In Support of Domain Structure for Operating Systems

Douglas Cook,
Computer Science Department,
University of Exeter,
England.

Abstract

One approach advocated in the search for better designed and more reliable operating systems is to base the design on the use of small protection domains. This paper presents empirical evidence to show that, with a suitable architecture, the overheads associated with using small protection domains do not make this an impractical approach.

Introduction

Using small protection domains as the basic building blocks for the construction of an operating system is not a new idea. For example, Linden [1] advocated this approach for the design of secure and reliable computer systems. From the design point of view it is aesthetically pleasing to be able to describe a complex software system in terms of a number of modules, each of a size which makes it easy to understand, which can interact only in a well-defined way. An architecture which implements these modules as separate protection domains, each with its own fully encapsulated address space, both encourages such modularisation and constrains the interaction of the modules. But the world of operating systems is a pragmatic one with no place for "elephants in best Carrara marble" [2]. Thus, despite Linden's optimistic and so far unfulfilled hope that "with appropriate hardware support, the overhead to switch protection domains could be comparable to that of a simple procedure call", the case for domain structured operating systems requires the support of empirical evidence from an actual implementation to show that the design does not impose an unacceptable loss of performance. This paper presents such evidence.

Most of the evidence was obtained in experiments carried out as part of an evaluation of the CAP project at Cambridge University [3]. An interim evaluation of the project is given in Needham [4] and the basic architecture of the CAP computer is described in Needham & Walker [5]. For the purposes of this paper we are concerned only with the features of the CAP computer which support the use of program modules which have their own fully encapsulated address spaces. Such modules are known as protected procedures and are the basic building blocks of the CAP Operating System. A protected procedure can only be entered on presentation of an ENTER capability for that procedure. The switching from one protection domain to another, by entering a protected procedure, is effected by microprogram, the instruction concerned being the ENTER instruction.

In the CAL system [6, 7] and in Hydra [8] protection domain switching is implemented entirely in software and consequently the overheads incurred by switching domains are high. This was one reason for the premature termination of the CAL project. In the case of Hydra the high overheads, for example 51 ms for the switch to the filing system domain [9], have meant that protection domains have in practice been much larger than would have been ideal from the design point of view [10].

The experience of the CAL and Hydra projects suggests that an operating system built from small protection domains will be unacceptably inefficient if domain switching is not implemented by hardware or microprogram. What can we conclude from projects where there is such support for domain switching? There are two considerations which warrant attention: (a) how long does it take to switch protection domains, and (b) how often is this done? Table 1 gives the time taken to switch from one protection domain to another and back again for the CAP system [3] which implements protection domain switching by microprogram. The time taken for a simple procedure call is also given.

time, t_1, for domain switch	time, t_2, for simple procedure call	ratio $t_1 : t_2$
0.24 ms	12 μs	20:1

Table 1: CAP instruction timings

These figures seem to lead us to the conclusion that the overheads are still uncomfortably high: they are certainly a good deal higher than those incurred by a simple procedure call. However, much less checking needs to be done to validate arguments than with a conventional system and this goes a good way to redressing the balance.

We now turn to the question of how often domain switching takes place in practice. It would be a simple matter to count the number of domain switches there are in a given time interval. However, it is of greater interest to assess how much extra work is being done because the operating system is composed of small protection domains by comparison with a more conventional design. Experiments were carried out with the CAP Operating System to investigate this question.

A simple 2-state machine, operating in one state when running ordinary programs and in the other when running so-called privileged programs, was taken as the basis for comparison. This was chosen in preference to a more modern segmented architecture such as Multics [11] because the author wished to compare the CAP operating system with one which was at the opposite extreme from the segmentation point of view. With such a comparison it is easier to see the benefits which come from the use of small protection domains. In this paper the two states are referred to as problem state and supervisor state respectively. When the computer is in supervisor state it runs without protection. Typically, when an ordinary program calls for one of the services provided by the operating system, the computer switches to supervisor state, executes the code to perform the desired service, and then reverts to problem state. Thus, the operating system runs in the same protection domain irrespective of which service it is performing. There is no protection while the computer is in supervisor state so the operating system's privilege is maximised rather than minimised: this is in marked contrast to the situation with the CAP Operating System [12, 13]. In the 2-state computer only a single protection barrier is provided whereas with a domain-structured system there are many, each corresponding to a switch from one protection domain to another.

In the CAP Operating System a user requests an operating system service by entering the appropriate protected procedure. This protected procedure may, in turn, call another, and so on. Although a protection domain switch is more complicated than the switch to supervisor state in a 2-state machine, the first domain switch in response to a call for an operating system service and the change to supervisor state both represent the first protection barrier encountered in providing the service. The differences between the protection provided in the operating systems of the CAP and a 2-state computer are that in the CAP Operating System (a) the first domain switch is not always to the same protected procedure, and (b) more than one domain switch may take place during the performance of an operating system service. Experiments were done in which each time an operating system service was called the number of domain switches which took place during the performance of that service was counted. Results were collected from the running of two programs, the Algol 68C compiler [14] and Genesis, the program used to generate the CAP Operating System and to initialise its protection environment. These two programs were selected because (a) they were much used, (b) they were substantial programs, and (c) one, the compiler, was compute-heavy whereas the other handled very large volumes of data.

