A NEW STRUCTURE

FOR AN

OPERATING SYSTEMS COURSE

by

G. Alan Creak and Robert Sheehan

(alan@cs.auckland.ac.nz)
(http://www.cs.auckland.ac.nz/~alan)

(robert-s@cs.auckland.ac.nz)
(http://www.cs.auckland.ac.nz/~robert-s)

Technical Report 162

Computer Science Department,
Auckland University,
New Zealand

1999 December 22

CONTENTS

INTRODUCTION.

This report has grown from a seminar which we presented to the Auckland University Computer Science Department in 1995. Though blandly entitled "An Operating Systems Course", it was in fact an account of changes which we had made in the nature of the department's final undergraduate year operating systems course since Alan took charge of it in 1985. Robert joined in 1988 while Alan was on leave; since 1989, we have presented the course jointly except during Alan's further leave in 1996.

The course has been changing from its original traditional form for many of the years since 1984. Most of the changes have been minor, and have followed developments in the subject, or filled in gaps which we have perceived in our treatment. The main topic of this report, though, is the single cataclysmic change which took place in 1989 (with a smaller, but still noticeable, aftershock in the following year). We made the change because we believed that the traditional presentation of operating systems as an academic subject was lacking in certain respects; we presented the seminar in order to put forward our point of view, and we now encapsulate it for posterity in this report.

The seminar followed the big change at a very respectful distance. The long delay was not occasioned by secretiveness or coyness, but by continued dissatisfaction with the state of the course. Similarly, the report has not crowded the seminar, but in this case the reason is simply pressure of work. A consequence of the delay is that we've forgotten many details of the process which got us to this point – so the story, while as true as we can make it, is to a significant extent reconstructed from our knowledge of what has happened, and approximately when. Our historical account certainly feels tidier than did the development at the time, so perhaps it should be regarded as how we ought to have gone about the task of development – but one can argue that history is always like that.

In the time since then, the course has evolved a little, but the overall structure, and our views, remain unchanged. While we don't expect ever to be completely satisfied with this, or any other, course, we are now fairly ready to accept that the development is (approximately) finished. Experience suggests that this must be an illusion, but hope springs eternal, etc. – and a publisher showed interest in the notes, which rather suddenly expanded to textbook scale when our recommended text for the course went out of print just before the beginning of the 1995 year. Correspondence with one of the authors raised hopes, which later turned out to be ill founded, that a new edition might be available for 1996, so we thought it best to muddle through 1995, and marvellously extended the notes to fill the gap. (Our dealings with the publisher are a saga in themselves, and they dragged on for over four years before the publisher decided against publishing, but that's another story.)

The future of the course is now uncertain. Alan has retired, so in 1999 the course was presented by Robert and Dr Mark Titchener, while in 2000 it will be presented by Robert and Dr Emilia Mendes. In 2001, Robert expects to be on leave, so the course will pass wholly into other hands. Life is like that; we are happy in the belief that we gave it our best, and had some good ideas.

CHAPTER 1.

WHAT'S THE PROBLEM ?

KNOW YOUR ENEMY.

Throughout this report, we are presenting an argument to justify our abandonment of a rather well established course of study on operating systems which has developed over many years, and our adoption of a very significantly reorganised presentation of a rather extended collection of material. We refer to the old regime, rather frequently, as the "traditional operating systems course". (We have firmly resisted the temptation to add another word and call it Tosca.)

We hereby acknowledge that this mighty Goliath against which we poor Davids do battle – or this windmill at which we poor dons tilt – is a myth. There is no cast-iron, copper-bottomed, ye-only-original traditional course, any more than there is an average family; there are about as many courses as there are lecturers in the subject, and each is likely to bring personal views and preferences to bear on the course content and organisation. We make many general statements in our discussion, and we shall not be at all surprised if they don't apply to some presentations which might otherwise fall into the overall pattern of the traditional course.

We don't think that there's much that we can do about it. If we are going to present our argument (and we are), we have to argue about something – and, if the something we want doesn't exist, we have to invent it. As our argument is about how we have changed our course, one of the somethings is necessarily the state of the course before we started; that's our "traditional course". This is a simple practical necessity; we can't keep adding all the qualifications which we would need to allow for all possible variations in the course every time we make a comment. The approximate intension might be not quite true, but dealing with it is practicable; the extension is true, but impossible.

Nevertheless, though our adversary is a straw man which we have set up for the purpose, he is a straw man with a fairly substantial skeleton. The skeleton is (as we shall show in the next chapter) most nearly visible in the currently available collection of textbooks on operating systems, where a rather well defined corpus of common material, and a fairly standard treatment of the material, are described. We suggest that this common approach is the operating systems analogue of the average family; each is a statistical myth, but each does typify useful features of that which it models, and is a reasonable basis for discussion. This is our "traditional operating systems course".

IN THE BEGINNING : SOME AUTOBIOGRAPHY.

The story starts with Alan, who at the end of 1984 took over the operating systems course in the Computer Science department. To understand what happened thereafter, it is slightly illuminating to know just a little about Alan's background at the time, and in particular about his encounters with operating systems. The relevant parts are tabulated below. The dates are more or less reliable, but the "mileposts" identify specific memories which act as anchors.

When	Milepost	Comment
Alan in pre-computing days : interested in computers, read books.		
1956,7 ?	Alan's first sight of a computer : visit to EDSAC.	Alan already knew something about computers, mainly the electronics of analogue and digital computing. There were no operating systems. He seriously considered changing his undergraduate course to electronic engineering, but didn't. He kept on reading about computers.
1961 ?	Alan's first use of a computer : an IBM1620, and Gotran.	(Gotran was a sawn-off Fortran which ran, interpreted, on the IBM1620.) There was no significant operating system in software, but lots of instructions for manual operations. He kept on reading.
1966 ?	Alan's first encounter with an operating system.	Alan knew about operating systems, but GEORGE (ICL 1903 ?) was the first practical experience. Reading continued. He was beginning to have preferences : languages and artificial intelligence were interesting, hardware was mildly interesting, operating systems were boring.
Alan becomes a computist :		
1969	Alan applies for a job in Derby : lectureship in mathematics.	He carefully didn't apply for jobs with any mention of operating systems in their descriptions. The job was lecturing, mainly on computing to support the department's new Higher National Diploma course in computer studies. Most of it was programming. There was some material on operating systems – which was sad, but part of the job, obviously unavoidable, and still not interesting.
1973	Alan applies for a job in Auckland : lectureship in the Computer Centre.	It wasn't a technical job; the original requirement was to instruct in Fortran, etc.. Alan was, informally, responsible for the educational activities of the Computer Centre, while colleagues were responsible for the technical functions. Again, a bit of operating systems was unavoidable : as well as programming courses, we gave talks about using the B6700 MCP and its programming language, WFL. The system was still not interesting, though the language was rather more so.
1976 ?	The Board of Computer Studies is established, offering undergraduate courses in computing.	The Computer Centre academics took part in the university's first organised offering of courses in computing; Alan looked after programming languages. Later, he took over half of Nevil Brownlee's computer studies operating systems course while Nevil was on leave. He coped adequately, but didn't find it fun.

1980	Alan moves to administration of the Computer Centre student computing service, based on a DEC-10 machine.	With the development of the Computer Science department, most of Alan's lecturing job disappeared. Alan didn't, so his work changed. He became more involved in student services, and therefore inevitably with the DEC-10 operating system (Tops-10), but without much enthusiasm. This was real hands-on experience, and very educational. It reinforced his impression of non-simple system behaviour and strong coupling between different system parts. He also became aware that there's a lot more to a real operating system than is mentioned in the textbooks. There is some evidence of activities in system monitoring[1], and of involvement with development of Zeno[2], intended as the university's student service operating system, but eventually abandoned.
1984	Alan moves to the Computer Science department	At last – back to academia, and a chance to take up once more his interests in programming languages and artificial intelligence. And goodbye to operating systems !
1985	Alan takes over the operating systems course	The best laid plans of mice and men .... But someone had to do it, no one else wanted to, so down to work.

The course as inherited from the previous incumbent (Richard Lobb, who will turn up again in a bad pun in chapter 4) followed the traditional pattern, using Deitel's textbook[3] and sticking to it reasonably closely. Alan survived 1985, the exigencies of working into the new course (and another at the same time) leaving him with not much leisure for thinking anyway. Predictably, the experience confirmed his unhappiness with the topic. He contrived to introduce two minor changes which helped to maintain his interest and such sanity as he was able to summon; in view of his stated interests, it is perhaps not surprising that these were short treatments of command languages and of programming languages used to write operating systems. Otherwise, the notes from Derby and from the Computer Studies operating systems course formed the nucleus from which, with the help of textbooks and Computer Centre experience, a rather traditional course grew in a rather traditional way.

It was an instructive year. Alan's first full-scale immersion in lecturing on operating systems thoroughly rubbed his nose in the subject – or, in more polite terms, was effective aversion therapy. It worked well, transforming a lack of enthusiasm for operating systems into a confirmed dislike. Or did it ? He still found the systems uninteresting, but boring rather than objectionable. The aversion was more to lecturing on the subject than to the subject itself.

More precisely, then, for "operating systems" read "that operating systems course". This realisation came from Alan's reflecting on the year, with the constraint that the course was going to happen again in 1986, and perhaps for some time thereafter (little did he know), and that the chance of escape – short of leaving the job and going elsewhere – was small. He had learnt more about operating systems from the closer contact experienced in the Computer Centre, and wondered whether there might be a better course in there trying to get out. In the interests of sanity, it seemed advisable to try to help it to get out if at all possible, so he directed his attention to seeking out possible directions of improvement. The first step (and this is where the story *really* starts) is to look for the factors which are at the root of the dislike. So here goes.

WHAT'S WRONG WITH THE TRADITIONAL COURSE ?

Something was clearly amiss with Alan's position at the time, for it was contrary to his own principles. For a long time, it has been true that :

Alan asserts that any subject is interesting once you get into it.

(Following this principle has not led him to fame and fortune, and certainly not to conspicuous academic success, but he nevertheless asserts that it has been great fun.) This is certainly, on the face of it, inconsistent with the remark already repeated ad nauseam in these pages that operating systems are not interesting.

To identify the source of the inconsistency, we can analyse the parts of that assertion one by one, in an order which is somewhat contrived for maximum effect.

Alan asserts that ....

Could he be wrong ? – unthinkable, to Alan, at the time, if not more generally. But, even if not, there is sound experimental evidence in support of the assertion : several subjects which Alan had initially viewed with deep suspicion turned out to be interesting once he had discovered more about them. Whether or not that's a general phenomenon is unimportant, because it's Alan's lack of interest which is under discussion.

.... is interesting ....

Well, *is* it interesting ? Alan asserts that it isn't. It is true that interestingness, like beauty, is in the eye of the beholder, but he offers an analogy in justification of the position.

In his early life, he collected stamps. He got off to a good start with the merged collections of his father and two uncles, and collected more (stamps, not uncles). He owned a Stanley Gibbons catalogue, knew about perforations, overprints, watermarks, gum, printing techniques, etc. Despite this expertise, though, he lost interest at around the age of 15, because the subject never seemed to get anywhere beyond that; it remained a collection of mildly interesting facts, which never cohered to produce any structure at a more general level.

The subject of operating systems, as embodied in the traditional course, has a similar structure, or lack thereof. There are several topics – processes, memory management, file systems, etc. – which can be discussed in some detail, but again they don't come together to give a unified account of the system as a whole.

.... once you get into it.

That begs the question of whether or not Alan was "into it", whatever that means in the context. As the context is Alan's assertion, only he can say what it is intended to mean, and he is reluctant to commit himself to a precise definition. He will go so far as to say that, as compared with other things into which he believes he is, he's into operating systems. As evidence he offers his reading (books, lots of papers, *Operating Systems Reviews* (to which he has conscientiously subscribed since 1986)), his active engagement in lecture courses since 1985 and in the past, and his direct experience in the Computer Centre. That's a good deal more impressive than his background in some other subjects which he finds interesting.

.... any subject ....

"When you have eliminated the impossible, whatever remains, however improbable, must be the truth"[4] : perhaps Operating Systems isn't a subject ? That also needs some interpretation, but if we assume that a subject is the sort of thing about which you might want to give a lecture course it makes some sort of sense.

That opens up a new train of thought. One could try to write down the attributes which might reasonably be expected of a subject, and seek them in the traditional operating systems course – but this is still Alan's story, and instead we shall follow his thoughts by considering his objections to the traditional course, as found in practice and embodied in textbooks of the time. There are about four of them.

1 : It's bitty. The analogy with stamp collecting is to the point; the course is a collection of bits, without much coherence. The material has not much depth; there's rather little to study, though there's plenty to know. No unifying principles appear, so the bits remain bits.

Those with an affection for the study of operating systems might feel that some complaint is appropriate, but should recall that we are discussing the traditional operating systems course, not the operating systems themselves. As a less emotionally loaded parallel, we might consider a course on lecture room science, in which one studies the contents of lecture rooms. There are several topics, each of which is quite likely to be relevant to any instance of a lecture room : illumination (lights and switches), timing (the clock), furniture (tables, chairs, etc.), presentation (blackboard, overhead projector, etc.). That gives us at least four simple subsystems – but we can study those for a long time without seeing the point of the lecture room.

2 : It doesn't work. The course material in fact doesn't account at all well for the actual behaviour of the system. Alan's Computer Centre experience with measurements on real running systems taught him that system behaviour is much more complicated, and very hard to analyse. Everything interacts inscrutably. The course material isn't wrong, but isn't sufficient to predict the behaviour of practical systems. It describes how all (well, some of) the parts work – but it doesn't add up to how the system works (and wouldn't even if the other parts were added).

Consider the lecture room again. In practice, the subsystems do interact, and we could study what happens in lecture rooms when they're used, but our study of the separate systems doesn't help us much in interpreting the overall patterns of behaviour. The interactions happen in ways which certainly don't just grow from the subsystems themselves – the behaviour of the whole system involves quite different principles.

3 : It isn't very adaptable. Again, this is a comment about the traditional course, which is commonly based on a model like a timesharing system, and doesn't extend very well without introducing many special cases. It certainly didn't apply at all well to software on the microcomputers available at the time, which was a particular case of some importance to the course, as the students' experience was gained mainly from just these microcomputers. Why didn't it apply ? Surely the microcomputers had operating systems ? – and, if so, shouldn't they be mentioned somewhere in the course ?

