary of the birth of CTSS, a manifestly appropriate time to review the history of CTSS.

In the rest of this commemorative brochure, we sketch the history of CTSS, include retrospective memories from several of the key CTSS personnel and others, discuss the uses of CTSS and the impact it had on the computing world, and provide a fairly comprehensive bibliography.

CTSS has been frequently mentioned in the archival literature of computing. In particular, volume 14, number 1 (1992), of the *IEEE Annals of the History of Computing* was largely dedicated to a discussion of the CTSS system.[1] In October 1988 prior *Annals* editors John A. N. Lee and Robert Rosin attended a 25th Anniversary celebration of MIT's Project MAC (the 27th anniversary of CTSS). At that time they held a group interview with several people who had been close to CTSS including Professor Corbató. That interview session and other materials on CTSS became the content of the 1992 issue of the *Annals* (the 31st anniversary of CTSS).

That prior effort was by personnel of the IEEE Computer Society was primarily about capturing the history of CTSS. Now we of the Society History Committee are able to give a proper celebratory salute to CTSS at the time of the 50th anniversary of its birth.

Note: Because we are using print-on-demand technology and on-line posting to disseminate this brochure, we can do an update if we learn more about CTSS.

Acknowledgments

Fernando Corbató was the primary motivating force for CTSS. He started the 1961 work on the system with Marjorie Daggett and Robert Daley. We are fortunate to have been able to talk to the three of them as we prepared this commemorative brochure in 2011.

Tom Van Vleck has been my most active supporter in the development and production of this brochure. He allowed his 7094-and-CTSS web page to be adapted and included as Section 2 of this document. He provided pointers to numerous sources of information about CTSS. He suggested who I should talk to, including making an introduction to Roger Roach. He provided photographs. He reviewed every page as this document was drafted and provided corrections, additions, and other insight and guidance. Thank you, Tom, for serving with me as editor of this commemorative brochure. Creating it would have been impossible without you.

We solicited comments about CTSS from other people who were involved with the system or had views of it. Their names are noted with their contributions in Section 4 (page 21 — they are also listed in the table of contents). Thank you, all.

[1] MIT92a.

In late 2010, David Alan Grier of the IEEE Computer Society brought to my attention that 2011 would be the fiftieth anniversary of CTSS, and he encouraged me to do something about this anniversary. Thank you, David.

Matthias Bärwolff spotted several typos in a draft of this document. Wally Weiner gave me a photo of the CTSS manual cover which I mimicked. Steve Peter talked to me about the typefont used on the cover of the CTSS manual. Roger Roach provided me with a copy of a DVD with scans of lots of original CTSS documentation. Paul Pierce enabled me to take a photo of the 7094 console in his computer collection. IBM Corporate Archivist Paul Lasewicz provided an image of a 7094 system. Karl Berry reviewed the accuracy of the colophon. The hard copy version was printed by Martin and Lily Sudarma of Copyman of Portland, OR. Luke Walden provided tips relating to images taken from an old video.

Hard copies of this brochure will be given to everyone who contributed to it as a token of appreciation. I hope that Tom Van Vleck and Roger Roach will accept my offer to post on-line copies of the brochure with their various web-based CTSS repositories and descriptions.

Most particularly, we are grateful to everyone who worked on CTSS over the years, helping to make it available and the fundamental influence on the post-1961 future of computing that it was. Hence the words on the dedication page.

MIT nomenclature and abbreviations

The MIT community typically refers to MIT buildings by their numbers, not names, for instance, Building 26, Building 39, and Building 20. Building 26 was the original location of the Computation Center and CTSS system. The Computation Center later moved to Building 39. Building 26 was next to the famous Building 20 (now gone) which housed MIT's Research Laboratory of Electronics (RLE) and other entities.

Perhaps curiously, the building housing Project MAC, 545 Technology Square in a complex of buildings known collectively as Technology Square, was typically not referred by number; rather one would say just say something like "I'm going over to Tech Square," meaning 545 Tech Square.

A couple of other abbreviations relevant to this document are ESL (Electronic Systems Laboratory), and DSR (Division of Sponsored Research).

We have not attempted to try for consistency among the alternatives of "Comp Center," "Computer Center," and "Computation Center."

David Walden, Chair
IEEE Computer Society History Committee
June 1, 2011
dave@walden-family.com

1. Early history of CTSS

Time-sharing was an idea for which the time was technologically ripe in 1961. Various people were talking about time sharing; and, over the next decade or two, many time-sharing systems were developed.

From the existing historical record,[1] we can piece together a sketch of the people and ideas that resulted in CTSS being created at MIT in 1961, becoming the first significant demonstration that time-sharing was really possible.[2] This chapter does not describe CTSS, only its early history. For a description of CTSS, read Section 2 before or after this chapter, or in parallel.

Fernando Corbató transferred from Caltech to MIT to work on his PhD in physics. At MIT Corby (as he is called by everyone who knows him), became a student of Professor Philip Morse. Morse was a physicist, and is called the father of operations research by some; he had insight that computers and computation were going to be a big deal before many others came to understand it. Morse arranged a research assistantship that allowed Corby to become familiar with the then current computer technology — from punched cards to programming the Whirlwind computer (the early core-memory-based, MIT-developed computer at which a generation of later computing innovators got their first taste of interactive computer access). As part of Corby's thesis work, which involved "doing energy band calculations in graphite, in carbon,"[3] he also learned to "write big programs, to debug them, to get a complete hands on feel for working with programs, organizing them, and learning the ropes of how to use a computer." John Little and Corby also used Whirlwind to create thick book containing a "numerical table of coefficients for expanding spheroidal wave functions."[4]

After his PhD, Corby was hired by Morse as a research assistant in the Computation Center, of which Morse was head. Morse had arranged for IBM to provide an IBM 704 computer to the MIT Computation Center. Morse put Corby in charge of managing other Computation Center research assistants, giving Corby a bit of administrative experience. Corby also got to attend meetings SHARE (the IBM user's group) to thus keep up with what else was going on in computing. Eventually Professor Frank Verzuh, who was associate director of the Computation Center (Morse was head), left and Corby took over as associate director.

The Computation Center had opened in 1957 and by 1958 and 1959, it was overloaded. People in science and engineering were using computers more, which FORTRAN had helped facilitate. However, typically it took repeated batch submissions to get a program working. Batch processing maximized utilization of the computer rather than optimizing throughput of getting programs written.

The people who had had hands on experience with computers such as Whirlwind, the TX-0 (an early transistor-based computer developed at MIT Lincoln Laboratory and

---

[1] Corbató63a (preface), Corbató89, Corbató06, Garfinkel99, Hauben97, McCarthy89, MIT92a.
[2] The vendors of the then prevalent batch processing computer systems were mostly not thinking in terms of time sharing in 1961. [3] Corbató06, p 5. [4] Corby has said that this was "one of the earliest examples of trying to do automatic printing."

later moved to RLE in Building 20 at MIT), and so on were particularly anxious to move computers and computing in the direction of interactiveness with the computer. In particular Professor John McCarthy, who had come from Dartmouth to be on the MIT faculty, was promoting the idea of time sharing. (According to Corby, McCarthy did the best job of articulating the need for interactive computing.) Also Professor Herb Teager was interested in hardware for time-sharing, and Morse had arranged some funding for him, and he was going to develop a time-sharing system.

Corby saw Teager's ideas as too grandiose and thought that a modest demo could be created on the IBM 709 in the computer center that would help people grasp why interactive computer use might be a good thing.

Corby states that he began talking with Marge Daggett (then Marge Merwin) and Bob Daley of the Computation Center staff about the system design in the spring of 1961, and the original demonstration was working by November of that year. They got IBM to provide an interrupt capability for the machine (an off-the-shelf change) which allowed them to take control of the machine. They created a special version of the operating system (which came to be known as CTSS). This operating system set aside 5 kilowords of memory (of 32kw total) for the time-sharing supervisor (and for buffering typewriter terminal input and output). Herb Teager had made four Flexowriters into special terminals, and these would work with the IBM 709. CTSS used tape drives to store the programs and files of the users of the four terminals. It was crude, but that was the original configuration for the desired demonstration of interactive computer use, and it allowed the traditional batch processing system to still operate (that was the "compatible" part of Compatible Time-Sharing System). Corby wrote the now famous first CTSS paper, with Daggett and Daley listed as co-authors,[5] in January 1962 for presentation in May 1962.

I asked Corby how it came to be that Marge and Bob were the other earliest involved people in CTSS. Corby explained,[6] "CTSS started out not as a planned project, but rather as an effort to see if the timesharing concept, advocated eloquently by John McCarthy, could be demonstrated by some judicious programming. Marge and Bob were both interested and were probably the two best programmers on the staff. So what began as a demo turned into a more and more viable system as the pieces began to fall in place."

Marge remembers that initially only Corby and she were working on the project, spending "a lot of time conferring and hashing and re-hashing problems." She also remembers, "having to do testing at odd hours and sitting at a listing after 5pm trying to figure out what went wrong." She remembers it was a big exciting day when it finally worked for two typewriters. "There was still a lot to do, but conceptually it was pretty much there — very exciting."[7]

Bob remembers Corby writing the core of the time-sharing system himself, using three tape drives to swap users out of memory and another three tape drives to hold user data. User data could be added to the end of the user data tape. This 3-user system demonstrated the basic principles of time sharing. It was demonstrated frequently, according to Bob, and he can't name a specific demonstration date.[8] Marge remembers that Corby certainly wrote a lot of code in the supervisor, with all of the work on queuing being his. She remembers her early involvement being with interrupt handling, saving the state of the machine, various commands (and what was a command and what was not), character handling, and also crude methods for inputting and editing lines for the demos. She remembers a lot of "back and forth" about what various states were — the distinction between dead and dormant, waiting, etc. Marge says that she "worked in the underpart of the supervisor."

The demonstrations of CTSS helped convince people at MIT about the value of time sharing, and Phil Morse in turn convinced IBM to upgrade the 709 (a tube machine) to a

[5] Corbató62. [6] 2011-03-05 email. [7] 2011-03-04 phone interview, page 22. [8] 2011-02-28 phone interview, page 24.

7090 (a transistor machine). This upgrade began in the spring of 1962 and was done by late summer. As part of that upgrade, they added an additional 32K words of memory (which allowed much more space for the time-sharing supervisor). An IBM 1301 disk memory was also added which replaced tape drives for swapping user program and storing user files. They obtained an IBM 7750 terminal controller which could support up to 32 terminals and then some early modems.

Marge Daggett married in February 1962, and took a leave of absence surrounding the birth of a child from approximately November 1962 through April 1963. In May 1963 she returned to part-time work and no longer participated in the mainstream of CTSS development.

Bob remembers that one day Corby gave him a document on a 1301 disk from IBM and told him that they were getting one of these. There was no existing software for the 1301. Corby had some ideas about how to use the 1301 but basically told Bob to figure out how to use it as part of CTSS. Bob developed the first user file system for a time-sharing system, including a capability for backups.

According to Corby,[9] by the late spring of 1963 they were "beginning to run CTSS with this new configuration involving the extra bank of memory, the 1301 disk drive, and the 7750 hardware. We were in place to be the workhorse for the Project MAC summer study which was to take place that summer."

In 1962 J. C. R. Licklider, also a visionary about time-sharing who had been doing his own experiments with John McCarthy and Ed Fredkin at BBN (see Section 5), became the first director of the Information Processing Techniques Office at ARPA. There he funded MIT's Project MAC.[10] ARPA also funded a second CTSS system (that resided at 545 Technology Square on the edge of the MIT Campus) for use by Project MAC. The second system was run primarily as a time-sharing system.

By 1973 it was time to retire the 7094s and CTSS. The Computation Center 7094 was returned to IBM in late 1968. The Project Mac CTSS system was shut off in May 1973.

Of the CTSS era, Corby says,[11] "What is becoming clear is that the details of those days are no longer crystal clear! In many ways, the system programming of those early days was cut-and-paste in that one used a working assembler or a compiler and glued them together. We gradually built CTSS piece by piece, adding a major telecommunications controller (IBM 7750), a second bank of core memory, and a large disk file for storage of user programs." Corby suggests that, because they did not have to deliver on any promises, they were able to take their quite successful incremental approach.

[9] Corbató06, page 11. [10] MIT92f. [11] 2011-03-05 email.

2. The IBM 7094 and CTSS at MIT

By Tom Van Vleck

Editors' note: Over 16 years, Tom Van Vleck's web page on the "The IBM 7094 and CTSS"[1] *has been growing incrementally. It is a mix of items and topics of which Tom says, "Basically I was trying to emphasize what I knew first hand, with information from others for context." We greatly appreciate Tom's work in adapting the content of his web page for use here.*

This note describes my experience at MIT with the IBM 7094 and the CTSS operating system.

One reason I wanted to go to MIT was my interest in computers.[2] When I entered MIT as a freshman in 1961, I often passed the glass wall that let passers-by in Building 26. see the Computation Center's 7090, in a computer room which seemed about the size of a basketball court. You could see the tape drives spin and the lights blink.

Figure 2.1: An IBM 7094 (Photo courtesy of the IBM Corporation.)

MIT's mainframe computers

The MIT Computation Center got its IBM 7090 in the spring of 1962, replacing a 709, and upgraded its 7090 to a 7094 by 1963. In the mid-1960s, IBM's 7094 was one of the biggest, fastest computers available, able to add floating numbers at a speed of about

[1] VanVleck11. [2] VanVleck95.

0.35 MIPS. A standard 7094[3] had 32kilowords of 36-bit-word memory (henceforth called 32K). Its data channels could access memory and run simple channel programs to do I/O once started by the CPU, and the channels could cause a CPU interrupt when the I/O finished. A 7094 cost about $3.5 million, back when that was a lot of money.

IBM had been generous to MIT in the fifties and sixties, donating or discounting its biggest scientific computers. When a new top of the line 36-bit scientific machine came out, MIT expected to get one. In the early sixties, the deal was that MIT got the computer for one 8-hour shift, 40 or so other New England colleges and universities got a shift, and the third shift was available to IBM for its own use. One use IBM made of its share was yacht handicapping: the President of IBM raced big yachts on Long Island Sound, and these boats were assigned handicap points by a complicated formula. There was a special job deck kept at the MIT Computation Center; and, if a request came in to run it, operators were to stop whatever was running on the machine and do the yacht handicapping job immediately.

The Deputy Director of the MIT Computation Center was Prof. Fernando J. Corbató, known to everybody as Corby. The Director was Prof. Philip M. Morse, who didn't seem to be involved in the day-to-day operations of the Center; Corby ran the whole thing. When you submitted batch processing jobs, you specified your problem number and programmer number. The lower your number, the longer you had been associated with the Comp Center. Corby's was 3.

FMS and Batch Processing. The 7090 and 7094 were operated in batch mode, controlled by the FORTRAN Monitor System (FMS). Batch jobs on cards were transferred to tape on an auxiliary 1401, and the FMS monitor read one job at a time off the input tape, ran it, and captured the output on another tape for offline printing and punching by the 1401. Each FMS user job was loaded into 7094 core by the BSS (Binary Symbolic Subroutine) loader along with a small monitor routine that terminated jobs that ran over their time estimates. Library routines for arithmetic and I/O were also loaded and linked with the user's program. Thus, each user's job had complete control of the whole 7094 — all 32K of memory, all the data channels, everything.

MAD Language. MIT and the University of Michigan were both 7094 owners, and their computation center people were colleagues who traded code back and forth. When I was a freshman in 1961, we used FORTRAN in the elementary programming course, but by the time I was a sophomore, MIT had installed Michigan's MAD (Michigan Algorithm Decoder) language,[4] written by Graham, Arden, and Galler, and was using that in most places that a compiler language was needed, especially computer courses. MAD was descended from ALGOL 58: it had block structure and a very fast compiler, and if your compilation failed, the compiler used to print out a line printer portrait of Alfred E. Neumann (MIT took that out to save paper). Mike Alexander says, "MAD was first developed about 1959 or 1960 on a 704, a machine which makes the 7094 look very powerful indeed." At that time MAD ran under UMES, the University of Michigan Executive System, derived from a 1959 GM Research Center executive system for the IBM 704 that was one of the first operating systems.

