Interview with Dennis Ritchie

Ritchie: He's [Ted Dolotta] involved with - I'm not sure whether it's as a job or whether it's a looser association with some magazine publishing group, that has magazines in Japan. So this was actually for a - to be published in a Japanese magazine.

MSM: Japan seems to be his new stomping ground.

Ritchie: Yes, he spends a lot of time there.

MSM: Yeah. Fortunately, on expense account.

Ritchie: Yes.

MSM: You worked on Multics didn't you?

Ritchie: That's right. I was a graduate student; I worked part time at Project Mac. And there that was sort of the beginning days of Multics. And when I came here - fairly soon after I came here, I also got involved in this part of the [inaudible].

MSM: [Inaudible].

Ritchie: Ah, well it was sort of two different things. At MIT mostly what I did was documentation. I sort of read things. Wrote some descriptions of various aspects of the file system. Did not really do very much programming at all. At least on Multics. When I came here it was divided up, and one of the things that was going on here was working on the terminal [inaudible]; output devices, input/output devices. Things had already begun to quiet down in terms of activity at the Labs. At one time there was a fairly extensive amount of work. Quite a few people involved. And this had pretty much been stopped except for the group here at Murray Hill. And there was a fairly small number of - and that involved Peter Norman, who was with us earlier, activist who was most interested, plus Joe Osanna, Thompson, and various other people would occasionally be involved. I sort of got into that fairly - when I first came I thought I'd stay away from it for awhile. But, then, as things worked out I soon got interested in what was going on there.

MSM: Then your special interest as a grad student?

Ritchie: My work was fairly theoretical. It was in recursive function theory. And in particular, hierarchies of functions in terms of computational complexity. I got involved in real computers and programming mainly by being - well, I was interested even as I came to graduate school. But, I guess it was even the first year I was a teaching fellow in the introductory programming course and I continued that for at least, I guess, at least three years. Being a teaching fellow in that course and also a couple of others, but, ah, this was at Harvard. The computer situation was fairly - reasonably backward there, it characterized [inaudible] quite well-known people. But they didn't really have a comprehensive program. Of course this was early 1963.

MSM: Your Harvard class was '63?

Ritchie: Yes.

MSM: Okay, then we overlapped by just a year then. I was Class of '60.

Ritchie: At that point there were really weren't any, any computer science probably was not even called that practically anyplace. Harvard has always been part of the sort of strange relationship with the subject. That's what's [inaudible]. At that time in particular there were individual courses that were for people who were quite well-known. So, as I said, my actual thesis work was on - was fairly theoretical but I spent a lot of time, also, doing programming.

MSM: Was your thesis work at Harvard?

Ritchie: Yes.

MSM: And in the recursive function area, who were you working with?

Ritchie: Pat Fisher. Who, when - I guess the last year or so he went to the University of British Colombia. He is now at Vanderbilt. Pat was sort of not really the Harvard advisor. I was fairly independent.

MSM: So you wandered down to Project Mac for a summer job?

Ritchie: It wasn't a summer job. I think by that time I guess I was just more interested in doing programming. I don't know how I even found my way to it, precisely. But it was a part-time job throughout the year. Basically to support myself.

MSM: How did you come to Bell Labs?

Ritchie: My father worked for Bell Labs. Hence I knew very much about the place. I knew it because also he was involved with telephony. I came to know the Multics work from MIT. I actually interviewed at several places at Bell Labs. I was out Indian Hill and Whippany as well as the research group here. And this certainly looked the most interesting.

MSM: Did you have an idea of what you wanted to do? A specific sort of agenda in mind?

Ritchie: Not a specific agenda, I would say a general agenda. I guess I definitely wanted to be - by the time finished school it was fairly clear that I didn't want to work anymore in, I didn't want to stay in theoretical. I was just more interested in real computers and what they could do. And particularly struck by the - how much more pleasant it was to have interactive computing as opposed to decks of cards and so forth.

MSM: Had you met that at MIT?

Ritchie: Yes. That's right.

MSM: You worked on CTSS there?

Ritchie: Yes I used CTSS because that was what was used during the development of Multics. I didn't work on that - I didn't have anything to do particularly with the development or maintenance of that system. I was just a user. And, ah...

MSM: At Harvard, at this time they were probably still throwing around card decks.