In fact, the microcomputer systems didn't fit the traditional pattern, and were therefore ignored – but that's surely not satisfactory : if there's a subject called operating systems, then it should work for *all* operating systems, not just fashionable ones. You should be able to take it and apply it to anything that might plausibly be an operating system, and it should work. The exclusion of early microcomputer systems has sometimes been justified by the assertion that they weren't *real* operating systems, because they lacked some function which is deemed essential for qualification as an operating system, such as multiprogramming. Apart from the dubious validity of appealing to some Platonic ideal of an operating systems which is presumably apprehended by revelation, the practical wisdom of studying a topic which arbitrarily excludes a large and increasing proportion of the world's computing activity must be questioned.

4 : It misses out much of the topic. Even apart from the microcomputer case, the traditional course gives little or no attention to several topics which are significant in operating systems. Some examples :

- Controlling the system. It is fairly unusual to find any treatment of topics related to the user interface for the system. More recent books are more likely to give some sort of treatment

to the topic, particularly now that graphical interfaces are common, but it remains superficial; in earlier texts, even the admission that there was a command language or some system component which interpreted it is rare indeed.

- How to get the system going. The course deals with phenomena observed in running systems, but not how to get them into that state. Topics such as system configuration and cold start procedures are rarely mentioned.

- How the disc is used for different purposes. In most practical systems – even microcomputer systems, starting rather early in their development – different parts of the available disc space are used for different purposes. The disc is used for filing, message buffering, cold-start sequences, virtual memory, and other purposes; whole disc partitions are used to contain different operating systems. There is more involved than the question of disc space allocation which might be the only topic discussed in early textbooks, and quite a lot of it has significant impact on practical use of the computer systems.

- Using devices. This is a notable omission. It might be the consequence of the early attitude that devices were at once so complicated to drive, so private, and so different from each other that there was little or no chance of presenting a unified treatment of devices as a class. Tanenbaum makes the point well in 1987 (note, after Alan started worrying) in the preface to his textbook[5] : "Most books and courses devote an enormous amount of time to scheduling algorithms, for example, which in practice are usually less than a page of code, while completely ignoring I/O, which is often 30 percent of the system, or more".

- Accounting. This topic is hardly mentioned in any textbook or syllabus we know, though it can have significant effects on system design and implementation.

If those accusations are true, we think that it is fair to suggest that the course falls short of what we'd expect of a "subject". One would hope that a topic worthy of study at university level would have enough content to permit the development of a coherent and effective treatment in terms which would apply over a wide range of circumstances. Incomplete coverage is less important, and could even be seen as desirable; one could argue that there should be more to say about a subject of reasonable stature than could be said in a single lecture course. At the same time, one would expect that the first lecture course on the subject would introduce the main features, and none of the items listed in the fourth point above is an obscure byway of operating systems.

It's true that some of the points made in our list beg the question, because so far we haven't defined what we mean by the title "operating systems". Indeed, that's one of the problems, and we shall discuss it later. For the moment, though, just consider what you buy when you buy an operating system; it will include many bits and pieces – such as those we've listed – which aren't mentioned in many of the standard books. If they aren't part of the operating system, why do you buy them ? If they are included in your operating system, do you complain about misleading product descriptions ? Well, perhaps you do, but not because your system includes a user interface or a bootstrap sequence.

We conclude that, no, it isn't a subject. It has some of the characteristics of a telephone directory : a big cast, but no plot. In terms of analogies with science subjects, there is little development of basic themes; instead, we find lots of bits, without much coherence. One might discern a resemblance to descriptive botany, before genertic theory and cell biology provided a sound foundation for development. (An analogy with engineering subjects might be more appropriate, but we are less well equipped to comment on that. We would expect similar comments to remain valid.)

Can it be made into a subject ? We believe that it can. To do so, it is necessary to identify some sort of deeper structure which provides a framework for the material of interest, and on which a systematic treatment of the topic can be based. Alan has presented the argument in more detail[6]. He

concludes that in a natural science it is appropriate to begin by studying phenomena, but the equivalent for a discipline built round an artefact is to study the design process.

Where is the deeper structure for operating systems ? If we accept Alan's argument (we do, but accept that there could be some bias), then it's much more likely to be associated with the purposeful manufacture of computing systems to satisfy known purposes than with the inherent nature of the components as in the natural sciences. We won't get there by studying the bits; rather, the conclusion points to an approach based on design. The purpose of the artefact is first determined, then structures necessary to achieve this purpose identified, and analysed step by step until implementation techniques are found. This view allies computing with engineering rather than science, but that isn't a big surprise.

Can we test the idea on another example ? Yes, we can, though with an element of contrivance. Recall our brief comments on "lecture room science"; that's an example of studying an artefact. Does the principle work ? Yes, it does. If instead of probing the nature of the lighting, furniture, and other attributes we consider what facilities are required in the context of a lecture, we begin to see much more clearly why the components are as they are, and how the whole functions together.

This argument falls far short of a logical proof, but we find it persuasive. More to the point, Alan found it persuasive when he reviewed his position towards the end of 1985 and the unedifying first year of his operating systems course. He did not then see it quite as we do now, in hindsight, but the nucleus of the idea was there. It implied that the academic presentation of computing could perhaps be better done using a different model. In particular, the argument applied to operating systems, with the possibility of leading to a course with a real cohesive structure of some depth.

There remained only a few questions. The first, and most important, was : what was he going to do about it ? Others followed; for example, what sort of operating systems course did he want ? What should go into it, and how should it be presented ? These took some time to work through, but led eventually to the course as it is today.

CHAPTER 2

WHAT PROBLEM ? – THE TRADITIONAL COURSE

In the previous chapter, we introduced a mythical being called The Traditional Course, and undertook to give evidence that the myth had some connection with reality. It did not seem appropriate to do so before the previous chapter, as we had not then explained our interest in the topic; equally, we can hardly defer it until after the next chapter, as within that chapter we have much to say (not quite all uncomplimentary) about the traditional course. Despite its interruption of the narrative at an exciting point, therefore, this seems to be the best place for our justification.

WHERE TO FIND THE TRADITIONAL COURSE.

There are several possible sources of information to which we might look in order to find out how people organise courses on operating systems.

Perhaps the most obvious source is published syllabi for the courses. These are not hard to find in print, and very easy to find through the internet. We did not avail ourselves of this wealth of information for two half-reasons. First, unless a syllabus is augmented with a significant amount of discussion (and few are), it gives you little more than a list of topics. We wanted to be able to find out rather more of how the topics were linked together. Second, we know from experience that a published syllabus does not always coincide with what actually happens in the course. While neither of these comments would necessarily apply to any particular syllabus, the collective possibility of uncertainty and unreliability was not what we wanted.

Perhaps the least obvious source is the course examination and test papers. We didn't even try that one, but it is noteworthy because Alan tried it as a way of finding out what had been presented in the course he inherited. The argument is that the examinations would be so constructed as to test those points which the lecturer considered important; in practice, it didn't work because apart from the final examination no useful relics could be found.

Perhaps the second most obvious source is textbooks. They serve our purpose very well; they include a selection of topics, and plenty of discussion which we might hope would lay bare the links between the topics and the reasons for the organisation chosen. They are also readily available, with specimens covering the last twenty or more years, which should be sufficient to show up any systematic change over time. This was our choice of source material. All in all, we looked at fifteen books. The criterion for selection was the presence of a copy of the book on our bookshelves, and therefore represents mainly the enthusiasm of publishers' representatives for sales to our operating systems students. We'd have used a more respectable criterion if we'd thought of one. They ranged from the classic Brinch Hansen text through to a comparatively recent offering from Silberschatz and Galvin. The full list appears on the next page.

COMPARING THE COURSES.

We have two reasons for wishing to compare the courses in our sample. First, we would like to compare those we have gleaned from other people's work, both to find out whether our belief in an identifiable "traditional course" is tenable, and, if there is such a course, to discover its construction. Our second reason is that we wish eventually to compare our own course with the other examples.

How do we wish to compare the courses ? Our real concern is with the course structure, but that is not an easy attribute to formalise, let alone compare. Instead, we compare the order of course topics, arguing that, while the relationship between topic order and structure is far from uniquely defined, some relationship will certainly exist. We were also undoubtedly influenced by the knowledge that our own change of course structure induced a change in the topic order that was both dramatic and easy to interpret; we hoped that other differences would also show up clearly.

Per Brinch Hansen	*Operating System Principles*	Prentice-Hall, 1973
Stuart E. Madnick, John J. Donovan	*Operating Systems*	McGraw-Hill 1974
A.M. Lister	*Fundamentals of Operating Systems*	Macmillan, 1979 (2nd edition)
Colin J. Theaker, Graham R. Brookes	*A Practical Course on Operating Systems*	Macmillan, 1983
Mamoru Maekawa, Arthur E. Oldehoeft, Rodney R. Oldehoeft	*Operating Systems – Advanced Concepts*	Benjamin/Cummings, 1987
William S. Davis	*Operating Systems – A Systematic View*	Addison-Wesley, 1987 (3rd edition)
Andrew S. Tanenbaum	*Operating Systems – Design and Implementation*	Prentice-Hall, 1987
Malcolm G. Lane, James D. Mooney	*A Practical Approach to Operating Systems*	Boyd & Fraser, 1988
Raphael A. Finkel	*An Operating Systems Vade Mecum*	Prentice Hall, 1988 (2nd edition)
H.M. Deitel	*Operating Systems*	Addison-Wesley, 1990 (2nd edition)
Ida M. Flynn, Ann McIver McHoes	*Understanding Operating Systems*	Brooks/Cole, 1991
Milan Milenkovic	*Operating Systems – Concepts and Design*	McGraw-Hill, 1992 (2nd edition)
Andrew S. Tanenbaum	*Modern Operating Systems*	Prentice Hall, 1992
William A. Shay	*Introduction to Operating Systems*	HarperCollins, 1993
Abraham Silberschatz, Peter B. Galvin	*Operating System Concepts*	Addison-Wesley, 1994 (4th edition)

The next requirement is fairly obvious : if we want to compare the order of topics we must decide on a list of topics to compare. It turns out that, though the requirement might be obvious, satisfying it is not straightforward. It is tempting simply to take the list of topics from our own course, which will certainly cater for our second reason for comparison, and which is certainly canonical enough for us. We have (properly) resisted the temptation, because most operating systems texts miss out sections of our course, so the list does not cater at all well for the first reason for comparison. We must therefore derive our list from the existing literature. The choice is difficult because of the very uneven levels of treatment accorded by the different authors to the topics which they deem appropriate to include in their books.

Some topics are relatively clear cut and are covered somewhere in all the texts. A good example is memory management, which is covered in a variety of ways. Other topics seldom occur; an example is system optimization, which nevertheless makes up a large section in some texts. As we pointed out earlier, some topics, such as devices, have been ignored because they are messy. Other topics naturally do not occur in the early textbooks as they weren’t significant at the time of publication; a noteworthy

example is distributed operating systems. It is interesting to follow the development of the treatment of distributed systems through the textbooks as it moves from being an afterthought to an essential part of modern systems.

Other topics might be treated more than once as the author explores their involvement in different contexts. This phenomenon is not uncommon in treatments of (again) distributed systems, with a topic appearing once in the context of simple systems early on in the book, and then again much later when discussing distributed systems. This is slowly changing as distributed systems become accepted as normal operating systems rather than exotic variations.

The amount of space devoted to each topic varies considerably across the books. The number of pages ranges from a handful to several chapters. For example, memory management is sometimes split over several chapters, while other topics might receive anything from a paragraph (or even nothing at all) to a chapter in different texts. We have not taken explicit note of this factor, even though it presumably reflects at least in part the authors' judgments of the importance of the topics concerned. A topic might be omitted from an early book simply because it hadn't been invented; in other cases, the level of treatment might be determined by an author's perception of the definition of an operating system rather than the significance of the topic itself. Instead, we have sought a selection of topics which appear in at least a few of the books, and which together give a thorough coverage of all the material.

In the end we decided on twelve different topic areas which seem to cover almost all information in all the text books : operating system concepts and history, processes, scheduling, concurrency, memory management, deadlock, input-output and devices, file management, multiprocessing and distributed systems, protection and security, optimization and modelling, and user interface and job control. As we suggested above, these topics certainly don’t represent equal percentages of the books.

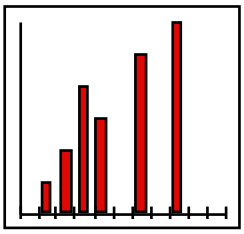
It would be possible to discuss the different orders of topics in the various textbooks by a series of pairwise comparisons, but it would be difficult to discern any broad patterns in the results. An alternative, offering some prospect of a more uniform treatment, is to choose a canonical ordering and gauge each of the courses against that. A potential difficulty with this method is that the usefulness of the results depends rather strongly on the quality of the canonical ordering chosen. If this is quite arbitrary, the results of the individual comparisons will be equally arbitrary, and it will be hard to identify significant features. At the other extreme, if we had access to an order which we had reason to believe was ideal, then each comparison would highlight every departure from the ideal and would be very informative.

In practice, we cannot attain the ideal (while we know that our own ordering should be the ideal, we are not sufficiently arrogant to expect anyone else to agree), so we had to find some way to define an order which had sufficient significance for the comparisons to be illuminating. We therefore based our comparisons on the "average order" of the topics in the textbooks; as there was not a wide divergence from a fairly common pattern, we would expect most individual cases to follow the average more or less closely, so that any real curiosities would become quite clear as deviations from the common pattern.

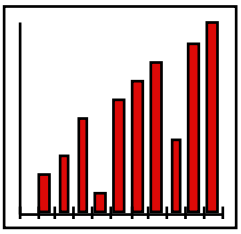
We worked out our "average order" by trying to determine how far into the total list of topics covered by a book each particular topic occurred. The first topic covered was given a value of 0% and the last topic was given a value of 100%. We averaged each topic's position over all of the books (which is a strange thing to do), and sorted according to the averages to get our canonical ordering. Overall, as we had expected, there was a good degree of conformity. Most books were based on an ordering not too different from that of our operating systems course many years ago.

This procedure resulted in a plausible order with only one serious anomaly; it was impossible to place the user interface topic in any really sensible position. Of the five texts which included it, two had it at the end and two near the beginning, putting it on average somewhere near the middle, but only one of the fifteen texts actually included it somewhere near the middle. We have therefore arbitrarily placed it at the end of the order in the comments presented below, and omitted it from the diagrams because we couldn't work out where to put it.

