A Technical History of National CSS

By Harold Feinleib

Written: March 4, 2005

CHM Reference number: R0363.2016

I knew I wanted to go into computers when I got my first program to work. That was quite a thrill. It happened in 1964 during my senior year at Cornell when I took the only course given in computers. I had to look to the Electrical Engineering school where we worked on a Burroughs 220, a computer filled with thousands of vacuum tubes that took up a good sized room. It was actually a very nice computer because it worked in decimal instead of binary. That memorable program was of course a Sort program—I had to sort a list of numbers. I wonder how many sort programs have been written before or since. It must be millions.

With great enthusiasm, I interviewed at the campus-recruiting program where companies sent people to find new employees for their organization. In the 1960's, unlike today, the job market was very active and no one had a problem getting started right after they graduated from college. I got some interest from three companies: Equitable Life to work in their actuarial program in New York City; IBM as a programmer to work in upstate NY; and with the Service Bureau Corporation to also work in midtown Manhattan. New York City was very appealing to me; I was 20 years old, and living in Manhattan was a very exciting prospect, so I took great interest in the Equitable and the SBC interviews. The two companies couldn't have been more opposite in their approach.

At Equitable I met with a senior executive who took me to lunch in their executive dining room at the top of their office tower. He was very cordial and didn't ask me any probing questions. I guess I was supposed to be impressed by their prestigious office. Shortly thereafter I received an offer letter with a good starting salary of $7500 a year in their actuarial program.

SBC was a different story. I was hardly greeted when I arrived. I was asked to wait in a somewhat disheveled conference room and after quite some time someone came in and slid a sheet of paper toward me and said I should work on it. There were 10 very tough questions on the paper, what we would call brainteasers. They didn't tell me how long I had or how I should show my answers. The only question I remember to this day was—if you had an orange and you could make four slices, what is the maximum number of pieces you could cut the orange into. I made the assumption that you had to hold the orange together while doing the cutting; otherwise the answer was simple. The others were logic or algebra questions—you know the type—if Tom is an X and Jim is a Y and all X's are also Y's, then what is...blah, blah, blah? Or if one train is going 60 miles an hour and another train is going 45 miles an hour...?? I wish I had a copy of the questions today; they were a very good aid for recruiting.

Well, I worked on the questions; I knew I couldn't answer all of them (they were really tough), but wasn't too concerned. Eventually someone came down and asked for my answers and went away. I waited some more and then was asked to come upstairs to speak with a young woman who asked me to go over each of the questions so she could see how I approached each one. I think I really impressed her when we went over the one with the orange, because I derived a formula, which showed that the maximum number was 15 pieces. I did this by first using a 1 dimensional line, given two cuts, then a 2 dimensional flat surface given 3 cuts and extrapolating to a 3 dimensional object with 4 cuts. The formula I came up with was 2^n-1, where n is the number of cuts. It is easy to see how a line can be cut into 3 pieces and a sheet of paper can be cut into 7 pieces. It is not so easy to visualize how a 3 dimensional object could be cut into 15 pieces.

I got an offer to start at $6700 from SBC. While this was 10% less than Equitable's offer I took the job. To this day I am glad I made that choice because it opened me up to a very exciting and personally rewarding career.

SBC was a great place to work. It was a subsidiary of IBM at that time, although later it had to be sold to Control Data Corporation as part of an agreement to settle a suit that CDC had against IBM. I was a member of an excellent staff that did consulting work to companies that needed programs written. Therefore the work was varied. All the people at SBC were young, in their 20's or early thirties except for a handful of executives (almost everyone in computing in those days was young). When I started work, I entered a 13-week training program that was awesome; we were taught several languages and how to

program. The instructor was terrific and every week that went by the class got smaller and smaller until only about half of the 30 starters were still there. The instructor's role was not only to train the new hires, but also to screen them by examining their work. And he knew pretty quickly if someone wasn't getting it. One thing that I have learned over the years is if someone isn't real good at programming, they shouldn't bother; it is not a forgiving profession. They won't produce anything worthwhile; all they will do is make a mess. The best people will easily do the work of 3 "average" programmers, and will be able to tackle problems that others couldn't get to first base with. Fortunately, I have had the pleasure of working with a number of the best.