Ritchie: They were still throwing around card decks. They had some things. They had - there was a thing called, there was a thing - an interactive system that was run out of some place in Southern California. The Color Freed System, which was a fairly special purpose math system. It wasn't symbolic. It really manipulated graphs and functions and could do pointwise arithmetic and functions. It was fairly crude. I mean it was sophisticated in some ways. It did things that were unusual and it had a very fancy special keyboard and all sorts of strange labels on it and so forth. And it was also a cross country connection if they managed to get a phone line in. A data line in to California which at that time was something really very fancy. But that was not, not really generally used. That was a very special sort of experiment of some sort. I was not directly involved with it. I just sort of saw it. The interesting thing of course was that within a few months of that I saw that someone had, at MIT, essentially a simulation of the system on CTSS. Considerably more greater convenience than trying to keep this cross country data link up. Anyway when I came here the Labs was in essentially the same state as far as computing was or a very similar state as Harvard. There was Multics, and the main conflict for the experimental [inaudible] the people who were working on the system was not wide spread. The comp center also had another GE645 which was a Multics machine. But they used it as the GE635 which was the standard large machine with an all-electric main. And at that time the system running on that was just a BAP system. Later they began getting a fairly crude interactive system built on top of [inaudible] which was the name of the operating system for the 635. But Multics was definitely the most sophisticated interactive system there was. And it was really the disappear - the combination of the disappearance of Multics, what it did two or three years later. And just the fact that Ken had always wanted to write an operating system of his own was what led fairly directly to Unix.

MSM: What attracted you to Ken's enterprise? Were you also interested in operating systems or was it that his work held some promise in getting back that interactive system that you just talked about?

Ritchie: I think it was probably a bit more of the later. I knew him already. We had been working together on Multics. There was this strange period which seems now to be longer than it probably was, of maybe 6 months, a little more in which we were agitating very hard to try and get a new machine of our own. A fairly substantial one in the half million dollar range. Nearly 25 years ago, it was a lot more even than it is today. And that sort of was at several stages that sort of almost seemed to be about to go through. And never did. And so we were sort of frustrated with that. Ken actually made a couple of starts at doing the system before there was any special hardware. In the very last days of Multics he actually wrote a very, very inner part of an operating system for the 645 machine standalone. I think he got as far as making it a few characters on the terminal. That didn't really go very far. And it was obvious it was pointless to waste time on it because the machine was not going to be around.

MSM: And you planned on following that?

Ritchie: What I'm getting at is that we had been sort of involved with working with each other in various ways. I had been doing other things as well during this period. I did a compiler for the Altran system which was Stan Brown's algebra system which was completely independent of Multics and that crowd. That was one of the things that made it particularly obvious that having an interactive system was nice. The Altran stuff was actually written on the terminal. I forget now whether the GE TSS stuff was around then or whether it was actually written in Multics terminals and somehow transmitted. But the representation of the program - in the first place was in Fortran, and in the second place it was obvious we were going for punched cards. Typing out on terminals. But the formatting was very rigid and whatnot. And there were batch jobs and object decks and stuff like that. And it really made it very obvious that having a smooth interaction with the machine was a lot more fun, a lot less clumsy. It was really a combination of things that got me involved. Just the fact that the style of thing that Ken was headed for seemed very much, very desirable. I know this just from the work with him.

MSM: What sparked exercise in file systems?

Ritchie: I think that was basically part of Ken's desire to do a system of his own. So I don't know what very specific thing sparked it. But he started about bosses and he started about paging systems and he actually wrote a simulator for paging behavior and then he and I and Rudd Canaday started drawing pictures on the blackboard or maybe white board. Drawing the structure of this proposed system which was in most ways the predecessor of Unix. Some of the ideas appeared even, I think the notion of special files appeared then. I think that was mine. Most of the ideas were Ken's. I think that particular idea was mine. I.e., the devices had a representation in the file system so that they could be opened and closed and whatnot.

MSM: How much of Multics was behind your thinking in this file system?