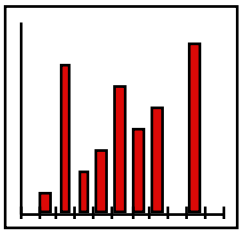
We have chosen to display the results as a set of graphs, one for each text, in which the actual order of each topic is plotted against the canonical order. Absent topics are not marked; we have used bar charts because they produce recognisable shapes. For a text which followed the canonical order precisely the resulting graph would be a sequence of bars of increasing heights, so any deviation from such a line marks a corresponding deviation from canonical order. (This form of display is as clear as we can get it. Experiments with three-dimensional presentations were unconvincing, as it was difficult to identify the results for different texts.)



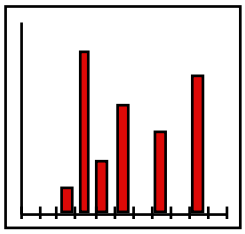
Brinch Hansen



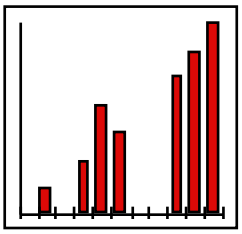
Madnick & Donovan



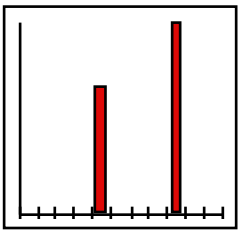
Lister



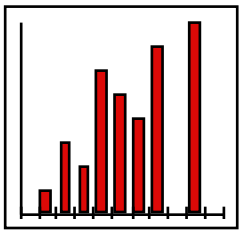
Theaker & Brooks



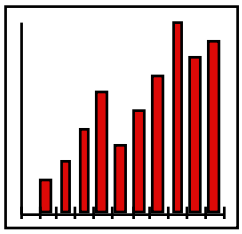
Maekawa & Oldehoeft



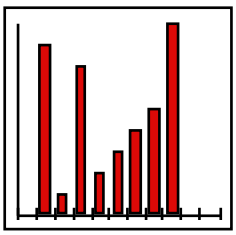
Davis



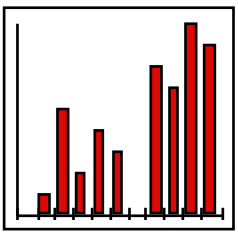
Tanenbaum 1



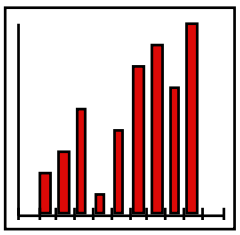
Lane & Mooney



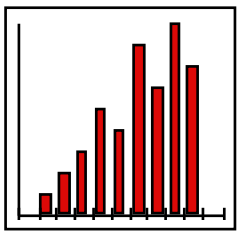
Finkel



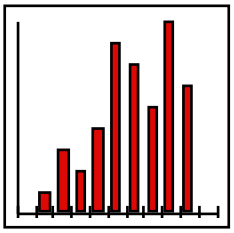
Deitel



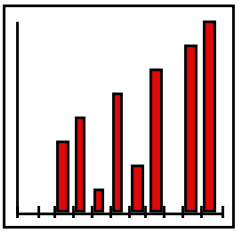
Flynn & McHoes



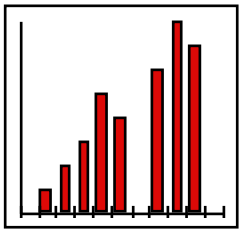
Milenkovic



Tanenbaum 2



Shay



Silberschatz & Galvin

NOTES ON THE TOPICS.

Topic	*Mean position*	*How many*	*Notes*
Operating system concepts and history	1	15	All text books have a section covering topics such as operating system concepts and the history of operating systems. If either of these sections is omitted, the other is present and contains the omitted material. The presentation can be seen as giving an overview of what is to come or as necessary glue to coordinate the separate but mutually dependent topics which make up the subject. For example, you can’t understand memory management until you understand processes and you can’t understand processes, until you understand memory management. Some texts devote a large section to an introduction on machine architecture and organisation as a foundation for the rest of the book. We have included such sections under this heading.
Processes	2.9	10	Most books have an early section or chapter on processes and their properties. In some books, process states aren't discussed explicitly, but the word and the concept are used as though it can be assumed that readers understand what is being talked about. Some authors, such as Davis, only mention these things in case study chapters. The one author who discusses processes late is Finkel. By this stage of his text readers have already looked at the problems of concurrency and are assumed to have a grasp of what processes are.
Scheduling and Concurrency	4	14	These topics are so closely linked that we have merged them into one. Low-level and mid-level scheduling and the problems involved with concurrency alternate places in the books. Just over half the texts discuss scheduling before concurrency, but almost all books discuss one or the other directly after talking about processes.
Memory Management	4.3	15	Usually this topic comes after the discussion of processes, though in three texts memory management appears very early.
Deadlock	5	11	Deadlock (or deadlocks as it is in many early texts) appears in all text books. Some books devote an entire chapter to it whereas in others it is only a few pages interspersed with the chapters on concurrency. Usually this is associated with concurrency (which seems sensible enough) but not always.
Input-Output and Devices	6.1	9	Only nine of the fifteen texts had identifiable sections on dealing with devices or input and output on anything except disk devices. When it did occur it was always either in the middle or towards the end of the topics, except in Shay's text.
File Management	6.8	12	Next comes file management. In all texts where both file management and memory management were mentioned, memory management came first.

Multiprocessing and Distributed Systems	7.7	10	Distributed systems play an increasingly important section in more recent texts, but they are almost always discussed after all of the usual topics.
Protection and Security	8	13	As in the design of many operating systems, the topics of protection and security seem to be add-ons to most texts in the sense that the topic always appears near the end. The closest many of the texts get to discussing security is file access control.
Optimization and Modelling	8.8	5	The few texts which include one or both of these topics mention them near the end.
User Interface and Job Control	5.8	5	As we mentioned before, most new texts ignore this area which many earlier texts saw as part of operating systems. Either an early or a late placement is understandable. Under our scheme it has to come at the beginning because we are starting with the user. Under a more traditional approach it comes near the end after the discussion of the underlying system.

NOTES ON THE TEXTS.

Per Brinch Hansen : *Operating System Principles* (Prentice-Hall, 1973) :

> (This is the book that started it all for Alan, though not the first operating systems text to be written.) The most intriguing feature of this book is the lack of a formal discussion of input-output and file systems; they are mentioned, but only in passing. There is a short chapter on resource protection with the warning, "The protection problem is not well-understood at the moment." The two categories of protection briefly discussed are type checking and process level access control.
>
> There is a case study of the RC-4000 system.

Stuart E. Madnick, John J. Donovan : *Operating Systems* (McGraw-Hill, 1974) :

> Madnick and Donovan present what they call a framework for the study of operating systems. They believe that the concept of a resource manager provides the structure they need. This doesn't help them choose the order in which topics are studied but it does provide them with a way of approaching each section of the subject. They start by describing how interrupt processing and input-output are handled on an IBM 370 in an attempt to provide a view of the hardware capabilities on which the operating system can be built. This is followed by a chapter on memory management before processor (not process) management. The rest of the treatment is conventional and includes a discussion of multiprocessor systems in the processor management chapter. They have a chapter on Interdependencies which illustrates the fact that different sections of an operating system affect each other in ways which a separate discussion of each section does not reveal.
>
> Case studies : a simple example system with complete source code, IBM 360/370 family, CTSS, MULTICS, VM 370

A.M. Lister : *Fundamentals of Operating Systems* (Macmillan, 2nd edition, 1979) :

> Lister explicitly declares that he is working from the bottom up. This is because he bases his text around a paper operating system and he designs this as the book progresses. Because of this the ordering of the chapters is very conventional with one interesting exception. He postpones any discussion of resource allocation until the basic system is up and running. As scheduling algorithms are crucial for questions of

resource allocation this means they don't follow his discussion of processes and appear very late. They also appear very late in our ordering. This is one of the five books with a section on the user interface or job control. Discussion of this is kept until the end because of the bottom-up development.

Colin J. Theaker and Graham R. Brookes : *A Practical Course on Operating Systems* (Macmillan, 1983) :

The book by Theaker and Brookes is the first of our books which doesn't have a section (or subsection) which describes processes. They are talked about, especially in the late chapter on concurrency, but the reader is expected to have knowledge of what they are. At the end of the book they include a chapter on job control systems.

Mamoru Maekawa, Arthur E. Oldehoeft, Rodney R. Oldehoeft : *Operating Systems – Advanced Concepts* (Benjamin/Cummings, 1987) :

Intended as a follow-up textbook to an introductory course in operating systems, this book only briefly touches on processes but spends more than half its pages discussing concurrency and deadlock, including the distributed versions of these problems. The areas of file systems, devices and input-output are missing in this text except as they relate to problems of concurrency. There are discussions of virtual memory, security and queuing models.

William S. Davis : *Operating Systems – A Systematic View* (Addison-Wesley, 3rd edition, 1987) :

As Davis states in his preface, "This is not a theoretical text". The book is constructed around case studies of MS-DOS, UNIX, IBM DOS/VSE, and IBM System/370, from a user's perspective. This means there is a large section on job control languages. Apart from introductory sections on operating system concepts and closing sections on distributed and database systems all other topics are covered from within the case studies.

Andrew S. Tanenbaum : *Operating Systems – Design and Implementation* (Prentice-Hall, 1987) :

Tanenbaum wanted to save students from a theoretical operating system course, so this text is based closely on Minix, the UNIX look-alike operating system and its implementation. Other factors we would like to see included in operating systems courses, such as the user interface and distributed systems, are almost completely ignored. In Chapter 1, we noted his comment on the relative importance of scheduling and input-output in operating systems.

Malcolm G. Lane, James D. Mooney : *A Practical Approach to Operating Systems* (Boyd & Fraser, 1988) :

This was the textbook we used, while it was still in print. After the obligatory introduction and history chapters there is a chapter on the overall design and structure of operating systems before a whole chapter devoted to the user interface, which, according to our analysis, is in the correct position. As with most texts, processes are then looked at in detail including states, scheduling, and deadlock, followed by input-output, memory management, and file management. The authors' style is to cover large amounts of ground, including topics that seldom get mentioned elsewhere or which normally appear as issues when discussing other subjects. In this category would go their chapter on error management. This is part of a sequence of chapters including reliability and security. Following chapters include performance analysis, standards and distributed systems.

Raphael A. Finkel : *An Operating Systems Vade Mecum* (Prentice Hall, 2nd edition, 1988) :

Finkel tries to organise the subject under two principles, the resource principle and the beautification principle. Beautification means hiding details of the underlying layers of the system. He starts by discussing scheduling and leaves concurrency and process construction and communication until near the end. Deadlock and other resource allocation problems are discussed before concurrency. There is a chapter on the user interface.

H.M. Deitel, Operating Systems (Addison-Wesley, 2nd edition, 1990) :

> This book is conveniently divided into several major parts. The second, following the overview, is process management (which does not include scheduling). Next is storage management, then processor management (which does include scheduling). Subsequent sections are on auxiliary storage management, performance, and networks and security. An interesting omission is a general discussion of input-output issues.
>
> Case studies : UNIX, MS-DOS, MVS, VM, Macintosh, OS/2

Ida M. Flynn, Ann McIver McHoes : *Understanding Operating Systems* (Brooks/Cole, 1991) :

> Flynn and McHoes intended their book to be useful to anyone who uses a computer. They say, "this book leaves off where other operating system textbooks begin." This leads to an interesting arrangement of chapters. The first section they discuss after a general overview is memory management. The reason for this is that they believe memory management is the simplest operating system component to explain.
>
> Case studies : MS-DOS, UNIX, VAX/VMS, and IBM/MVS. All the case studies have a section on the user interface.

Milan Milenkovic : *Operating Systems – Concepts and Design* (McGraw-Hill, 2nd edition, 1992) :

> Milenkovic divides his text into four sections. The first, Fundamental Concepts, is like a traditional operating systems text. The second is the description of the implementation of a demonstration operating system. The third deals with details of multiprocessing and distributed systems, and the fourth includes the case studies.
>
> Case studies : MS-DOS, UNIX, iRMX 86,

Andrew S. Tanenbaum : *Modern Operating Systems* (Prentice Hall, 1992) :

> Tanenbaum's most recent operating system textbook is divided in two, reflecting the increased importance of distributed systems. User interfaces are also described in the case studies. The first section of the book is a traditional look at traditional topics : processes, memory management, file systems, input-output and deadlock. The second section of the book, on distributed systems, focuses on the issues of communication and synchronisation before looking at processes and file systems. Both sections of the book take the "what other problems do we have at this level" approach to introduce the material.
>
> Case studies : UNIX, MS-DOS, Amoeba, Mach

William A. Shay : *Introduction to Operating Systems* (HarperCollins, 1993) :

> For a recent book this text is surprising in that distributed systems are only mentioned in an extended summary in the introductory chapter. As in the book by Flynn and McHoes, the first topic is memory management. There is no separate section on processes and mention of process control blocks is included in the chapter on scheduling. Input-output processing is the next section and it precedes the discussion of scheduling and concurrency. There is also a chapter on modelling and queueing.
>
> Case studies : MS-DOS, UNIX, VMS, MVS

Abraham Silberschatz, Peter B. Galvin : Operating System Concepts (Addison-Wesley, 4th edition, 1994) :

> This text has been around a long time and not surprisingly it has changed in its emphasis over time. In the preface to the second edition (by Peterson and Silberschatz) , the authors comment on the late placement

of processes. "Almost every other text places this material at the beginning as Chapter 2. In our experience, this arrangement does not work. " By the fourth edition the ordering had reverted to a more traditional style with processes and concurrency being seen as central and hence dealt with early on. There was one glaring omission (just as in the Deitel book) : no section on input-output, apart from secondary storage. This has been corrected in the 5th edition.

Case studies : UNIX, Mach, and a variety of older systems

WHAT WE FOUND.

It is amazing that introductory operating system texts have changed so little over the years. Of course, there has been development, but large sections of early texts are still current and would not be out of place in a modern book. By the early 1970s the development of operating systems was well founded and topics such as virtual memory and time slicing algorithms had been the subject of extensive theoretical work. File systems had developed into the now ubiquitous directory tree structure, and protection and security was becoming a more important topic.

Since then the major developments have been in distributed computing systems and the growing importance of personal computing. This last area of growth has stimulated a flourishing of user interface design study; oddly enough, this has coincided with disappearance of sections on job control languages – the equivalent of the user interface in older systems – from recent text books. Some major changes in operating system design, such as the move to microkernels, have meant little from the point of view of the topics which must still be handled.