Part of the Michigan/MAD code was a replacement for the standard FORTRAN output formatter routine (IOH). (Programs written in **FAP** — FORTRAN Assembly Program, the 7094 assembler — could use special characters, such as parentheses, in external names. I/O library routines were often given names that FORTRAN and MAD programs could not generate, to avoid name conflicts.) The MIT/Michigan version of IOH supported additional format codes used by the MAD language and had other internal improvements over the IBM version. One change was to use the extra index registers that the 7094 had and the 7090 didn't. And buried deep in the code, there was the line of code

```
SXA  VR16,1   'SOMEBODY HAS TO SAVE IT' SAYS BOB CRABTREE
```

[3] Weik64. [4] MAD62, MAD66.

Noel Morris and I made up the Bob Crabtree Society, open to people who knew where that comment was and what it did.

Early Time-Sharing at MIT

MIT professors, such as Herb Teager and Marvin Minsky, wanted more access to the machine, like they had had on **Whirlwind** in the fifties and the **TX-0** in the sixties, and quicker return of their results from their FMS jobs. Professor John McCarthy wrote an influential memo titled "A Time Sharing Operator Program for Our Projected IBM 709" dated January 1, 1959, that proposed interactive time-shared debugging. These desires led to time-sharing experiments, such as Teager's "time-stealing system" and "sequence break mode," which allowed an important professor's job to interrupt a running job, roll its core image out to tape, make a quick run, and restore the interrupted job. McCarthy's Reminiscences on the History of Time Sharing[5] describes his and Teager's role in the beginnings of time-sharing. Teager and McCarthy gave a presentation titled "Time-Shared Program Testing" at the ACM meeting in August 1959.

CTSS development had started the same year I entered MIT, led by Corby, Bob Daley, and Marjorie Merwin-Daggett. A version of CTSS that swapped four users to tape was demonstrated on MIT's IBM 709 in November of 1961. This system could support four Friden Flexowriter terminals directly connected to an I/O channel of the computer. CTSS was described in a paper at the 1962 Spring Joint Computer Conference,[6] even though the software wasn't quite working on the IBM 7090. Much of the CTSS research was funded by US National Science Foundation grants to the Computation Center. Development continued through 1962 and 1963, and system capabilities and usage continued to expand. Service to MIT users began in the summer of 1963.

Corby was interviewed by John Fitch for the Science Reporter program on WGBH-TV on May 9, 1963. He demonstrated CTSS running on MIT's 7090.[7] The program aired on WGBH-TV on May 16, 1963, and was titled "Solution to Computer Bottlenecks."

Figure 2.2: Corby on WGBH

CTSS

CTSS was called "compatible" in the sense that FMS could be run in B-core as a "background" user, nearly as efficiently as on a bare machine, and also because programs compiled for FMS batch could be loaded and executed in the "foreground" time-sharing environment (with some limitations). Background jobs could access some tape units and had a full 32K core image. This feature allowed the Computation Center to make the transition from batch to timesharing gradually, and to retain the ability to run "dusty decks" and software from other institutions. The Comp Center got the CTSS RPQs added

[5] McCarthy83. [6] Corbatø62. [7] Corbató63b.

Sidebar: My Undergraduate Computer Experience

As a freshman, I took course 6.41, *Introduction to Automatic Computation*, which taught me machine language and FORTRAN. The 6 in the course number mean that it was offered by the Electrical Engineering department. You couldn't major in computers at MIT in 1961 — or anywhere else either. There were about a dozen computer-related courses available in the catalog, in four or five departments, and I wanted to take them all. I was in the Mathematics department, which didn't seem to believe in computers at all: I remember my Numerical Analysis laboratory, with its rows of Marchant calculators.

I took course 6.251, *Introduction to System Programming*, as soon as I could. When I took the course, there were a few exercises in MAD, and then we were presented with CAP (Classroom Assembly Program), a simple assembler for the 7094, and assigned a series of problems to improve it. The assembler itself was written in FAP, and we submitted batch jobs in the form of update runs that applied "alters" (symbolic patches) to the CAP source, reassembled and linked it, and ran a standard test suite. You got so many points for adding the * meaning "the current instruction counter," so many more for putting in a simple expression evaluator, and so on. The textbook was by Corbató, Poduska, and Saltzer.[8]

I had programming jobs back home each summer, and after taking 6.251, I looked for part-time programming jobs at MIT. In 1963–1964 I had part-time work at MIT Project MAC, using CTSS, in Professor Teager's group.

In 1964–1965, I found a different job as a part-time programmer for Prof. Ithiel de Sola Pool in the MIT Political Science department. I wrote event simulation, survey analysis, and statistical code for the 1401, 7094 FMS and CTSS. As part of this activity, I learned as much as I could about CTSS, and started to make user contributions. I worked with another student programmer, Noel Morris. We shared a windowless office in the library stacks: the good thing was that it had an IBM 1050 terminal.

to the blue machine and began running CTSS in 1965. (RPQ stood for *Request Price Quotation*. IBM, in those days, would engineer and sell features not part of the standard product, for a price.) The configuration for both machines included about a dozen tape drives, a swapping drum, and a 1302 disk file with a capacity of about 36 megabytes, shared among all users of the machine.

The CTSS supervisor provided up to about 30 "virtual machines," each of which was an IBM 7094. One of these virtual machines was the background machine, and had access to tape drives. The other virtual machines were foreground users: these virtual machines could run regular 7094 machine language code at 7094 speed, and could also execute one extra instruction, which invoked a large set of supervisor services. These services included terminal I/O and file system I/O. Programs could be written for the foreground environment in any language available for the 7094; libraries were provided to allow compiler languages such as MAD to call on supervisor services. CTSS provided a file system that gave each registered user a separate directory of disk files.

The key features of CTSS were virtual machines, hardware isolation of users from the supervisor and from each other, and a per-user disk file system. Because the user virtual machine supported the same architecture and instruction set as the 7094, CTSS was able to support a large body of applications originally developed for the FMS batch environment at MIT and elsewhere.

CTSS software included not only the supervisor but also a set of "foreground" command programs and subroutine libraries. Many of these commands read files from the user's file system storage and wrote other files; for example, the MAD compiler read a disk file containing a program source and wrote a disk file containing machine

[8] Corbató63c

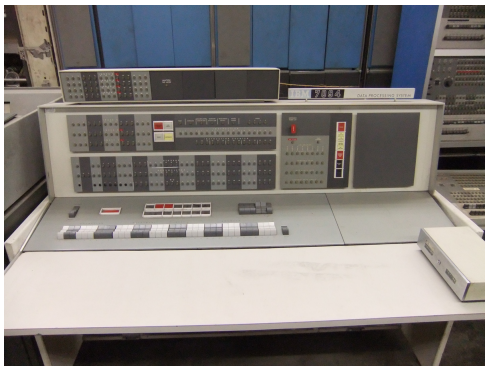

Figure 2.3: 7094 console

instructions in BSS (binary) format. Most of the foreground "MAD command" (the program invoked to process a command line beginning with MAD) was identical to the background MAD compiler on the FMS system tape, with the addition of a wrapper to handle command line options, and replacements for the input and output routines to read and write disk files instead of tape, and to write error messages to the user's terminal.

Project MAC

In 1963, MIT obtained funding from ARPA for Project MAC, an interdepartmental laboratory to explore development and use of time-sharing. Its first major activity was the 1963 Project MAC Summer Study, where many of the big names in the computing world spent time at Project MAC using the brand new idea of a time-shared computer. Project MAC's plans included the development of a next-generation time-sharing system called Multics. CTSS was to be used as the programmers' tool to develop this new system, as well as the tool to support other research projects, such as database and language research. The Director of Project MAC was MIT Professor Robert M. Fano, a well respected electrical engineering professor and one of the authors of the canonical circuit design textbook. His Assistant Director was Dick Mills, who had worked on Whirlwind and contributed to early CTSS development.

Figure 2.4: Prof. Fano using a Model 35 TTY

YouTube has a movie clip from 1964 of Professor Fano describing time-sharing and using CTSS on a Model 35 Teletype.[9]

[9] Fano64b.

Sidebar: My CTSS Experience After Graduation

When I graduated in 1965, I transitioned to a full time programming job, working for the Political Science department, as did Noel Morris. We were able to get offices in Tech Square, near a public terminal room, and worked on multiple projects. At the same time, we explored and investigated how CTSS worked and clever things we could do with it, and contributed some software to the CTSS community. My interests were more in the operating systems area than in data crunching, and in time I transitioned over to working for Project MAC, helping Dick Mills administer CTSS.

The Multics project had begun and was a big consumer of the MAC CTSS. There was much more demand than one 7094 could provide. People resources were also stretched: Corby concentrated on building and leading Multics and gave up his role as Deputy Director of the Comp Center. Dick Mills was given the new post of MIT Director of Information Processing Services, reporting to the Provost. In 1965 the Comp Center got the RPQ changes necessary to run CTSS and the additional hardware needed, and began to run its own CTSS. Almost all of the original designers moved on from Comp Center to Project MAC, or to other jobs such as IBM Cambridge Scientific Center, where CP/CMS was built. The MAC people who understood CTSS, like Bob Daley, were under pressure to not waste their time tweaking it—just stabilize it to make it a usable tool, and provide a copy to Comp Center. So I started out at Project MAC as a flunky taking over the CTSS jobs that other more important folk were supposed to relinquish. I was ecstatic: CTSS was arguably the best working timesharing system in the world, and there was a ton to learn and as much responsibility as I could handle. As the Multics project use of CTSS expanded, I helped sign up users from Bell Labs, who would use the MAC CTSS by dialup from New Jersey. I may have set up CTSS accounts for Ken Thompson and Dennis Ritchie, among others.

As time went on, I took on projects for Multics as well as CTSS, initially writing design papers on how to do resource accounting and system admin for Multics. CTSS required less support and once Multics became self-supporting, MIT went down to one 7094 running CTSS, moving the "red machine" from MAC to the new Information Processing Center in Building 39. The Comp Center folks assembled a team of interested undergraduate and part-time system programmers to support CTSS, and Roger Roach was one of the leaders. I worked on Multics system boot and other facilities, and then took a job at the Information Processing Center managing system programming for CTSS, an IBM 360/67 running CP/CMS, and the features of Multics that IPS would need to be able to run a service on it. Roger became the leader of the CTSS team.

By the time of the 1963 Summer Study, CTSS was in use as a service on the Computation Center 7090, supported by hardware RPQs. Computer researchers from all over the country, from academia and industry, used CTSS to try interactive computing.

In October 1963, Project MAC got its own 7094 (paid for by ARPA) in the new Tech Square building behind MIT. The Computation Center machine, upgraded to a 7094, ran CTSS and FMS in the background; it had the standard blue side panels. The Project MAC machine had red side panels, and staff often referred to "the red machine" or "the blue machine."

By 1965 most of the CTSS development group had moved over to Project MAC and was beginning the design of Multics. The last big CTSS project was a new, much better, file system, as a kind of parting shot, to ensure that the system would be adequate to support Multics development, and that the Computation Center CTSS service would be robust.

CTSS Features

Shared Files. The biggest discovery from the CTSS experience was the importance of online file systems and file sharing. The initial 1960 vision timesharing was focused on access to CPU and memory. When that was implemented, feeding input into programs and capturing their results required a way to have rapid access to per-user data; a disk filled that need. Some early systems stopped there, essentially dividing the disk storage into per-user chunks. As CTSS developed, we provided ways for users to share their files on disk, through "common files" and "linking," and the collection of on-line shareable data became the seed of a community of information sharers. We are now seeing a shift in computer usage from individual piles of files on individual PCs to storage and sharing "in the cloud" that parallels the discoveries about what is important that we saw with CTSS in the 1960s. It may be that "cloud computing" will encounter again some of the same hard lessons about security and community that we learned at Project MAC.

7094 CPU Modifications. The hardware **RPQs** that made CTSS run on MIT's 7094s added an interval timer (B/M 570220) and memory boundary and relocation registers (RPQ E007291) to the 7094 processor. In addition, the MIT machines had two 32K core memory banks instead of one (RPQ E02120). Core bank A held the CTSS supervisor, and B Core was used for user programs. More importantly, the RPQ made the 7094 a two-mode machine. When the machine was running a program in B-core, many instructions were forbidden: executing them caused the machine to trap and take its next instruction from A-core. In particular, I/O instructions were forbidden as were attempts to switch core banks.

By CTSS convention, user programs called on supervisor services, such as the file system, by executing a `TSX ADDR,4` where `ADDR` held `TIA =HWRFLXA` (where `WRFLXA` was the name of the entrypoint being called). The `TIA` instruction was illegal in B-core mode, so the 7094 CPU trapped into A-core. The CTSS supervisor running in A-core invoked a module named `PMTI` (Protection Mode Trap Interpreter), which looked up the BCD name of the supervisor entrypoint `WRFLXA`, found its location, and made the transfer.

Memory Boundaries and Swapping. CTSS used the modified 7094's memory boundary register to limit user jobs' access to only part of B-core, so that the supervisor didn't have to swap 32K to the drum for every job. An algorithm called the "onion skin" left parts of a big job's core image in memory if a little job ran next, and handled all the complicated space accounting and reassembly of images. If the drums filled up, swapping spilled to disk.

(I remember visiting SDC in Santa Monica, in 1967, and seeing their AN/FSQ-32 timesharing system there, which lacked these features. When you issued a command, the swap space was statically allocated, and if there wasn't enough to go around, you might have to keep trying. When the system printed `LOAD OK ON 9`, it meant you had successfully claimed enough drum to run.)

Indirect Addressing. The 7094 didn't recognize the indirect flag in an indirect word, but it had the `XEC` opcode, which executed an instruction out of line, and if you did `XEC *` the CPU would sit there taking "I-cycles," uninterruptible, until an operator manually reset the CPU. People were warned not to do this. If they did it while CTSS was running it froze the system, and operations took a core dump and restarted CTSS, causing all active users to lose their work. We system programmers would analyze the dump and identify and fuss at the user. (One such crash was caused in 1965 or so by Joel Moses, who was running a big symbolic integration program written in LISP. When we complained, he said it was an honest mistake, caused by using the `RPLACD` LISP command. I have been reluctant to use `RPLACA` and `RPLACD` ever since.)

Password File Story. The one time `XEC *` was used in a good way was the day in 1966 that a certain Computation Center administrator's mistake caused the CTSS password file and the message of the day to be swapped. So everybody's password was displayed to each user who logged in. (The editor in those days created a temporary file with a fixed name.) This was before (and was the source of) the idea of one-way encrypting passwords. Bill Mathews of Project TIP noticed the passwords coming out, and quickly entered the debugger and crashed the machine by entering an `XEC *` instruction. Naturally this happened at 5 PM on a Friday, and I had to spend several unplanned hours changing people's passwords. (The problem is described and analyzed in Corby's Turing Award Lecture.[10])

Terminal Access. By 1963 CTSS provided remote terminal service to dial-up terminals connected by modem to a specialized telecommunications computer, the IBM 7750. Model 35 Teletypes were used at first, followed by IBM 1050 terminals and IBM 2741s. The 7750 and the IBM Selectric terminals were designed for other uses, such as stock trading and airline reservations, and CTSS adapted to the design of these devices. We did specify that certain RPQs had to be added to the 2741, so that its keyboard would not lock up after every line. We also required that remote terminals on CTSS have a "terminal ID" that would be sent at dialup time, and CTSS users would look at the output of the `who` command to see where their friends and colleagues were connecting to the system from. Terminal access for Teletype terminals was at 110 baud. The 1050 and 2741 terminals could support 134.5 baud. All of these devices were supported over dial-up modems, accessed via a private phone exchange at MIT.

Notable CTSS Applications

There were other significant improvements to CTSS, some contributed by the user community.

Electronic Mail. Noel Morris and I wrote a command, suggested by Glenda Schroeder and Louis Pouzin, called `MAIL`, which allowed users to send text messages to each other; this was one of the earliest electronic mail facilities.[11] (I am told that the Q-32 system also had a `MAIL` command in 1965.)

`RUNCOM` and the Shell. Louis Pouzin also invented `RUNCOM` for CTSS. This facility, the direct ancestor of the Unix shell script, allowed users to create a file-system file of commands to be executed, with parameter substitution. Louis also produced a design for the Multics shell, ancestor of the Unix shell.