SBC exposed me to a wide variety of problems. Their biggest account was IBM itself, where SBC worked on developing operating systems and other systems programs. So, not only did I get experience in developing applications, but I also learned a little bit about systems programs, which at that time were considered a higher form of the art.

This gave me a double perspective of computers: how business people wanted to use them and also how systems programmers developed tools to make it easier to create business applications. One perspective was focused outward and the other inward.

One job I worked on was TSS/360, a huge IBM undertaking to develop a time-sharing system for the new IBM 360/67, IBM's first production machine with virtual memory. TSS was being developed at IBM's Mohansic Labs and there were hundreds of people working on it, both IBMers and contractors; there were a lot of very smart people (I was very impressed at the time, especially with my limited experience). They all felt that they were inventing the future and the sense of excitement was very high.

TSS was a very well designed system, except for one thing; it wasn't buildable at that point in time. The 360/67, although a very powerful computer at that time, was still not powerful enough to run TSS. TSS' memory requirements were too demanding to perform well. Memory at that time was very expensive—256K, yes 1/4 of 1 megabyte cost $400,000. Today 256 megabytes costs about $38 and is at least 100 times faster. That is an improvement of 1 billion times To understand this, 256 megabytes is 1000 times as much memory at a cost of 1/10,000 with a speed improvement of 100 times. Multiplying these together yields: 1000*10,000*100 or 1 billion.

It was great working in Mohansic. I lived on 60th street and Third Avenue and SBC had me lease a car so I could pick up some others and drive back and forth to Mohansic, which was north of NYC on the Taconic Parkway. This gave me a car in Manhattan, and the place I leased it from let me park it in their garage, which was right across the street from where I lived. Imagine that, living on the Upper East Side with a car, and I was barely 21.

Since the Vietnam War was heating up in 1965 and I was of a prime age to be drafted, SBC decided that I should work on a project for Lincoln Laboratories in Lexington, Massachusetts. Lincoln was a government run science lab, which would enable me to get a defense deferment. I learned a great deal at Lincoln working with Frank Belvin and Jack Arnow (both of whom went on to form Interactive Data Corporation). After working there for a short while, Jack offered me a position as a Staff Member at Lincoln, a prestigious position, which gave me a more secure draft deferment.

Now it so happened that Lincoln Labs was also one of the prime customers for TSS/360 together with Bell Labs and a couple of other big companies. So after Jack Arnow got to know me, he sent me back as a customer liaison to work at Mohansic Labs on TSS. What he wanted me to do was evaluate the progress IBM was making on TSS because he needed to provide a better way for the scientists and engineers to run and test their programs. MIT had developed its own time-sharing system for its students to work on, called CTSS and was working with GE to develop the next generation Multics system.

After a few months I told Jack that I thought TSS had a long way to go. Its performance was terrible and I wasn't sure they could fix it. They never did and eventually scrapped the project after spending hundreds of millions on the software although they did deliver about 10 copies to select customers.

Jack was well connected in the Cambridge community. He knew Norm Rasmussen who headed IBM's Cambridge Scientific Center. Norm had some really smart MIT students working on a little project they called CP/40. It ran on a specially modified 360/40 that had virtual memory. One of the objectives was to develop an operating system that would allow developers to test and debug different operating systems at the same time, all sharing the same machine. It actually gave each user a "virtual machine" which was used to run their programs. All virtual machines shared the same physical machine. The memory for each virtual machine would reside on a storage device and be brought into main memory as the program referenced different "pages" of its memory. The pages could be loaded into any portion of the physical memory of the machine and the machine's Dynamic Address Translation (DAT) box would translate the addresses that the user's program thought it was referencing to the actual physical address that page was loaded into.

I guess Jack saw the potential of CP/40 and convinced Norm to modify the system to run on Lincoln's 360/67, a much more powerful machine with virtual memory designed in. Norm's group had also developed a single user operating system called CMS, a command line operating system that enabled programmers to write and run their programs. Each user ran CMS in their virtual machine. So the combination of CP and CMS enabled several programmers to share the same machine while testing their programs. CMS was also designed to use the standard IBM Fortran compiler and the 360 assembler.

It is worth noting that the IBM PC's DOS operating system introduced more than a dozen years later took a lot of ideas from CMS.

At that point I met Dick Bayles who headed up the CP portion of the project. He and a couple others, including Mike Field, worked at the IBM Cambridge Scientific Center and ported CP to Lincoln's 67.