Understandably the emphasis in the texts has changed over the years. Brinch Hansen[7] says in his preface, “This book tries to give students of computer science and professional programmers a general understanding of operating systems – the programs that enable people to *share* computers *efficiently*.” (The emphasis is ours.) Nowadays the emphasis in operating system design is the ease with which the system enables people to get their work done, and in the first chapter of the book by Silberschatz and Galvin[8] this is stated clearly : “The purpose of an operating system is to provide an environment in which a user can execute programs. The primary goal of an operating system is thus to make the computer system *convenient* to use. A secondary goal is to use the computer hardware in an *efficient* manner.” (The emphasis is in the original.) Given this evidence that the shift has been recognised, it is perhaps curious that recent texts show little evidence of any corresponding shift in treatment.

WHAT IS THE TRADITIONAL COURSE ?

Our investigations give strong support to our belief that "The Traditional Operating Systems Course" is real. They also give us some insights into what it is.

Some of the observations are not surprising. For example, the content of the course and its general form are just those with which we have been familiar for many years. There have been changes as operating systems have developed, but no revolutions.

But one of our conclusions was – to us, at least – very surprising indeed. We had assumed that the overall structure of the course could be regarded as a bottom-up development of the implications of having to drive computer hardware in such a way as to get useful results. The textbooks give some support to this view insofar as they follow the pattern of first talking about processes and related areas and then moving out to memory, file systems and protection – but this is a long way from a true bottom-up treatment. Such an analysis would start with the hardware and describe the operating system software layer by layer in some plausible order. We found little evidence of any such discussion in the textbooks; instead, the underlying operating system framework seems to be assumed, with most texts taking a descriptive approach, and presenting a discussion of this assumed model.

We therefore suggest that the common ordering is based on a pinch of bottom-upness and a smattering of the traditional computer scientist's fascination with the CPU, but mostly on a widespread folk memory of how operating systems ought to be. We suppose that when the early operating systems came together from a set of primitives which obviously had to be there, they set a pattern which has since evolved as technology has developed, but that practitioners have become so accustomed to the nature of operating systems as they learn about computing that it has simply been assumed as a background when the same practitioners came to write textbooks. As the background has, in practice, been a very effective and practically useful background, there has been little need to question it, so the folklore has been perpetuated.

Do we really believe that ? Perhaps not quite – but our surprise at the lack of analysis behind many assumptions in the textbooks remains, and we are sure that a more careful treatment is needed if the subject is to be regarded as an example of good scholarship. We believe that the course structure which we advocate here is more firmly based, and a better grounding for academic study.

CHAPTER 3

WHENCE AND WHITHER THE OPERATING SYSTEMS COURSE ?

THE TRADITIONAL COURSE AGAIN.

We left Alan in 1985 when we embarked on the pseudo-philosophical speculations which ended the first chapter. At the beginning of 1986, Alan had not thought as far ahead as that. What he had done was something in the nature of our discussion in Chapter 2, though without either the comparatively careful analysis or the benefit of literature not yet published at the time, and his conclusions were in some respects close to those we presented. In particular, his inspection of the textbooks led him to conclude that they were all much the same, and not at all in line with the structured development from purpose suggested by the discussion in the Chapter 1. (The single exception was the text by Brinch Hansen[7]. This had been used by Nevil Brownlee in presenting the university's first course on operating systems, and was notable for its emphasis on topics such as system design, provability, and so on. That seemed to Alan more like an analytical approach, and might have led to his thinking about top-downery, which will emerge later. But Brinch-Hansen's treatment was based on the resource-manager model, and was also bitty, covering much the same bits as the other books.)

Alan's impression was that the textbook material followed a bottom-up pattern, starting from the raw computer and (optimistically interpreted) asking what was required to make it work. (In the pessimistic interpretation, there is no asking; you're just told how an operating system works.) Typically, the aim of the exercise was not clearly defined, but left understood as making the computer work in the manner to which we're accustomed; that isn't obviously a very good basis from which to develop new sorts of system, which one might reasonably expect to be among the targets of an operating systems course. He didn't develop the hypothesis of self-perpetuating folklore which we presented in Chapter 2, but his "pessimistic interpretation" and unease about the ill-defined aims of the analysis demonstrate his unhappiness about the treatment. Whatever the details, he thought it worth seeking a better way.

The manner to which we're all accustomed was assumed (as we recognised at the end of our analysis) to be a straightforward, typically Unix, timesharing system. At the time, such systems were very widely used, particularly in university computing departments. But they weren't used in our department in Auckland, and it was painfully clear that Alan's students were not at all accustomed to the model. They had been brought up on microcomputers, but these were hardly mentioned in the traditional course, and offered no timesharing experience. Further, future prospects were not encouraging; the old familiarity with operating systems of any sort seemed likely to diminish (it did) as Macintoshes replaced more conventional machines.

As the use of microcomputers seemed likely to grow, their almost complete absence from the subject matter of the course could be regarded as a significant omission, and we've already commented on the anomaly. Unfortunately, there appeared to be no simple alternative. Look at the textbooks of the time; microcomputers rarely, if ever, get a mention. Attitudes were beginning to change, but for a long time microcomputer systems were simply ignored. There is still a strong belief that "MS-DOS isn't a *real* operating system" – which, considering that for many years it supported a significant proportion of the world's computing, is hardly sensible.

As an example of the reason for our concern, consider techniques for memory management. Microcomputer systems then typically used a simple form of memory management in which the upper and lower limits of available memory were recorded by the system, and changed as new software was installed or executed. It is a perfectly good method, but was ignored in the books and courses. Surely that is wrong. If we're studying memory management, we shouldn't leave out techniques simply because they're not used in trendy systems. Simple lowerlimit – upperlimit memory management has been used from the earliest monitor systems up to today; quite recently, it turned up again in systems for integrated circuit ("smart") cards[9]. A treatment of memory management should at least acknowledge its existence, not just ignore it; and it should preferably say why it's inferior. (Which, for very small systems, it isn't.)

Microcomputers were not the only evidence of departure from the supposed standard timesharing system. Other evidence of variety abounds now, and abounded then. The most obvious imminent change then was in the rapid development of different forms of user interface, and much variety in such interfaces seemed likely. Other perturbations to the old settled pattern of computing included communications networks, multimedia, and parallel computing; specialised machines (for work in offices, for database operations, supercomputers) were already in use (though not all have survived); dataflow and reduction machines[10] (and later neural networks) were discussed. Later developments have included hypermedia systems, pen interfaces[11], IC cards[9], and so on.

How could all these developments be accommodated in the traditional old course structure ? Not very easily. The basic material was there, of course, and those new features which were direct developments of the traditional systems could be described by continuing with the same material beyond the traditional bounds of the course. New sorts of computing presented more of a problem; it is not clear that operating systems for dataflow machines would be direct descendants of those for von Neumann machines, and the traditional file systems might not be appropriate for, say, systems with pen-based interfaces. Without an obvious development path, these new ideas could be incorporated only by introducing more and more special cases – which gets us back to stamp collecting. In all cases, the traditional method describes particular existing systems, but gives no guidance on how to design a new one.

How do we escape from this impasse ? One answer is suggested by inquiring further into our case study : why didn't the traditional course easily accommodate microcomputers ? It was because, instead of being seen as useful systems in their own right, they were seen as very poor imitations of real operating systems. And why was that ? It was because the assumed definition of an operating system excluded them. One heard assertions such as (to continue an earlier train of thought) "MS-DOS isn't a *real* operating system because it can't handle multiprogramming".

That gets us back to the complaint expressed earlier, but which bears repeating : if there's a subject called operating systems, then it should work for *all* operating systems, not just fashionable ones. (Chemistry works for all chemicals; not all of it applies in any individual case, but every case fits in somewhere. And if it doesn't, we stretch chemistry. "Methane isn't a *real* chemical because it doesn't contain a benzene ring" ?) You should be able to take it and apply it to anything that might plausibly be an operating system, and it should work. That is not to say that we have nothing to learn from the older treatments – that would be a ridiculous suggestion. It *is* to say that a definition which excludes a significant part of the field of interest isn't a very good starting point.

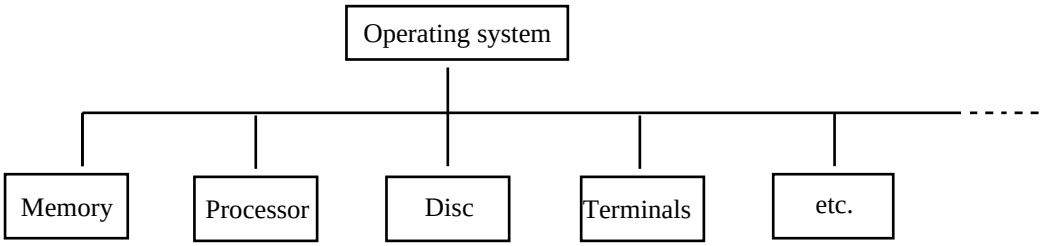
It is as if one were to define a motor vehicle as something with a motor and four wheels. Perhaps if all the vehicles one has ever seen are cars and buses, that's understandable, but it eliminates motorcycles, tracked vehicles, and other useful devices. The real definition isn't primarily to do with the composition of the object (except that it must have a motor); it's to do with the nature of a vehicle, which is to carry things around. If we redefine the motor vehicle as something which has a motor and carries things around, we have a much more useful and productive definition. What is our corresponding definition of an operating system ?

WHERE WE COME FROM : OLD MODELS.

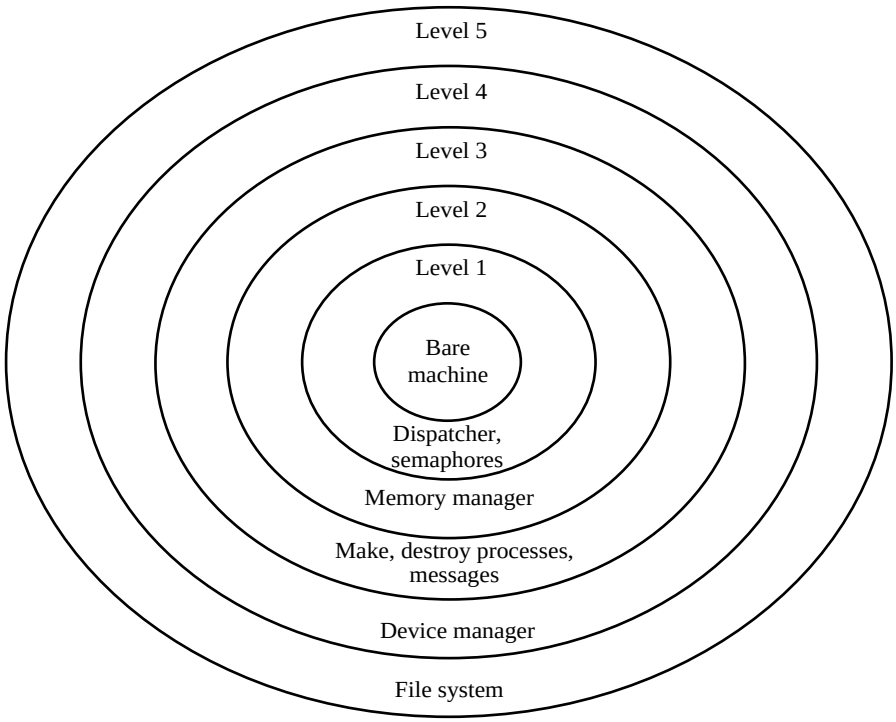
It is instructive to begin by reviewing models which have been used already. All those we know are variants on a theme; here are three examples.

Resource manager : This is the theme, and turned up quite early in the story. The function of the operating system is to manage the allocation of the system's resources – processors, memory, devices, etc. – to various activities as they are requested by programmes. The purpose of this allocation was left to be understood as something like "using the computer most efficiently". It was also understood that some sort of application to getting people's work done was a good idea, so

that achieving 100% processor use from the programme "X : GOTO X" wasn't satisfactory, but there was no obvious principle from which this would follow in the course of orderly development. The lack of such a principle makes it possible to select parts of the system for study quite arbitrarily. In practice, the result was that the operating systems course would address interesting and comparatively simple topics (memory management, scheduling, using discs) but omit equally essential but "unacademic" topics such as maintaining data on users, providing a consistent and useable set of system instructions, and microcomputer systems.



Onion : The onion model is a development of the resource manager model, taking into account some of the dependencies between the resources. This diagram is (redrawn a bit) from the early and influential textbook[12] written by Madnick and Donovan, published in 1974 :



This is unquestionably a useful and helpful extension to the model – but it still gives no insight into the reasons why we have to do those things. Of course, we know why, but it would surely be better if that knowledge were stated and used as a starting point in the development, not merely taken for granted. There is still no component of the model which constrains one to include the unfashionable topics (accounting, user management, etc.) – or, for that matter, which gives one a reason to include the fashionable topics, like process management.

If we press the onion analogy rather too far, we can find what might be a clue to the solution of the problem. Like the model, the real onion presents itself as a collection of concentric spheres (this is an ideal real onion) and it is up to the intending consumer to impose an interpretation upon this entity. A human consumer addresses an onion from the outside, and can, should he so wish, consider each layer in turn while working in to the centre. In contrast, a maggot might work from the inside outwards, probably comparatively uncritically, perhaps regarding each new layer as a delightfully crunchy surprise. It is interesting that the traditional course should

choose to follow the maggot. (The crunchy bit is not part of the metaphor.) Do we want our operating systems to be designed to cater for our computing activities – or do we want them to grow because now that we can do this clever trick, gee whiz, we could do that one if we wanted, and maybe it would sell ?

In hindsight, then, it might be more precise to say that the structure shown by the diagram was not used in such a way as to bring out the reasons which we seek. Roughly speaking, the reason for the existence of one layer of the onion is that it has to be there to support the next layer out. Perhaps a more satisfactory treatment would result if we started outside the onion, and worked inwards, asking what each layer was required to do in order to support the outside that we want, rather than following the traditional order by starting at the middle and working out. In other words, when you know what you're looking for (how the system architecture supports the desired functions) you can find it.

There is another advantage of working from the outside in : you know when to stop. This feature is absent from the usual development.

Client-server : The client-server model developed rather later than the others, and became particularly prominent when computer networks came into common use. One can see the stratification of the onion model in terms of clients and servers, with the inner layers the servers for the services required by the clients in the outer layers. The advantage of the client-server model is that it is more flexible. The onion model, for all its multidimensional image, imposes a one-dimensional order on the dependencies between the operating system components, and in practice it is difficult to extend the model beyond a fairly crude picture of the operating system. The pattern of dependency is such that some components must be placed on the same level, as each depends on the other for some service in some circumstances, and there are questions of what sort of communications within a level should be permitted.

