WORKING NOTE AC95

Alan Creak
30 December 1994

ADVANCED OPERATING SYSTEMS COURSE

This note exists because Bruce Hutton asked me to write down what I'd expect from an advanced operating systems course. I'd never really tried to analyse my thoughts about operating systems at stage 4 before, because I'd never wanted to be concerned with such a course. Now I've done it, I find that the exercise has helped to clarify some ideas about the stage 3 course, and has otherwise been interesting and illuminating; I hereby thank Bruce for pushing me into it.

The note is written in the context of the current operating systems course and fragment thereof in which the subject is presented in this department's undergraduate course : an introduction at stage 2 as part of a Computer Systems course, followed by a full Operating Systems course[1] at stage 3. The ideas presented are those which I see as appropriate to a postgraduate course following on from the existing material.

The context is significant, because it causes me to begin from a state which I don't think is ideal. I'd prefer to get rather more of our current stage 3 material (which is essentially introductory in nature) into stage 2, where I would like to see all the major system components introduced and described as separate entities. We could then spend more time in stage 3 studying the integration of, organisation of, and interactions between, the parts, and looking in more detail at real operating systems; that's in line with my wish[2] to define the stage 3 level as that which assures competence in current practice. My Grand Plan then defines stage 4 as concerned with current research and possible future developments.

In practice, we don't get that far, though recent proposals[3] for the stage 2 systems course (largely due to Bruce) are encouraging. My discussion here therefore represents a compromise between my ideal view and reality.

HOW COULD THE COURSE BE APPROACHED ?

An advanced course is usually expected to develop material presented in a preceding course in some selection of the directions of greater complexity (more complicated examples, etc.), greater rigour (deeper theory, etc.), more realistic cases (case studies), recent developments (modern systems, research in progress) and doubtless others which haven't come to mind. All these would work for operating systems, and in many cases overlap.

Complexity.

There is no shortage of complexity; pick any two system components, and study their interactions. Measurements of the performance of real systems or simulation studies can provide the raw material. This is interesting and illuminating, and demonstrates the trade-offs and compromises which characterise operating system design – and also shows how hard it can be to get useful information out of real systems !

On the other hand, apart from illustrating the problems, it doesn't do much. So far as I know, there's little enough in the way of solutions to the problems.

This approach emphasises the interactions between the parts of the system. System architecture might fit in well.

Rigour.

It's possible to give (mathematically or logically) rigorous treatments of some operating system components, but always (I think) of components in isolation, and sometimes, at least, of simplified models. Examples are aspects of security, virtual memory, timing in distributed systems, scheduling, etc. This is good for giving insight into some of the problems, but in many practical cases complications from interactions with other factors obscure the ideal behaviour.

This approach won't give full coverage, because there is as yet, so far as I know, no overall theory of operating systems. At the system level, one can consider approaches like queueing models for statistical behaviour, or Petri nets for problems of synchronisation and parallelism, but again neither copes very well with the complexity of real systems.

This approach emphasises the system behaviour (and sources of trouble) at a low level. The effects of, and problems arising from, interactions are likely to be missed.

Realism.

Rather than concentrating on the behaviour of the system components, in isolation or in interaction, one can begin with a whole system and analyse its operation. For example, one could inspect some running systems, and observe how they provide services and solve problems. Experiments with real systems would again be appropriate.

This view would be particular suited to introducing the unfashionable, but necessary, parts of systems, such as accountancy, error management, system standards and conventions, etc.

This approach concentrates attention on complete systems. Perhaps system architecture would fit in here, too.

Advances.

The substance of the course is the study of new developments in research and practice. New material can be compared with current practice, and explicit improvements, drawbacks, brilliant insights (not often), etc. identified and (as far as possible) quantified.

The coverage possible would depend on the availability of topics to discuss, but there is usually quite a lot going on; and one can argue that in an advanced course concentration on current developments is entirely appropriate. There are certainly recent developments in the areas of communications, distributed operating systems, security techniques, object-oriented systems, open systems, distributed and real-time file systems, etc., all of which could reasonably be included in an advanced course.

This approach largely concentrates on components, because developments are typically fairly specialised in nature.

WHAT'S THE OBJECT OF THE COURSE ?

I've suggested that the stage 4 course should be about current research and future developments, but I don't think that it's sufficient just to list some interesting topics. The aim of studying the material should be to draw something from it, not just to admire it in its own glory.

I suggest that an appropriate aim is to identify guidelines and principles which might help in developing new and better operating systems. (In these terms, my aim at stage 3 is to identify the structural principles of current operating systems so far as they are determined by satisfying people's requirements for service.) Here I can step out bravely into the unknown, because I'm not likely to be doing the course. I don't know just how I'd achieve this aim; my best idea at the moment is to ask at each stage of the material what we can learn from the topic under discussion which might be a useful guide in the future.

WHAT I'D DO.

Looking at my (undoubtedly partial) list of approaches and my statement of the course object, it's clear that there are very many ways in which a stage 4 course could be organised. That being so, my own guess isn't necessarily very significant, and is likely to reflect my own preferences at least as much as the imperatives of the course requirements. For what it's worth, though, here it is.

The course has two parts. The first elaborates the realism approach, and the second concentrates on the advances approach.

Part 1 is based on comparative case studies of a few (about three ?) very different modern systems (e.g., a pen-based system, Amoeba, a real-time system), with emphasis on the different ways in which the same problems are solved in each case. For example : how is the user interface managed ? – how is the file system arranged ? – what sort of communications are supplied ?

The answers to the questions should take into account the different demands on the systems, and the constraints under which solutions had to be developed.

Part 2 is based on discussion of areas of current active research interest. (It would obviously be sensible to choose the topics discussed in part 1 with an eye on part 2, though they need not be identical.) A possible approach would be to look (critically) at some recent publications, and see how the material they describe fits into the existing operating system structures.

REFERENCES.

1 : G.A. Creak, R. Sheehan : Course notes for 07.340, 1994.

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

3 : Proposed syllabus for the 415.210 course : various authors, December, 1994. (See appendix.)

APPENDIX

Excerpt from the proposed syllabus for the 415.210 course

(Transcribed from the original electronic mail from John Hosking, 25 November 1994.)

An elementary introduction to the UNIX operating system. In particular, including using a terminal emulator, logging in and out, changing a password, use of the C shell, file and directory management, moving around directories, redirecting input/output, printing files, using the mail system, man, make, cc, grep, and find commands, and a UNIX editor. Use of control characters to kill or stop processes, etc. A general idea of the typical commands, system calls, and library routines available on any system, such as UNIX.

The principles behind multi-user operating systems. Hardware features necessary to achieve computer security. System registers. Virtual memory. User and kernel mode. Privileged instructions. System calls. Exception and interrupt handling. Process management. Context switching. Input/output. Achievement of exclusive access to operating system data structures. Operating system startup. Creation and termination of processes, and execution of programs. Flaws in computer security.

Function and organization of peripheral devices. Buses, processors, memory, terminals, printers, tapes, and disk drives. Representation of files and directories. An elementary introduction to data communications.