Ritchie: The relationship of Multics to this is actually interesting and fairly complicated. There were a lot of cultural things that were sort of taken over wholesale. And these range from important things, the hierarchy file system, tree structure file system, which incidentally did not get into the first version of Unix on the PDP-7. This is an example of why the whole thing is complicated. But any rate. Things like the hierarchical file system, choices of simple things like the characters you use to edit lines as you're typing, erase character, [inaudible] character, were the same as those we had. I guess the most important fundamental thing is just the notion that the basic style of interaction with the machine, the fact that there were the notion of command line, the notion was an explicit shell program. In fact the name shell came from Multics. A lot of extremely important things that we completely internalized, and of course this is the way it is. [Inaudible]. A lot of that came through from Multics. As far as other technical details, there were enormous differences. The scale of not only of both of effort and of resources, of machine resources, was just incomparable. The whole, I forget now how big the Multics machine was, it wasn't very big of course by today's standards, but the first PDP-7 was tiny. It was only 8k words of memory, 68k [inaudible]. So there was a vast difference in complexity. And a lot of that was forced just by conditions. But a lot of it really was a taste as well. One of the obvious things that went wrong with Multics as a commercial success was just that it was sort of over-engineered in a sense. There was just too much in it. And it certainly explained why it took so long to get going. Heavily consumptive of resources, of machine and certainly in terms of people required

to produce it. I think there was a real reaction in the design of Unix. But this is sort of going back to the other side again. Another strong resemblance is the way that the I/O system worked; the read and write system calls were essentially taken directly from Multics. At least they're the form of the call. On the other hand Multics had a fairly complicated thing called the I/O switch. The argument, the handle that you used to do a read and write was not a file descriptor as in Unix which is a fairly simple object. An I/O screen in Multics that implied a fairly complicated thing called the I/O switch that sort of [inaudible] lots of different modules had a chance to massage the calls as they went around accessing modules and whatnot. And in fact in some ways the Multics I/O switch was like the screen stuff that came much later. On the one hand the read and write I/O system, part of the I/O system in Unix was modeled after that of Multics or at least a simplified version of it. But there was nothing at all of the Multics virtual memory system. That in fact was the most characteristic feature of Multics. That was the thing that made it technically unique. The fact that it so-called one level stored, that the objects that programs manipulated were segments which were named, and the segments caused the memory segments, the things that are visible to the program, that's the named object that you can store into [inaudible] responded also to objects in the file system. That was the absolutely crucial distinguishing thing about Multics. And that just didn't, that was just completely out of Unix. Partly because it seemed very hard to do, partly because it required fairly sophisticated hardware that was memory mapping hardware, segmentation hardware that was just not on any of the machines we had. Until just over the last few years has not been available on [inaudible] of machines. Furthermore, particularly in hindsight and to us, or to me anyway, there was some sort of conceptual problems, in that Multics really wanted to make a very uniform view of the world based on these segments in which the paradigm for all activity was all relationship with the file system was to institute operations on segments in memory. But of course that's not really possible because you always have devices of some sort. We have terminals and whatnot. And those just can't work on the same model. We've got to have a different model for dealing with I/O. And so there were various between these two possibilities in Multics. In particular there was a thing called the FSIM, the File System Interface Module, that really made a segment, a memory object look like a device. Then you could treat this memory thing in the same sort of way. In fact they cleaned things up a little bit later. But in the original thing the only way you could deal with very large objects was through this FSIM which, if something was too big to fit into a given segment it would use several of them. [Inaudible]. That was something that was inconvenient to do by hand. At any rate, my view is that even though Unix is usually understood does not have this segmentation notion. It only has an I/O notion. If you're going to restrict to one main Paradigm, the one Unix has chosen is more powerful in some sense because in the sense it can apply to objects other than addressable memory. In particular that's how things like pipelines and our current form of interprocess communication and so forth all work. By using this same scheme of doing reads and writes. Usually of course people have begun adding something of the segmentation idea, so called mapped files and so forth. They haven't really made it into any of the standards, probably because they're not really a very portable idea, at least with current hardware. Not all machines can handle the notion.

MSM: You said you're interested in getting away from theory, and getting into real computers. One of the things that I've always found interesting about Unix, is that although it might not be theory driven, it nonetheless it's had a strong theoretical component [inaudible]. Neat and sophisticated programming going on here but generally exemplifies something theoretical or has a theoretical base. How far has the theory been from your activities? Have you had it in mind? Thought it out? Have you chosen to do things for theoretical reasons?