Command Abbreviation and Instant Messaging. Noel and I created a replacement command processor, `.SAVED`, partly inspired by the promised Multics shell features. It supported user-defined abbreviations, multiple commands on a line, and inter-user instant messaging. The combination of `.SAVED` and `RUNCOM` allowed CTSS power users to define their own productive command environments.

`RUNOFF`. Jerry Saltzer wrote one of the very first computer word processing programs, `RUNOFF`, for CTSS in 1963 and 1964. This program is the ancestor of Unix `roff`, `nroff`, and similar text formatting facilities for many other systems. Users edited the input files for `RUNOFF` with a special editor, `TYPSET`, that supported upper and lower case letters.

Jerry has placed the original CTSS documentation for `RUNOFF` online.[12] The source of `RUNOFF` is included in the Pierce Collection tapes.[13]

Graphics. The ESL[14] Display, familiarly known as **The Kludge**, was an interesting device attached to the Project MAC 7094. It consisted of a PDP-5 computer with a vector display scope and a controller that interfaced directly with channel D. This device could be used

[10] Corbató91. [11] VanVleck01a, Morris11. [12] Saltzer64, Saltzer66. [13] Pierce03, Pierce04.
[14] Electronic Systems Laboratory.

for interactive graphics with light pen and mouse input. It is described in a report by Dan Thornhill et al.[15].

QED Editor. QED was a text editor contributed to the CTSS community by Ken Thompson, a Multics programmer at Bell Labs. This line-oriented editor was influenced by the character-oriented QED editor on the SDS-940. One of Ken's major additions was regular expression searching and substitution. QED was ported to Multics BCPL by Ken and Dennis Ritchie. QED was programmable: it supported multiple buffers, and a user could execute the contents of a buffer containing editor commands. Some remarkably arcane editor applications were written using QED.

Corby's Leadership

Corby inspired his whole team with the vision of interactive, powerful, personal computing. Time sharing was necessary because powerful computers were so expensive that they had to be shared: hence the "computer utility." Our goal was to empower the individual programmer. Batch processing was still quite young, but its proponents were vicious in putting down any newer idea.

Corby also fostered a wonderful set of design and implementation skills in his team. Honesty, peer review, unselfishness, concentration on essentials rather than frills, and rapid production of results were things we learned from him. Tom De Marco wrote much later about "jelled" teams:[16] my time working for Corby on CTSS and Multics was when I first enjoyed one.

Story: Corby's Hardware Transient and the Blackout

During the fall of 1965, the Computation Center was trying to bring up CTSS on the blue machine, and requesting help from the former CTSS developers at Project MAC when they ran into problems they couldn't handle. The blue machine kept crashing in the disk backup software, in a way we'd never seen, and the dumps didn't make sense. Hardware diagnostics showed no problem; the IBM customer engineers insisted the machine was fine. Bob Daley and Stan Dunten, in particular, were called on repeatedly to try to figure out what was going on. Corby involved himself too, and hypothesized a "*hardware transient*" that zapped some registers. Finally Mike Padlipsky and Charlie Garman took over the blue machine in the middle of the day, and loaded a one-card program with a simple test loop, and left it running. After about ten minutes of execution, the failure count in an index register suddenly began counting up: they had caught the bug! (The bug was something like: the TXL and TXH opcodes sometimes reversed roles — in only one of the two core banks — if a big enough truck went by on Vassar Street.) The programmers heaved a sigh of relief, turned the machine over to the IBM customer engineers, and headed over to Tech Square House[17] to celebrate. Quite a few people came down to the bar for a drink, when suddenly the lights went out. I remember I ran up nine flights of stairs to help crank the tape out of the tape drives by hand, because the tapes could be damaged if they were in the drive when power came back on. I needn't have run: it was November 9, 1965, and the lights were out over the entire East Coast, and stayed out until the next morning.

Old Age

In late 1968, the MIT Computation Center returned the blue 7094 to IBM. The red 7094 from Project MAC was moved from Tech Square to the new Information Processing Center in building 39, and Multics developers shared CTSS resources with general MIT time-sharing users until Multics was self-supporting in early 1969.

CTSS continued in use for several more years, with gradually declining usage, as MIT computer users switched to OS/360 batch, CP/CMS timesharing on the 360/67, or to Multics. One project, called the Overlap Project, was funded to support 7094 usage until equivalents for some CTSS social-science tools were found or created on other machines.

[15] Thornhill68. [16] DeMarco99. [17] VanVleck01b.

There was one kind of funny problem with this long decline of CTSS: every year, there was a fire drill as we tried to remember what you had to do to create a system for the next year. The **Chronolog** clock attached to tape channel E of the machine provided a date and time in BCD, but without the year; that came from a constant in the supervisor, and each year we had to recompile one small module, and relink the supervisor, the salvager, and a few other standalone utilities. As the original CTSS crew gradually moved on to other jobs, fewer and fewer people knew how to carry out this task, but we always managed to figure out how. And each year, we considered making the constant a "configuration item" somehow, and decided not to bother, because the system wouldn't last another year.

The 7094 was a fine machine, but by the time it was over ten years old, it had become expensive to maintain, hard to find expert programmers for, and had been deserted by most users in favor of faster and more modern machines. A group of students proposed that they'd take over the machine and run it for student computing; the floor space, air conditioning, and maintenance costs made this impractical, but some of these students went on to found MIT's Student Information Processing Board,[18] which supported student computing at MIT in many ways, and still exists today.

When CTSS shut down, it was time for it to go: the hardware was becoming ancient and flaky. But looking at what it could do, and what people did with it, shows that many subsequent systems were less capable and vastly more wasteful.

There was a ceremony held in 1974, the year after we shut CTSS down for good, to honor those who worked on the system and those who supported it. A list of people involved was compiled.[19]

Figure 2.5: I was there the day Roger Roach finally shut down the red machine for good, on July 20, 1973.

[18] Chuu01. [19] VanVleck03

Documentation, Simulator, and Other Time-sharing Systems

The website version of this section includes a list of nine CTSS documents. These are included in the Bibliography section of this brochure.[20]

Dave Pitts has created a version of CTSS running on a 7094 simulator he created by substantially rewriting a 7090 simulator originally written by Paul Pierce. You can log in, execute commands, and compile and run MAD and FAP programs.[21]

During more or less the same era as the CTSS era, other time-sharing systems were developed elsewhere in the country:

- RAND's JOSS system was operational on JOHNNIAC in 1963 and was later ported to a PDP-6.
- The PLATO II system was built by Donald Bitzer at the University of Illinois and demonstrated in 1961.
- RAND's SDC Time-sharing system on the AN/FSQ-32, led by Ed Coffman, Jules Schwartz and Clark Weissman, was operational in 1963.
- Andy Kinslow at IBM built a system called the Time-Sharing Monitor System on a modified 7094. A paper describing it was published in 1964.
- Dave Evans and Harry Huskey at Berkeley built a system called Project GENIE on a modified SDS 930 in 1964.
- John Kemeny and Thomas Kurtz built the Dartmouth time-sharing system in 1964 on a GE-255.

Section 5 of this brochure describes several of those other time-sharing system that were developed in the MIT and Cambridge environs.

[20] Brandel99, Corbató62, Corbató63a, Creasy81, Crisman65, Fano64a, Pierce03, Pierce04, and Saltzer65. [21] Pitts10.

3. Uses

Our reading of the literature relating to CTSS suggests several reasons why the time was right for the system's development.

- The MIT Computation Center was swamped with demand for computing capability, and something needed to be done.
- Some users wanted faster (i.e., interactive) access to the computer — doing the same things they had been doing but without the delays inherent in batch processing in their edit-compile-run-debug cycle.
- There were some inherently interactive activities that could not be done in batch mode. Joe Weizenbaum's ELIZA program[1] that simulated the interactions between a patient and his or her mental therapist is an example of a system that was first implemented once time sharing was available.
- The idea of time sharing was "in the air." Someone was going to do it soon, and MIT computer researchers naturally were at the forefront of such developments. Also, getting CTSS demonstrable provided additional funding opportunities for research.

MIT was also a natural environment in which to develop a powerful computer time-sharing. A significant subset of users was willing to expend effort to learn something new and potentially beneficial. Thus, the system did not need to be oversimplified. For instance, it did not need to be a single language time-sharing system like the JOSS[2] system. With CTSS good balance between simplicity and powerful capabilities could be found.

CTSS was also able to evolve continuously. The system programmers were users of the system, and they put in changes rapidly and incrementally — not infrequent "releases." The focus on a mostly local community of users made dissemination of information about these frequent changes quite possible; and it got even easier when the documentation was moved to be online.

Relatively early in the development of CTSS, it became clear that the initial focus on providing machine cycles to users more effectively was only half the benefit of time sharing. An equally big, perhaps bigger, impact came from having an online file system and the resulting capability for sharing of data and programs and collaborations by users. Related to this, Tom Van Vleck has said,[3]

> *When CTSS was envisioned, we thought that interactive computing and remote debugging would be the big impact. It wasn't. Providing the on-line file system, storing user files on the machine instead of in card decks in the users' offices, turned out to be a much bigger impact, for two reasons. One was that people began to develop tools to operate on these shared files: to generate program source, to use data in multiple ways, to create new things from old. The other was that people found ways to share information through the file system. This started with "common file directories" and led to electronic mail . . . and later to inter-computer communication. The sharing effect created a community of users, able to cooperate on big projects and to evolve ideas rapidly.*

The rest of this section notes some of the ways CTSS impacted computing and some of the particular kinds of uses (and users) of CTSS.

---

[1] Weizenbaum66. [2] Ware08, p. 111. [3] March 2011 email.

The most direct impact of CTSS was with the creation of Project MAC and then the development of the Multics time-sharing system.[4] Corby and other MIT faculty and staff involved with the implementation or use of CTSS wanted to build a follow-on system to improve on CTSS, and that happened as part of the Project MAC laboratory that had been created partially based on the existence of CTSS.[5]

Because Bell Laboratories people were collaborators in the Multics effort before they left the project and created Unix instead, some of the MIT Multics people feel that Multics influenced Unix (as it no doubt did to some extent). However, the connection between Unix and CTSS may be even closer; Unix co-inventor Dennis Ritchie has said,[6]

> *In most ways UNIX is a very conservative system. Only a handful of its ideas are genuinely new. In fact, a good case can be made that it is in essence a modern implementation of MIT's CTSS system. This claim is intended as a compliment to both UNIX and CTSS. Today, more than fifteen years after CTSS was born, few of the interactive systems we know of are superior to it in ease of use; many are inferior in basic design.*

As already mentioned in this section and elsewhere in this brochure, time sharing was, in the early 1960s, an idea whose time had come; and CTSS provided a working demonstration of the feasibility of time sharing that no doubt influenced both the designs and motivation to build other time-sharing systems. CTSS's existence was also used to spread the idea of time sharing. For instance, the U.K. computer scientists David Barron and Maurice Wilkes gave demonstrations of CTSS to people in the United Kingdom.[7]

> *The demonstration of time-sharing at Oxford in 1963 was a demo of CTSS (Compatible Time Sharing System) . . . Also, the Oxford demonstration wasn't the first: CTSS had been demonstrated by Wilkes and myself at the BCS conference at Edinburgh a year or two earlier, and before that a few of us had used CTSS at Cambridge courtesy of the GPO. All the demos used a 10 cps Telex connection.*

Additionally, CTSS provided a source of real measurements on how time-sharing worked that impacted the theory of time sharing and the design of other time-sharing systems.[8]

CTSS also has been used over the years as an example or case study in classes on operating system design.

Tom Van Vleck makes the points[9] that time sharing at MIT was rapidly adopted for a couple of reasons: (1) "There was some pent-up demand for computing cycles. Researchers who had always wanted to compute stuff and finally could. Other researchers thought up stuff to do because the resource became available." (2) Some of the quick uptake in use was perhaps fostered by the funding agencies suggesting that researchers use the new time-sharing technology to see what they thought about it. "Many different departments participated in this largesse. Computer time cost hundreds of dollars per machine-hour in those days, but you could get an allocation of time by writing a research proposal and some progress reports."

One can get a feel for the expanse of early use of the CTSS system by looking at the tables of contents of the progress reports on Project MAC from its inception to July 1965 that are included on pages 353–356 of Fano79.[10] In the rest of this section we list a few specific, and representative, projects that used CTSS. These systems, in turn, in some cases had great impact on their respective fields.

- Under the leadership of Professor Charles Miller, MIT's Civil Engineering department developed the Integrated Civil Engineering System (ICES), with its COGO (COordi-

[4] Corbató65. [5] MIT92f. [6] Ritchie79. [7] June 27, 1995 email found on the Internet: from David Barron to `mhouston@mh01.demon.co.uk` on with the subject line `Linux@UK:Lettertotheeditor`.
[8] For examples, see the interviews of Scherr and Denning on page 35. [9] March 14, 2011 email.
[10] Not all of the activities listed in those progress reports were carried out on CTSS, but undoubtedly most of them were.

nate GeOmetry) and STRESS (STRuctural Engineering System Solver) programming languages that allowed non-computer-programmer engineers to make effective use of the computer. ICES included a number of other subsystems relating to various common civil engineering activities.

- Doug Ross's work with Automated Engineering Design (AED) and Computer Aided Design (CAD) resulted in the AED-0 programming language and changed the way computing was done and what was thought of as computable. In particular, it addressed interactivity and data abstraction. AED-0 was used, in turn, to help implement many other MIT computer system projects.
- The Kludge display connection to CTSS was mentioned in Section 2 (page 12). The "AED Applications" notes[11] list many many MIT research activities that used this display and thus CTSS. One of these applications was a project led Professor Cyrus Levinthal that was the beginnings of interactive molecular graphics.[12]
- Joe Wiezenbaum's most CTSS famous project was ELIZA, but he also developed the OPL-1 system on CTSS, an incremental programming system providing an early model for how a user to augment both his program and his data base during successive, widely separated sessions at his terminal.[13]
- MIT's Project INTREX[14] was another use of CTSS. This early, innovation library information system project, led by Professor Frank Reintjes, was in some sense the earliest search engine.

  An important and widely demonstrated component of INTREX was the TIP (Technology Information Program) system developed under the leadership of Dr. M. M. Kessler.[15] TIP was the first system to do "bibliographic coupling," to allow saving of search output to be the basis for later searches, and to enable "stem searching."
- OPS-3, developed by Professor Martin Greenberger et al.,[16] was an "on-line process synthesizer," a simulation system depending on CTSS to enable the user to iteratively develop the simulation.
- Finally, CTSS was constantly used for ordinary mathematical calculations (and various other kinds of problem solving not requiring user interactivity) by various MIT research projects, because of the convenience and turn-around speed of the system, and because of the availability of TYPSET[17] on the system.

  The DYNAMO (DYNAmic MOdels) system, created by Jay Forester's team, for doing systems dynamics simulations[18] is one such system that benefited from running on an interactive time-sharing system. While DYNAMO existed before CTSS, and batch mode users could input punch cards specifying the simulation network and its parameters and then run the simulation, being able to immediately see the results of one simulation run and change the simulation network or parameters and do another run, and to repeat the cycle over and over, helped users get a useful result faster.

CTSS had big impact in various application worlds not only because it was the first time-sharing system. MIT was also a major early computer science research center, and the Institute already had relatively major digital computing facilities and many researchers in many disciplines willing to jump into the new digital and then interactive digital world. Many MIT areas of research benefitted from the availability of CTSS, but CTSS was in the right environment to have its capabilities utilized for many things and

---

[11] `http://www.csail.mit.edu/timeline/timeline.php?query=year&from=1963`
[12] Francoeur02. [13] `http://publications.csail.mit.edu/lcs/specpub.php?id=578c`
[14] Burke98. [15] Bourne03. [16] NANCE96. [17] CTSS command names were stored as a maximum of 6 characters in the 7094's 36-bit word. [18] Richardson81.

to have new capabilities developed on top of it. It is not surprising that computing technology and use of computing technology developed rapidly in the MIT environment.