By adopting the client-server model, more complex patterns of dependency can be catered for much more easily. This more flexible model caters well for systems in which different functions are performed by clearly identifiable distinct entities, as is the case with distributed systems and microkernel systems.

It isn't an accident that these models are used in the traditional course : they're the traditional models. They were once exactly right, and in some ways they still are. The early operating systems were developed very specifically to provide facilities for more efficient use of the computer hardware, and offered very little else. They dealt with the obvious problems that could be solved by software in an effective way. As an example of the consequences of this view, the early systems contained little or nothing in the way of code to manage input and output, because the sorts of interface which might be used by devices were outside the control of the system, so uniform treatment was difficult to manage. For this reason, early textbooks often didn't say much about devices.

The resource manager model and those derived from it are primarily descriptive in nature, and they lack criteria with which we can judge what *should* be there. This isn't just a feature; it's a bug. Not having any criterion which you can use to determine what should be included and what shouldn't gives you the freedom to put in the bits you like, and keep out the bits you don't, according to your whim. It doesn't even lead to a very good description if you take the broader view of operating systems, as it misses out anything that doesn't fit into the neat traditional pattern.

On the other hand, alternative models are not thick on the ground. There is only one way out : if we want a model, and the usual models don't work, and there is no other, then we must invent a new one. Or two.

A NEW MODEL ?

What sort of new model do we require ? The preceding discussion suggests a new model which is more robust to change and which would encompass all the new developments – such as microcomputers (in 1985), perhaps middleware (in 1997), and more generally any which haven't happened yet – naturally and comfortably. It should also make provision for the bits and pieces of software which were conveniently omitted from the textbooks. If they are legitimately to be regarded as operating system components, then they should fit into the model; if not, the model should make clear why not in some more convincing way than pointing to an arbitrary list of topics.

This is perhaps an unusual requirement. In other subjects, and even in other parts of computing, we don't often worry about whether there are well defined rules about what's in and what's out – but computing changes so fast that perhaps we need them. In operating systems particularly, which is not well defined to begin with, some sort of guide is likely to be useful. Old ideas fossilise very quickly, and (to follow the geological metaphor) sedimentation is very rapid. Because it is so easy to construct complex artefacts in software, speculation becomes practice very easily, and there is little incentive to hold on to old methods if new ones seem to offer advantages. The result is that it is very difficult to keep up to date with everything, to assess and judge all new ideas and ways of working before they in their turn become obsolete. In such circumstances, some model of the field of operating systems within which new developments could at least be placed, showing their relationships with other system components, would be valuable.

That's a technical reason for wanting a new system model. There is also an argument from the academic side, connected with our desire for deeper structure in the course. What do we need in a university course on operating systems (or anything, for that matter) ? We would like more than stamp collecting; a description at the level of "how it is" will suffice if we've nothing better (botany didn't die), but leaves much to be desired. This is roughly the level of the traditional course, which in some cases became a rather detailed description of Unix. "Why it is how it is" would be an improvement; a discussion of how systems interact with their environments leads to a valuable link with design considerations – and incidentally needs more than Unix, because different environments should be considered. The model becomes an important component of the deeper structure which we seek.

We emphasise the phrase "a discussion of how systems interact with their environments". It is perhaps our first explicit move towards the deeper structure we advocated in the Chapter 1, where we suggested that it should be derived from "the purposeful manufacture of computing systems to satisfy known purposes". Even in the traditional course, alternatives were canvassed (is paging better than segmentation ?), and the benefits and problems associated with different methods were discussed (what sort of scheduling algorithm should we use ?) – but it is not common to see these discussions related to the design of a system to satisfy particular requirements.

One further step might therefore be desirable. At the level described, we would discuss how the operating system interacts with its environment – but the natural expression of the "purposeful manufacture of computing systems to satisfy known purposes" puts the purposes first. We could describe this approach to the operating system as a discussion of "How circumstances make it how it is", and we argue that this is at least among the best available views. It answers the question "What's a good operating system ?" – a system which satisfies known purposes – and it can cope with changing circumstances. We begin with certain requirements of the system and certain resources with which we can construct the system; these are the circumstances, and we can proceed from there by a process of design to derive the sort of operating system which will satisfy the requirements. The emergence of cheaper hardware, new devices (pen interfaces), special requirements (people with disabilities), and so on fits in naturally at some point in the analysis. We can go further : such topics *must* fit in naturally, because the new features are introduced to satisfy certain requirements, and the requirements are our starting point.

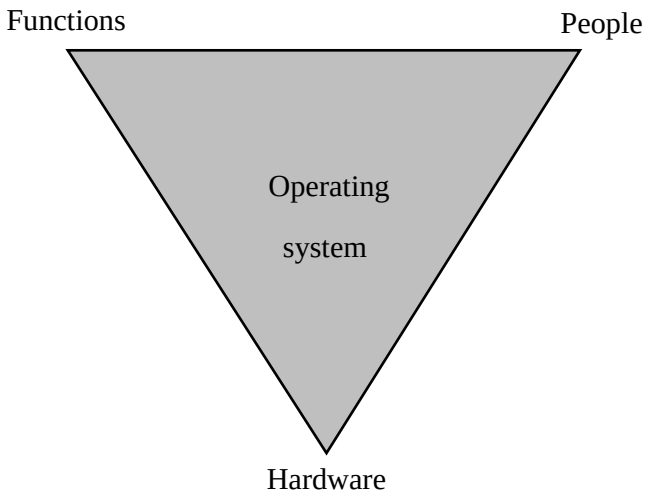
WHERE WE COULD GO : SUGGESTIONS FOR NEW MODELS.

In seeking other models, we begin with a question : what really does determine what you get when you buy an operating system ? With the package you get some stuff without which the machine won't work, and you get some stuff without which it's very hard to use the machine, and you get some stuff that's quite nice, and you get some stuff. None of it is stuff you're likely to produce for yourself. That suggests our first new model :

The dustbin : The operating system is the collection of stuff which people want to use and which isn't done anywhere else.

This model incorporates two features which, if you have been convinced by our arguments so far, are immediately attractive : it explicitly includes people's requirements, which introduces purpose; and it is realistic – the operating system really is a collection of soft(ish)ware which does everything that the hardware can't do and no one else wants to have to do. It's also a realistic model of how systems grow; new features are added because they are thought to offer new facilities which people want but are unlikely to be able, or willing, to do for themselves.

An interesting view of the operating system can be extracted from that description. It is the entity which coordinates the activities of people, the programmes they wish to run, and the hardware in which they wish to run them. In the triangular diagram below, the operating system is the material which occupies the interior of the triangle :



(It is interesting that broadly similar diagrams representing broadly similar ideas have turned up – so far as we know, independently – in at least two other places[13,14]. While this doesn't prove anything, it does suggest that the idea that linking things together is significant is not confined to our discussion here.)

An important consequence of this inclusive definition is that many topics related to effectively using a computer gain a place within the subject. Some of these – user interface management, archive systems, etc. – are rarely dealt with in traditional courses on operating systems, and never dealt with anywhere else. It would be possible to argue that a computing syllabus which completely omits any significant study of operations essential to the use of most computers most of the time leaves something to be desired.

This model is a considerable step forward, but still permits us to include all manner of things that happen to take our fancy. We have a sort of rule for putting things in, but we're slack on keeping things out. We would like to move towards a model which resembles the dustbin in its ability in principle to accommodate any potentially desirable item, but we also require a selection

criterion which will ensure that we accept only items which make a significant contribution to the system's operation.

The introduction of "people" is a very significant novelty, because it gives us a fixed point. This is what we wanted to give depth and structure to the material. Hardware and software change all the time, but people don't. Whatever happens to hardware and software, we'll want an operating system to make them work for people who want to use them – and people's requirements (so far as they're not imposed by advertising agents) don't really change much. In fact, we might take as one of the foundation stones of our new approach the assertion that

PEOPLE ARE THE ONLY INVARIANT

– which doesn't necessarily mean that they must be the central figures in the description of the system. Atoms are, in a sense, central to chemistry, but you don't have to develop everything in a chemistry course from atoms. If you don't make sure that the structure is there, though, there remains a tendency for atoms to pop up again whenever you seek an explanation for any phenomenon. We might expect that people would play a broadly analogous part in the development of operating system theory.

That being so, it might be appropriate to investigate how the emphasis in the model might be moved towards the part played by the people in the system, rather than representing them as equal partners with hardware and software as implied by the dustbin model. In particular, we might hope that such a shift of emphasis could be combined with the requirement for a selection criterion. Such considerations lead us towards the notions embodied in our final model, in which the selection criterion is connected with the provision of desirable services.

The service provider : The operating system is that which provides computing services to people.

In this model, in some sense a combination of the ordering of the onion with the universality of the dustbin, both people and structured development take their places. We think it worth emphasising that, in conformity with our proclaimed invariant,

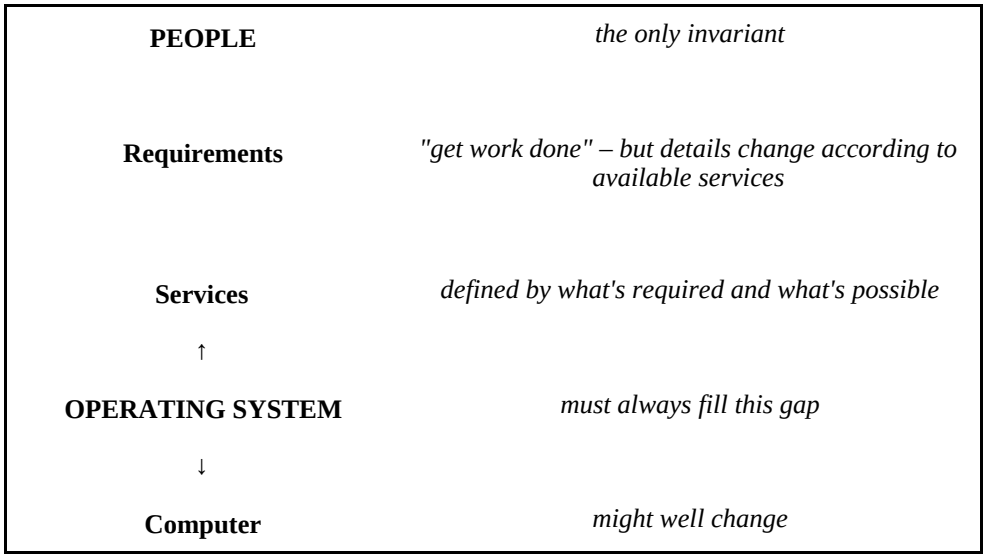
the starting point is people's requirement for computing services.

The target is a way of providing these services using the facilities available, which is primarily the hardware and software, but not necessarily limited thereto. This outline leads naturally to a *top-down* approach, with the hardware as a rather clearly defined bottom, operating in a comparably well defined manner. This contrasts with the traditional treatment, where the hardware is the starting point, and development proceeds in the direction of a rather ill-defined collection of facilities, with matters becoming more complicated (and less comprehensible) as the development proceeds.

Identifying the "top" of the analysis with the people has two specific advantages which are worthy of note. First, it makes the analysis responsive to people's requirements as they change with time. People's expectations of computers are very different today from expectations held even twenty years ago, not only in the great expansion of the domestic use of computers but also in the much greater variety of applications for computing in commercial and industrial milieux. Second, it makes the analysis much less dependent on the details of currently available hardware. One of the original points of dissatisfaction with the traditional course pointed out in the previous chapter was that a course based on a timesharing model was not well suited to students whose experience was almost wholly shaped by microcomputers; with the hardware taking a less prominent rôle in the development, this objection loses much of its force.

More generally, the introduction of the service motif provides the criterion lacking from the dustbin model; analysis based on the requirements for service will lead only to useful features, and will therefore automatically exclude items which are more decorative than useful, and include some things that people don't want, but ought to have, if the system is going to give good service to everyone. (We might think of limits on usage, access restrictions, and other similar restrictive but valuable system attributes.) Emphasis on service concentrates the analysis on what people should have. It does not determine which people should be considered, and we must seek other criteria to decide whether we wish to build a system for Unix freaks, or accountants, or web surfers, or other groups. Even there, though, sensible answers are easy to find; it is still possible to identify a set of requirements corresponding to a common and generally acceptable operating system – and the analysis can also be skewed towards specific requirements if so required.

The diagram shows, rather schematically, how the several topics are related. It is intended to depict only the influences on the operating system; a more complete account of the interrelationships would include such matters as the dependence of people's requirements on their experience of using operating systems, the development of hardware in response to people's expectations, and so on. Fortunately, these issues are well outside our terms of reference, and we shall ignore them.



DOES IT WORK ?

Euphoria aside, we might ask whether the proposed model works in practice. Does it indeed lead us to a more satisfactory academic treatment of the subject matter of operating systems, and to a better and more useful lecture course ? At the end of our report, we shall attempt a guess at the answer based on our experience. For the moment, we suggest that this approach does seem to satisfy several of our requirements rather effectively. For example :

- It includes the microcomputer systems. If operating systems had always been thought of as service providers, the microcomputer systems might have been incorporated into the computing mainstream earlier. Instead, they were regarded as toys until much too late.

- It includes traditional services (memory management, process management, file system, etc.), but also many components which used to be called "utilities". With modern highly modular systems and good (or, at least, more easily accessible) application programmer interfaces it's easy to bolt new parts onto the system, so facilities such as file compression, virus detection, and so on are widely available. Whether all (or any) such "extensions" should be regarded as operating system components is a matter for judgment, but that they are so easy to attach, and how to control their

interactions, are surely matters which bear on the operating system design and structure, and could usefully be discussed.

- It is easy to adapt to accommodate new requirements. Consider, for example, the suggestion of "network computers", which rely on external service providers for much of their software support and storage space. Our analysis of basic requirements leads to the identification of a need for some storage medium for more or less long-term preservation of materials in the computer system. Traditionally, we satisfy this requirement by introducing a disc system; for a network computer, we need only observe that the same requirement can be satisfied by the resources of a network. The result is to move the emphasis on disc management to a corresponding treatment of network management – which, in the circumstances, is certainly appropriate.

All in all, the effect of our suggestion is to move from studying a chunk of software produced by a computer manufacturer to studying the computing environment we want, and how to make it happen. This may include manufacturer's software, but is just as likely to include additional components acquired from other suppliers or home-grown, as well as more abstract features such as the establishment of standards. We sum up the change by suggesting that the focus of the course has moved from

developing software to make a computer work efficiently

to

developing an environment to get a task done effectively.