Ritchie: No I would say that I haven't, really. The kind of work that I was doing as a graduate student was in some sense fairly abstract. The kind of theorems that were being proved referred - it was kind of strange actually. It was the area of computational complexity. What functions are inherently more complex take longer time to compute than others. This particular bulk of hierarchies of functions. But the interesting is that that hierarchy began at exponential time. So that was sort of the unit [inaudible] size of its input or the value of its input. And, this really is - in some sense dealing with it was partly like programming because when you design a function to exemplify something it bears a strong resemblance to programming a machine. But the results you get are really totally theoretical because, of course, exponential time things are typically too hard. And the reason it's amusing is that about two years or so after I finished the whole business of NP completeness and all this interesting structure and questions within things that take exponential time as opposed to building on top of that. Suddenly the hot field was the stuff that I just sort of said was sort of baseless. We won't examine this because it all seems a little bit too complex whereas we can get a nice smooth structure. I guess the point is that this particular aspect of things really does not have very much direct applicability.

MSM: The sort of stuff that Al Aho was doing when he came.

Ritchie: There are many aspects of theory that do have applicability. Certainly a lot of the insights into languages.

MSM: Has that been an area of interest to you at all?

Ritchie: Not of great interest. It was the sort of thing that was included in [inaudible] and whatnot. Actually, most of the interesting work in parsing came after I was out of school. [Inaudible]. The notion of parsing generators, and so forth. I think to get beyond just me, though, I think that one of the things that is very refreshing is the fact that there are all sorts of people doing quite a range of different things nearby. In particular both theoreticians and more practical-oriented people. And it's a lot easier to keep in touch with everything they're doing here than it seems to be in most places. So yeah, there definitely have been people who have contributed things that are not just sort of hacking out solutions to the problems but doing programs or systems based on some sort of theoretical principal.

MSM: When we speak of it sometime as a programming environment and yet it occurred to me as I was talking to Al this morning, it's almost a meta-programming environment. It just happened?

Ritchie: Well, I don't think it just happens, or at least it seems to me that there has been a conscious effort to hire people who are fairly broad. Whom they really looked on as a plus, even if you're theoretically inclined, if you have say some programming experience. And the pressures to turn something into something theoretical into some kind of practice, they can be quite subtle and not burdensome, but I definitely think that they're there in this organization. And of course sometimes people justify things in saying - they justify it in sort of theoretical terms, or at least when they're following the area of computers. In a lot of cases it really does happen. Things like AWK for example which was the condition of the program to produce parsers from the grammars. I'm not even sure what the exact historical period is. I don't believe that that idea was his, [inaudible], had already been realized that this was the technique by which the grammar describing the language had produced a machine that would recognize the language. He was one of the first actually to do this and produce a package that allowed people really to use this, as opposed to either proving that it could be done or demonstrating it as some sort of private program of their own, but rather producing a

general thing. These people had a language that they wanted to try. That they were using tools to help take the description of the language and turn it into the compiler. I think that was the kind of work I really very much encouraged.

MSM: You designed C.

Ritchie: Yes, although not from scratch.

MSM: I see. An adaptation of B?