Also, unlike some of the other well known early time sharing systems (e.g., the Dartmouth Time Sharing System in its initial implementation as a system for small student projects written in Basic), CTSS from its earliest operational use provided the entire computer and all capabilities that could be run on the computer to its users. Building an "open system" instead of a "closed system" was a big job, but it let users build big things on top of it, including letting combinations of user, programming languages, databases, and subsystems work together.

4. **Memories and views of CTSS**

Parts of this commemorative brochure were drawn from existing historical records, such as those listed in the bibliography. However, we also took the opportunity to talk (by phone and via email) with various people who were associated with CTSS over the years or who have a historical view of CTSS. These notes and transcripts follow. The order of the notes and transcripts is approximately the order of each person's initial involvement with CTSS.

Marge Daggett and Bob Daley were project founder Fernando Corbató's initial collaborators in the design and implementation of CTSS. Dick Mills, Tom Van Vleck, and Roger Roach were (as Tom sees the history) the first, second, and third and last system administrators (to use modern terminology) for CTSS. Allan Scherr and Peter Denning studied CTSS as part of their thesis research. David Alan Grier has considered CTSS from the computer historian's point of view.

In what follows, we did not attempt to completely reconcile slightly differing memories of historical details.

Fernando Corbató[1]

Phil Morse was the head of the Computation Center. It was his initiative and shrewdness that led to the formation of the Computation Center in 1956. He proposed the deal that MIT would get one eight-hour shift of time on an IBM 704 computer in return for MIT administering a second eight-hour shift of time for the 40 or so New England Colleges. In turn, IBM would maintain the computer system and get to use the third eight-hour shift for its own purposes.

Thanks to Morse's initiative, MIT suddenly had to find a location for the new Computation Center. Building 26 was at that time already under construction with a plan for the building to be "floating" on stilts, (the fad of the day). To the architects chagrin, he was forced to create a ground floor, computer room, and basement to accommodate the new Computation Center. The architect got his revenge by making the ground floor be without exterior windows (only transoms) and encasing the outside walls with blue tiles to simulate the upper floors of the building floating in the sky.

Morse was also instrumental in getting funding for a dozen Research Assistantships from the Office of Naval Research, and later getting funding from the National Science Foundation for the second 32K word bank of memory to be added to the by then IBM 709 for CTSS to use. Morse's style of management was to spend one intensive day a week meeting with the senior staff of the Center. My recollection is that it was at one of those Meetings in 1959 that John McCarthy presented to Morse his memo outlining the three key changes (RPQ's) that would make the IBM 709 suitable for timesharing.

I have lost track of who carried the rental of the initial IBM 7750 which controlled the various terminals, but it may have been IBM. Similarly, the large IBM disc file that was

---

[1] After publication of the first version of this document, Corby sent the editors (2011-06-24 email) these notes relating to some uncertainty he sensed in our understanding of Professor Phil Morse's role in the management of the Computation Center.

brought in for program and data storage may have been by IBM. A key reason for thinking so is that the IBM liaison to the Computation Center at that time was Loren Bullock, and I remember he tried very hard within IBM to persuade the upper management of the importance and potential of the MIT timesharing work.

Marjorie Daggett[2]

Marge majored in math, taught for two years, and then spotted an opening in a group in Building 20 doing calculations, differential equations, and so on for an engineering lab.

In the mid-1950s the Whirlwind computer was opened to the MIT Community. Marge's co-worker (another woman) was sent there for training, and then the co-worker came back and taught Marge about coding for Whirlwind.

A Prof. Frazier had lots of calculations to do, and they decided it was appropriate to do them on the card punch calculator that was part of the activity run by Prof. Frank Verzuh. She and her co-worker coded this application and others, and it seemed clearly faster and better than how they had been doing them with more manual equipment in their lab. Marge and her co-worker were spotted by Frank Verzuh, and they were offered jobs in approximately the summer of 1957 when he was staffing the computer center. Early on they took a course in symbolic assembly language programming given by Corby.

There were four of them in the group, and they did a bit of everything: programming for people, doing demonstrations, and consulting to people.

Frank Verzuh was involved in a project to compare the effectiveness of programming in assembly language versus FORTRAN. Marge recoded equations from before, and it was easier. One of her first assignments was to learn FORTRAN and then to code a program in FORTRAN that a co-worker was coding is assembly language. The FORTRAN program took a considerably shorter time and gave a clearer statement of the algorithms being used.

Marge consulted to various MIT professors and research assistants who wanted to use FORTRAN, including helping people understand error messages from FORTRAN. She helped lots of people with lots of things because she was one of the first people doing this work. For a short time she was the "FORTRAN expert" until others in the group learned it.

John McCarthy was also around, in particularly looking at the compiler and suggesting various beneficial subroutines, and Marge coded these for him. Working with McCarthy helped her see a bigger picture of computing.

Some programs run in the computer center were quite large, and people were thinking about run times, ways to speed things up, etc.

There was also a development systems group "upstairs," which was working on an assembly language operating system [or was it an operating system oriented to assembly language programming?] which Marge called the MIT Operating System. Corby was there as was Dean Arden who was looking into modifying for efficiency the assembly program, SAP. Others who were upstairs included Sheldon Best and a couple of other people who had come from the Whirlwind Project. Arnie Segal was also upstairs.

Marge's group was downstairs and had 6 or 7 programmers who did consulting for other computer users and who had learned the operating systems in order to fix

[2] These notes are based on a 2011-03-04 telephone interview of Marge by David Walden. Walden explains: I started by asking Marge my first question, "What is the background that got you to MIT, to the 7094, and then as part of the CTSS project?" Marge began to answer and I never asked another of my prepared questions — just an occasional follow-up question. I took hand-written notes on the conversation and after the call typed up my notes as a more complete, third-person narrative of what I understood Marge to have said. I sent the more complete narrative to Marge for review. Marge sent me her corrected and updated version of the notes in an email of March 6. That revision is included here.

problems people ran into. The downstairs group was responsible for operations and thus gained experience with all the standard systems and programs running at the Comp Center.

At some point Frank Verzuh left, and there was a big reorganization with Corby taking Verzuh's position as deputy director of the Comp Center. By this time the other people upstairs had left (e.g., to go to industry), and it was clear that symbolic assembly language programming was not the wave of the future. With responsibility for the entire Comp Center, Corby moved downstairs. This group became responsible for both maintenance and eventually the time-sharing project.

Marge continued to learn things. At one point Dick Steinberg and she were went to IBM's "Inside FORTRAN" course. Modifying FORTRAN was not an easy possibility. After that Marge was sent to Michigan for a short course on "Inside MAD."

Marge and others were familiar with all the systems they were running, and Corby starting thinking about time-sharing with them. Marge said that at the same time Herb Teager and John McCarthy were talking about human-machine interaction with the goal of making debugging easier.

Marge said that Corby certainly wrote all of the first paper on the experimental system, and Corby certainly wrote a lot of code in the supervisor — all of the work on the queues was his. At the beginning Corby and Marge were the only two working on it, and they spent a lot of time conferring and hashing and re-hashing problems, etc.

Her early involvement was with interrupt handling, saving the state of the machine, various commands (and what was a command and what was not), character handling, and also crude methods for inputting and editing lines for the demos. She remembers a lot of "back and forth" about what various states were: the distinction between dead and dormant, waiting, etc. And then there was the whole area of queuing and what was ready to run which was done by Corby. She worked in the "underpart of the supervisor." She remembers having to do testing at odd hours and sitting with a listing after 5pm trying to figure out what went wrong. She also remembers it was a big exciting day when it finally worked for two typewriters. There was still a lot to do, but conceptually it was pretty much there — very exciting.

I told Marge that Bob Daley had told me that he came to work for Frank Verzuh in a non-programming position, that she had spotted him, and invited him to join the programming group; and I asked how that came about. Marge said the it was obvious to all that Bob, whose job was keeping various pieces of equipment such as punch card machines going (and whatever else he was doing for Frank Verzuh) was clearly a bright and motivated person who was studying (and had already taught himself a tremendous amount) and eager to learn, and he should be given a chance.

I asked Marge how she felt when working on CTSS, and she said that it seemed like it might unfold into something big, and it was terribly interesting to work on in the beginning. On the other hand, a lot had to change (particular in the hardware) to enable time sharing. The system grew over the years, and then the evolution into Multics, and the beginnings of Multics was very exciting.

Marge married in February 1962; and, after the birth of her child (for which she took a six months leave — two months before and four after), she came back part time in May 1963. After that she was no longer involved in mainstream development. She fixed things, she worked on specific commands, etc.

At some point Marge went to Michigan for 3 or 4 days to take a course from Bruce Arden, Bob Graham, and Bernie Galler on MAD. At MIT they had some big programs that tested MAD's capabilities. When there were problems, they tried to get help from Michigan, but the Michigan people didn't really have the time to help. So Marge learned a lot of programs that used MAD, which could be called as a subroutine.

Marge mentioned R. J. Creasy (who died in 2005). Her memory is that Bob Creasy had some highly responsible project that was part of developing the CTSS supervisor.

I told her Bob Daley told me that Creasy had "gone downstairs" (in the Tech Square building) to help IBM get into time sharing. Marge noted that Lynda Korn also went to IBM.

Marge also mentioned Dan Roos who was one of the research assistants that came from different departments, who she remembers working on F2PM — the FORTRAN 2 post-mortem software.

Marge shared an office at one point with J. R. (Dick) Steinberg. When the interval timers came, she and Dick Steinberg talked about the necessary subroutine expansion for time-sharing, and she coded it and later expanded it from the prototype Dick had written. Dick was operations manager when Corby became deputy director, and the programming staff worked on the operations staff and on time sharing.

Dick Orenstein was hired as a regular employee; he was not a research assistant, and he also worked on the post mortem routines.

Marge added during her edit, "I also would like to say or probably add to what others have said: how supportive and fair Corby was in all our work and how generous in his credits. Some authors of papers, though not in computing, relegate the co-workers to footnotes."

Robert Daley[3]

Bob stated that his getting to MIT, the 7094 and CTSS was serendipity. He was bored silly in mechanical engineering at Tufts and dropped out. He then went to night school to learn to program accounting machines, found it very exciting and aced it. Programming was entirely based on plug boards and timing was critical, based on which row of the card was being read at the time. This turned out to be a big advantage in dealing with I/O on the 704.

He saw an ad for the MIT Computer Center and was hired by Professor Frank Verzuh, to take care of the SHARE library. He also did accounting for the IBM 704 system on a 407 tabulator downstairs. It seemed silly to not be doing the 704 accounting on the 704 itself, and Bob taught himself to program to do that. There was an IBM guy (Dewey Manzer) in the IBM liaison office, helping Bob learn 704 assembler language. Bob tried to talk to him about how to do I/O. The IBM guy said I/O was too complex, and one should just use the SHARE library subroutine calls. Bob took this as a challenge to learn all about I/O and to become an I/O expert.

Marge Merwin (later Merwin-Daggett) was leading the programming team, saw what Bob was doing, gave him a test which he passed, and he became a DSR [Division of Sponsored Research] staff member doing programming.

An aside: Professor Frank Verzuh taught 6.251, and he had Bob grade student assignments, e.g., of Bill Poduska.

In time Corby took over running the Computer Center and he had a vision of dividing computer time among multiple users using a time-slicing algorithm. Corby began to talk to Marge and Bob about this. (I asked Bob why he was one of the two people Corby was

[3] These notes are based on a 2011-02-28 telephone interview of Bob Daley by David Walden. Walden explains: I started by asking Bob my first question, "What is the background that got you to MIT, to the 7094, and then as part of the three person initial CTSS design team of you, Corby and Marjorie?" Bob began to answer and I never asked another of my prepared questions — just an occasional follow-up question. I took hand-written notes on the conversation and after the call typed up my notes as a more complete narrative of what I understood Bob to have said. I send the more complete narrative to Bob for review, and he sent back his edits in an email of March 2. Then, on 2011-05-31, Bob answered three clarifying questions, and the following is the final result.

talking to about a time-sharing system design, and Bob noted that he was young and energetic, but couldn't say more than that. It sounds like Corby recognized some spark in Bob.)

Bob says that Corby wrote the core of the time-sharing system himself, using three tape drives to swap users out of memory and another three tape drives to hold user data. User data could be added to the end of the user data tape. This 3-user system demonstrated the basic principals of time sharing. It was demonstrated frequently, according to Bob, and he can't name a specific demonstration date.

Eventually Corby began to back off from the actual programming and Bob took over the supervisor and Marge took over the rest. They also hired some more people. He remembers Linda Korn and a Norwegian woman [Gunn Lytell according to Marge Daggett]. I asked if the additional people were full-time hires or students working part time, and Bob thought some of each. Dick Orenstein was an early addition to Marge's staff.

According to Bob, Corby had/has always been championing time sharing. He had a vision of multiple users each with the belief that each user had total personal use of the machine (but he noted there was an omission in that vision that he would get to later). Briefly, the early vision (at least as Bob understood it) did not include the concepts of data file sharing across users and the access controls that would need.

In time they got a disk drive. One day Corby gave Bob a document on a 1301 disk from IBM and told him that they were getting one of these. There was no existing software for the 1301. Corby had some ideas about how to use the 1301 but basically told Bob to figure out how to use it as part of CTSS. Bob developed the first user file system (with a backup capability) for a time-sharing system. Later he added a capability for incremental backups, hierarchical directories, and access control (needed with the ability for users to share data files).

Bob noted three major innovations involved in CTSS. [He noted in his round of edits that there were probably more.] The first was having the disk drive resulting in a file system for user data. The second innovation was the RPQ that resulted in IBM providing a protection and dual core system. The supervisor could reside in core A and a user program could reside in core B. If the user program did an I/O or other protected operation, control was transferred to the supervisor in core A. The third innovation was using the 7750 communications processor for dealing with a large number of communications lines to time-sharing terminals.

The initial version of CTSS supported only 3 Flexowriter typewriter terminals. It was later upgraded to using IBM 1014 terminals (this is what is being used in the John Fitch video,[4] where Bob is the guy in the corner operating the computer). Next they had IBM 1050 terminals. By the time they had a large number of terminals they were using the IBM 2741. Both the 1050 and 2741 terminals required an RPQ to allow a "reverse break" where the user spoke first to the computer rather than following the IBM assumption that the computer always controlled user communication and the computer asked the user if it wanted something from the user. In time CTSS would support more modern terminals such as DEC displays, etc.

CTSS users used assembly language and MAD, and Fortran could be run in the background.[5]

From the beginning, users wrote application programs, for instance, Jerry Saltzer's TYPSET and RUNOFF, which Jerry wrote to help him write his thesis but which became a model for other word processing and typesetting systems.

Returning to the protection thread, if a user program did I/O, etc., the system

[4] Corbató63b. [5] Tom Van Vleck explained in a 2011-02-28 email, "CTSS users could not use FORTRAN in foreground. It was a hairy multipass compiler and would have required deep source changes to make it work in CTSS. Not sure we had the source of this IBM program product, even back in the good old days. By 1964 or so, there was a CTSS command called MADTRN which pre-processed FORTRAN into MAD and compiled that."

trapped, and then the supervisor would attempt to simulate the I/O. Well behaved FMS programs could run in the background.

When Bob went to Poughkeepsie to check the protection RPQ, R.J. Creasy went along. Bob Creasy was Bob's office mate. The idea of I/O emulation was a key idea to Bob Creasy, and he went downstairs (from the 5th floor of the Tech Square building to the 4th floor where IBM was), and the concept became CP67. (Bob also remembers that Arnie Miller did the changes that turned the 65 into the 67.) Dick Bayless also went downstairs and created CMS. The first version was CP40, implemented on a 360 model 40. The model 360 model 67 came later and CP40 became CP67 (later renamed to VM370).

As Bob understood it, Corby's vision was a personal machine for each user, and they originally implemented that; but Bob says that they did it too perfectly. They discovered that users want to share files, and thus they needed to implement access control and Bob did that for a second version of the file system. (Then Bob moved over to Multics where he implemented the Multics file and virtual memory system. This was a big evolution, since it combined the concept of data files and virtual memory. Multics had the first industry implementation of data file mapping into a virtual address space.[6])

The idea as the Multics project began to heat up was that the CTSS developers would move to Multics over time leaving the Computer Center staff to run CTSS. However, the transition was a gradual one and took perhaps 18 months.