(We noted in Chapter 2 that Silberschatz and Galvin made the same point in their 1994 text; perhaps they'd been to our lectures.)

Whether or not you see this change as a good direction of development for the study of operating systems depends on your point of view. We suggest that it is an appropriate change, given the shift in emphasis of the computer industry as a whole from efficient use of the computer hardware to efficient use of the people who do the work.

In fact, the change in course material is not as great as might have been expected; it is clear that the second statement of intention above includes the first, so we are forced (not unwillingly, we should add) to return to the traditional material because that is our means of providing the services which we discuss. As evidence to support this assertion, we might appeal to the 1989 ACM curriculum[15] for computing, where there is a sort of definition of the subject of operating systems. The basic statement is :

"This area deals with control mechanisms that allow multiple resources to be efficiently coordinated in the execution of programs.".

We have no difficulty at all with that definition – though we'd want to define "efficiency" (which they don't) in terms of people, not machines. Without such a redefinition, it is hard to justify what appears as a waste of time spent on drawing windows and other graphic ornamentation on the screen. The ACM recommendations also include a section on human-computer communication, but it is strongly directed towards computer graphics. While cognitive psychology and "modes of interaction that reduce human error and increase human productivity" are mentioned, they form only a small part of the topic; we would prefer to see them take a much greater part, and in our course they do. It is interesting that even though the authors introduce design as one of three "paradigms" for computing, and they even list "major areas of design and experimentation in the area of operating systems", they produce nothing at all even approximately related to our design approach to the overall system. The list contains examples of design within the subject areas traditionally regarded as proper components of operating systems, but nothing about designing the system as a whole.

We find ourselves, then, with a conviction that the traditional course is not altogether satisfactory, a scheme for a new sort of course which we believe might be better, and a general understanding that to provide the structure we seek we should adopt some sort of top-down analysis starting from the position that the system is there to provide service. What next ? We take up Alan's story once again.

CHAPTER 4

MAKING IT ALL WORK

DEVELOPMENT.

In practice, not all the considerations outlined in Chapter 3 were clear to Alan at the end of 1985, though some components (most important, the "invariant") were there right from the start. There is even some documentary evidence; he wrote it down in 1985[16]. Despite this, though, the next task, that of working out the ideas into a plausible lecture course, turned out to be not nearly as straightforward as he expected. One of the reasons for his difficulties is that the way from a conviction that there should be a new course to the course itself is by no means uniquely defined.

The major difficulty was inherent in the nature of top-down analysis. In effect, there is Alan at the top – but which way is down ? The answer must be determined in the context of the scheme outlined in Chapter 3; Alan views the world from the item "People", and must therefore assess the meaning of "Requirements" in order to define the "Services" which must be provided, but just how this should be done was not self-evident. The standard advice for top-down analysis is something like "Split each component into three or four until you get to items small enough to handle". But what are the components of "providing service" ? – and how do you split them ?

The first guess was that the components were different sorts of service which might be provided. Alan spent a little time (actually quite a lot of time) elaborating this approach. This is how it works according to the original document, somewhat paraphrased and abbreviated for ease of exposition but not materially changed. First the services and computer resources of the scheme are specified :

Services required :

Different people need : commercial services, information, arithmetic, control systems, word-processing .....

Systems must provide for : packages, utilities, compilers, files, communications, security, accounting, diagnostics .....

Computer resources available :

Computers (different sizes, one processor or many, array processing, dataflow machines, etc.);
Memory (internal, external – discs, etc.);
Devices (terminals, printers ...);
Services (communications, videotex).

In case you wanted to know : "videotex" was a generic name for a number of techniques for transmitting text and low-resolution pictures by telephone wire or in the flyback periods of a television signal. It was right up to date at the time, but has been overtaken by the internet so far as computers are concerned.

Now what ? Well, the operating system has to fit between those two lists, in the sense that it must use the resources to provide the services. This gives a sort of system specification. The task is to design a system which matches the specification, so a set of tools available to the system designer is listed. (The heading in the original paper is "What should happen"; the precise meaning of that phrase is now lost in the mists of time, but "tools" seems to be a reasonably good title.)

Tools :

The system Genie[17] – the mental model built up by someone using the system;
Metaphors (Computer, Virtual machine, Desk top, Office, Database);
Operating system languages (Traditional, Microcomputer systems, Word processor systems);
Help system (How to provide help, What sort of help is needed);

Control structures : tasks and jobs ("Macro" processors – define your own instructions; batch jobs).

That list has a strong smell of desperation. It looks not so much like the bottom of the barrel as what you find was under the barrel after you've rolled it away. It is possible to speculate that (though unfortunately impossible to remember whether) Alan wrote it down as a set of not-very-good best guesses after spending quite some time trying to do better. The proposal finishes with a section on implementation including all the traditional course and some other material, which in the context is, if anything, somewhat less convincing than the list of tools.

The scheme is undoubtedly based on the principle of top-down design : see what we want (W), see what we've got (G), work out what the people should do to use G to do W, decide how to make it go. Good intentions or not, though, the result is unimpressive. Some comments :

1 : It doesn't do the job; the desirable strong structure is not evident. It isn't quite descriptive botany, but (to strain the metaphor) is stuck somewhere in the Linnaean classification, with similarities, differences, and some relationships noted but not really pursued. The result is mildly interesting, but less than effective as a unifying principle. It gives essentially no help in answering questions about what happens when someone wants something new or when the available resources change; a new member is added to a list somewhere, but nothing constructive ensues.

2 : It's too prescriptive. Just listing things isn't enough; everyone wants different software and facilities, so that isn't very promising as a unifying principle. This is to some degree because the chain of reasoning required to carry out the last two design steps (work out what the people should do to use G to do W, decide how to make it go) isn't articulated at all – and that was because Alan couldn't see how to articulate it, given the lists of things to be taken into account.

3 : It does not look like a promising foundation for a lecture course. It is still too bitty – see the "Services required" entry in the first table. Diversity has been achieved, but simply by listing different possibilities, not by introducing any new unifying ideas.

4 : On a more positive note, the development suggests that one can justifiably assert that there is more to operating systems than is found in the traditional course. Analysing what is going on in computer systems leads to some topics at least which are not found in the traditional list.

5 : There are some interesting changes from the traditional pattern in the order in which topics appear. For example, security appears among the requirements, right at the beginning of the discussion, while in traditional treatments it more commonly appears rather later. (For example, in the text written by Silberschatz and Galvin[8] this statement appears in chapter 3 : "If a system is to be protected and secure, precautions must be instituted throughout it". Except for a short section on file protection, the topic then disappears until chapter 13.) This is one example of an overall picture which is interesting : the new proposal is something quite like the old traditional course done backwards. This is consistent with the change of approach from bottom-up to top-down analysis, but nevertheless the magnitude of the change came as something of a surprise.

That is not a favourable report, and shows that there is something lacking either in the proposed scheme or in the analysis. That isn't to say that the proposal was wrong – just that it wasn't appropriate for the requirements of the lecture course. (It was interesting to see much the same structure turn up again recently as a proposal for constructing a commercial system server[18], where careful consideration of the range of services was necessary.) Back there in 1985, though, Alan recognised that something was missing, and deferred action until further progress had been made.

The process turned out not to be simple and straightforward; there has been an iterative refinement, which is why it took a long time. There is not a lot of documentary evidence of the course of

development, which was certainly slow and spasmodic, so it is convenient here to turn to a more explicitly historical account of events.

EVOLUTION.

To speak of events implies that there were some, and signals a change in emphasis from what might be considered the theory of our operating systems course to ways and means for putting it into practice. So far in our development, there has been very little practical to report; but now we have reached the point at which Alan – and later Robert – began to implement the ideas in earnest. What actually happened ?

The answer depends on how you try to work it out. In our first sketch of this material, we collected information from the course descriptions distributed to the students each year, in which the topics covered are listed. They don't tell any lies, but there isn't much detail – so the overall picture is right, but the details are less reliable, or sometimes not there at all. The account we present here is therefore based on the notes we've distributed through the years, and is probably a much more precise – and certainly a much more detailed – picture of what really happened.

BC – 1984 : ("BC" here stands for "Before Creak". Alternatively, think of it as the years of our Lobb.) We have no very precise data on the details of the course as it was presented before 1985, but the scanty evidence available points to a rather traditional pattern something like this :

> What's an operating system ?
> Processes.
> Processor management.
> Concurrent processes.
> Memory management.
> Disc management.
> Other resources and peripherals.
> Safety.
> (Case studies, etc.)

1985 : Alan followed that pattern with very little change. He introduced some topics leaning a little more towards the human interface side of the system – command languages, command files, languages for writing operating systems, etc. – in which he reached out towards the user interface, but didn't really get there. He introduced this material partly in the interests of preserving sanity, but also because it wasn't there, and (following principles as brought out in the previous chapters, though initially without formulating them very clearly) he thought it should be – though he wasn't yet really convinced that further extension in this direction could be justified. Most of the new material was added at the end of the traditional course, and amounted to only one or two lectures. As it turned out, it replaced the case studies; that wasn't intentional, but the consequence of time running out. The case studies stayed in the official course prescription for a couple of years, but made no other appearance, and it finally became clear that there wouldn't be enough time for a satisfactory treatment even without extending the original course. (Examples of practical applications were not ignored; instances were mentioned throughout the course where appropriate.) Clearly, Alan's way of presenting material wasn't the same as Richard Lobb's, an observation which came as no surprise to anyone. (It has been suggested that the only point of agreement between Alan and Richard is that if they agree on anything then one of them must be wrong.)

1985 –1987 : The course didn't change much during this period, mainly because it was always threatened with major revisions "next year" or so; these typically took the form of broodings at the department level about "rationalising the systems courses", meaning that perhaps we should blend operating systems, computer hardware, and maybe network stuff, and do something vague to

unspecified with it, or, then again, perhaps not. Documentary evidence is limited, but there is one fragment written by Alan towards the end of 1985[19], in response to Richard's suggestion for course reorganisation[20], in which Alan rather despondently observed "It's hard to find a unifying theme that comes close to including even most of the bits and pieces which turn up in reasonably ambitious operating systems today", and went on to contemplate splitting the course into pieces, but otherwise silence reigns. During this dark age, Alan made occasional minor changes to the traditional course in response to what he saw as developing needs, but without introducing any major revisions, and without any change in his distaste for the course as it then was. But he kept on thinking, and slowly a different organisation emerged.

1988 : Alan was on sabbatical leave – still thinking, but still only thinking.

1989 : There was not much change to the course material, but a different sort of non-trivial change to the course : Robert arrived, and the dawn came. (Robert protests at this association of ideas. Alan replies that Robert hadn't worked through the night, so isn't in a position to judge.) The revitalisation of Alan's ideas for change was significantly stimulated by Robert's participation. He turned out to be encouragingly receptive to Alan's arguments; himself a survivor of the course from a few years earlier, it could be argued that he was already indoctrinated. Alan was not heard to complain about Robert's background, and from his point of view Robert's support, which was not uncritical, was welcome.

The new movement was marked by a new document[21] describing the state of affairs at the time. Here is a summary of the course taken from that document :

> What do we need from a computer system ?
>
> How do we use computer systems ?
>
> Batch and transaction patterns.
>
> System management.
>
> Communication : instructions, information, assistance.
>
> Data management : store, move, and change data.
>
> Work : execute programmes.

This has shuffled out a lot as compared with the earlier attempt, though it's still guesswork, and it's still fragmented. It continues to depend on lists of topics, rather than a natural development. In structure, it's a tree, with major branches (system management, communication (by people), data management, executing programmes) determined by things the system does. Unfortunately, the subject matter is more like a lattice than a tree. Trying to force it into a tree, and imposing an order on the branches leaves you with many cross-links, and you can't find an order that will sort them out. (Alan found that out – or, at least, became more convinced of it – when he tried to describe the course in Hypercard[22]. The intention was laudable : to present *all* the types of structure to the students, so that they would be able to follow different sorts of dependency link, such as client-server dependencies, design-implementation dependencies, system-component dependencies, and others. After pursuing the idea for some time, he gave up on the grounds of too little time, and far too much frustration with Hypercard.)

The next step – perhaps the most significant of the whole process – must have happened after that document was written, but if there was ever any direct documentary evidence it has been lost. (Some indirect evidence will appear shortly.) This is the step which made the whole course organisation feasible by getting rid of the lists. It was based on the realisation that the real service offered was access to service – that the lists were all very well, but the really clever, and really general, trick was to offer the low-level services which make it possible to think about the lists. These are such universal services as access to the computer system through all sorts of user interfaces, provision for storing files, and provision for performing computations on them. All the

higher-level services of the lists depend completely on these facilities, and catering for these functions is clearly the real task of the operating system.

It is interesting to record that this new understanding was rather embarrassing for Alan. He had for years been advocating what amounted to exactly this approach to the computing course as a whole, and it was some little time after writing it down in 1989[23] that he made the link.

1990 : This year is notable as the date of the first try at a significantly reorganised course. That's one small sentence for a report, but a mighty leap into the unknown for the operating systems course. It was not quite as foreshadowed in the 1989 document; here is the list of topics in the order of presentation :

> What's an operating system ?
> Historical development of operating systems.
> People and computer systems.
> Security.
> Devices attached to computers.
> Disc management.
> Processes.
> Concurrent processes.
> Processor management.
> Memory management.

The first two items are not part of the top-down development, but we thought they were necessary to set the scene for the development. We shall discuss these topics in a little more detail later, but in the meantime they can be regarded as establishing a clear background for the treatment, and as showing how views of the computer system had developed over time.

After that, the analysis begins. "People and computers" and "Security" come first, because together they form the top from which down happens. The argument proceeds in this way :

> To get their work done, people need, primarily, reliable communication with the system, and reliable performance of work inside the system, so they need devices to do it with. We therefore start with the human-computer interface. Then they need somewhere to store the data, which introduces discs. The object of having data is to do something with it, so we proceed to discuss processes, processors, and execution. And, before we can make processors work, we want memory.