Meanwhile I worked with Frank Belvin to build up CMS to support Lincoln's requirements. We developed a batch processing environment and also developed an EXEC language so terminal users could string commands together instead of having to type them in over and over again. We brought on John Skodon, an MIT student for the summer. He was terrific and developed an interactive editor that made it easier to edit source code. I also worked on trying to improve the performance of the simple CP dispatching algorithm, which had some critical limitations.

In 1968, I heard from Dick Bayles that he had been contacted by a couple of guys in Connecticut who wanted to start a time-sharing company and wanted him to join. They wanted to use CP/CMS as the basis for the service. Dick asked me if I was interested, so I went down with him to speak with Bob Bernard and Dick Orenstein and thought it was a good idea and was eager to join as well. When I told Jack Arnow what I was thinking of doing he told me that he and Frank Belvin were also in the process of starting a company (Interactive Data Corporation) that would sell time-sharing services using CP/CMS. Both companies gave me almost identical offers. I had a real hard time deciding what to do.

The overriding factor for me was moving to Connecticut. I had married Judy Kaufman in April 1967. Her father had died in 1965 and she wanted to be closer to her mother who lived alone in Queens. So I decided to join the CSS (Computer Software Systems) team. I brought the software with me, the system we were running at Lincoln, which was more advanced than what IBM had since it was being used in a real-world environment. When developing software, there is nothing like testing it with real-world users, otherwise you simply do not know where it stands. The earlier you have real users the better.

It is interesting to note that software at that time was not considered proprietary property. The money was made by selling hardware, and software was viewed only as a way of making it easier to sell the hardware. That thinking didn't really change for many years to come. Even though we called ourselves Computer Software Systems, we still sold machine cycles, not the results the customer was getting from those cycles, which depended on the software they were using.

CP/CMS was more successful than TSS because it was simpler, but we were not yet out of the woods. At Lincoln we were able to get about 15 users on the system before performance degraded. The problem wasn't that the system got a bit slower for each user we added, it would get to a point where the system

started chasing its tail, and once it did that, the performance for all the users went into the tank.

I moved to Connecticut in September 1968. CSS hadn't gotten the machine yet so there really wasn't much work that anyone could do. Those couple of months were among the most difficult in the company's history. Everyone was very anxious to begin, but couldn't really do anything other than find space and prepare for the machine. The initial team was in place, Dick Bayles, Nick Pisarro (a recent graduate from RPI who Bob Bernard knew), and I made up the technical team, and Joe McCarthy and one or two salespeople made up the sales team. Joe had worked for SBC and knew a few other good sales people that he was lining up. Bob Bernard was CEO, Dick Orenstein was VP, and Ken Bridgewater was another VP. Ken was still doing contract work to bring in some money for the company. I don't think the venture funding had closed when Bob hired this team, but I didn't know that. So I imagine that Bob was quite stressed, having a payroll that I assume he had to pay out of his own pocket. The funding did close shortly after I joined. The company raised $600K from Clark, Dodge and Roberts Rudder(?) and possibly another firm. It was quite a tribute that Bob could get this funding because I am certain the investors had no clue about what we were doing. We were inventing an industry, and there wasn't much to point to back then that was at all similar. I guess they must have thought computers were going someplace, so why not invest in CSS.

The 67 was delivered in November of 1968. Dick Bayles did a great job in planning the facility setup and worked with IBM to install the machine. We had to hire a crane and remove the office building windows so we could get these huge, heavy boxes to the second floor where the computer room was. It was exciting to watch this happen.

I couldn't imagine what the IBM people must have been thinking. This machine cost close to $100K a month to rent so I am sure they were quite concerned to accept the order from a bunch of young guys like us. Even most big companies didn't have machines this powerful. This was their latest technology and we were one of the first to order one. Dick Glazer, who was our IBM salesman can tell that part of the story much better, which I am sure is very interesting. Dick saw a good thing in CSS, made a sizeable investment in the company, and joined the company several months later. That too, took considerable imagination and courage.

The machine was up and running in late November and we gave initial customers free time. Joe McCarthy and his small sales crew had enough contacts in the tri-state area to get some good interest. I think we started charging in December and actually did about $5K in business, but I am not sure if it was December or January when we actually billed customers. In any case, in the following month revenue grew to about $25K, and then went quickly to $40K, then $60K, and then we hit the wall. The system started thrashing when we added more users and response went to hell.