Bob went on to talk about others, particular a succession of office mates of his, who left CTSS and had impact on other time-sharing systems. Already noted above are the people who "went downstairs" to create CP/CMS. Stan Dunten went to Dartmouth where he was involved with Kurtz et al. Jerry Clancy went to Data General to do RDOS which was CTSS like. Bill Poduska, who later taught 6.251, went to Prime where they did PRIMOS which was CTSS like. Dick Orenstein, who had worked with Bob on the file system, went to "ADP (or something like it)." Bob also notes that CMS "Script" is a clone of TYPSET. Also, Tom Hastings left to go to DEC where CTSS concepts evolved into TOPS-10.

Bob himself did an editor called EDL on CTSS, which he reimplemented on Multics as EDM. (Then, using the EDM architecture, he built qedx for Multics.) All of these implementations were patterned Saltzer's TYPSET "dual-file editing philosophy" as Bob wanted to provide an environment for programmers that was as good as the TYPSET environment was for writers.

While Bob was working on CTSS, he was excited about the project, but he did not realized how seminal it would be.

I asked where had Marge had gone, and he said he thought she simply had gone off [as was the custom in those days] to have a family.

Richard Mills[8]

Q: Please tell me a bit about how you got to MIT, then into computing at MIT, and then involved with Project MAC and CTSS.

---

[6] Tom Van Vleck has a high opinion of Bob Daley, saying:[7] "Bob was the 'chief engineer' or 'implementation architect' for CTSS and then the initial development of Multics. He was the final arbiter for what features were in and what were not, and the strategist for how things could fit together. His good taste, his ability to focus on which issues were essential, and his flat-out programming chops were invaluable. We could not have gotten Multics implemented without him.

[8] Interview of Dick Mills by Dave Walden. Walden explains: I sent an email to Dick with some questions, and he sent back his answers in a 2011-03-15 email. I sent some follow-up questions to which he replied in a 2011-03-19 email. I merged the two sets of answers into a single set of questions and answers and sent it to Dick for review. Dick did some revision of the merged version and sent the final version back to me in a 2011-04-01 email.

A: Not-so-quick summary of my MIT careers (plural):

September 1949. Arrived to start a five-year-plus-three-summers program leading to a BS and MS in Electrical Engineering. In 1951, switched to BS in EE plus BS in Management on the same timetable. In 1953 met Bob Fano, Corbató, Doug Ross, Jay Forrester, Phil Morse, many other computing earlybirds. Programmed Whirlwind I to process data for my EE/Management thesis. Graduated June 1954; continued to pursue thesis topic using Whirlwind remotely — by snail-mail; week or two turnaround time.

June 1954 to September 1958. A non-MIT period, except for continuing remote use of Whirlwind (I was on a GE management training program, interrupted by a period of active military duty as Air Force pilot).

September 1958 to June 1960. I returned to enter the MIT Sloan School. I was during both grad-school years a research assistant in the MIT Computation Center. I worked with or met Corbató, Merwin, Daley, Teager, Dennis, Licklider, Saltzer, McCarthy, and others. For my masters-degree thesis I designed, wrote and ran on the 704 a program to implement and then experiment with an uncommon algorithm for solving linear-programming problems.

June 1960 to March 1963. I helped to start and grow a Route 128 entrepreneurial consulting and contract-programming company. In early 1963 we took it public; I sold my interest soon after.

March 1963. Bob Fano hired me to help him start Project MAC. Soon I was named Assistant Director of the Project. My first assignment as MAC's first employee was to play a major role in planning and administering the 1963 Project MAC Summer Study that launched the Project. During the Study I met many of the then leading lights in computing at the time — Wilkes, Samuel, Schwartz, Amdahl, Engelbart, Sutherland, Fredkin, Minsky . . . on and on. During four years as the Project's Assistant Director and business manager, I was also involved in decisions on Project direction, strategy, Multics hardware specification and acquisition, CTSS evolution, refinement, operation, and communication-system integration. The latter, incidentally, was a major undertaking, given the voice-optimized analog telecommunications plant that existed in the United States, and the world, at the time.

September 1967. I was asked by MIT President Johnson to leave MAC and assume the position of Director of Information Processing Services (DIPS), reporting, alongside the five academic deans, to the Office of the President. My charge was to rationalize and coordinate (de facto, direct) all computational services and facilities on the campus. All campus computing facilities reported directly to or were closely coordinated by the DIPS. In this role, with its ex-officio faculty appointment, I was able to accomplish a major transformation and rationalization of the management of computing resources on the MIT campus.

December 1969. With my mission accomplished, and the CIO job at a major international bank in view, I departed MIT for what was to be the last time. I have looked back only once, very briefly.

Q: What was your role with CTSS?

A: Primarily, administration and operation, first as Assistant Director of Project MAC, later as DIPS. I am not a good source on the pre-1963 days of CTSS, but I think that by the time of my June 1960 graduation from Sloan, the concept existed and some early experimentation by Herb Teager had taken place. When I came back to MIT in March of 1963 to become Assistant Director of MAC, CTSS was fully, if sometimes feebly and failingly, in full operation on the Comp Center's highly modified 7094. The

system soon became useable enough to support the Summer Study. At MAC I was the Project's administrative chief. I "owned," operated, and resource-allocated the entire plant — computer access and capacity, space; the works.

As to my contribution to core CTSS development, it was very small. Relatively late in the development process, when the system had become quite stable, I took Corby's concept of a multi-level priority-queue scheduling algorithm and tweaked it to fit the real-world need that emerged from our "computing utility" conception of time-shared systems. Thus I played a leading role in transitioning CTSS from a computer-science experiment to a multi-user computing-service facility.

Q: In taking "Corby's concept of a multi-level priority-queue scheduling algorithm and revised it to fit the real-world need that emerged from our 'computing-utility' conception of our system," did you motivate and supervise this transition, or did you do coding yourself, or both?

A: A really long answer is needed here, but let's see if I can give you enough to meet your need. Tom Van Vleck, by the way, is a heavy-hitter in exactly this area — he was an invaluable resource involved in designing, implementing, and especially in running the complex resource-allocation system that we created for "production" CTSS.

The shortest answer to your specific question is "all of the above." I motivated, supervised, and did a lot of the coding. As far as the core-CTSS code is concerned, I created nothing new, but I did some tweaking of the existing scheduling and queue-management code. One of the basic design concepts implemented in the system (due to Corby) was that it must defend itself against overload. Illustration: as one tiny part of carrying this concept into our approach to scheduling and resource management, I used a dynamic doubling of the length of the "tick" — the length of time each user was allowed to run between clock interrupts — in response to an indication of impending overload. This of course had the effect of more or less halving interrupt-processing overhead. Was I the first to do such a thing? I don't know. I doubt it, but more important, I could not possibly care less.

We (Who is "we"? Well, certainly "I", but surely there were others involved. And again, I make no claim of primacy, a subject in which I have zero interest.) Whatever.

Anyway, "we" created the "standby user" concept. We set a fixed number (which increased as the system improved) as the nominal number of users who could be given an acceptable level of service. But after that number had logged in, as long as the system was providing acceptable responsiveness, we allowed additional users on — with the caution that they were subject to, and they indeed suffered, the dread "automatic logout" if things began to slow down. We thus implemented another basic concept: we chose to give acceptable service to a smaller number of users rather than poor service to a larger number.

External to core CTSS, Tom and I and others created a complex process, which ran as a privileged CTSS user, that each night at midnight updated the time-used vs. time-allotted accounts of registered users. We called this "turning the crank," and Tom was a principal authority in managing and operating this machinery — which a small group of us, led by me, had designed and coded.

Anecdote: Corby once quipped to me that I was taking the "carburetor-designer approach" to the modification of system scheduling — "we'll drill a new hole right here . . . why? Well, because it just feels right!" He was correct. I never based the work we were doing upon a solid underpinning of analytical results; instead we sort of just made it work — often by "drilling a hole here" because we thought it might help. Maurice Wilkes never stopped chiding me for never publishing any of this, and he finally wrote his own paper on the subject, which in part described some, but by no means all, of what we did.[9]

[9] See "Additional note" BY Tom Van Vleck on page 33.

Q: My understanding is that having CTSS in existence helped with the creation of Project MAC. If this is correct, please tell me about how you saw CTSS and Project MAC's relationship.

A: CTSS was first a demonstration of accessible computing — that the capabilities of a large computer could be presented to a number of users much more nearly on the user's terms (as to time, place, and ease of access) than the then-traditional ritual presentation of card decks, 24-hour turnaround, and massive bundles of paper from high-speed printers. CTSS, in its latter manifestations, was a fully functional multi-user computer service facility, accessed from distant terminals.

MAC was created to take the capability realized, demonstrated, and used "in production" in the CTSS instance to a new level by defining a hardware platform that had been designed to cater to the special needs of multiple access and to develop an operating system built from the ground up to manage that hardware as a time-shared resource. Time sharing on the 7094 was only made possible by standing the stock 7094 on its head by means of fundamental hardware modifications — e.g., bolting on a second bank of high-speed memory and creating an interrupt capability where none had existed.

For a significant span of time, CTSS ran on two independent copies of the enhanced-7094 hardware that existed at MIT, one in the Comp Center (the "Blue Machine" in local parlance) and the other at Project MAC (the "Red Machine"). The former was run as a service facility, providing time-sharing service to campus users, most of whom probably had no interest whatever in computers, but all of whom needed computing service. The MAC machine was the central resource to MAC-based research and, for a significant period, to the developers of the Multics system. (It would be wise to get Corby's endorsement, or correction, of the above. I fear that my recollection of the detailed time line for the intertwined events of CTSS-MAC-Multics has become very foggy over the years.)

Q: How did the system at Tech Square come about?

A: MAC was created, in part, to reduce to practice the capability of Multi-Access Computing, and to explore the benefits thereof, including its possible role in what we called Machine-Aided Cognition. CTSS demonstrated the former and was an early platform for the latter. Our initial vision of the progression of the work of the Project was that it would (1) continue to develop and to exploit the 7094-based CTSS platform while (2) developing an improved time-shared operating system that would run on current-technology hardware and (3) while collaborating with firms in the computer industry to specify, implement and exploit new-technology hardware that would have been, with MAC-developed insights, designed from the ground up to be time shared (instead of being forced by ad hoc modifications to support, awkwardly, a time-sharing operating system). As events rolled out, the Project was able to skip step two of that sequence, because sooner than expected we found a platform — the GE 635 — that was close enough to what we needed so that we could jump — with a significant amount of hardware-development pain — to step three. All of that is background to the answer to your question: MAC needed an instance of the then version of CTSS (1) to develop further, while at the same time, (2) to provide users with the means to begin to learn by hands-on experimentation what truly accessible computing meant in their various fields of endeavor. This need was, of course, inconsistent with the (nominal) mission of the Comp Center, which was to provide stable, dependable computer service to the campus. Therefore MAC needed to implement its own instance of CTSS on what became the "Red Machine" in MAC quarters in Technology Square.

Q: What was your role in making that happen?

A: I was directly responsible for all aspects of the build-out and the operation of the Tech Square facility, which of course included the acquisition, installation, and operation of the Red Machine.

Q: Please tell me what you can about how the CTSS project was organized, where the staff came from, how big was the staff, who was on the staff, how the project was managed, etc.

A: Corby is your man here. Most of the early CTSS development happened during the time I was either in the Air Force or working for GE. Of course, a lot of refinement continued after my March 1963 return to MIT. It was done primarily using Comp Center, not Project MAC, staff, though the line between the two was often pretty blurred. Except for briefly, as indicated above, there really was only one development staff, albeit a growing and evolving one. In internal-bookkeeping terms the developers were initially charged to other than MAC accounts (initially, of course, there was no MAC), but whichever account paid them, and regardless of where they sat, it was the same — growing and evolving — crew. You should probably check that statement with Corby, but I think he will agree.

Q: Did changes to one of these systems migrate to the other?

A: Initially all concerned exerted considerable effort to keep the systems as close to identical as possible. On some date between 1963 and my move to become Director of Information Processing Services in 1967 — I don't remember when, but Corby will — MIT hired from the outside a new Associate Director of the Comp Center to replace Corby in that position (MAC needed all of Corby — or better, two or three of him!). That individual, and a cluster of students gathered around him, chose to encourage the Comp Center staff to become less dependent upon the version of the system running on the Red Machine and on its development and support staff. As I recall, even with their new independence, the Comp Center staff actually made few — probably no significant — mods to core CTSS on the Blue Machine. They fiddled with the scheduling algorithm and such, but not much else. But even so, as a result (and in my personal opinion) the quality of Comp Center service to the campus' non-CS community declined materially during that era. The new Comp Center Deputy Director departed MIT more or less simultaneously with my assumption of the newly created DIPS role and, temporarily, of the position of Director of the Comp Center.

Q: Can you tell me about the steps in the evolution of the system from an idea to a design, to a prototype, to something demonstrable, to something operational.

A: By the time I returned to MIT to become MAC's assistant director (March 1963), CTSS on the Blue Machine was fully demonstrable, and by the start of the 1963 Summer Study (in June), it had stabilized to the point at which it, with a few wobbles, did support experimentation with time-shared computing by a stream of top-drawer practitioners, each of whom spent up to six weeks as guests of the Study. Major refinements in CTSS were still to come, such as enhanced access control, a totally new file system (which defined to a great extent the file systems of today), and resource-management strategies.

Q: When you became Bob Fano's deputy for Project MAC (i.e., Assistant Director of MAC), did you sooner or later take on responsibility for the system in Building 26?
A: Yes, but later, not while I was at MAC. I acquired formal ownership of the Comp Center (Building 26) organization and equipment in 1967, when I became Director of Information Processing Services. As soon as the DIPS position was created, with me as its first occupant, I immediately assigned to myself the additional position of Director of the Computation Center. It was only later, after I had firmly (and sometimes painfully) established the service-only, as distinct from CS-research, mission of the Comp Center, that I hired Weston J. ("Wes") Burner to replace me as Comp Center Director.

But Corby, and therefore MAC, and therefore I, had significant involvement with the Comp Center even after we installed the Red Machine at MAC. Remember that the Red Machine was MAC's development system. The Blue Machine, even in 1963, was nominally a campus service facility, although some interests in Building 26 continued to view it as

a CS research facility, fiddling with the OS, etc., sometimes to the detriment of reliable service to campus users who needed reliable computational service.

Q: In your first answer you said you were, "Asked by MIT President Johnson to leave MAC and assume Dean-rank position of Director of Information Processing Services (DIPS), reporting to the Office of the President, with the mission of rationalizing and coordinating (de facto, directing) all computational services and facilities on the campus. All campus computing facilities either reported directly to, or were closely coordinated by, the DIPS. In this role, with its ex-officio faculty appointment, I was able to accomplish a major transformation and rationalization of the management of computing resources at MIT." Were CTSS and Multics part of this?

A: By "CTSS" in the above, you must refer to the Red Machine at MAC, because CTSS on the Blue Machine was a part of the Comp Center — the campus computer-service facility — and perhaps more obviously my responsibility. But the short answer is still yes, of course. MAC computers were campus computing facilities, so responsibility for them was, de jure, part of my brief. However, as they were dedicated to supporting the specific purpose and goals of Project MAC, and were explicitly excluded from the campus computer-service plant, in my DIPS role I had little to do with them other than providing such operating support as was needed. However, in my role as interested ex-Assistant Director of Project MAC, I stayed close with the work of my colleagues there.

Q: In addition to its influence on Multics (or at least influencing Corby et al.) to want to build a second time-sharing system, what influence do you see from CTSS to other computer operating systems?

A: I have already mentioned the (final) CTSS file-system design. As far as I know, without confirmatory research, the user-name/password method of user authentication was first used by CTSS. In the hardware, the clock-trap interrupt, which ensured that the operating system could always wrest control of the system from even the most pathologically flawed of user programs, was crucial. (Actually, even after the extensive CTSS hardware modifications, there were a rare few operations of the 7094 hardware that the clock interrupt could not intercept. The AI Lab people found them all in short order, but showed admirable good citizenship by only occasionally using them to crash the system.)