Ritchie: It was adaptation of B. That was pretty much Ken's. He actually started out as Fortran. Ken one day said the PDP-7 Unix system needed a Fortran compiler in order to be a serious system. So he actually wrote and sat down and started to do the Fortran grammar. This was before AWK. He actually started in TMG. It took him about a day to realize that we wouldn't want to do a Fortran compiler at all. He did this very simple language called B, got it going on the PDP-7 and whatnot. B was actually moved to the 11. A few system programs were written in it, not the operating system itself, but the utilities. It was fairly slow because it was an interpreter. There were sort of two realizations about the problems with B. One was that because the implementation was interpreted it was always going to be slow. And the second was that unlike all the machines we used before which were word oriented we now had a machine that was byte oriented. In that some of the basic notions that were built into B which in turn was based on BCPL were just not really right for this byte oriented machine. In particular, B and BCPL had notions of pointers which were names of storage cells. But on a byte oriented machine in particular, and also one in which the - it had 16 bit words - and I don't think it did originally, but they were clearly intending to have 32 bit and 64 bit floating point numbers. So that there were all these different sizes of objects. And B and BCPL were really only oriented towards a single size of object. From a linguistic point of view that was the biggest limitation of B. Not only the fact that all objects were the same size but also just the whole notion of [inaudible] object didn't fit well with B. So more or less simultaneously I started trying to add types to the language B, and fairly soon afterwards write a compiler for it. The language changes came first. For a while it was called NB for New B. It was also an interpreter. I actually started with the B compiler. [Inaudible]. Because C was written in a language very much like itself and at every stage in the game. So it must have started with the B compiler and sort of merged it into the C compiler and added the various - the type structure. Then tried to convert that into a compiler. That incidentally was sort of another - that the basic construction of the compiler of the co-generator for the compiler was based on an idea that I heard about from someone at the Labs at Indian Hill - I never actually did find and read the thesis, but I had the ideas in it explained to me so that the co-generator for NB which was based on this Ph.D. thesis. It was also the technique used in the language called EPL which was used for switching systems in ESS machines - it stood for ESS Programming Language. The first phase of C was really, it was two phases in short succession of first some language changes from B really adding the type structure without too much change in the syntax, and doing the compiler. The second phase was slower; it all took place within a very few years but it was a bit slower so it seemed. And it stemmed from an attempt, the first attempt to rewrite Unix. Ken started trying it in the summer of probably 1972. He gave up. And it may be because he got tired of it or whatnot, but there were sort of two things that went wrong. One was his problem in that he couldn't figure out how to run the basic sort of co-routine in multi-programming primitives. How to switch control from one process to another. In relationship inside the kernel of different processes. The second thing that he couldn't easily handle, from my point of view the more important, and that was the difficulty of getting proper data structure. The original version of C did not have structures. So to make tables of objects, process tables and file tables and this tables and that tables, it really fairly painful. One of the techniques that we borrowed from seeing it used in BCPO was to define names who are actually small constants and then use these essentially as subscripts. Basically you would use a pointer offset by a small constant that was named to do the equivalent of naming a field of a structure. It was clumsy. I guess people still do the same sort of thing in Fortran. It was a combination of things that caused Ken to give up that summer. Over the year I added structures and probably made the compiler somewhat better. Better code. And so over the next summer that was when we made the concerted effort and actually did redo the whole operating system in C. That was fairly successful. It took the winter of '73 to do that. And there were no really tough problems.

MSM: People, yourself included have said that Unix was a distillation of some of the best ideas in operating systems in the '60's. I want to come back to that point but let me take a variation on that. Is C a distillation of ideas on programming languages as they had emerged during the '60's?

Ritchie: No, only one aspect. It's definitely a limited language variety in various ways. I think Unix probably covers a lot more of the real important ideas of that period than C does. C is quite low level; all the objects it deals with are really very concrete. The operations it provides on them are very [inaudible]-oriented. The control flow in [inaudible] is conservative in terms of [inaudible] straight line. None of the more interesting things like co-routines of various kinds.

MSM: What I found interesting, I counted when I was writing a list interpreter in C was that even the assignment operator returns a value so that I found that at a certain point that my C routines were beginning to look like LISP routines. I had my little C functions and I started embedding them and I started to get a lot of families of parentheses that were long enough and I said to hell with it, I'm going to do a list by [inaudible] instead. But it does have that applicative quality to it. Was that conscious? Am I right in discerning it?

Ritchie: I think you may be exaggerating it. Either that or you found a particular style and application in which it became more visible [inaudible]. Particularly if you - well, I guess there is a style in which there are a lot of different operators that are of one kind or another that are implemented as functions. But since they are not syntactic operators they have - the functions have to be embedded in arguments to each other. I don't think that's unique to see.

MSM: It wasn't something you had consciously?

Ritchie: Let's see... what are the other ways in which C is sort of [inaudible] covers a small part of the space of languages. I guess that's it mainly. I mean - oh, storage. The different kinds of storage that are possible in C itself are very simple in just the automatic variables, stacked variables and static, statically allocated things. There's no built in off-stack heat type storage. There's no garbage collection. Whereas language design - I mean, probably because it's easier to do a language than to do an operating system, there's less work in producing the compiler. Language design has sort of always had a little more freedom associated with it. There are lots more ideas that are really very different from each other as compared to systems. So no, I wouldn't call C a distillation except at the very specific part of it. I guess the thing that has made it successful - well, there are

really two things that are making it successful. One is the association with Unix. It just got carried along. The second is the fact that I think that there is a surprising amount of work [tape reverses] so you can keep a [inaudible] the real hardware details. In particular so that the portability is possible with a little care. There are lots of machine dependent things that can be visible to C programs. Part of the art is to learn what sort of things you can depend on, and what you shouldn't. I guess the only real explanation for success is that there were a lot of people writing in the language, typically prose, are now able to write in C. The programs are likely to be a lot better for it. They do have a chance of moving. I think portability, and keeping portability in mind is something that you really have to learn to do. You have to convince yourself that it's important. It's easy to say, look, this is just a one odd program, and I know it's not going to anywhere but here. Why should I care that this is something that will never work on another machine? Sometimes people consciously do that or just below that.