While that is a fairly plausible development, it turned out to leave something to be desired in practice. The set of topics covered the ground well (in fact, there was rather little change from the previous course except in emphasis), but the order of treatment was not as effective as we had hoped. That is because we chose the wrong direction from top to bottom for our top-down treatment : unable to escape from our traditional roots, we pointed the course towards the traditional fairly large topics, more or less. In fact, as we found, to do so is artificial, in that it prejudges some of the structure which should come from the top-down analysis. The result is a structure partly defined by analysis and partly by prejudice, and it is not satisfactory. To develop a sound structure, you have to apply the same analysis consistently, so the top-down analysis must be used within the topics as well as around them. The result of the more consistent analysis is to move towards a structure in which all the design considerations are reviewed before the implementation is discussed.

That calm and detached description and analysis of 1990 omits certain features of the change which should be included, if only to deter others from following our lead unless they are very firmly convinced of the eventual benefits. In practice, the changeover was much harder than appears on the surface, and turmoil prevailed throughout much of the year. We had foreseen the requirement for new course notes, particularly important for the new organisation because the textbook provided no backing. (We retained a fairly conventional textbook because there was no alternative and we thought an independent source of information was important.) We had not foreseen that the pattern of course assignments would be completely changed, with the first assignment now dealing with topics which had previously only turned up when students had a much more extensive background in operating systems. Similarly, the mid-course test switched to a completely new set of topics. All in all, it was an exceptionally strenuous year, and not recommended for fun.

1991 : If 1990 was an earthquake, 1991 was an aftershock. The major changes followed from the more consistent application of top-down analysis according to the justification given above. The most obvious change was in the discussion of files and devices; rather than wanting devices, the analysis leads to a requirement for files, regarded in the first instance as abstractions defined as persistent stores for data. It will eventually turn out that files "accidentally" require discs (or something), which in turn require devices, but that is now clearly an implementation topic. The result is that several of the topics which we had moved forward towards the beginning of the course split into two portions, one for design which remains in the early position, and one for implementation which goes back to somewhere near the position of the topic in the traditional course.

The new organisation was deemed (by us) to be much better, and it has persisted, with minor changes, ever since. It is certainly not perfect, and a notable problem is the discussion of processes and memory in the middle. Here we had great difficulty in separating design from implementation, and some work remains to be done on these components of the course.

1992 : The final big change had practically no noticeable effect, because it's a recognition of something that has already happened. It is the classification of the course topics into major groups :

Early developments.
Requirements specification.
Basic requirements : support for execution.
Basic requirements : execution.
Implementation : facilities.
Implementation : methods.

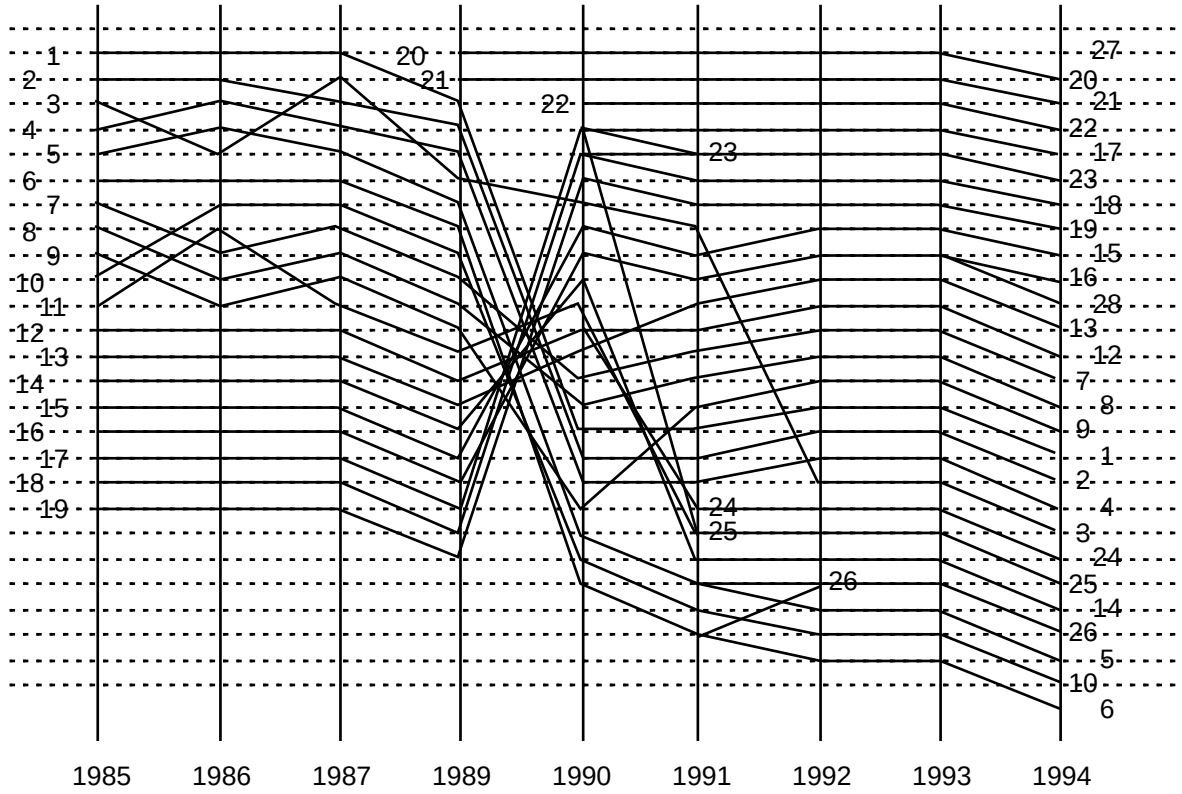
This description emphasises the notion of design far more strongly. Paraphrasing the titles, the course is seen to begin with a survey of the field, then a specification of the characteristics of a desirable system. The specification is refined, and then implemented. We are happy with this view of the course.

1998 : The pattern has not changed much since that time. There has been a little more shaking out as specific improvements suggested themselves, and a few newer developments have been introduced. Perhaps the most obvious change has been notional; we have renamed the last two major topics Implementation and Management, which give a clearer picture of the material covered. And we are beginning to see how to resolve the processes-and-memory knot in the middle[24].

And that's the end, as Alan retired from the workforce in 1998, and half the course passed into other hands.

THE BIG PICTURE.

The diagram below shows the changes in gratifyingly dramatic form. Each line represents a course topic, and the order of topics is as given in the course descriptions for successive years.



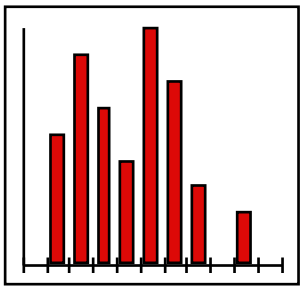
KEY TO TOPICS :

Processes	1
Concurrency	2
Languages to write OS	3
Processors	4
Scheduling	5
Starting and stopping	6
Memory	7
Virtual memory	8
Thrashing	9
Deadlock	10
Disc management	11
Disc files	12
Files and streams	13
Devices	14

Protection	15
Security	16
HCI	17
JCL	18
Managing people	19
What's an OS ?	20
History	21
People and computers	22
GUI	23
Devices	24
Special Devices	25
Configuration	26
What is "operating systems" ?	27
Authentication	28

The slow expansion of the course, so far as distinct topics are concerned, is clear, and the cataclysm of 1990 is very prominent. The shakeout which followed in 1991 is also clearly defined.

We can show the same property of our course in another way : we can compare it with the canonical order of topics for the traditional course which we derived in chapter 2. Here is the graph of topic order against canonical order :



This is very clearly different from the graphs for the various textbooks shown in chapter 2 in a way which has no counterpart there; the predominant slope of this graph is of the opposite sign to that of all the examples given for the textbooks. This strikingly demonstrates the reversed order of topics.

TOPICAL COMMENTS.

The diagram points up certain features of the reorganisation which are not very obvious in our description so far, but which are quite interesting. There are two main sorts of path in the graph, associated with orderly and disorderly behaviour. It is also interesting to note the innovations which appear from time to time, as they are indicative of developments on our thinking on what sort of material should be included in the course.

Orderly behaviour occurs in the movements of significant groups of topics. These groups are the entities which rearranged themselves in the great divide. Within the groups, subtopics might be rearranged in order, but the identities of the groups persist. Examples, with topic numbers from the diagram, are :

Process topics : processes (1), concurrency (2),
processors (4).
Memory topics : memory (7), virtual memory (8),
thrashing (9).
Scheduling topics : scheduling (5), deadlock (10), starting
and stopping (6).
Device topics : disc management (11), disc files (12),
devices (14).
Safety topics : protection (15), security (16),
authentication (27).
People topics : HCI (17), JCL (18), managing people (19),
people and computers (22),
GUI (23).

Disorderly behaviour is exhibited by certain topics which followed less obvious paths in their positions in the course as we tried to fit them in to appropriate contexts. These are more interesting, because they throw light on the development of our ideas of how the course should be organised, and sometimes raise questions about the completeness of the scheme. The examples which appear on the diagram are :

Languages to write operating systems : This topic (topic 3) was usually covered in just one lecture (or quite commonly no lecture, if things hadn't gone too well) of the course, but proved very difficult to settle into a comfortable niche. In content, it is not intended to introduce any very profound or

analytical considerations about the languages, but simply to raise the question, and suggest that there are certain attributes we'd like in a language used for writing operating systems. It was one of Alan's earliest extensions to the original course. It was first placed in the context of processes, because those are perhaps the major structures which the language has to handle, but that never really worked well. Then in 1990 it clearly became a (rather specialised) part of the user interface, which, given a broad definition of "user", does make some sense, but the topic was never very comfortable there because it was only of concern to rather specialised users. It is also significant that it makes more sense to discuss the languages when one knows what sorts of entity one will be writing programmes about, suggesting that a comparatively late position in the course would be more appropriate. It wasn't until we invented the "major topic" grouping in 1992 and introduced implementation as a major topic in its own right that it found a real home. It sits there very happily.

Disc topics : Topics related to using the system disc proved troublesome right from the start. On the one hand they are related to the file system and its management, and on the other to device management. That they are also used for virtual memory is a fact remarked in the older texts, but rarely related to the other topics of disc management. In our course, the topic was originally represented by disc management, covering lower level topics such as sectors and disc space allocation, and disc files, covering higher level material such as blocks, records, and file storage. In practice, the treatment was at the same time too general (insufficient detail on disc management, with no mention of configuration or partitioning) and too specific (for example, discs were not discussed along with other devices). These problems were not solved in 1990, but the 1990 event clarified things sufficiently for us to see a solution in 1991. This was a component of the separation of design from implementation which we mentioned earlier, and in the context of disc topics led in particular to an early treatment of files in the abstract, followed much later by a discussion of how files were implemented on a disc, by that time in the context of other requirements for the disc such as virtual memory, spooling, and so on.

That reorganisation proved fruitful. The abstract files gave an opportunity to talk about what we really want, and to distinguish files from streams (which we had always done, but if files are only discussed in the context of discs it doesn't carry much conviction). The separate treatment of implementation made it natural to place the disc among the other devices, which is sensible, and also to introduce a new class of special devices. These are devices which do several things at once, in contrast to simple devices (printers, keyboards) which only do one thing at a time.

GUI : Alan had introduced elements of HCI as early as 1985, and the topic had been growing steadily though undramatically since then. It had been treated under two headings : the GUI itself, and how it fits into the system, etc., came under the heading of HCI, while just what the interface does was a part of the section identified as JCL. Implementation was not very well discussed. This question was also resolved in 1991. At this time, GUI became a topic in its own right, covering the behaviour and design of the interface itself, while the implementation fitted in very satisfactorily in discussing the function of a terminal as a member of the class of special devices mentioned above. That was very neat, and also made more sense of some of the material we had covered under the HCI and JCL headings.

Devices : The general low-level treatment of devices had always been lateish in the course, though not necessarily for any particularly good reason. Nevertheless, we dutifully moved it to a much earlier position in 1990, because for any sort of top-down discussion you obviously have to talk about devices before discussing specific topics like discs and files. The change turned out to be a catastrophe. In hindsight, that is probably because there really wasn't a good reason for the original late position of devices – that is, it wasn't there for a definable bottom-up reason, but just by accident. We made a sort of sense of it in the 1990 "argument" presented above as part of the historical summary, but on closer examination it's hard to detect much real basis for the assumption. As much as anything, the anomalous position of devices in the course was the factor which triggered the minor rearrangement of 1991.

Innovations are the third category of interesting irregularities. They come into being in several ways – new topics become of interest in the current literature, an existing topic is divided, or components of existing topics are merged under the new heading. Here they are in roughly chronological order.

Introspection arrived in 1989, in the form of sections on the nature of an operating system and on the historical development of operating systems. There were two reasons for introducing these topics. First, we had found that as our students became more accustomed to the Macintosh operating system, their notions of what the system was became less precise. Earlier students, brought up on comparatively old-fashioned CP/M systems, had to know quite a lot about how the systems worked in order to survive their early years in the department, and entered the operating system course with quite a clear idea of the structure of the system; Macintosh students didn't. It was therefore appropriate to spend a little time identifying the purpose of the system (and in any case we wanted to do that to introduce our service-provider model); a brief treatment of the history of operating systems was also helpful, as it emphasised the shift of focus from efficient use of machines to efficient use of people, and, by showing what people expected of computers, gave a valuable grounding in the eventual aims of the system. You can't do a top-down analysis unless you have some idea of the bottom, and the historical treatment gave this useful background.

People and computers branched out as a topic on its own in 1990. Some of the material had been presented in earlier years as part of HCI and JCL, but now suddenly it had become the main topic of the course, and reasonably required more emphasis. In particular, topics such as system metaphors and people's mental models of the system became important.

Configuration – including both system configuration before you buy it and configuration on system startup – arrived in 1992. It wasn't really brand new, as again there had been some material in the discussions of starting and stopping, with another link from scheduling at the management level, and also several references to the importance of being able to gather system information as a base for system development and designing new systems. It sprouted anew because suddenly, with the 1992 relabelling, there was a natural place for it to fit, and collecting the scattered material together gave a more coherent and satisfactory treatment.

IS IT FINISHED ?

More or less. Or, to put it another way, no, and it never will be.

This ambivalent non-answer is the best that we can offer. We believe that the current organisation is fundamentally satisfactory, in that it embodies useful principles which define the subject and order the material in helpful ways. It is an accommodating structure in ways which represent a significant improvement on the older model, in that new topics easily find a natural place within the course, and any consequent reorganisation follows automatically from the same principles.