Q: Tell me about your attitude toward the project while it was happening, e.g., just another programming job, something exciting that could change history, or something in between?

A: Those in leading roles in the project, certainly including me, never doubted for a moment that we were changing the world. Accessible computing, on the users' terms? A true computer utility, with computer power available at a wall socket? Multi-Access Computing and Machine-Aided Cognition? Oh, we knew what we had set out to do.

Tom Van Vleck[10]

Q: Please tell me about how the CTSS project was organized, where the staff came from, how big was the staff, who was on the staff, how it changed over time (e.g., as MIT students joined the staff and left it) how the project was managed, etc. And, of course, what was your role and how did it evolve?

A: I do not have first hand knowledge of the beginnings of the CTSS project, or of the politics surrounding the acquisition and use of computers at MIT at the start of the 60s.

[10] Interview of Tom Van Vleck by David Walden. Walden explains: I sent an email to Tom with some questions, and he sent back his answers in a 2011-02-23 email. On 2011-07-01 Tom edited the interview to eliminate overlap with Section 22.

There was a committee on computation consisting of professors; there was skepticism among other professors about the value of computers relative to their high cost. (When I took a Numerical Analysis course, we had to use Marchant electomechanical calculators.)

I went to MIT as a freshman in the fall of 1961, so I missed out on the very beginnings of timesharing. (As mentioned on my website, I had interest and experience in computing when I came to MIT from a high school project: this was rare in those days.) I got a part-time job in fall of 1963 working for Prof. Herb Teager at Project MAC, using the brand new MAC CTSS.

Q: What do you know about the steps were in the system evolving from an idea to something demonstrable — and whatever you know about how Daley and Daggett original got involved in the project.

A: This happened before I joined the project. Bob Daley started out as a computer operator and became a brilliant programmer and designer. Most of the CTSS file system is his work as well as many other parts of CTSS.

Q: You mentioned Prof. Fano, Dick Mills, Jerry Saltzer, and Roger Roach. I know what I have read in Bob Fano's oral history and in Corby's oral history something about Bob's connection to the project; however, beyond that I don't know what his role was.

A: Fano was one of the leading EE Department professors. He wrote the book (Chu, Fano & Adler) on circuit design. He bet his career on time sharing and became the first Director of Project MAC. Whether this was because he saw it as a way to get research funding or as a way to transform the world is a question you should ask him.

Q: I also know that Jerry Saltzer did `TYPSET` and `RUNOFF` and in time wrote lots of documentation for the project.

A: Saltzer showed he had programming chops with `TYPSET` and `RUNOFF` on CTSS, but he also made major intellectual contributions to computer science, solidifying the idea of "process" and then establishing fundamental principles in the security area.

Q: I don't know of Dick Mills (except having heard his name) or Roger Roach at all. What were their roles in the project?

A: Dick Mills was an early contributor to CTSS. (I inherited a listing of CTSS Version 14 (tape swapping) with his handwritten notes on it, and donated it to the Computer History Museum in Mountain View.) Mills had been a B-52 pilot, worked on Whirlwind, and became Assistant Director of Project MAC. Before Multics was launched as a project, Project MAC did the 1963 Summer Study which gave influential computer scientists from all over a taste of timesharing on CTSS. There were many projects using CTSS scattered around MIT and at other New England colleges and universities. Dick Mills dealt with many different tasks as Assistant Director, dealing with ARPA, managing personnel, etc: but one of the biggest jobs he did was to be the system administrator of CTSS. You could make a case that he was the first "sysadmin" in the modern sense. He assigned user account IDs and passwords; set up projects; designed, programmed and ran the usage accounting; programmed changes to LOGIN and the supervisor; understood the arcane interface to the IBM 7750; and did far more.

Roger Roach led the team that maintained CTSS at the MIT Computation Center, and was the one who finally pushed the button to shut CTSS down for the last time in 1963.

Q: Tell me about your attitude toward the project, e.g., just another programming job, something exciting that could change history, or something in between?

A: Most of these have been covered or are in my online stuff. Being part of a team led by Corby was a particular pleasure.[11]

[11] See page 13.

Q: How did you feel when CTSS was shut down? Looking back, how do you think about the CTSS project: the experience of working on it, it becoming an operational reality, and what impact you think it had on computing?

A: When CTSS shut down, it was time for it to go: the hardware was becoming ancient and flaky. But looking at what it could do, and what people did with it, shows that many subsequent systems were less capable and vastly more wasteful. When the Windows NT design came out in the mid 90s, it seemed to me to be a pretty nice version of CTSS . . . with a graphic interface.

Additional note[12]

I forget when Prof. Maurice Wilkes visited Project MAC. Probably 1966.

It was at a time when CTSS was very overloaded. We had tried assembling the system to support up to 50 users, but it was unusable. It seemed like about 30 was as much as it could handle, depending of course on what these users were doing. Big compilations might take hours, e.g. the EPL compilations for Multics modules.[13]

Complex rules were added to the CTSS login routine to decide who could log in. These rules reserved some slots for different groups of users: so many for Multics development, so many for the School of Science, etc.

I think Dick Mills was the primary inventor of the "load leveler," which was a facility that monitored the scheduler queue length and decided when the system was overloaded. When the system was overloaded and had been overloaded for some period, the load leveler would pick a user and initiate an automatic logout.

There were complex rules that influenced which user would be bumped: some users were protected from bumping, e.g., the daemon (that is, the special user scanning the disk and dumping modified files to tape).

There was a set of rules applied at LOGIN time to decide whether a user could be logged in as "PROTECTED FROM PREEMPTION" if their load control group had not used all its slots, or "SUBJECT TO PREEMPTION" if the system had not reached the maximum number of users but this user's group had all its slots used. A subject-to-preemption user could be promoted to protected if other group users logged out. (The standby-login stuff was my code, done under the direction of Dick Mills.)

The user chosen for preemption was the unprotected user who had been logged in the longest. When a user was bumped, a "blast" message was typed on their terminal giving them 3 minutes to save files and log out. People hated that. Then the user was logged out and whatever they had running at the time was saved as a file LOGOUT SAVED, which the user could RESUME when they next got logged in.

There was much tweaking of the system parameters to find the least-bad set of values. Some folks raised the issue that warning users and giving them time to clean up caused them to initiate a burst of demand in their last few minutes, running the load up higher, causing more bumping to go on, and possibly leading to oscillation. Nobody knew how to deal with this issue. I remember proposing that we write a DYNAMO simulation: but it would have required values for a lot of simulation constants that we would have to guess at or do a measurement study to determine.

Professor Wilkes heard about this controversy, and pointed out that this kind of problem had an analytic solution. He wrote some difference equations for the load, number of users, etc.. and then mapped them onto the continuous case differential equations, and plotted the transform in the complex plane. Whaddaya know, the equation system had a pole some distance from the origin, i.e. it was unstable, and the oscillation effect was intrinsic, if any warning was given.

---

[12] 2011-03-30 email. [13] EPL was an early PL/I compiler that ran very slowly.

I was enlightened: Real mathematics applied to operating systems. No need for elaborate programming and parameter fudging.

This is described in Wilkes71.

Roger Roach[14]

Q: What is the background that got you to MIT, to the 7094, and then involved with CTSS, and about your?

A: I came to MIT as a student in 1963. In my sophomore year, I got a student job working on a project to use the computer to format a thesaurus of textile terms for the Mechanical Engineering Department which migrated to an information retrieval project called TIRP which utilized CTSS to access abstracts of textile papers with a very early form of a relational database. We also used an IBM system called IBSYS to sort the data for the database we were developing. I was lead programmer in the project and there were two other students working with me. We were using more computer time than was then allocated for projects so we made a deal with the Computer Center to maintain CTSS in return for access to the spare time on the computer. The Computer Center was converting to the IBM 360 and all of their programmers were working on that. I did not work on the CTSS at MAC, but I talked to them when I need help understanding an issue. I became the de facto administrator on the CC CTSS, a role that Tom had performed before and who had moved to MAC to work on Multics. in 1967 I joined the Computation Center as a full time programmer to maintain CTSS and later as head of the system maintenance group on Multics. As leader in the system maintenance group in the Computation Center I was also in charge of maintaining the CP67/CMS system the CC installed for a project in Urban Planning. This group had between 3 and 4 other programmers working with me to perform system maintenance functions on these three systems. We did not do development, but took changes from the system programmers and did integrated testing, prepared the releases and kept the system libraries organized. For CTSS we did the normal maintenance functions including analyzing crashes, applying fixes and some minor enhancements for projects using the system. We worked with a group from the management school to utilize CTSS as a very primitive war gaming system in which various teams use what is now called instant messaging to trade information between themselves and form alliances to defeat other teams.

Q: Tell me what you can about how the CTSS project was organized and its organization evolved over time.

A: I do not know how the early CTSS team was formed or how it was managed. They did document what they were working on with notes. I captured the ones I could find on the DVD[15] I am sending you in the PSN (Programming Staff Notes) folder. That was before my time.

Q: Tell me about the key people on the project during your time on the project.

A: To me the key people were Tom Van Vleck for administration and overall knowledge, Noel Morris for file system and scheduler knowledge, and Stan Dunten for communications (e,g, the 7750). These were my go to people when I hit snags. Obviously Corby (F.J. Corbató) was the master of all this, but I did not have any interaction with him until I started working on Multics.

Q: Tell me about your attitude toward the CTSS project during your period of involvement: just another job, something exciting that could change history, or something in between?

A: The ability to use a system like CTSS was extremely exciting for me. It became my

[14] Interview of Roger Roach by David Walden. Walden explains: I sent an email to Roger with some questions, and he sent back his answers in a 2011-02-27 email. [15] Roach11.

"PC." I think I was the first student to use CTSS to write his thesis. I also used CTSS to do some bookkeeping for some of the outside projects I was working on. It was obviously the way of the future and I was very glad to be in on it. This also applied to my interest in Multics and CP/CMS (aka VM/370) — a forerunner of VMWare.

Q: After the system was operational, were further enhancements made over time?

A: My team made incremental changes. The most extensive were the modifications for the gaming system to allow users to send "immediate" messages to other users and the ability for referees to monitor and hold messages. We also added features to RUNOFF. We created a set of mini-commands to be used in runcoms (set of commands in a file) to provide dynamic data such as date and time for file names.

Q: There was a copy of CTSS at Project Mac. Were you also involved with that?

A: No, when I got involved, the MAC CTSS as being used mainly for Multics development. For compatibility, we did take their releases and applied them to the CC CTSS, but they were not interested in the changes we were doing unless we found a bug that would affect their system in which case they took the change and applied it to their system.

Q: How long did you stay with the CTSS project, and what did you do next?

A: I was responsible for maintained CTSS until the final shutdown.[16] I think Tom has a picture of me on his web page taken on the final shutdown. I worked on Multics until 1988 when MIT finally discontinued it. I was manager of systems programming until 1985 when I became director of Operations and Systems which then became director of IT Services until I "retired" in 2005. I did some consulting for a few years and then took a job as director of IT for the Whitehead Institute of Biomedical Research (affiliated with MIT but not strictly part of MIT) in 2007 where I am now.

Q: How did you feel when CTSS was shut down?

A: I felt sad although I was pretty involved in CP/CMS and Multics which by then I considered the next evolution. I did save as much of CTSS as I could and that was used by Dave Pitts recently in constructing an emulator for the 7094 and thus CTSS still lives![17]

Q: Looking back, how do you think about the CTSS project: the experience of working on it, and what impact you think it had on computing?

A: CTSS changed the way we deal with computers. Timesharing would have evolved eventually, but CTSS was a very visible proof of concept. Without it, there would have been no Multics and likely no UNIX which developed out of Multics, It might be a stretch, but without timesharing to allow mortals to use computers, the idea of PC's might have been delayed much further.

Allan Scherr[18]

I was one of the original graduate students with Project MAC. My PhD research was essentially to model the performance of time sharing systems. I created both detailed simulation models as well as simple mathematical ones. So as to test the models, I also took performance measurements of the CTSS system: number of users, response times, service times, etc. When I got my degree in June, 1965, I went to work for IBM right in the middle of what I called "the time sharing wars." IBM was in competition with Multics and General Electric for time sharing system business. IBM's entries into the fray were

[16] See page 13. [17] Pitts10. [18] Dave Walden explains: I solicited a paragraph from Allan regarding his involvement with CTSS and its impact. He sent back in a 2011-03-06 email this description of how CTSS was a part of his PhD research, from which he went on to a distinguished career in industry.

specialized systems that were incompatible variations on the System/360 line that IBM was introducing. I joined IBM as probably the only person in the world who understood the performance dynamics of time sharing systems.

In 1965, it was commonly thought that virtual storage (i.e., paging systems) were a necessary aspect of time sharing. Since System/360 did not support virtual storage, it was commonly thought that time sharing wouldn't work. I wound up leading a task force to come up with a general-purpose, System/360-based time sharing strategy. I showed that a practical, general-purpose time sharing system did not require virtual storage, and proposed what became the "Time Sharing Option" of OS/360. I became the design manager for the project, and my career as an operating system designer and leader was launched. Ironically, the TSO work established the feasibility of adding virtual storage to OS/360. I later went on to lead the creation of the virtual storage version of the operating system (MVS) which in turn is the basis for IBM's main frame operating systems of today.

I was named an IBM Fellow in 1984 in recognition of my work on operating systems and networking.

It is interesting to note that even today the time-sharing concepts apply to many of the client-server systems being created.

Additional note[19]

There is an incident that I finally confessed to at the 25th anniversary of CTSS and Project MAC..

My PhD research involved measuring, modeling, and simulating the performance of the CTSS system. The performance simulations were very detailed and I had created a simulation system based on MAD to run them. The actual scheduling program that was the heart of the CTSS operating system was part of the simulator. In any case, I needed many hours of computer time to run my simulations. Unfortunately, I had only been allocated something like 4 hours of processing time on CTSS per semester. I exceeded this time well before I had completed the simulations.

Because I had embedded measurement code into the CTSS operating system, I had access to the listings of the system. I discovered that the cumulative time usage for each account was loaded into the operating system every time the associated user's core image was swapped into memory. At the end of the time slice, the usage was updated and swapped out. I simply added a `XEC*` (indirect execution) instruction to the performance measurement code I had in the operating system so that I could indirectly execute an `STZ` (store zero) instruction to zero out my usage any time I wanted to. So I would rack up usage time until I got close to 4 hours and then zero it out and start all over.

This worked well until the spring of 1962. What happened was that someone on the CTSS staff (I think Bob Daley) came to me and said that the space my measurement programs used was needed and, since I was done measuring, they were going to remove my programs from the operating system. At the same time, my privileges to modify the operating system were revoked. By the way, the account number ("problem number" as it was called) for system programming was M1416 as I recall.

Being desperate, I had to find a way to regain access to the operating system so I could re-insert my `STZ` code. I finally found a way to do it. All of the passwords for the system were stored in a file called `UACCNT.SECRET` under user M1416. There was a way to request files to be printed offline by submitting a punched card with the account number and file name. Late one Friday night, I submitted a request to print the password files and very early Saturday morning went to the file cabinet where printouts were placed and took the listing out of the M1416 folder. I could then continue my larceny of machine time.

I did two things to spread the risk. First, I contacted J. C. R. Licklieder (the ARPA administrator for Project MAC). Because he was interested in computer security, I tempted

[19] 2011-06-25 email.

him by offering him a copy of all of the system's passwords. After being granted immunity I told him what I was doing and sent him the password files. I later learned that he had used the passwords to log onto Fano's account (and maybe others) and leave taunting messages.

The second thing I did was to tell two other struggling PhD students how to get more time on the system. One was Don Carroll (at the time an Assistant Professor in the Sloan School) and Peter Denning. Both were doing simulations, Don of job shop scheduling and Peter of disk based systems using the simulation software I had developed for my thesis.

As it turns out, one night Bob Daley noticed someone logging onto the system (every time a logon occurred, the ID was printed on a line printer in the machine room) who he knew was not there. He almost caught the person doing it (it was Don Carroll I think). After some investigation, Bob figured out that the culprit had modified code that I had left in the system (the purge of my code left one small vestige). He couldn't figure out why there was an XEC* being inserted, and I didn't help him figure it out.