MSM: When did you decide that Unix had to be born?

Ritchie: That was not in there in the beginning. It did have roots. Part of the claim of Multics was that it was going to be portable because - and nobody ever did anything to try and prove that. They sort of upgraded different newer models of the same machine, but nothing less than that. The business about Unix came because there were a couple of things that were happening. A lot of these interesting tools began being developed for Unix. Things like Yak. And of course since it was not a widely used system at the time there was an effort to make them available to other people. And the best way to do that was to have a C compiler on other machines. Even though the idea in general was quite important, a lot of the Altran project was concerned with a Fortran system. And they were very explicit about making that portable and wrote tools to make sure that the dialect, the subset dialect of Fortran was sort of the intersection of all possible of all possible Fortrans in the world. The real insight that we got - I was responsible for grasping this - was the fact that when moving these programs to - these various Unix tools to - the IBM systems and the GE or then Honeywell systems, the difficulties that we ran into were really not due to the underlying machine architecture, even though C made that visible, or made it possible to see, but really had a lot more to do with the operating system differences. How you do I/O. How you write the so called portable libraries. There were messes in implementing some of the - keeping these programs portable. It had nothing to do with the program itself. It had to with its interaction with the rest of the system. And the thing that really was most important was the thought that instead of really spending all this time trying to make individual programs portable and fighting the operating system, why not move the operating system along, too. In general, the portability aspect of work around here has been - but it took a few years before it was specifically called Unix. Although I have to say that Ken had a - one point early in the game. I think we were probably still on the PDP-7. He came up with this idea of writing in B the very simple operating system that would be very, very portable. It would run on all sorts of microcomputers that were just here. It would not be ambitious. The idea would be that this is something that could be distributed widely. I think what he was getting at was on the one hand something perhaps like UCSD Pascal which was a [inaudible]. Maybe something like, some of these operating systems that people like [inaudible] did. In [inaudible], it was written in fairly portable form. I think that was after this... It was the sort of idea that other people had and it never went anywhere as such. The idea was just put out one day and I don't think he spent any time at all working on it. Even though nothing came of that immediately, or very directly, that was the first sort of specific impulse towards Unix portability. Upright system portability.

MSM: This raises a question of perhaps the timing, your first request - you came off a big machine coming off the big GE's. The first request was either for a PDP-10 or a Sigma 7. You either had to settle for a discarded PDP-7, which is a small machine. And in the end Unix has wound up going on small machines.

Ritchie: Not as small as the original though.

MSM: [Laughs] Well, small is a relative term.

Ritchie: Yes, relatively small.

MSM: But I think also what goes with small machines is not only the question of machine size but also an attitude toward who's going to have the machine, what commitment is one making when one buys the machine, because if you buy yourself a small machine, you have an idea that you're going to use it for a few years and you're going to replace it and it may or may not be a machine by the same family. Maybe next time I'll get myself a Mac or I'll get this or I'll get that. I had the sense that the Labs around 1970 were beginning to make a transition as an Institution from the idea of big central computing facilities and big machines to small machines. Smaller machines.

Ritchie: Yeah, but it took longer. It was not until a few years later that it really began to happen.

MSM: Now, did Unix respond to this or did you help to foster it?

Ritchie: No I think we helped to foster it. I think there was - well, it went along with it. There were a couple of things happening. First, that was when the machines like the PDP-11 began to appear and there had been these earlier machines that tended to be for fairly quite specific purposes. The PDP-7 was a graphics engine. That was the time when a lot of projects began developing what are called OSS's, for Operations Support Systems. The way these worked out were that they tended to be mini computers dedicated mainly to doing a specific support sort of thing for the telephone company. A very typical one was one which would keep - collect and keep trouble records. The ESS machine or Crossbar machine or something like that, you know, spits out in one form or another records of things that have the text going wrong, with individual lines or whatnot. Traditionally these had been punched on cards and then sorted somewhere else by a process I don't understand. The technicians would understand it. And so that an Operation Support System might be something that collects these automatically and puts them in bins and whatnot and makes printouts. These things existed for billing and for taking orders and just all sorts of telephone things. And this was all quite independent of the [inaudible] itself. A lot of people began buying machines like PDP-11's to do this sort of stuff. And some fraction of them - this exemplifies what you're saying, that there were these things away from the main comp center. But at least groups like these, the product they were producing was something that was sort of inappropriate for comp center anyway, it was really a new thing. But of course they had to do their own computation someplace. You know, when you're producing a support system it's not as if it doesn't involve a lot of programming. So I guess the idea of getting your own machine, the departmental machine, along with being more sensible - the availability of reasonably powerful and affordable machines I think was the most important thing. It was not a software issue, it was this. And one of the great strokes of luck was that Unix was a system that appeared at just the time that this new phenomenon that was beginning to - was occurring, because there was really no chance