On the other hand, there remain some areas where we are less than satisfied with our current analysis. Typically, these are areas in which we have not managed to achieve a clean separation between design and implementation, or where we find that several topics seem to be inextricably interconnected. We remain (reasonably) confident that there are solutions to these problems, and that they can be found by careful analysis and the elimination of preconceptions. Other similar problems have turned out to be tractable; the clearest example is the case of the disc operations, which were cleared up by insisting on a stage of abstract design of how a file should be, without introducing notions depending on experience with disc files. The result is a clearer understanding of the topic, and is certainly better adapted to coping with files on any new media which might appear in the future. We therefore hope that the remaining problems will yield to similar careful analysis, and that the result will be a more useful description of the areas concerned.

The biggest current knot in the course is (and has been for some time) the memory-processor-processes combination. These are so tightly linked together that it's difficult to find a property of one that isn't qualified by properties of the others, so, technically speaking, they form a single topic. It gets even worse with virtual memory, which also involves a disc, but is so closely associated with processor hardware and with processor exceptions that we want to talk about that too. Pedagogically speaking, though, it's hard at the moment to see how to present it that way. The three component topics (four with virtual memory) are clearly identifiable, and easy to discuss. That's why virtual memory still precedes processes in our scheme, though it's very clearly just an implementation technique for providing memory services, and should come much later. We are at present cautiously optimistic that we have a prospective solution to the problem[24], and we toyed with it in 1998, but the attempt fell short of a serious implementation.

More generally, we are still not entirely satisfied with the separation between design and implementation, but we accept that in one sense at least perfect separation is impossible. This is a consequence of having a bottom to the analysis; whatever you decide at the top must eventually be implemented at the bottom, and if the resources available at the bottom cannot provide the functions required at the top, then the design effort is doomed to failure. In such a case, implementation questions affect the whole of the design. To give a fanciful example, if you wanted to build an aeroplane and the only material available was bricks and mortar, the task might not be impossible, but you would certainly be unable to use many of the design techniques of modern aircraft engineering. Closer to home, if your specification included real analogue computation, you could not satisfy the requirements if you were restricted to digital components.

To that extent at least, therefore, pure top-down design is an illusion, as there will always be some bottom-up constraints. (That was one of our reasons for including the historical review in the course; it gives some idea of the sorts of behaviour we can expect at the bottom levels of our systems and of the nature of the components from which our systems must be constructed.) The first few steps down from the top are relatively free, and you can invent any useful abstractions you like, leaving them to be implemented at the next level down. As you get near the bottom, though, you have to be careful that your abstractions can be implemented effectively with the resources which are there, and that's a constraint.

That does not mean that you must only design systems which you know that you can build. It is important to consider the design of systems which at present you can't build in order to identify the facilities you will need to build them in future. Requirements perceived for software systems development have frequently influenced hardware design; interrupts, memory protection, and direct memory access are cases in point. To give another example, it would be nice to have the system offer a wide variety of memory models to run different sorts of programme, and that's what you'd design if you could; we mention this topic in the memory part of the course. The received wisdom is that, in practice, running foreign memory models without hardware support is just too slow, and you have to put up with what you're given – but it's interesting that suddenly everyone is very happy to run Java virtual machines, so perhaps the received wisdom is questionable, and our discussion is not as impracticable as it might have seemed.

HOW IT IS NOW.

The general resemblance of the course development to a design process will be clear; we begin by drawing up a specification, then explore means of satisfying it. We emphasise, though, that the resemblance remains at the schematic level. We are deliberately not trying to design a single system in our course, as an important part of our aim is to offer a general coverage of the subject of operating systems. The clear strong connections come about because we have ordered our approach following a design model; first we review requirements, then proceed to discuss possibilities for high-level design, low-level design, and so on.

CHAPTER 5

NOT REALLY CONCLUSIONS

We end with some final remarks which don't really conclude anything except the report. They cannot properly be regarded as conclusions of our arguments, because nothing is really concluded; we have presented our views, and devised, and delivered, a lecture course in which these ideas are embodied. The result has not been to prove anything, except that it's possible to present the lecture course, and we knew that. We are reasonably happy with our experience, but we do not propose to try to convince anyone that this is the only possible way to go.

We would go so far as to suggest that the ideas we have put forward are worthy of consideration in any discussion of operating systems as an object of academic study. If nothing else, they offer an alternative view to that promoted by the traditional course in several topics, and some of the views are illuminating; our comments below point out some of the illumination which we specially noticed. Whether the type of course which we advocate is the way of the future is for the future to determine, but we'd like to think that at least some of our work will bear fruit, and we're happy to have got there (so far as we know) first.

WHAT'S NEXT ?

Apart from the improvements in the memory-processor-processes area which we have discussed, we have formed no detailed plans for immediate further developments. We believe that the structure as we've described it is now flexible enough to adapt easily to new features in operating systems as they come along, and that the next requirement is for comparatively gentle evolution as we assess such features and incorporate them where we believe it to be useful for the course.

So far, it seems that the structure copes very well with new developments. We shall illustrate this adaptability by suggesting solutions to an impending problem.

The impending problem is the question of **middleware**. The meaning of "middleware" seems to be less than precisely defined[25], but it is usually connected with some aspect of building useful activities in which several different programmes cooperate to produce the desired result. Client-server systems are perhaps simple examples; others are the Macintosh OpenDoc and Microsoft OLE techniques[26].

The first question is whether we should take any notice of middleware. We discussed the question of how to decide what was and what was not a legitimate topic to include in the course, and the criterion we adopted was whether or not the topic was effective in bringing computing services to people. By this measure, middleware certainly qualifies; the ability to combine the services of several programmes in one activity is useful in many fields.

We therefore ask how we should incorporate a description of middleware into the course. How can support for such systems be provided at the operating systems level ? The structure as presented in our course copes well. The primary change is in the *requirements specification* : what people require can no longer be provided by executing a single programme, so we have to direct attention to activities which involve simultaneous cooperative execution of several programmes. With this change in definition, everything else follows. Processes turn up in much the same way, but the process state becomes a tree of threads rather than a simpler structure; interprocess communication also appears, but that would probably turn up at about the same place anyway. The designer may choose whether to run the threads concurrently or not, and the methods we already describe for process management and interprocess communication and synchronisation can be used in natural ways. The detailed presentation is a question which we have yet to address, but it seems likely that rather little change will be necessary.

How might one add a discussion of middleware to the traditional course ? It is undoubtedly possible, but just how to manage it is not so obvious. Should one introduce the added complexity of

interprocess communication, and perhaps processes with multiple threads, right from the start, with rather little in the way of motivation, or should it be regarded as an advanced topic, treated after the simple cases have been discussed, and necessitating a review of those cases with modifications for the more demanding context ? We prefer to solve the problem in our way.

DOES IT WORK ?

The original aim when this development process began was to devise an operating systems course which better represented the subject and embodied a more satisfactory and effective structure. Now we have our new course; does it meet our expectations ? This is not an easy question to answer, because we have not carried out the sorts of measurement which might give us relevant information so far as the effectiveness in presenting the material is concerned, and judgments about "more satisfactory and effective structure" are not easy to make. The best we can manage in this final assessment is therefore to present some notes based on anecdotal evidence. We are not unhappy with the result, and so far as we can tell it does the student no more harm than the older course. We let the rest of the evidence, feeble though it might be, speak for itself.

- It has turned out to be a "people-centred" course. This is not an unwelcome development in any way, but it was not an explicit aim from the start. We had unexpected confirmation of this view from a student who took the course (most unusually) in the context of a sociology degree. She was an excellent student, and performed brilliantly throughout the course. Hearsay evidence from a mutual acquaintance reports her opinion that she enjoyed it because it fitted very well with the principles she had learnt in sociology. We are intrigued by this remark, and might some day follow it up. (We have resisted suggestions that we should require Sociology 101 as a prerequisite.)

  It is unfortunate that the emphasis on people rather than machinery turns out to be trendy, which was certainly not our intention. It is much more significant that the switch in view corresponds reasonably well to the switch from the use of computer-centred to people-centred criteria in operating system design.

- We are satisfied that the new organisation is conducive to better scholarship. It gives a deeper structure to the material, and accounts for why everything's there. While the same was true, at least to some extent, for the older course, the reasons were external to the explicit course structure; in the new approach, they are explicitly taken into account, and any change in the assumptions or requirements leads directly to the consequences for design.

- The boundaries of the subject have changed a little. At the high-level end, the criteria for including topics under the heading of operating systems are inclusive rather than exclusive; anything which helps you to get your work done and is of general utility fits in somewhere. We believe that this is a realistic approach, and properly represents how people build up their systems. At the low-level end, the situation is not quite so comfortable. The effective boundaries here are set by other courses presented in our department, and are inevitably artificial. The most direct interactions are with the hardware and data communications courses, and in both cases the interactions are significant. For example, the use of processor caches complicates scheduling in multiprocessor systems; and the operation of distributed systems depends intimately on the data communications facilities used. Perhaps the reorganisation ideas[20] of 1985 advocating some sort of merger between the operating system, hardware, and communications courses weren't so silly, but it remains unclear how to present the unified view in practice.

- We have already remarked that the structure seems to work insofar as incorporating new material is concerned. We mentioned the question of middleware as an example; other topics which we have considered include pen interfaces (which easily fits, but is not deemed sufficiently important for more than a passing mention), and log-structured files and multimedia files in real-time (which also fit, and are described in a little detail as examples of useful techniques of present and future importance).

- Another of Alan's principles, on a par with "Every subject is interesting" is "Every good idea gives you new insights". Are there any new insights from this view of operating systems ? Yes, there are. Some are matters of emphasis : the relationship between segments and pages becomes very clear, with segments appearing as a natural consequence of the behaviour of processes, and paging no more than an expedient for implementation. Others offer new views of old ideas : interrupts come out of the dispatcher and appear as one of five scheduling levels. Some are even rather spectacular : computer memory, far from being a vital part of a computer system, is seen as an inconvenient and clumsy implementation requirement[24]. It is possible that not everyone will believe all these assertions – but that is unimportant; if people are encouraged to think about the alternative views, that will be enough.

- The students' point of view is more difficult to assess. The change does not seem to have driven people away – indeed the course enrolments have fairly steadily increased, but as there's no alternative that means little. The students' assessments of the course went up a bit when we started (from utterly abysmal to abysmal), but haven't changed much since. There is always dissatisfaction among the fanatical computists because we don't speak C much and don't spend all the time talking about Unix, but we wouldn't do that in a traditional course either. There's some confusion because we don't always say in the lectures precisely what's in the textbook, but that has nothing to do with the relative merits of the two treatments.

  Conversations with a non-random selection of former students of the course, strongly biased towards students continuing to postgraduate studies (and excepting Robert), is on the whole mildly positive, and occasionally strongly so. The extreme case was one student who was so impressed by the notion that everything about operating systems could be related back to what people wanted to do with computers that he declined to take the postgraduate operating systems course on the grounds that it had nothing more interesting to say – gratifying, but by no means what we intended to happen.

- It is obvious that our approach is not the only way to cover the topic. It is equally obvious that we have more or less arbitrarily imposed a structure upon the material. On the other hand, one can say the same about any other structure, and we continue to believe that ours at least gives as coherent a treatment as any other that we know.

- It's worked on one level – Alan is mildly more interested than he was before.

REFERENCES

1 : G.A. Creak : *Monitoring the DEC-10's performance*, unpublished Working Note AC40 (29 August 1984).

2 : G.A. Creak (ed) : *The Zeno system*, Auckland University Computer Centre Technical Report #22 (June, 1984).

3 : H.M. Deitel : *An introduction to operating systems* (Addison-Wesley, 1983).

4 : A. Conan Doyle : *The Sign of Four*, 1890. (See *Bartlett's Familiar Quotations* (15th Edition, Little, Brown and Co, 1980).)

5 : A.S. Tanenbaum : *Operating Systems – Design and Implementation* (Prentice-Hall, 1987), pages xiii, xiv.

6 : G.A. Creak : *Academic structure in the study of operating systems*, unpublished Working Note AC121 (March, 1998).

7 : P. Brinch Hansen : *Operating System Principles* (Prentice-Hall, 1973).

8 : A. Silberschatz, P.B. Galvin : *Operating system concepts* (Addison-Wesley, fourth edition, 1994).

9 : P. Paradinas, J.-J. Vanderwalle : "New directions for integrated circuit cards operating systems", *Op.Sys.Rev.* **29#1**, 56 (January, 1995).

10 : P.C. Treleaven, D.R. Brownbridge, R.P. Hopkins : "Data-driven and demand-driven computer architecture", *Computing Surveys* **14**, 93 (1982),

11 : A. Meyer : "Pen computing", *Sigchi Bulletin* **27#3**, 46-90 (July, 1995).

12 : S.E. Madnick, J.J. Donovan : *Operating systems* (McGraw-Hill, 1974).

13 : B.E. John, J.H. Morris : "HCI in the School of Computer Science at Carnegie Mellon University", *Human Factors in Computing Systems INTERCHI'93 Conference Proceedings* (Addison-Wesley, 1993), 49.

14 : I.M. Flynn, A.M. McHoes : *Understanding Operating Systems* (Brooks/Cole, 1991)

15 : P.J. Denning, D.E. Comer, D. Gries, M.C. Mulder, A. Tucker, A.J. Turner, P.R. Young : "Computing as a discipline", *CommACM* **32#1**, 9-23 (January, 1989), page 20.

16 : G.A. Creak : *Operating systems Proposed new course structure*, unpublished note (June, 1985).

17 : R. Tagg, M. Sandford : "Where to now that the mouse has arrived ?", *Computer Bulletin Series 2* **#42**, 2 (December, 1984).

18 : F. Manola : "Interoperability issues in large-scale distributed object systems", *Comp. Surv.* **27**, 268-270 (1995).

19 : G.A. Creak : *"Computer organisation" reorganisation revisited*, unpublished Working Note AC43 (25 November 1985).

20 : R.J. Lobb : *"Computer organization" reorganization*, unpublished note (November 1985).

21 : G.A. Creak : *Operating systems course structure*, unpublished Working Note AC75, (4 December, 1989).

22 : G.A. Creak : *Three Hyperforty : the structure of a course in operating systems*, unpublished Working Note AC76 (October, 1979).

23 : G.A. Creak : *Computing – a course structure*, unpublished Working Note AC73 (November, 1989).

24 : G.A. Creak : *Introducing processes in a course on operating systems*, unpublished Working Note AC113 (August, 1997).

25 : P.A. Bernstein : "Middleware : a model for distributed system services", *Comm. ACM* **39#2**, 86-98 (February, 1996).

26 R. Shah : "Componentware on the horizon", *Sunworld Online* **10#2** (February, 1996) (http ://sunsite.icm.edu.pl/sunworldonline/swol-02-1996/swol-02-connectivity.html)