I finished my simulation work in May of 1965, completed my thesis, and left for IBM. Later I heard that Don Carroll had been caught and called onto the carpet by Prof. Fano. I ultimately had the opportunity to confess to Prof. Fano in person at a 25th anniversary get together. He assured me that my PhD would not be revoked.

Peter Denning[20]

Q: When and how did you first hear about or come in touch with CTSS? Did its existence influence you toward MIT for your graduate work?

A: I fell in love with electronic computers in high school and won three science fair awards with homemade computers. In 1960 I entered Manhattan College EE in the hope of studying computers, but they were traditional EE with no computer courses at the time. In 1964 I chose MIT for graduate school because they were at the forefront of computing technology. I had not heard of CTSS until I arrived and Jack Dennis, my advisor, gave me many reading materials about the system. Even before I got to use CTSS, he sent me to his lab, where the year before he had completed an experimental time-sharing system for the PDP-1. That was my first experience with interactive computing. I loved it.

Q: Once at MIT, what was your involvement with CTSS? What were your first reactions?

A: I got my first CTSS account in 1964. I used CTSS for my masters project, a simulation study of disk storage access times under different scheduling policies. CTSS was a wonderful agglomeration of really useful tools, including the hierarchical file system, ability to share files with other users, primitive graphics for 2-D screens, interactive text editors, debugging tools, implementation of the language MAD (a derivative of Algol), and Saltzer's TYPSET and RUNOFF word-processing programs. I used all these tools to program my simulator and report its results.

I learned that CTSS was not just an experimental time-sharing system; it was an attempt to gain acceptance for time-sharing by showing that it could harmoniously co-habit a traditional batch processing system. To me, a novice unfamiliar with the history of batch processing, the co-habitation objective seemed to be a political issue that I did not understand.

Q: When and why did you get interested in the technical aspects of CTSS?

A: Almost immediately. I studied the technical documents and talked with the established

[20] Interview of Peter Denning by David Walden. Walden explains: I sent an email to Peter Denning with some questions. He adjusted the questions to be more appropriate and sent back answers to those questions in a 2011-03-10 email. In the email Peter also noted that he received an EE SM from MIT in 1965 and a PhD in 1968, and was a member of Project Mac from 1964–68.

people so that I could learn how CTSS did all its amazing things. I was intrigued by the CPU time slicer and the multilevel scheduling queues. I thought the design of the user memory was clever — a memory bound register defined the upper limit available for a user's job and an "onion skin" swapping strategy did not swap anything above that limit. I thought it was ingenious to have two 32K-word (128K bytes) memory boxes, one for user programs and data and the other for the supervisor (operating system). CTSS used an interrupt to call a supervisor routine, which simultaneously transferred control to a trusted location in the other memory and put the CPU into the supervisor-privilege state.

Q: What did you contribute to CTSS?

A: Al Scherr, who was finishing up his PhD during my first year at MIT, showed me how to program skillfully in FAP (the assembly language for the 7094 process of CTSS), how to use the debugger, how to program in MAD, and how to build a preprocessor that extended MAD into a simulation language. I noticed that the CTSS scheduler and swapper strongly favored small programs with short running times, and that the input-output (IO) library, borrowed verbatim from Fortran, made CTSS programs too long for optimal treatment by the scheduler. I wrote a bare-bones IO library for MAD users, which I called MADIO. MADIO cut my program sizes by 80 percent and got super-fast response times. It became popular in the CTSS community.

Q: How did what you learned about the behavior of CTSS impact your thesis work?

A: Jack Dennis saw my interest in performance analysis and encouraged me because we needed to learn how to design time-sharing systems for predictable throughput and response time. I vividly remember Corbató's claim in a technical paper that the multilevel scheduling queue not only gave fast response to the shortest jobs, but it kept the CPU utilization at 50 percent or more by starting jobs at a level of the queue where the time slice was longer than the swapping time. I wanted to know how to guarantee processor efficiency when swapping would be a standard part of computing systems. My PhD thesis was a quest for an answer to that question.

Al Scherr had discovered that the Machine Repairman model from queueing theory accurately predicted throughput and response time of CTSS. It amazed me that such a simple model worked for such a complex system. System administrators put his model to good use because it told them how many users they could allow to be logged in while maintaining response time under 3 sec. Scherr showed me how to use some of the same modeling techniques in my masters thesis. They became a central part of my PhD thesis.

Q: Did you make any recommendations for improvements to CTSS, or was the development focused on Multics by that time?

A: As noted, my direct contributions to CTSS itself were small. The Multics design was occupying everyone's attention by 1965. Multics had some pretty serious design issues in memory management, which entered the virgin territory of combining virtual memory with multiprogramming. Early reports from research projects around the world indicated an unexpected and devastating problem called thrashing, the sudden performance collapse of the system when multiprogramming level got too high. Understanding and solving thrashing was the holy grail of my PhD thesis.

Q: Once you left MIT and were teaching and writing about computer systems, did you use CTSS as an example? If so, how?

A: CTSS was always a reference example for me. Its design was extremely efficient, parsimonious, and predictable. I never understood why operating system kernels had to be large when I knew from my own experience that one of the first kernels providing the functions was small. I wrote a lot about how to keep an OS design simple and kernel small.

I was fascinated with the design of the user interface, then called the CTSS command language. Someone had extended the command language with a macro facility that

allowed users to write scripts of commands. The experience from that led to the Multics shell, which was an extraordinarily simple, highly efficient command interpreter. I found later that writing a simple shell was a marvelous project for students of operating systems classes because it helped them appreciate the power of the time-sharing interface.

Q: To what extent to you think CTSS influenced later time-sharing and other computer systems (or at least made developers of other systems more willing to understand their development efforts)?

A: CTSS had a great deal of influence. It showed that the design of a computer system could follow a small set of clear principles. The CTSS designers talked frequently with their counterparts in other time-sharing research projects, and incorporated the best ideas known at the time. CTSS passed its learning to Multics, which influenced many later systems. CTSS is like a centipede, with hundreds of footprints in many operating systems over the years down to the present day.

Q: Thank you for taking the time to answer questions about CTSS.

David Alan Grier[21]

CTSS: The balance of hardware and software

In the history of computer science, the work on CTSS represents one of those points where researchers began to realize that the nature of their field was starting to shift. Throughout the 1950s, most computer researchers devoted themselves to problems of hardware: faster switches, memories that took less space, more efficient architectures. While many researchers looked to make computers more accessible, they rarely articulated the idea that software and hardware might have an equal role in computer society. The work on CTSS showed that software had an equal place with hardware and that software might ultimately be the key component of any computing system.

Much software research in the 1950s were was intended to make computers more accessible. The major languages of the time, Fortran, LISP, ALGOL and COBOL, were all presented as tools that would relieve programmers from the details of programming. Even some preliminary research on timesharing was done to make more efficient use of computing hardware and hence make computers accessible to more people. The prime goal of time-sharing research, explained the IBM STRETCH research team, "is to allow more of the component units of a computer system to be kept in productive use more of the time."[22]

Most researchers knew that time-sharing programs would be complex systems and they attempted to control that complexity by developing time sharing on top of existing hardware. In common with most researchers of the time, the CTSS system, the project leader, Fernando Corbató acknowledged that computer hardware would have to be adapted for time-sharing applications. "There are several problems which can be solved by careful hardware design," he wrote. However, he argued that there were "also a large number of intricate system programs which must be written before one has an adequate time-sharing system."

In developing the CTSS system, Corbató went one step beyond his peers and argued that software was no longer a secondary part of computer development, but was the crucial element of computer science. "An important aspect of any future time-shared

[21] David Walden explains: I sent David Grier a request for a paragraph about his view of the impact of CTSS on computing history. He sent back a page in a 2011-02-23 email, which was finalized in an email of 2011-03-14. [22] Codd59.

computer is that until the system programming is completed, " he concluded, "the computer is completely worthless."[23]

CTSS did not mark sudden change in computer operation. Long after work on CTSS stopped, programmers were able to work on the native hardware without dealing with an intervening layer of software. However, CTSS would mark the point when researchers recognized the limitations to such an approach. The future of general purpose computation would lie not only in systems that combined both hardware and software elements. In these systems, the hardware would be of limited use, if not "completely worthless" unless the system program was present and operating.

[23] Corbató62.

5. Also from the MIT/Cambridge environment

In the years following the birth of CTSS until time-sharing was replaced by the personal computer, the MIT/Cambridge environment was a hotbed of time-sharing system development.

At MIT Prof. Jack Dennis led the development of a time-sharing system running on a PDP-1 donated to MIT in 1961 by Digital Electronics Corporation. The time-sharing system became operational in May 1963.[1]

MIT's Multics (Multiplexed Information and Computing Service) system was in a very real sense an outgrowth of CTSS as the developers of CTSS, again led by Fernando Corbató, began thinking in 1962[2] about what we today would call a "next generation" system, and actual development work began in 1965. The actual development project initially was a joint effort among MIT, Bell Telephone Laboratories, and General Electric (Multics was based on a GE-645 computer). The system was operational by 1968–69 and over the years approximately 65 systems were deployed by Honeywell which took over GE's computer business (and, in turn, Bull which later took over Honeywell's computer business).[3]

In 1969 the Bell Laboratories left the project, and some of the Lab's people who had been involved with Multics went on to develop Unix, no doubt at least somewhat influenced by their Multics experience.[4]

In the late 1960s, researchers at the MIT AI Laboratory, eschewing Multics, built their own time-sharing system known as ITS (Incompatible Time-sharing System, playing with the CTSS name). ITS was based on a DEC PDP-6, and many innovative applications were developed one it. Later ITS migrated to a PDP-10.

Across Cambridge at Bolt Beranek and Newman Inc., J. C. R. Licklider was hired by Leo Beranek to pursue Licklider's vision of making computing personal. Licklider in turn hired Ed Fredkin, who hired John McCarthy and Marvin Minsky as consultants (the same John McCarthy who had pushed time-sharing at MIT and thus provided some of the impetus for creating CTSS). They set out to build a time-sharing system on a PDP-1b computer (the first PDP-1 sold by DEC), and an initial demonstration was done circa mid-1962. Shelly Boilen and Bill Mann got the system really working by the end of 1962.[5] There was a good bit of communication between Ed Fredkin and Jack Dennis as their respective PDP-1 time-sharing systems were developed.

Based on its work with that initial time-sharing system, BBN undertook an NIH contract to deploy time-sharing in Massachusetts General Hospital, based on the PDP-1d system from DEC. A second time-sharing system was implemented (called Exec II) but never worked reliably, and then a third system was implemented (Exec III) which worked reliably for many years at BBN in support of Mass General (the original ARPANET IMP software was developed on this machine).

The work with the "hospital system" spawned three follow-on projects: a time-sharing joint venture between BBN and GE called Medinet, which did not succeed as a business; BBN's own time sharing bureau called Telcomp, which was quite successful;

---

[1] Ackerman67 and Dennis66. [2] Hauben97. [3] Multics was described initially in a set of six papers presented at the 1965 Fall Joint Computer Conference: Corbató65, Glaser65, Vyssotsky65, Daley65, Ossanna65, and David65. The papers explained what was intended which in some cases was later revised in light of experience.. [4] Ritchie79. [5] McCarthy63.

and a time-sharing spinoff of Mass General called Meditech (which continues to operate in the medical information technology field to this day).

There is an important distinction between time-sharing as implemented on big machine such as the 7094, big 360 machines, and DEC 6 and 10 machines and time-sharing implemented on little machines such as the PDP-1. Ed Fredkin and Jack Dennis were making an important contribution by showing the time-sharing could be implemented on such a small machine. Of course, ultimately this lesson was made manifest by the creation of Unix on a PDP-11.

Meanwhile, IBM began development in Cambridge of a 360-based time-sharing system, influenced by CTSS and involved some people who had been involved with CTSS, that came to be known as CP/CMS.[6] The first installation of this system was on a 360/67 at MIT Lincoln Laboratory. The MIT Urban Systems Laboratory obtained funding that supported the installation of an IBM 360/67, running the CP/CMS time-sharing system, in 1968.

Returning to BBN, when its original PDP-1 systems were growing old and tired, BBN bought an SDS-940 running the time-sharing system developed at UC Berkeley to which BBN made some additions. Then BBN developed the TENEX system, adding a BBN developed paging hardware to a DEC PDP-10.[7] This became a popular time-sharing system in the early days of the ARPANET, and eventually was licensed to DEC where it became TOPS-20. The TENEX developers were very much aware of the Multics design as they designed a virtual memory system.

The connection between MIT, BBN, and DEC is notable. The founders of DEC had come from MIT's Whirlwind project. BBN took delivery on the first PDP-1 sold by DEC, and DEC's PDP-1 designer, Ben Gurley (another MIT computer person) made changes to the system to support time-sharing (which both Ed Fredkin and Jack Dennis needed for their PDP-1 time-sharing work). Various of the MIT and BBN PDP-1 people eventually worked at DEC as did some of MIT's CTSS people.

In this era, time-sharing bureaus were popping up all over the world, including in the area around MIT and Cambridge. We already mentioned BBN's Telcomp System. Some of the people involved with the CP/CMS system at Lincoln later started a time-sharing bureau, Interactive Data Corporation, in Waltham, MA. Another time-sharing bureau based on CP/CMS was National CSS (in Connecticut) which also had staff members from MIT. Lou Clapp who had been part of the MIT community started Dial Data in Watertown, MA (later acquired by Tymshare). Undoubtedly there were others.

Naturally, time-sharing was also happening elsewhere in the country and the world (for example, the Dartmouth Time Sharing System and its descendant GE Timesharing (Mark I, Mark II, Mark III). Time sharing was a development for which computer technology and economics were ripe. We make no claim that all of this evolved out of what happened first at MIT and around Cambridge (although TYMSHARE, for instance, employed Norm Hardy who had left IBM's Cambridge Scientific Center and also employed other folks from MIT). However, we also speculate that the various people who were around MIT and CTSS at various times, for instance during the Summer Study of 1963 (page 27), maintained contact with what was going on with CTSS after they returned to their own organizations.

There can be no doubt, however, that McCarthy's original idea for time-sharing and the Corbató-led substantive, solid, early implementation of time sharing in CTSS at MIT were important stimuli for the era in computing that continues to this day — the era where users themselves have direct contact with a computer, telling the computer what they want it to do from moment to moment.

---

[6] Varian91. [7] Murphy11.

Bibliography

Ackerman67. Ackerman, W. B., and W. W. Plummer. An implementation of a multiprocessing computer system, *Proceedings of the ACM Symposium on Operating System Principles*, Gatlinsburg, TN, October 1–4, 1967.

Bourne03. Bourne, Charles P., and Trudi Bellardo Hahn. *A history of online information services, 1963-1976*, MIT Press, Cambridge, MA, 2003, pp. 41–48, available online at `http://books.google.com/books`

Brandel99. Brandel Mary. 1961 — Learning to share, *Computerworld*, June 29, 1999, `http://www.cnn.com/TECH/computing/9906/29/1961.idg/index.html`

Burke98. Burke, Colin. A rough road to the information highway — Project INTREX: A view from the CLR archives, in Bourne03.

Chiou01. Chiou, Stefanie, Craig Music, Kara Sprague, and Rebekah Wahba. A Marriage of Convenience: The Founding of the MIT Artificial Intelligence Laboratory, rough draft of project report for MIT course 6.933J/STS.420J (The Structure of Engineering Revolutions), December 5, 2001, `http://www.scribd.com/doc/33123208/MIT-AI-Lab-History`

Chuu01. Chuu, Chian, Michael Lei, Chiyu Liang, and Alida Tei. The Student Information Processing Board: the social and technical impact of an MIT student group, prepared for MIT course 6.933, Structure of Engineering Revolutions, December 14, 2001, `http://web.mit.edu/6.933/www/Fall2001/SIPB.pdf`; this report also includes a section on CTSS.

Codd59. Codd, E. F., E. S. Lowry, E. McDonough, C. A. Scalzi, Multiprogramming STRETCH: feasibility considerations, *Communications of the ACM*, vol. 2 no. 11, November 1959, pp. 13–17.