that we would, just as there was no chance that we, or at least, didn't persuade our own management to spring for all this money for this untried thing. There was really very little chance that Unix at that time would [inaudible] someone who was running a big comp center. It was just too unformed. It wasn't even a just a matter of being risky, it just didn't do enough. There were just too many things that inaudible]. That was when people were - the combination of having machines that were sort of affordable and did more or less lead to the effort; they really did more or less lead - ultimately lead to the near overthrow of comp centers in the traditional sense, or at least the great weakening of their power. Software for all these machines tended to be pretty [inaudible].

MSM: You think that within that [inaudible] this operating system that worked on these small machines? Give people added insurance. [Inaudible].

Ritchie: Yes, although there were plenty of projects that were either working on their own operating systems or using manufacturers' systems [inaudible]was clearly a very fertile [inaudible].

MSM: So that was Ken's system, and then he tells me that you guys figured out a way to get the console to act as a second terminal. I mean the display to work as a second terminal. So then it became Ken and Dennis's machine.

Ritchie: Most ideas in the system, and actually most of the working out of the ideas were his, but Thompson was in it from the start, and [inaudible] at least in some phases we sort of worked together, and periods in which we sort of wrote programs together.

MSM: What I was getting at was that issue of leadership/ownership; you must've had a terrific sense ownership the two of you had in the beginning. There must also have been a point at which you had to loosen that sense of ownership to open the thing up. Were you conscious of that happening?

Ritchie: I was conscious of it on and off. You have to remember that this did take, this had been going on over [inaudible] there was no particular moment in which there was a sort of [inaudible]. Even in the '70's there was this history started that Bell Labs [inaudible] that did the programmers [inaudible]. And the point about these groups is that in fact sort of one of the interesting things about the whole development of the system is there always were people outside our own group who were actually [inaudible] never a matter of distributing something [inaudible] other people. There were always lots of people who opened up the box. They always did have independent rules and needs. So it really never was a stage acceptance. The very, very beginning of which he had complete control on what was going on. By continuing to make progress and generally by doing a reasonably good job of producing sort of the next generation of things was an effective, a very strong [inaudible] just because other groups, and this date was later sort of extended [inaudible]. There was always a very strong influence because people tended to pick up what we did and it continued for years afterwards with a diminishing fraction of the total system. As long as the USG had PDP-11's, the compiler that Aho did for the PDP-11 was the one that they used. Even by this time they had their own version of the operating system, that they controlled and wasn't ours, until - probably until this very day [inaudible] at this point we've got [inaudible] fraction at least, and AWK, too, was still owned by the AWK conspiracy, or consortium. So there's obviously been an enormous loss of control over [inaudible], a fraction that we can do anything about. It really isn't spread out over a long time. I guess the biggest realization I have now is occasionally getting into search of things that the Summit People do or the standards committees do or Berkeley does or whatever is just the - things have such an enormous momentum, that to deflect them requires enormous effort. Occasionally we get involved in something and say this is wrong. We really should do this instead of that. Or just not do that. I don't care what you do as long as it's not that. And, it really does take a lot of effort.

MSM: You're anticipating a question I wanted to ask Ken. It's the counterpart to your question about, or to the observation that Unix is a distillation of the best ideas. Were there bad ideas of operating systems in the '60's and about computing in the '60's you didn't want to see? [Inaudible] Whatever else is going to be, it is not going to be this, and it's not going to have this [inaudible]. Do you have any bad examples in mind that you were working early on?