Everyone started panicking. With the cost of the machine close to $100K a month and only being able to generate $60K, how could the business survive? Those were exciting times...customers loved the service...the clock was running...the demand was clearly there as our build-up of revenue indicated, but we had big technical problems, and cash was flowing out rapidly.

So I started to work on solving this problem. I worked very closely with Nick Pisarro, one of the most brilliant programmers I ever worked with, and I continue working closely with him to this day. I did the design and Nick wrote the code. Here is what I remember about solving this problem of thrashing, a well-known problem with time-sharing systems at that time.

What made this problem unique was that we didn't know how much program memory each virtual machine would use because the underlying operating system CP didn't know anything about what was running in each of the virtual machines. Whatever it knew it had to discern by taking measurements. I did know that we had to let each virtual machine get enough real memory so it could do meaningful work. More primitive time-sharing systems were designed where the user program and the underlying system were closely coupled. With CP, there was a clear separation between the two, which enabled different operating systems to run in each of the virtual machines and the system running in a virtual machine could run on a physical machine without CP in place. That is what made the design so powerful. This

concept is now coming back with Virtual Servers in the Windows Server environment. Companies want multiple applications to share the same server hardware and be assured of strong isolation between the applications. In this way they can save on buying a server for every application. Most applications do not use all the resources that their servers are capable of delivering; many of them only use 5 to 10 percent of their capacity. Large companies have data centers filled with thousands of servers and using those servers more efficiently can save millions of dollars in hardware and very expensive data center space.

So here is how I solved the problem. I imagined that I had climbed into the machine and could view the users outside demanding service. I knew I had to handle those users in a more orderly way, instead of the first-come, first-served way the CP dispatching algorithm was written. I said to myself: it took a certain amount of time for CP to bring in enough of the pages of the user's program into real memory (what we call paging); we had to allow that user program to run so it got a reasonable amount of work done. If we brought in too many users they would all be fighting to get at the limited real memory and we wound up throwing out the pages for older users to handle new users before the older users made use of those pages, causing them to be brought in again. That is what thrashing is.

I have come to realize that people also thrash when they have more projects to work on than they can reasonably handle. It takes a certain amount of time to get into a project where you can do useful work. When you have to switch to another project, all that time getting into the first project is wasted. A good manager closely monitors how many projects a person can handle well. Poor managers just keep throwing work at their people and wonder why nothing gets done.

In order to solve this thrashing problem, we created two queues. The Working queue consisting of those users we would allow to request more memory and the Waiting queue, which was made up of users whom we didn't service at that time. In a time-sharing environment there is also a third group, those who are not requesting any service because they are either inputting the next command or getting output printed or not doing anything at all.

I decided we had to measure how much memory we had left at any given point and limit the number of users we were putting into the Working queue at any given time so we could enable them to get the memory they needed. If the system was fully busy with the users in the Working queue, and another user started demanding service we put that user in the Waiting queue. Not until we released a user from the Working queue did we bring a user in from the Waiting queue. If the users in the Working queue were not getting enough work done because they couldn't get enough memory, then we reduced the number of users in the Working queue so the remainder could get more memory and get their work done. The user we took out we put at the head of the Waiting queue. We would start new users from the Waiting queue on a first come, first served basis. This kept the system running so that the machine was spending at least 60% of its time on user work rather than on system overhead. One other point to remember is that we were dealing with very short periods of time. A user could get a lot done in a fraction of a second. So we were still switching between users very rapidly.

Nick and I constructed the new algorithm and tested it. Lo and behold it worked very well; the machine didn't degrade all at once. It degraded gracefully as we added more and more users. We were never spending an inordinate amount of time in system mode, so users would get a meaningful portion of the machine to get their work done. Remember, we only charged users for the time that their programs actually ran, not for system overhead. With the original algorithm, when the system started thrashing, the amount of time we spent in system overhead went through the roof and the time spent on user work went down significantly. We would use almost all the CPU paging things in and out, not delivering any CPU time to users. So revenue generation when the system thrashed was very low.