Details of experiments

The structure of the CAP Operating System is such that there is no clear dividing line between what is part of the operating system and what is not, so a more-or-less arbitrary decision had to be made as to which protected procedures were considered to be included in the operating system. When a user logs in he is allocated one of a stock of USER processes and that USER process is able to call a number of protected procedures as soon as it is allocated. Those protected procedures were considered to be in the operating system. The operating system services available in a USER process are referred to as primary services.

Monitoring code was inserted into the microprogram and operating system to detect when a USER process requested a primary service and to count the number of ENTER instructions obeyed until that service had been completed, referred to as the consequential ENTERs for that primary service. Further details are given in Cook [3]. The results are summarised in Table 2 and show that, although the CAP Operating System is built from small protection domains, the source of a typical protected procedure being only about 200 lines of Algol 68C, use of the operating system does not involve many more switches between protection states than the single switch per call of the simple 2-state design.

	Algol 68C	Genesis
Number of primary service calls	11786	1733
Number of consequential ENTERs	2810	6135
Consequential ENTERs/primary service call	0.24	3.54

Table 2: CAP Operating System-consequential ENTERs

Both programs made use of primary services provided by the five protected procedures which look after file directory management, store management, allocation of i/o devices, interactive i/o, and spooled i/o. In addition Genesis made one call to the protected procedure which provides the interface to the central coordinating process. The main difference between the two programs is that, because the Algol 68C compiler is engaged in processing the information stored in segments, it generates a lot of input and output activity whereas Genesis is concerned mainly with manipulating complete segments and consequently makes more use of directory and store management services.

Evaluating the benefits which accrue from the use of small protection domains is more difficult than assessing the costs. One advantage is the ability to separate logically distinct operating system services into separate domains. As already mentioned, the Algol 68C compilation called for primary services from five different protected procedures and in running Genesis primary services were called from six protected procedures.

Conclusion

The subjective reaction of those involved in developing the CAP Operating System has been that their work was made easier and more effective by the underlying domain structure. This paper has presented empirical evidence to support the claim that, with appropriate hardware or microprogram support, the costs associated with switching protection domains are low enough for domain structure to be a realistic basis for the design of an operating system. The evidence suggests that in deciding on the number and size of domains to use, the designer need not be unduly constrained by the cost of switching from one domain to another. However, the results are based on a limited experiment with a single implementation and additional results from other systems are needed to add weight to the case. It is hoped that the results presented here will encourage such further experimentation.

References

1. Linden, T.A. "Operating Systems Structures to Support Security and Reliable Software", Computing Surveys, Vol.8, No.4, December 1976, pp. 409-445.

2. Needham, R.M., Hartley, D.F. "Theory and Practice in Operating System Design", Proc. 2nd Symposium on Operating Systems Principles, Princeton, October 1969,pp.8-12.

3. Cook, D.J. "The Evaluation of a Protection System", Ph.D. Thesis, Cambridge University Computer Laboratory, April 1978.

4. Needham, R.M. "The CAP Project: an interim evaluation", Proc. 6th Symposium on Operating Systems Principles, Purdue University, 16 - 18 November 1977, pp. 17-22.

5. Needham, R.M., Walker, R.D.H. "The Cambridge CAP computer and its protection system", Proc. 6th Symposium on Operating Systems Principles, Purdue University, 16 - 18 November 1977, pp. 1-10.

6. Sturgis, H.E. "A postmortem for a time sharing system", Ph.D. Thesis, University of California, Berkeley, 1973.

7. Lampson, B.W., Sturgis, H.E. "Reflections on an operating system design", CACM, Vol. 19, No.5, May 1976, pp. 251-265.

8. Wulf, W., Cohen, E., Corwin, W., Jones, A., Levin, R., Pierson, C., Pollack, F. "HYDRA: the kernel of a multi-processor operating system", CACM Vol.17, No.6, June 1974, pp. 337-345.

9. Almes, G., Robertson, G. "An Extensible File System for Hydra", Proc. 3rd International Conference on Software Engineering, Atlanta, Georgia, May 1978, pp. 288-294.

10. Cohen, E., Jefferson, D. "Protection in the Hydra Operating System", Proc. 5th Symposium on Operating Systems Principles, Nov. 1975, pp. 141-160.

11. Schroeder, M.D., Saltzer, J.H. "A Hardware Architecture for Implementing Protection Rings", CACM, Vol.15, No.3, March 1972, pp. 157-170.

12. Cook, D.J. "Measuring memory protection", Proceedings 3rd International Conference on Software Engineering, Atlanta, Georgia, May 1978, pp. 281-287.

13. Cook. D.J. "Measuring memory protection in the CAP Computer", Proc. 2nd International Symposium on Operating Systems, IRIA, France, October 1978.

14. Bourne, S.R., Birrell, A.D., Walker, I. "ALGOL 68C Reference Manual", Cambridge University Computer Laboratory, July 1975.