Ritchie: I've never been convinced that Multics's virtual memory model was right, and in an interesting experiment is coming back. I think it was over sold. Of course it was not the common idea even then. It wasn't pioneer enough to do it because very few other people had it. I don't know, I guess we didn't think in those terms. I guess the thing that, particularly in retrospect, it may have happened, some of it unconsciously, but honestly I wasn't very conscious of, is the extent to which things were not put into Unix because we didn't yet know how to - well, a combination of we didn't know how to do them and we didn't actually need them. And this actually caused general weaknesses as far as [inaudible]. The whole business of inter-process communication of which [inaudible] sort of classic [inaudible], the whole business is very cleverly designed to do sort of exactly what we wanted and very little more. It fits together...

MSM: You were talking in your evolution article about essentially the hack of the message this never received. God help the system if it ever worked.

Ritchie: Right, in the later development that went away and the whole business of float [inaudible] and so forth [inaudible]. All this works very, very neatly and there's no need to talk to each other by means of pipes which we had sort of set up. A little more general scheme was put in place and as a result it was a very big succession of word schemes that people, because they felt they really needed to do something. Having one process call another out of the blue.

MSM: I think they were communicating [inaudible] sequential processes was about that time. That was an issue of some [inaudible].

Ritchie: Yeah, everybody else sort of knew that this was required. In the first place it was not required by anything we wanted to do. And therefore no scheme was worked out.

MSM: Did the size of the system, was the size of the machine and the size of the system you were working on dictate that?

Ritchie: In a general way, but not in any particular thing. There was nothing we decided that we could not do because it wouldn't fit. That never happened. There was no specific feature. Put it that way. It induced an approach to things.

MSM: I tend to think that a process communications been using on big central systems and you weren't building a big central

system [inaudible] size constraint.

Ritchie: I think more it was a matter of [inaudible] things the style of computing - we were just interested in using the machine in a limited number of ways. It wasn't required for the kinds of things we were interested in doing.

MSM: And those ways, where were they coming from?

Ritchie: It was just a matter of wanting to, most of the programs are either straightforward and interactive commands that we type, or perhaps they derived it. Or perhaps some actual computation that takes in the place in the background. Nothing to do with autonomous events happening wanting to tell somebody that this event is happening. What's a better example? Well this isn't quite in the same track but it does serve. Some people's interprocess communications problems came for reasons that were directly related to the machine being small. And that is that they had something that was logically one single process but it wouldn't fit. And therefore they had to chop it up. And then the pieces had the problem of communicating. That of course is an artificial example. Theoretically it's a sort of artificial example because we shouldn't have chopped it up. But of course they had to do it. [Inaudible]. They had to work because of artificial restrictions. [Inaudible]. Except that there are definitely limitations on Unix. Many of them they are simply because we did not [inaudible] bother with that and the result of something that is a lot more coherent, because we're not going to solve all the problems. I guess the sort of genius in both Unix and C for that matter is the fact that while both languages have this approach they have a fairly strict approach about what they are willing to do and what they're not willing to do. The fact they manage to cover sufficiently large fractions of what people really needed, or at least some group really needed, while still retaining a sort of smallness of compactness of design. It really was not planned that feature, that aspect was not planned in an explicit way. It can't be. You can try but it's very hard to do. And I suppose the reason is that things work that way is that we did in fact start with some fairly well formed ideas on what we wanted. It wasn't [inaudible]. But, these designs were pushed to the point that they still showed that they solve problems that we are interested in solving themselves as opposed to satisfying all the things or doing all the things on a bulletin list, or surveying all the people that we could find and ask them what it was that they wanted. It didn't really have to prove a point in the same sense that most other people do. It obviously was important to make something available as other people became interested. But we didn't have to make a product. We didn't have to write enough pages to get a thesis. It really was a matter of being able to push things in directions that are interesting to go. And you only had to push as far as seemed to be worthwhile. [Inaudible] I think the keys to this that made this succeed where people or talents might not have made such a...

MSM: And similar tastes?

Ritchie: And tastes. It might have been more constrained than in other environments. As far as the success is concerned there was a good deal of luck involved in being in a position to ride this particular wave of the growth of machines that was suitable. Lucky to have the license it required - that the licensing stuff worked the way it did, too. There was a long period being able to have lots and lots of universities. A small number of commercial and government firms kept it on very good terms. A lot of people were interested in the system. [Inaudible].

MSM: It seems ironic it's effective marketing when marketing was not the intention and there was no product to sell.

Ritchie: Yes. On the other hand marketing takes a rather luxurious approach, namely to say we'll give it away for nearly free for ten years. And then start - in most cases you can't arrange that.