It took us only a couple of days to get this new algorithm running and the difference was amazing. We were able to double the number of users we could handle, effectively taking the revenue up to about $150K per month on a 256K box. I said that if we added another 256K we would be able to deliver even more CPU time to users. When we added the extra memory, we were able to generate $250K per month, which made the business viable.

A bit later on we improved this algorithm further by remembering the pages that a user had in real memory when we switched them out of the working queue. Instead of bringing those pages back in one by one as the user generated “page faults”, we brought in all the pages in their “working set”, which was much more efficient. Bob Jay (he later changed his name to Bob Jesurum) did the work to develop that algorithm. This made the system even more efficient, allowing us to handle more users, and generate more revenue without adding hardware to the system.

In May 1969 while we were in the midst of solving this problem, the company needed more capital. The sales curve looked very promising, so the investors anted up another $850,000 and this enabled the company to continue building its business.

A key event was an ad that we placed in Time magazine that said, “Debug COBOL programs 40 times faster!” That ad, which I think was placed only once, brought in a torrent of business. We had implemented the IBM COBOL compiler so our sales people could go to insurance companies and banks and sell VP/CSS (we had renamed our version of CP/CMS) to debug their COBOL programs and then take those programs in- house to run them in production.

In those days, COBOL programmers could get one or possibly two turnarounds a day because their in-house machines were used for production work most of the time. They worked with punch cards, so any slight error would waste a run and a whole day. On our system, which used the same IBM COBOL compiler, they could get dozens of turnarounds a day. They would use an IBM Selectric Typewriter terminal to edit and enter their program, and then they could run and debug it. We enabled them to put in test data and even added a symbolic debugger so they could debug their programs interactively. This program development service sold like hot-cakes. In late 1969, we opened a second data center in Sunnyvale CA, which serviced customers in the west.

In 1970 we went public and raised $3M. And that was the beginning of the next set of problems.

I guess Bob Bernard had dreams of creating a big software company. He set up a group at Brown University under the auspices of a professor there. It would be our think tank. Meanwhile no one was paying attention to the business, like sending out bills and collecting what people owed us. So we started bleeding cash. I remember working furiously on our accounting system, and when I got it running we realized we were almost out of the millions we had raised. That brought on what was called the “night of the long knives”. The Board made Dick Orenstein CEO, we closed down the Providence research center, and laid off dozens of people we had recently hired. I remember we had to tell a fellow to go home the day he reported for work. But the company’s business was solid and we were able to recover, but not without quite a bit of pain. To tell the truth I was too young and too involved in the technical side of things, so I didn’t fully realize the severity of the situation. I knew we weren’t doing the right things, but I didn’t have a clear picture of the consequences. I just put my head down and did what I thought I had to do to make things happen and focus on servicing our customers.

One incident worth noting was a problem we had with Bell Labs. They were a very good customer, spending a lot of money each month. I had developed a special file system for them so they could have multiple users updating the same file at the same time. It was kind of neat and they liked it. Then one day something happened to the disk they had their files on. It was a hardware failure. Now that wasn’t a problem because we did routine backups of all our customer files. Except, for some reason we didn’t backup Bell Labs files. I remember John Pryor, the salesman for the account calling me and asking “What should we do?” Without hesitation I said tell them exactly what happened--that we screwed up and didn’t back up their files. Explain how it happened and then say that we will do anything to help them get their data back. He sucked in his gut and did it. After the initial shock, the folks at Bell Labs rolled up their sleeves and together with our staff gathered stacks and stacks of printouts that were used to get their data keypunched and reloaded into the system. The way we handled this problem so impressed Bell Labs that they went on to become a much larger customer than ever before. I have always felt that it is best to deal very directly with customers. Never try to hide anything; they understand that technology is fragile, and having confidence in their vendor relationship is the most important thing.

I look back at these early years at CSS and think about how much I have learned from that experience. First, when you have a group of dedicated people working well together, all trying to solve a common problem; you can pretty much do anything. Second, it was a magical experience. A number of people who were part of it all have commented to me that they searched for a similar experience elsewhere, but couldn't find it. I attribute that to the task orientation we had at CSS. It was an open, cooperative environment, especially during the early years. We were all focused on succeeding. The third thing I learned is to go with your gut. I remember being so focused that there was no time to "think" things out. The most difficult thing is to trust your gut. Too often in my later career I wish I had gone with my gut sooner.