Corbató62. Corbató, F., M. Merwin-Daggett, and R. Daley. An Experimental Time-Sharing System, *Proceedings of the Spring Joint Computer Conference*, vol. 21, 1962, pp. 335–344, `http://larch-www.lcs.mit.edu:8001/~corbato/sjcc62/`

Corbató63a. Corbató, F. J., M. M. Daggett, R. C. Daley, R. J. Creasy, J. D. Hellwig, R. H. Orenstein, and L. K. Korn. *The Compatible Time-Sharing System: A Programmer's Guide*, The MIT Press, Cambridge, MA, 1963.

Corbató63b. A TV interview of Fernando Corbató explaining about time-sharing, interviewed by John Fitch for the *Science Reporter* program on WGBH-TV, recorded on May 9, 1963, shown on May 16, 1963,
`http://www.youtube.com/watch?v=QO7PhW5sCEk&feature=related`

Corbató63c. Corbató, F. J., J. W. Poduska, and J. H. Saltzer. *Advanced Computer Programming: A Case Study of a Classroom Assembly Program*, MIT Press, Cambridge, MA, 1963.

Corbató65. Corbató, F. J., and V. A. Vyssotsky. Introduction and overview of the Multics system, *Proceedings of the 1965 Fall Joint Computer Conference*, pp. 185–196.

Corbató89. An interview of Fernando Corbató, conducted by Arthur L. Norberg on 18 April 1989 and 14 November 1990, Charles Babbage Institute call number OH 162, `http://www.cbi.umn.edu/oh/pdf.phtml?id=92`

Corbató91. Corbató, F. J. On building systems that will fail, Turing Award Lecture, March 5, 1991, http://larch-www.lcs.mit.edu:8001/~corbato/turing91/

Corbató92. Corbató, Fernando. Excerpts from Corbató63a, *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 31–32.

Corbató06. Oral History of Fernando Corbató conducted by Steven Webber on February 1, 2006, Computer History Museum reference number X3438.2006, http://archive.computerhistory.org/resources/access/text/Oral_History/102658041.05.01.acc.pdf

Creasy81. Creasy, R. J. The Origin of the VM/370 Time-sharing System, *IBM Journal of Research and Development*, vol. 25 no. 5, September 1981, pp. 483–490; this article describes the roots of CP/CMS in CTSS: http://pages.cs.wisc.edu/~stjones/proj/vm_reading/ibmrd2505M.pdf

Crisman65. Crisman, P. A. *The Compatible Time-Sharing System: A Programmer's Guide*, second edition, The MIT Press, Cambridge, MA, 1965; the following URL is to a version updated through 1969: http://www.bitsavers.org/pdf/mit/ctss/CTSS_ProgrammersGuide_Dec69.pdf

Daley65. Daley, R. C., and P. G. Neumann. A general-purpose file system for secondary storage, *Proceedings of the 1965 Fall Joint Computer Conference*, pp. 212–230.

David65. David, E. E., Jr. and R. M. Fano. Some thoughts about the social implications of accessible computing, *Proceedings of the 1965 Fall Joint Computer Conference*, pp. 243–248.

DeMarco99. De Marco, Tom, and Timothy Lister. *Peopleware: Productive Projects and Teams*, second edition, Dorset House Publishing, Inc., New York, 1999.

Dennis66. Dennis, J. and E. Van Horn. Programming semantics for multiprogrammed computations, *Communications of the ACM*, vol. 9 no. 3, March 1966, pp. 143–155.

Fano64a. Fano, Robert. The MAC system: a progress report (describes the usage of CTSS with examples), MAC-TR-12, Project MAC, MIT, Cambridge, MA, October 9, 1964, http://www.bitsavers.org/pdf/mit/lcs/tr/MIT-LCS-TR-012.pdf

Fano64b. Fano, Robert. Video of Robert Fano explaining scientific computing, 1964, http://www.youtube.com/watch?v=sjnmcKVnLi0

Fano66. Fano, R. M., and F. J. Corbató. The time-sharing of computers, *Scientific American*, September 1966, pp. 128–140.

Fano79. Fano, Robert M. Project MAC, in *Encyclopedia of Science and Technology*, edited by Jack Belzer, Albert G. Holzman, and Allen Kent, vol. 12, pp. 339–360, http://simson.net/ref/lcs_35/Fano_On_the_social_role_of_computer_communication_systems.pdf

Francoeur02. Francoeur, Eric, and Cyrus Levinthal. The Kluge and the origins of interactive molecular graphics, *Endeavour*, vol. 26 no. 4, 2002, pp. 127–131, http://www.cgl.ucsf.edu/home/tef/pubs/EndeavourVol26Issue4Pgs127-131.pdf

Garfinkel99. Garfinkel, Simson L. *Architects of the Information Society: Thirty-Five years of the Laboratory for Computer Science at MIT*, MIT Press, Cambridge, MA, 1999.

Glaser65. Glaser, E. L., J. F. Couleur, and G. A. Oliver. System design of a computer for time-sharing applications, *Proceedings of the 1965 Fall Joint Computer Conference*, pp. 197–202.

Hauben97. Hauben, Michael, and Ronda Hauben. *Netizens: On the History and Impact of Usenet and the Internet*, Wiley-IEEE Computer Society Press, 1997. See also http://www.columbia.edu/~hauben/netbook/

Lee92a. Lee, J. A. N. About this issue, *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 3–4.

Lee92b. Lee, J. A. N. Time-sharing at MIT: Introduction, *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 13–15.

Lee92c. Lee, J. A. N. Claims to the term "time-sharing," *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 16–17.

Licklider60. Licklider, J. C. R. Man-Computer Symbiosis, , *IRE Transactions on Human Factors in Electronics*, volume HFE-1, pages 4Ű-11, March 1960, on-line starting on page 7 at `http://memex.org/licklider.pdf`

Licklider92. Licklider, J. C. R. Excerpts from Licklider60, *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, p. 24.

MAD62. *A User's Reference Manual for the Michigan Algorithm Decoder (MAD) for the IBM 7090*, Digital Computer Laboratory, Graduate College, University of Illinois, June 20, 1962.

MAD66. *The Michigan Algorithm Decoder (The MAD Manual)*, revised edition, 1966.

McCarthy63. McCarthy, J., S. Boilen, E. Fredkin, and J. C. R. Licklider. A time-sharing debugging system for a small computer, *AFIPS 1963 Proceedings of the Spring Joint Computer Conference*, ACM New York, 1963.

McCarthy83. McCarthy, John. Reminiscences on the history of time sharing, 1983, `http://www-formal.stanford.edu/jmc/history/timesharing/timesharing.html`

McCarthy89. McCarthy, John. An interview of John McCarthy, conducted by William Aspray on 2 March 1989, Charles Babbage Institute call number OH 156, `http://www.cbi.umn.edu/oh/pdf.phtml?id=192`

McCarthy92a. McCarthy, John. Reproduction of McCarthy83, *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 19–24

McCarthy92b. McCarthy, John. John McCarthy's 1959 memorandum, *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 20–23.

McCarthy92c. McCarthy, John. Minsky and Teager. *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 28–29.

MIT92a. Special issue on time-sharing at MIT, *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, containing Lee92a, Shriver92, MIT92b, Lee92b, Lee92c, MIT92c, McCarthy92a, McCarthy92b, Licklider92, Teager92, McCarthy92c, MIT92d, Corbató92, Rosin92, and MIT92e.

MIT92b. Biographies of participants and authors, *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 10–12.

MIT92c. The beginnings at MIT, *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 18–30, containing an introduction without a listed author (pp. 18–19) and McCarthy92a, McCarthy92b, Licklider92, Teager92, Minsky92, and MIT92c.

MIT92d. Long Range Computation Study Group's recommendation for a time-sharing system, *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 28–30.

MIT92e. References and Bibliography. *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 52–54.

MIT92f. Special issue on MIT's Project MAC, *IEEE Annals of the History of Computing*, vol. 14 no. 2, 1992. This special issue was edited by J. A. N. Lee with articles and interviews by the same people who participated in MIT92a, and others. There is some mention of CTSS in this special issue which has the following introductory note by Lee: "The story of Project MAC is a continuum from that of CTSS, marking the transfer of an experimental system into the practical academic field and thence into the commercial field. Two men were key to this recognition of the value of the work by Corbató et al. — Joseph C.R. Licklider and Robert M. Fano." The two special issues both resulted from involvement of John Lee and Robert Rosin in a celebration at MIT, in the fall of 1988, of the 25th anniversary of the Laboratory for Computer Science, the then name of Project MAC.

Morris11. Morris, Errol. Did my brother invent e-mail with Tom Van Vleck?, a five-part blog series, New York Times website, June 20–24, 2011, http://opinionator.blogs.nytimes.com/tag/tom-van-vleck/

Murphy11. Murphy, Daniel. TECO, TENEX, and TOPS-20 Papers, a website, http://tenex.opost.com/

Nance96. Nance, Richard E. A history of discrete event simulation programming languages, in *History of Programming Languages—II*, edited by Thomas J. Bergin and Richard G. Gibson, Addison-Wesley, Reading, MA, 1996, pp. 367–427.

Ossanna65. Ossanna, J. F., L. Mikus, and S. D. Dunten. Communications and input-output switching in a multiplexed computing system, *Proceedings of the 1965 Fall Joint Computer Conference*, pp. 231–242.

Pierce03. Paul Pierce's collection of actual computers, http://www.piercefuller.com/collect/main.html

Pierce04. Paul Pierce's collection of source code for CTSS from about 1972, http://www.piercefuller.com/library/ctss.html?id=ctss

Pitts10. Pitts, Dave. A version of CTSS created by Dave Pitts running on a 7094 simulator written by Paul Pierce (Pierce03, Pierce04), http://www.cozx.com/~dpitts/ibm7090.html

Pouzin00. Pouzin, Louis. The origin of the shell, November 27, 2000, http://www.multicians.org/shell.html

Richardson81. Richardson, George P., and Alexander L. Pugh III. *Introduction to System Dynamics Modeling with Dynamo*, Pegasus Communications, Cambridge, MA, 1981.

Ritchie79. Ritchie, D. M. UNIX Time-Sharing System: A Retrospective, *Bell System Technical Journal*, vol. 57 no. 6, October 1978, pp. 1947–1969.

Ritchie84. Ritchie, Dennis. The evolution of the UNIX time-sharing system, *Bell System Technical Journal* 63, No. 6, Part 2, October 1984, pp. 1577–93. See also http://cm.bell-labs.com/cm/cs/who/dmr/hist.html

Roach11. Roach, Roger. One-time CTSS system administrator Roger Roach has a DVD with scans of a large collection of CTSS documentation, listing, correspondence, etc.

Rosin92. Rosin, Robert, and J. A. N. Lee interview of Fernando J. Corbató, Allan L. Scherr, Douglas T. Ross, and Martin Greenberger. *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 33–51.

Saltzer64. TYPSET and RUNOFF, memorandum editor and type-out commands, M.I.T. Computation Center Memorandum CC-244 and M.I.T. Project MAC Memorandum MAC-M-193, November 6, 1964, http://mit.edu/Saltzer/www/publications/CC-244.html

Saltzer65. Saltzer, J. H. CTSS Technical Notes, MAC-TR-16, Project MAC, MIT, Cambridge, MA, March 15, 1965, http://www.bitsavers.org/pdf/mit/lcs/tr/MIT-LCS-TR-016.pdf

Saltzer66. Saltzer, J. H. Manuscript typing and editing, from Crisman65, section AH.9.01, December 1966 revision, http://mit.edu/Saltzer/www/publications/AH.9.01.html

Shriver92. Shriver, Bruce D., and Merrill Buckley. Welcome messages from the IEEE Computer Society and IEEE presidents, *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, p. 5.

Teager92. Teager, H. Teager's recommendation for an IBM 7030, *IEEE Annals of the History of Computing*, vol. 14 no. 1, 1992, pp. 24–27.

Thornhill68. Thornhill, D. E., R. H. Stoz, T. T. Ross, and J. E. Ward. *An Integrated Hardware-Software System for Computer Graphics in Time Sharing*, Report Nos. ESL=R-356 and MAC-TR-56, MIT, Cambridge, MA, December 1968, http://www.bitsavers.org/pdf/mit/lcs/tr/MIT-LCS-TR-056.pdf

Tordo10. Tordo, Emily. Guide to the Records of the Massachusetts Institute of Technology Computation Center, a finding aid (AC.0062), Massachusetts Institute of Technology, Institute Archives and Special Collections, MIT Libraries, Building 14N-118, 77 Massachusetts Avenue, Cambridge, MA, March 24, 2010, 14 pages. The materials in this collection are mostly from before the CTSS era. It does, however, refer to other MIT collections including papers relating to CTSS and the work of Fernando Corbató.

VanVleck95. Van Vleck, Tom. How I got started in computers, 1995, http://www.multicians.org/thvv/boyd.html

VanVleck02. Van Vleck, Tom. 1401s I have known, an evolving (starting in 1995) website, http://www.multicians.org/thvv/1401s.html

VanVleck01a. Van Vleck, Tom. The history of electronic mail, an evolving (starting in 2001) website, http://www.multicians.org/thvv/mail-history.html

VanVleck01b. Van Vleck, Tom. Tech Square, an evolving (starting in 1993) website, http://www.multicians.org/tech-square.html

VanVleck03. Van Vleck, Tom. CTSS creators and users, 2003, http://www.multicians.org/thvv/ctss-list.html

VanVleck11. Van Vleck, Tom. The IBM 7094 and CTSS, an evolving (starting in 1995) website, http://www.multicians.org/thvv/7094.html

Varian91. Varian, Melinda. VM and the VM community: past, present, and future, April 1991, http://www.leeandmelindavarian.com/Melinda/neuvm.pdf; 1997 revision: http://www.leeandmelindavarian.com/Melinda/25paper.pdf

Vyssotsky65. Vyssotsky, V. A., F. J. Corbató, and R. M. Graham. Structure of the Multics Supervisor, *Proceedings of the 1965 Fall Joint Computer Conference*, pp. 203–212, http://www.multicians.org/fjcc3.html

Waldrop01. Waldrop, M. Mitchell. *The Dream Machine: J. C. R. Licklider and the Revolution That Made Computing Personal*, Viking Press, New York, 2001.

Ware08. Ware, Willis H. *RAND and the Information Evolution A History in Essays and Vignettes*, The RAND Corporation, Santa Monica, CA, 2008, http://www.rand.org/content/dam/rand/pubs/corporate_pubs/2008/RAND_CP537.pdf

Weik64. Weik, Martin H. *A Fourth Survey of Domestic Electronic Digital Computing Systems*, Report No. 1227, Ballistic Research Laboratories, Aberdeen Proving Ground, MD, January 1964, pp. 168–169, http://ed-thelen.org/comp-hist/BRL64-i.html#IBM-7094-II

Weizenbaum66. Weizenbaum, Joseph. ELIZA — A Computer Program For the Study of Natural Language Communication Between Man And Machine, Communications of the ACM 9 (1), January 1966, pp. 36–45.

Wilkes71. Wilkes, M. V. Automatic load adjustment in time-sharing systems, *Proceedings of the ACM SIGOPS Workshop on System Performance Evaluation*, Association for Computing Machinery, New York, 1971, pp. 308–320.

Colophon

This brochure was typeset with the LaTeX system created by the famous computer scientist Leslie Lamport (and since extended by many other people) and based on the TeX system created by the famous computer scientist Donald Knuth. Both of these systems were revolutionary in their time, and they remain vibrant parts of the computer world's longest running open source success story.

The typefaces are from the extensive Lucida typeface family of fonts designed by Charles Bigelow and Kris Holmes from 1985 onward. The Lucida family was specifically developed with computer output devices in mind.

Change history

Rev. 01, 2011-06-24: Corrected various typos in the print edition.

Rev. 02, 2011-07-02: Added new content from Corbató and Scherr; dropped redundant content from Van Vleck.

Rev. 03, 2011-12-05: Updated the sentence about Dave Pitts creating a version of CTSS running on a 7094 emulator.

Rev. 04, 2012-01-11: Added a person to the Acknowledgments and added an item to the bibliography.

Rev. 05, 2012-12-15: Updated the URL for the Melinda Varian document in the bibliography; updated the URL for the location of this brochure on the Computer Society website.